Understanding Code Changes Practically with Small-Scale Language Models

Cong Li
Zhejiang University
Hangzhou, China
chifei.lc@antgroup.com

Zhaogui Xu
Ant Group
Hangzhou, China
zhengrong.xzg@antgroup.com

Peng Di
Ant Group
Hangzhou, China
dipeng.dp@antgroup.com

Dongxia Wang
Zhejiang University
Hangzhou, China
dxwang@zju.edu.cn

Zheng Li
Ant Group
Hangzhou, China
zheng.lz@antgroup.com

Qian Zheng
Ant Group
Hangzhou, China
zhengqian.zheng@antgroup.com

ABSTRACT

Recent studies indicate that traditional techniques for understanding code changes are not as effective as techniques that directly prompt language models (LMs). However, current LM-based techniques heavily rely on expensive, *large* LMs (LLMs) such as GPT-4 and LLAMA-13B, which are either commercial or prohibitively costly to deploy on a wide scale, thereby restricting their practical applicability. This paper explores the feasibility of deploying *small* LMs (SLMs) while maintaining comparable or superior performance to LLMs in code change understanding. To achieve this, we created a small yet high-quality dataset called HQCM which was meticulously reviewed, revised, and validated by five human experts. We finetuned state-of-the-art 7B and 220M SLMs using HQCM and compared them with traditional techniques and LLMs with ≥70B parameters. Our evaluation confirmed HQCM's benefits and demonstrated that SLMs, after finetuning by HQCM, can achieve superior performance in three change understanding tasks: change summarization, change classification, and code refinement. This study supports the use of SLMs in environments with security, computational, and financial constraints, such as in industry scenarios and on edge devices, distinguishing our work from the others.

CCS CONCEPTS

• **Software and its engineering → Software maintenance tools**.

KEYWORDS

code change, code review, language model, LLM, SLM

ACM Reference Format:
Cong Li, Zhaogui Xu, Peng Di, Dongxia Wang, Zheng Li, and Qian Zheng. 2024. Understanding Code Changes Practically with Small-Scale Language Models. In *39th IEEE/ACM International Conference on Automated Software Engineering (ASE '24), October 27-November 1, 2024, Sacramento, CA, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3691620.3694999

ASE '24, October 27-November 1, 2024, Sacramento, CA, USA

ACM ISBN 979-8-4007-1248-7/24/10.
https://doi.org/10.1145/3691620.3694999

1 INTRODUCTION

Software is evolving with continuous code changes (or changes). In a code change, developers may fix existing bugs, add new features, update accompanying documents, or refactor their code. Code change understanding (or change understanding) is fundamental for various important, downstream tasks such as code refinement [49], risk detection [33], and impact analysis [40].

Yet, code change understanding in practice is difficult. Developers struggle with resolving the rationale of code changes and concern about potential code breakage due to them [45]. To automate the process, many prior works chose to understand code changes through general-purpose change summarization, or through developing end-to-end learning models for specific, change-related tasks. For example, FIRA proposes a fine-grained code graph to represent code changes [7]. CC2Vec [13] and CCRep [28] encode code changes into distributed vectors. CodeReviewer [23] and CCT5 [24] are pre-trained for general-purpose change understanding.

Even though all these works have achieved promising results in the time, a recent study [6] observed that the following two issues still exist in them, hindering their practical uses:

(1) *Restricted code-change understanding*. They do not effectively understand the syntax and especially *semantics* in the change while overly focusing on the change marks, i.e., "+" and "-".
(2) *Biased code-change datasets*. Their change understanding behavior is positively influenced by the ratio of the *pattern*s implicitly embedded in their datasets used for (pre-)training.

A recent study [54] seeks language models (LMs) to address the first issue because LMs benefited a variety of code-related tasks [12, 12, 32, 36] recently. The study suggests that LM's code understanding capability can well translate to code changes that only include incomplete code portions (namely *code chunk*s in this paper). It shows that prompting LMs to summarize code changes is more effective than traditional techniques that do not incorporate LMs. However, it primarily focuses on expensive, *large* LMs (LLMs) with ≥10B parameters such as GPT-3.5 [2] and LLAMA-13B [47], which are either commercial or excessively costly to deploy widely spread. Furthermore, it is also infeasible to deploy such LLMs in environments with security, computational, and financial constraints like in industrial cases or on edge devices.

This implies the need for resource-friendly, *small* LMs (SLMs). Nevertheless, we realized that the second issue of "biased dataset" poses a challenge for SLMs. State-of-the-art (SOTA) datasets for change understanding, such as the MCMD dataset [44], are automatically collected from popular code repositories without human intervention. This inevitably introduced dirty summaries like "1", "fix", and "Polish", or summaries with patterns like "Merge PR from ..". Like LLMs, SLMs have also demonstrated impressive in-context learning abilities and can generalize effectively across tasks with minimal examples [2, 21, 35, 38, 51]. Providing biased examples (dirty or with patterns) can typically mislead them.

In summary, the following question regarding change understanding still remains open in the current era of LMs:

Is it feasible to deploy SLMs in practice while maintaining a competitive (comparable or even superior) performance with LLMs?

The HQCM Benchmark. We provide a positive answer in this paper. Our key insight is that *a small yet high-quality dataset for change understanding suffices to finetune SLMs to competitive performance.* Thus, we introduce the High-Quality Code-change bench-Mark (HQCM). It comprises a small yet comprehensive dataset with the same name HQCM, our finetuned SLMs for change-related tasks, and a suite of state-of-the-art baselines for accessing change understanding capabilities.

The HQCM dataset consists of 5,129 pairs of code changes and summaries. It sets apart from SOTA datasets in two aspects. Firstly, the dataset was meticulously reviewed and revised by three human experts, and then validated by two human experts to prevent biased data and to improve the quality. Additionally, we followed "Conventional Commits" [5], a common practice in the industry for code changes [1], to classify the dataset into 8 categories by the same human experts: *style*, *docs*, *test*, *build*, *cicd*, *fix*, *feat*, and *refactor*.

To answer the open question, we finetuned three SLMs with 7B and 220M parameters and compared them against state-of-the-art baselines and/or LLMs with ≥70B parameters. We focused on three change-related tasks: change summarization, change classification, and code refinement.

Change summarization, also referred to as commit message generation, involves generating a summary or a commit message for a code change. In this task, we finetuned Llama2-7B [47], CodeLlama-7B [41], and CCT5-220M [24] to determine if the HQCM-finetuned SLMs could generate better summaries than three SOTA baselines: NNGen [29], FIRA [7], and the raw CCT5-220M [24] which was finetuned using the MCMD dataset [44].

Change classification classified each pair of code change and summary into one of the aforementioned 8 categories. As no existing baselines were found for this task, we created two baselines using two LLMs GPT-4 (>175B) [34] and Llama2-70B via few-shot prompting [2]. We compared the HQCM-finetuned Llama2-7B against the two baselines to assess whether the finetuned SLM has the ability to classify a code change more accurately.

Code refinement refines a code chunk based on a refinement suggestion. A recent study [12] revealed that GPT-4, if appropriately prompted, demonstrated superior performance to state-of-the-art techniques. We thereby created baselines using GPT-4 and Llama2-70B following their settings. We evaluated if the SLM Llama2-7B, after finetuning by the HQCM dataset, could outperform the two larger baselines in generating more reasonable code chunks.

Our findings suggest that a high-quality dataset like HQCM, even small, can drive SLMs to reach competitive performance with existing baselines (traditional techniques or LLMs) in all three change-related tasks, demonstrating competing change understanding capabilities. Notably, all SLMs finetuned with the HQCM dataset produced summaries that are 2× preferred by GPT-4 and human experts. In change classification and code refinement, HQCM-finetuned Llama2-7B outperformed other baselines by approximately 15% on our cleaned dataset. We believe that this offers a positive answer to the open question and provides evidence to deploy SLMs for change understanding in environments with security or resource constraints.

Contributions. Our major contributions are:

- *Dataset*: We provide a small yet high-quality dataset called HQCM for augmenting the current era's SLMs with an improved understanding of code changes. It is a dataset that has been reviewed, revised, and validated by human experts.
- *Models*: We finetune SLMs Llama2-7B, CodeLlama-7B, and CCT5-220M using the HQCM dataset, all of which are confirmed to have competitive capabilities than state-of-the-art baselines and LLMs in change summarization, change classification, and code refinement.
- *Benchmark*: We package the HQCM dataset with our scripts to finetune SLMs into the HQCM benchmark to facilitate future research in code change understanding: https://github.com/codefuse-ai/codefuse-hqcm.

2 THE HQCM DATASET

Previous studies in change understanding typically leverage machine learning or deep learning techniques, with the datasets used for training models playing a crucial role.

State-of-the-art datasets include the MCMD dataset (comprising 2.25M pairs of code changes and summaries) [44], CodeReview (4.311M) [23], and CodeChangeNet (2M) [24]. In the past era, these datasets were intentionally created to maintain a large scale and were designed to train or pre-train understanding models from scratch. As a result, models trained or pre-trained through them have shown promising results in various evaluation metrics.

Despite this, recent studies showed that these models' understanding of code changes is still under par: They are unable to generate satisfactory, human-preferred summaries [6] or do not perform well on change-related tasks [12]. This is due to the fact that these datasets are automatically collected from popular code repositories without human intervention. Despite undergoing tiered data sanitization pipelines, they still inevitably introduced considerable amounts of "dirty" data such as unclear, ambiguous, or meaningless summaries (e.g., "1", "fixed", and "Polish"), and summaries with implicit patterns (e.g., "Merge pull request from <branch-name>" and "[#issue-id] fixed ...")—this situation is also observed by the recent study [54]. These biased data, either dirty or with patterns, positively influence the understanding behavior of their trained or pre-trained models. When presenting such data to the current era's LLMs, especially to SLMs which demonstrate exceptional but quite limited learning and generalization capabilities [2, 50], the biased data typically drive them to generate inaccurate or irrelevant change understandings. We display two such examples in Figure 3 and Figure 4 and detail them in Section 4.

This motivates us to create a small yet comprehensive, high-quality dataset for change understanding.

2.1 Dataset Collection

We constructed the HQCM dataset based on MCMD [44], a large-scale dataset recently released for commit message generation for five programming languages: C++, Java, JavaScript, Python, and Go. In this paper, we focus on the Java[1] subset consisting of 450,000 pairs of data. Hereafter, we denote the Java subset as MCMD for simplicity.

Reviewing and revising. To obtain a high-quality dataset, we conducted a rigorous reviewing and revising process involving 5,000 randomly selected pairs of code changes and summaries from MCMD. In this process, three experts, each with extensive experience in software development and unrelated to this project, were employed to review and revise each pair of data.

Initially, for each pair of code change and its summary in MCMD, we prompted GPT-4 and Llama2-70b to refine the summary into two additional summaries given the code change. Following this, the three experts independently selected the optimal summary that best summarizes the major behavior of the code change from the three summaries: the original summary in MCMD, the GPT-4 refined summary, and the Llama2-70b refined summary. In cases where there were differing choices, the three experts discussed the pair of data until reaching a consensus. If none of the three summaries could be chosen, the experts collaborated to write an optimal summary for the code change.

Subsequently, the three experts revised the optimal summary to ensure that each one should:

- Outline the primary behavior of the code change.
- Be concise, fitting for the "<description>" component in "Conventional Commits" [5], a widely used specification for code change summaries (or commit messages) in the industry [1].
- Start with a capitalized verb in the present tense.

Validation. After obtaining the 5,000 pairs of code changes and their optimal summaries, two authors of this paper independently validated whether the revised optimal summaries met the specified requirements outlined above. For pairs where the summaries did not meet, the two authors and the three experts engaged in collaborative discussions until reaching a consensus. If they could not find a consensus for a pair, they removed it from the HQCM dataset.

2.2 Dataset Categorization

In software development, a code change may involve fixing an existing bug, adding a new feature, or refactoring the code. Classifying code changes into different categories is beneficial for automating processing tools and assisting developers in understanding the change more fluently. Such categories are widely recognized as important in practical development activities and are integral components to Conventional Commits [5].

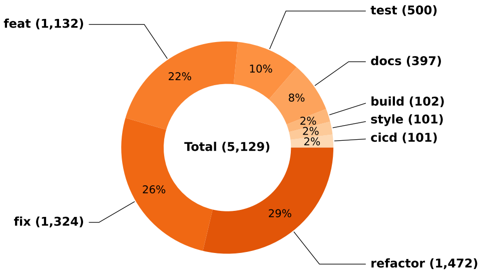


Figure 1: Category distribution of the HQCM dataset.

Categories. We considered 8 commonly used categories in the industry such as Angular [1]: *style*, *docs*, *test*, *build*, *cicd*, *fix*, *feat*, and *refactor*. We excluded *perf* and *chore* from considerations as our experts found it difficult to distinguish them. Yet, we deem finding better ways to incorporate them into HQCM as important actions in the future. Our benchmark contains detailed descriptions for them.

Categorization. We categorized the data pairs along with our reviewing/revising/validation process (Section 2.1). Specifically, after the three experts obtained the optimal summary for each pair of data, we requested them to further classify the pair into one of the 8 categories independently. They discussed their classification results until reaching a consensus. In the validation process, the two authors validated the categories assigned by the three experts to ensure they reflected the major behavior of the corresponding pairs. For unmet pairs, the five experts discussed until reaching a consistent category; otherwise, the pair was removed.

2.3 The HQCM Dataset

After being reviewed and revised by three experts and validated by the two authors, 4,987 pairs of code changes and summaries were included in the HQCM dataset. Furthermore, to prevent categories with (much) fewer data pairs from being overwhelmed by others (especially during supervised fine-tuning), we expanded them by randomly selecting new data from MCMD. If we were unable to select new ones, we leveraged GPT-4 to generate new data and requested the five experts to refine them following the same process of review, revision, and validation. It should be noted that we tried our best to minimize the set of newly added data, to ensure that the final HQCM dataset is primarily composed of non-synthetic, realistic data, reflecting real-world distributions.

Statistics. Finally, the HQCM dataset consists of 5,129 high-quality pairs of code changes and summaries, obtained over ~45.93 effective human days (~1 natural month), with an average of 2.79 data pairs included per human hour. The distribution of different categories is presented in Figure 1, where we observed that *refactor* is the most prevalent, while *style* and *cicd* are the least prevalent. Additionally, *refactor*, *fix*, and *feat* are ≥2× more popular than all the other categories. On average, each code change involves 1.01 changed files and 5.59 changed lines (excluding context lines without change marks). The average length of code changes and summaries are 339.82 and 9.28 tokens, with a range from 101 to 2,807 and from 2 to 73, respectively (computed by Llama's tokenizer).

[1] We were unable to commit additional resources due to budget constraints. Our decision to focus on one programming language was inspired by [54], showing that LM's change understanding capabilities can effectively translate among popular programming languages. We finally selected Java due to its prevalence and its primary use within our organization and among our partners, enabling us to deploy HQCM-finetuned SLMs for our developers and in our products.

Among the 5,129 pairs of data, only 1.09% (56) are chosen from the original MCMD summaries without any modification; the percentages for GPT-4- and Llama2-70b-refined summaries are 25.97% (1,332) and 23.98% (1,230). Additionally, 28.54% (1,464) of the summaries are revised from MCMD within 5-word modifications. The revisions for GPT-4 and Llama2-70b account for 1.29% (66) and 1.09% (56), respectively. Our experts rewrote all the rest (18.03%, 925) summaries. These results conform to [54]'s finding that LLMs are already effective in summarizing the behavior of code changes.

3 BASELINES AND METRICS

Based on the HQCM dataset, we finetuned SLMs and explored their change understanding capabilities through three change-related tasks: change summarization, change classification, and code refinement. For each task, we selected SOTA techniques or LLMs as baselines for comparison.

3.1 Change Summarization

Change summarization, also referred to as commit message generation, involves generating a summary or a commit message for a code change. This is a fundamental task to understanding code changes for two reasons. Firstly, it is observed that developers often lack the time to write high-quality commit messages in their development activities [8], highlighting the need for automated commit message generation techniques [6, 44]. Secondly, the summaries encapsulate the primary behavior of code changes, streamlining the developer's understanding to the code from various aspects (e.g., rationale, types), serving as a pivotal component for a variety of subsequent tasks, such as change classification, code refinement, and risk detection [45]. In this task, our objective is to determine if SLMs can result in improved summaries than SOTA techniques, after finetuning by HQCM.

Baselines. We selected the following baselines, which were shown to outperform other techniques in existing work [6, 44]:

- NNGen [29]. NNGen summarizes code changes using a nearest neighbor algorithm. It represents code changes as "Bag-of-Words" (BoW) vectors. When summarizing a code change c, NNGen first identifies a set C of the top-K closest code changes within a specified dataset, using the cosine similarity of their BoW vectors. Subsequently, it selects the code change $\tilde{c}$ from C that has the highest BLEU-4 similarity (with regards to their BoW vectors) to c. The summary accompanying $\tilde{c}$ is finally reused as the summary for c. NNGen was the most effective technique [44] before the introduction of FIRA [7].
- FIRA. FIRA summarizes code changes by learning from code graphs–their specialized graph representations of code changes. The graph compiles code tokens at various levels (i.e., subtokens and N-gram tokens), and encodes the editing operations between the code chunks pre- and post-changes. Based on the graph, FIRA develops an encoder-decoder model comprising a GNN encoder and a Transformer decoder to learn the correspondence between code graphs and their summaries. FIRA achieved superior effectiveness over NNGen [7].
- CCT5 [24]. Unlike NNGen and FIRA which are specialized in change summarization, CCT5-220m (or CCT5 for short) is a pre-trained model for general-purpose change understanding. It was pre-trained from five change-related tasks through token infilling (a.k.a. Masked Language Modeling) and completion. Like other LMs, CCT5 can function as the base model for various downstream, change-related tasks. In this task, we used the MCMD-finetuned CCT5 [24] specialized for commit message generation as a baseline.

In this task, we finetuned three state-of-the-art SLMs and compared them against the aforementioned baselines: Llama2-7b[2] [47], CodeLlama-7b [41], and the pre-trained CCT5.

MCMD-Picking. Comparing these techniques using the HQCM dataset may disadvantage the baselines, as the training and testing set randomly split from HQCM share the same distribution. Conversely, using the MCMD dataset is unfair to HQCM-finetuned SLMs because our baselines FIRA and CCT5 were trained or finetuned from or partially from the MCMD dataset[3].

To maintain a fair comparison, we further created a dataset called MCMD-Picking (MCMDP) from MCMD. Unlike HQCM, the development of MCMDP was entirely automated to avoid human preferences. Yet, we tried our best to ensure the selected data pairs were of good quality, excluding biased data. To this end, we leveraged GPT-4—the LLM that was well evaluated to have good code and change understandings by a series of work [11, 12, 19, 32, 36, 54]—to randomly pick data pairs from MCMD. We iteratively executed the following steps until each category includes 50 data pairs or no new data pairs could be found within 10 minutes for a category:

1. Request GPT-4 to analyze the behavior of a code change and explain its accompanying summary. Based on this, ask GPT-4 to assign a score determining whether the summary outlines the primary behavior of the code change:
   - Score -2: The summary does not reflect any behavior of the code change.
   - Score -1: The summary captures little behavior of the change.
   - Score +1: The summary captures behavior that is not central to the code change.
   - Score +2: The summary accurately outlines the primary behavior of the code change.
2. If the data pair is not scored +2, discard the data pair, pick the next data pair randomly, and turn back to the first step. Otherwise, process to the next step.
3. Instruct GPT-4 to modify the summary by altering its expression without changing the underlying meaning, to distinguish it from MCMD.
4. Employ GPT-4 to analyze the modified summary and the code change, then classify the data pair into one of the 8 pre-defined categories as mentioned in Section 2.2.
5. Include the modified pair of data into the MCMDP dataset.

This procedure finally led to 383 data pairs where the *cicd* category consists of only 33 pairs. We included MCMDP in our benchmark.

Metrics. Following prior work in change summarization [7, 24, 29, 53], we adopted the widely recognized ROUGE and BLEU. Both metrics focus on the literal similarity of change summaries and do not consider their semantics. In particular, we employed ROUGE-2 and the B-Norm variant [30] of BLEU-4.

[2] We chose Llama2 as Llama3 had not been released at the time of our experiments.
[3] FIRA was trained from CoDiSum's dataset [53]; we found it is nearly MCMD's subset.

We also measured semantic similarities. For this purpose, we utilized MPNet [43], specifically the "all-mpnet-base-v2" model, which currently leads the Sentence Transformer Leader Board [48]. MPNet embeds summaries into fixed-width semantic vectors and calculates their cosine similarity. We refer to this metric as "SEMSIM".

Finally, we calculated a "BRSA" value to reflect the average similarity by combining ROUGE, BLEU, and SEMSIM, weighted at 0.25, 0.25, and 0.5, respectively. We applied a twofold weighting on SEMSIM to mitigate the limitations imposed by BLEU and ROUGE that, require the evaluated techniques to produce a similar set of tokens.

3.2 Change Classification

Change classification classified each pair of code change and summary into one of the aforementioned 8 categories. This task is beneficial for automating processing tools, helping developers understand code change more easily. Categorizing code changes and their summaries are widely used in practical development activities and are integral components to Conventional Commits [5].

Baselines. Considering that we did not find any existing tools for this task[4], we developed two baselines using LLMs GPT-4 (>175B) and Llama2-70B, the most powerful LLMs in their respective families. We employed few-shot prompting [2] to provide them with a concrete example to assist their categorization. In addition to the code change and summary, we further ask them to analyze the primary behavior step by step in a "thoughts" field. The prompt used for this task is provided below, simplified for brevity; the text within `{{...}}` are placeholders and will be replaced at runtime. The complete prompt is included in our benchmark.

```
A git commit can typically be classified into specific categories
by examining its code change and summary. This includes:

{{category-descriptions}}

For a given git commit, we can inspect its code change by its
unified-diff representation and its summary via its commit message.
Let's think step by step.

Change: {{example:unified-diff}}
Summary: {{example:commit-message}}
Thoughts: The code change rectified a parameter error where
`oldValue` should be passed as the argument of `onDropFromCache`
rather than `value`. Therefore, it is a "fix" commit.
Category: fix

Change: {{categorizing:unified-diff}}
Summary: {{categorizing:commit-message}}
Thoughts:
```

In this task, we selected the SLM Llama2-7B to assess its capability in classifying code changes.

Dataset. For this task, we randomly divided `HQCM` into a training set (80%) and a test set. To maintain the distribution of different categories, our division was performed per category. We did not reuse `MCMDP` for evaluation as the dataset was generated by GPT-4.

Metrics. We chose the standard Precision, Recall, and F1 Score as in other classification tasks. We considered both the macro and micro variants in multi-class classification, wherein the macro variant assigns equal weight to each category, while the micro variant assigns equal weight to each pair of code change and change summary.

3.3 Code Refinement

Description. Code review is often time-consuming, demanding significant human effort. To address this issue, recent studies suggest automating the process through techniques such as suggestion generation or code refinement [12]. Code refinement involves refining a code chunk based on a refinement suggestion. This is useful in code review systems, where the reviewer offer advice for improving a code change during the review process.

Baselines. Recent studies [12, 52] show that, when prompted appropriately, GPT-4 demonstrated superior performance in code refinement than CodeReviewer [23], the state-of-the-art tool before the studies. Following them, we created two baselines using GPT-4 and Llama2-70B as in the change classification task. We organized our prompt in line with the optimal prompting style recommended by the study [12]: A piece of description consisting of the code chunk before changing and the refinement suggestion, accompanied by a preceding "scenario description" outlining the role and responsibility of the LLM. In addition to this, we provided LLMs with a concrete example for code refinement to let LLMs learn from. Below is the simplified prompt used for this task. The complete version is included in our benchmark.

```
// Please refine the given "Code (to refine)" (a code chunk a
"File") by strictly following the given "Suggestion".
// Your refinement can involve editing or removing existing code,
or adding new code.
// You may pay more attention to lines marked by "// !!attention".
If no such marked lines, do everything by yourself.

## Code (to refine) ##
//// File: {{example:file-name-before-changing}}
{{example:code-chunk-before-changing}}
//// Suggestion: {{example:suggestion}}
## Code (after refinement) ##
//// File: {{example:file-name-after-changing}}
{{example:code-chunk-before-changing}}

## Code (to refine) ##
//// File: {{refining:file-name-before-changing}}
{{refining:code-chunk-before-changing}}
//// Suggestion: {{refining:suggestion}}
## Code (after refinement) ##
//// File:
```

We again finetuned Llama2-7B through `HQCM` to test if the SLM could refine code chunks better than the baselines.

Dataset. For this task, we applied the same dataset splits as in the change classification task for finetuning and evaluation.

Metrics. We selected the following four metrics:

- ExactMatch. This metric measures the percentage of generated code chunks that are literally identical to their ground truths.
- ExactCodeMatch. This metric considers two code chunks identical if their token sequences after removing code comments are identical. Similar to ExactMatch, this metric calculates the percentage of identical code chunks.
- CodeBLEU [39]. This metric is a common choice for code generation tasks. It shares similarities with BLEU, as it conceptually considers correctly generated (N-gram) tokens. It also incorporates code syntax through abstract syntax trees and code semantics via data-flow relations; however, this complexity makes it difficult to apply to all programming languages. Another limitation is its potential imprecision when used with incomplete code chunks as in our case.

[4] The categories are typically created by developers while writing summaries.

- CrystalBLEU [9]. This metric was recently proposed as an improvement over BLEU for code-related tasks. It assigns lower weights to trivially shared N-grams to focus on truly similar ones. It is designed specifically for code based on some common observations, although it does not capture the syntax and the semantics. It can be applied to any programming language and even incomplete code chunks. We believe combining CodeBLEU and CrystalBLEU provides a more comprehensive evaluation.

3.4 Other Methodology Details

We finetuned SLMs Llama2-7B and CodeLlama-7B by adding LoRA [14] adapters to their attention layers, without any further modifications. For CCT5-220M, we exercised the finetuning scripts in its artifact by substituting MCMD with HQCM. In the cases of LLMs GPT-4 and Llama2-70B, we did not finetune them. Instead, we prompted them with natural language instructions and few-shot examples. We prompted or finetuned all open-source LMs (Llama2, CodeLlama, and CCT5) on a Linux server with two 32-core Intel Xeon CPUs (120 GiB RAM) and an NVIDIA A100 GPU (80 GiB VRAM). For GPT-4, we utilized OpenAI's ChatCompletion APIs. Furthermore, we applied greedy decoding to eliminate potential randomness during text generation.

4 BENCHMARKING LLMS

Research questions. We focus on the following questions:

RQ1 *Quality of* HQCM: After being revised and validated by five human experts, can HQCM demonstrate superior performance compared to the SOTA dataset MCMD in finetuning SLMs?

RQ2 *Basic Understandings*: In the context of change summarization, the foundation of other change-related tasks, how do HQCM-finetuned SLMs compare with SOTA baselines?

RQ3 *Advanced Understandings*: For tasks more advanced than change summarization, i.e., change classification and code refinement in this paper, does HQCM enhance SLMs with competitive understandings than much larger baselines?

4.1 RQ1: Quality of Our Dataset

To ensure a sound evaluation, we initially assessed the quality of the HQCM dataset to ascertain that it surpasses SOTA datasets in change understanding and better aligns with the current era of LMs, before experimenting with the three concerned tasks.

Baselines. In this experiment, we compared the HQCM dataset against MCMD [44], from which our dataset originates. In particular, we chose the change summarization task as it is the foundation of the other tasks. To ensure a comprehensive evaluation, we included three different kinds of SLMs for finetuning: Llama2-7B [47] which is a SOTA general-purpose SLM, CodeLlama-7B [41] which is a SOTA code-related SLM, and CCT5 (220M) [24] which is recently released, specially pre-trained for change-related tasks.

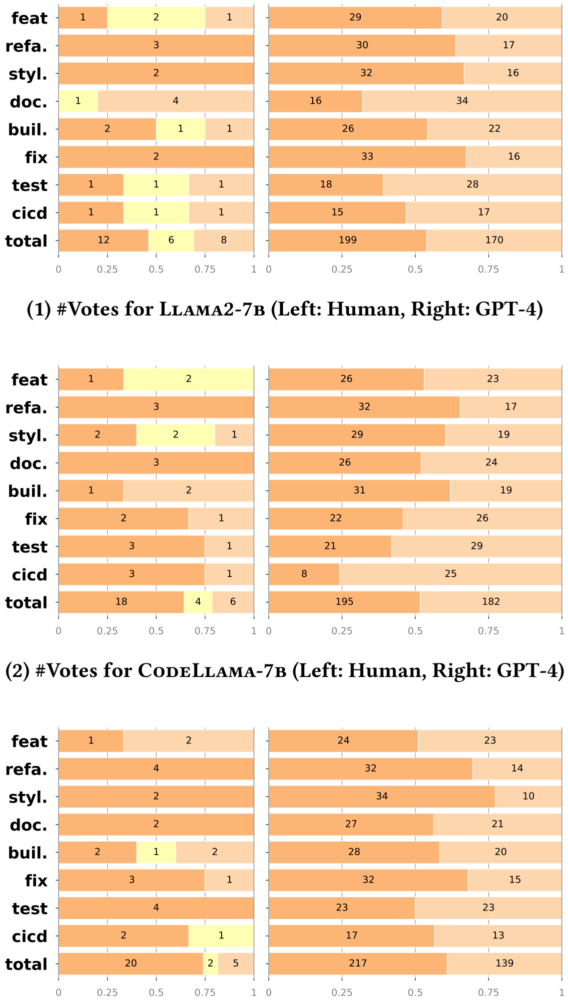
Dataset and Metrics. We finetuned selected SLMs respectively through HQCM and MCMD. We compared them with the metrics chosen for the change summarization task using the MCMDP dataset (Section 3.1). Furthermore, we computed the average number of data pairs required to achieve a 1% improvement in BRSA of all techniques; this is denoted “Profit” in this paper.

Table 1: Comparison results of SLMs finetuned respectively through the small-scale HQCM dataset and the large-scale MCMD dataset. “Profit” measures the number of data pairs required to achieve a 1% improvement in BRSA.

Cat.	Metric%	CodeLlama-7B			Llama2-7B			CCT5-220M		
		MCMD	+/-	HQCM	MCMD	+/-	HQCM	MCMD	+/-	HQCM
feat	BLEU	12.84	-4.74	8.11	11.27	-5.25	6.02	12.92	-7.23	5.69
	ROUGE	18.11	-6.60	11.50	15.04	-5.04	10.00	16.79	-6.48	10.31
	SEMSIM	55.15	+0.41	55.56	53.56	-0.28	53.28	49.44	+4.54	53.98
refa.	BLEU	15.44	-3.83	11.61	14.75	-5.54	9.21	8.44	-0.41	8.03
	ROUGE	14.03	-1.38	12.65	14.62	-4.66	9.96	9.26	-1.44	7.81
	SEMSIM	46.93	-1.53	45.39	47.22	+0.99	48.21	42.81	-2.01	40.81
styl.	BLEU	10.77	-0.81	9.96	7.66	+0.40	8.06	9.85	-3.40	6.45
	ROUGE	14.86	+0.41	15.27	17.46	-3.45	14.00	11.65	+0.16	11.81
	SEMSIM	45.82	-1.69	44.13	40.83	+2.23	43.06	34.49	+8.19	42.68
doc.	BLEU	14.81	-5.13	9.68	17.62	-9.60	8.02	12.32	-4.56	7.76
	ROUGE	19.23	-4.71	14.52	20.74	-8.78	11.96	16.29	-3.89	12.40
	SEMSIM	57.34	-4.72	52.63	55.24	-7.61	47.63	51.57	-5.17	46.41
buil.	BLEU	25.57	+5.53	31.10	24.89	+15.14	40.03	20.48	+8.19	28.66
	ROUGE	36.01	+6.01	42.02	34.51	+11.47	45.98	28.61	+9.64	38.26
	SEMSIM	69.87	+2.06	71.93	68.36	+3.16	71.51	66.19	-2.74	63.45
fix	BLEU	14.96	-7.10	7.87	13.38	-4.88	8.50	8.92	+0.73	9.66
	ROUGE	18.20	-8.03	10.17	15.53	-6.34	9.18	10.67	-0.60	10.07
	SEMSIM	50.44	-2.57	47.87	48.13	-0.70	47.43	49.26	-3.83	45.43
test	BLEU	26.29	-16.25	10.04	9.66	-6.14	3.52	12.82	-5.58	7.23
	ROUGE	24.57	-9.94	14.63	22.35	-12.80	9.54	16.09	-4.96	11.13
	SEMSIM	58.95	-9.42	49.53	55.10	-11.84	43.27	56.21	-11.63	44.57
cicd	BLEU	15.72	-6.54	9.18	12.53	-6.12	6.41	12.63	-4.26	8.37
	ROUGE	18.99	-6.95	12.04	17.90	-5.33	12.57	13.20	-1.58	11.63
	SEMSIM	60.41	-5.98	54.43	59.82	-5.81	54.01	50.58	-1.04	49.54
Ave.	BLEU	18.10	-4.48	13.62	15.24	-1.66	13.58	12.76	-1.60	11.17
	ROUGE	20.70	-3.85	16.85	19.88	-4.23	15.65	15.55	-1.10	14.44
	SEMSIM	55.40	-2.80	52.61	53.25	-2.33	50.92	50.05	-1.74	48.31
BRSA		37.40	-3.48	33.92	35.41	-2.64	32.77	32.10	-1.55	30.55
Profit		12K	↑79.57×	**151**	12K	↑81.19×	**156**	14K	↑83.51×	**167**

Anonymous Evaluation. In addition to the objective metrics, we included an anonymous evaluation involving GPT-4 and a human expert. Specifically, for each SLM, we presented a code change with two anonymous summaries to each practitioner: One generated via the HQCM-finetuned one and the other by the MCMD-finetuned one. The practitioner was asked to (1) vote for the summary better outlining the primary behavior of the code change and (2) provide a clear reason for their voting. If the practitioner found both summaries equally good, we considered it a tie. If the practitioner was unable to choose a better one, we considered both finetuned SLMs to fail and discarded the pair. For GPT-4, we presented all the data pairs in the MCMDP dataset for anonymous evaluation. As for the expert, we randomly selected 5 data pairs for each category.

Results. Table 1 illustrates the comparison results in terms of BRSA. Despite being much smaller than MCMD (~87× smaller), SLMs finetuned through HQCM demonstrated comparable performance on these metrics. On average, to achieve a 1% improvement in BRSA, an SLM requires ~81× fewer data pairs when finetuning using HQCM, with CodeLlama requiring the least numbers. Specifically, the difference between HQCM- and MCMD-finetuned SLMs is within 3.5% on average, where the greatest difference is shown on CodeLlama and the least is shown on CCT5. It is worth noting that these significant profits and marginal differences are evident not only in 7B SLMs



(1) #Votes for Llama2-7b (Left: Human, Right: GPT-4)

(2) #Votes for CodeLlama-7b (Left: Human, Right: GPT-4)

(3) #Votes for CCT5-220m (Left: Human, Right: GPT-4)

Figure 2: Anonymous evaluation results for different SLMs finetuned respectively by HQCM and MCMD. The dark orange is practitioner's #votes for summaries generated by HQCM-finetuned SLMs; the light orange is their votes for MCMD-finetuned SLMs; and the yellow is for ties.

but also in 220M SLMs. We consider this implies[5] that a small yet high-quality dataset is already compute-optimal to finetune the current era's SLMs which were already pre-trained with substantial FLOPS and tokens following Chinchilla Law [20]. Additionally, even though these metrics are useful for evaluating certain aspects of the generated summaries, we argue that they cannot fully capture the semantics between LM-generated summaries and the ground truths. We thereby conducted an anonymous evaluation.

The anonymous evaluation results are shown in Figure 2. Either GPT-4 or the human expert found that HQCM-finetuned SLMs significantly outperform MCMD-finetuned ones. Specifically, HQCM-finetuned SLMs received at least 7% more votes from two practitioners for all three kinds of SLMs. The gap between their votes ranges from 7% to 56% for GPT-4 and 28% to 214% for the human expert. This suggests that both practitioners preferred HQCM-finetuned SLMs. Similar to BRSA, CCT5 achieved the highest vote differences for both GPT-4 and the human expert, with a 214% and 56% increase, respectively. We believe that these results confirm the quality of the HQCM dataset and evidence the necessity of pre-training models for general-purpose change understanding.

```
  language: java
  jdk:
    - oraclejdk8
+ before_install:
+   - pip install --user codecov
  after_success:
    - mvn clean cobertura:cobertura coveralls:report
-   - bash <(curl -s "https://codecov.io/bash")
+   - codecov
  branches:
    only:
      - master
```

○ **Expected**: "Add codecov installation to before_install in .travis.yml"

○ **MCMD** (A): "add codecov" (BLEU: 6.41, ROUGE: 30.00)

★ **HQCM** (B): "Install codecov before installing" (BLEU: 0, ROUGE: 8.70)

○ **GPT-4**: "The code change indicates that a "before_install" step is added, which includes a pip command. This step is clearly meant to install the codecov tool before the installation process of the build. Summary B directly refers to this action, matching the semantics of the code diff. Summary A is less specific and could be interpreted as simply adding codecov to the project in some form, which is not as accurate as Summary B in describing the specific change made in the code diff."

○ **Human**: "The core of the change is to move codecov installation to before_install, which is reflected by Summary B."

Figure 3: A *cicd* example where the HQCM-finetuned CCT5 achieved lower BLEU and ROUGE scores than MCMD-finetuned CCT5 but provided a more comprehensive summary. The example has been simplified for brevity.

Figure 3 shows a simple example where the MCMD-finetuned CCT5 achieved higher BLEU and ROUGE scores. However, GPT-4 and the human expert favored the summary generated by HQCM-finetuned CCT5 because it accurately captured the core idea that codecov was added before installation. A more complex example is depicted in Figure 4, where developers captured an assertion failure and used a try-catch block to log debugging information. In this case, the MCMD-finetuned CCT5 incorrectly treated the changed method `testRestoreBehaviourWithFSH()` as a newly added method. However, after finetuning by HQCM, CCT5 successfully identified the primary behavior. We present the (simplified) reasons of both practitioners in both figures. Our benchmark includes more examples.

> ***Finding 1.*** The HQCM dataset is small yet high-quality. Leveraging it, finetuning SLMs can be faster, requiring significantly less data while achieving a competitive (comparative or even superior) performance with SOTA large-scale datasets, such as the MCMD dataset.

[5] We acknowledge that providing formal proof to support this statement is still necessary in the future; we were not aware of any such papers when writing this paper.

```
  void testRestoreBehaviourWithFSH() ... {
      ...
      deadline,
-     (jobStatus) -> jobStatus == JobStatus.FINISHED,
+     JobStatus::isGloballyTerminalState,
      TestingUtils.defaultScheduledExecutor());
-   assertEquals(JobStatus.FINISHED, jobStatFuture.get());
+   try {
+     assertEquals(JobStatus.FINISHED, jobStatFuture.get());
+   } catch (Throwable e) {
+     // include additional debugging information
+     StringWriter error = new StringWriter();
+     try (PrintWriter out = new PrintWriter(error)) {
+       out.println("The job did not finish in time.");
+       out.println("allowedInitialiizeCalls...= " ) + ...;
+       ...
+       out.println("threadDump= " + generateThreadDump());
+     }
+     throw new AssertionError(...);
+   }
    assertThat("We saw illegal restores.", ...);
  }
+ private static String generateThreadDump() {
+   ...
+ }
  private static class UnboundedSource implements SourceFunction<
      String> {
  private volatile boolean running = true;
```

○ **Expected**: "Refactor ZooKeeperHighAvailabilityITCase for more detailed and useful information on timeout error"

○ MCMD (A): "Add testRestoreBehaviourWithFSH" (BLEU: 0, ROUGE: 14.26)

★ HQCM (B): "Capture exception and print detailed information" (BLEU/ROUGE: 0)

○ **GPT-4**: "The code change shows that the test method "testRestoreBeh..." has been modified rather than added. The most important section in the change is the addition of a try-catch block around an assertion to capture any exceptions that occur and the printing of detailed debugging information. This matches the description in Summary B. Summary A does not match because the test method was not added; it was already present and has been modified."

○ **Human**: "Summary B exactly conveyed the core idea of the change: log additional debugging information."

Figure 4: A *test* example where the HQCM-finetuned CCT5 achieved lower BLEU and ROUGE scores than MCMD-finetuned CCT5 but provided a more comprehensive summary. The example has been simplified for brevity.

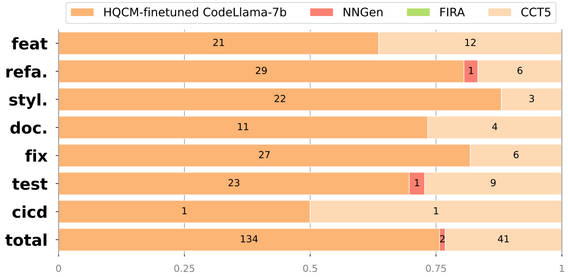


Figure 5: GPT-4's anonymous evaluation results for change summarization over our finetuned SLMs and all baselines.

4.2 RQ2: Basic Understandings

After validating the quality of the HQCM dataset, we finetuned SLMs and compared them with the selected baselines: NNGen, FIRA, and MCMD-finetuned CCT5, on top of MCMDP. Since FIRA only supports Java, we conducted separate evaluations: (1) NNGen, CCT5, and finetuned SLMs on the whole MCMDP dataset (383 data pairs), and (2) NNGen, FIRA, CCT5, and finetuned SLMs the Java subset (changes only to Java files) of MCMDP, including 181 data pairs.

Table 2: Comparison results between HQCM-finetuned SLMs and baselines (including FIRA) for change summarization. The results for the *build* category are all 0.00s as all *build* changes are not for Java. Llama2 represents Llama2-7b; CLlama stands for CodeLlama-7b; CCT5 is CCT5-220m;

Cat.	Metric%	Baselines			SLMs		
		NNGen	FIRA	CCT5	CLlama	Llama2	CCT5
feat	BLEU	0.00	1.64	**14.09**	7.26	4.77	6.61
	ROUGE	1.07	4.39	**15.57**	9.11	7.34	10.77
	SEMSIM	13.90	20.69	48.65	**53.46**	49.38	52.06
refa.	BLEU	6.62	0.00	9.79	**12.55**	9.36	9.47
	ROUGE	5.72	0.98	10.00	**13.01**	8.67	9.63
	SEMSIM	22.18	12.54	40.55	43.02	**45.83**	41.98
styl.	BLEU	5.70	7.32	8.97	**10.18**	7.13	9.77
	ROUGE	4.75	**17.33**	9.36	15.85	15.65	16.41
	SEMSIM	22.52	32.79	32.62	**44.94**	43.14	42.80
doc.	BLEU	3.30	0.00	6.79	**11.87**	6.18	4.65
	ROUGE	4.00	1.90	11.68	**14.58**	8.64	8.87
	SEMSIM	21.60	16.01	43.66	**53.51**	40.63	36.63
buil.	BLEU	**0.00**	0.00	0.00	0.00	0.00	0.00
	ROUGE	**0.00**	0.00	0.00	0.00	0.00	0.00
	SEMSIM	**0.00**	0.00	0.00	0.00	0.00	0.00
fix	BLEU	4.36	3.01	**8.54**	7.41	7.99	6.42
	ROUGE	4.51	7.89	**9.41**	9.40	8.54	9.15
	SEMSIM	18.97	22.62	**49.25**	46.56	45.88	43.71
test	BLEU	2.17	0.00	**13.84**	8.12	3.91	6.70
	ROUGE	3.42	2.49	**17.51**	11.10	8.10	10.60
	SEMSIM	26.46	20.50	**54.37**	47.59	41.38	45.12
cicd	BLEU	0.00	0.00	**23.66**	0.00	0.00	0.00
	ROUGE	0.00	7.69	**25.64**	4.00	0.00	4.00
	SEMSIM	30.17	44.58	**61.21**	32.21	39.85	35.05
Ave.	BLEU	4.03	2.09	**11.41**	9.43	6.90	7.40
	ROUGE	3.98	5.73	**12.71**	11.69	9.24	10.90
	SEMSIM	20.94	20.92	45.53	**47.45**	44.79	44.32
BRSA		12.47	12.42	28.79	**29.00**	26.43	26.73

Anonymous Evaluation. Like the previous experiment, we additionally included an anonymous evaluation involving only GPT-4[6]. The evaluation considered all our baselines and the HQCM-finetuned SLM that achieved the highest BRSA. Specifically, we presented a code change along with four anonymous summaries to GPT-4 and asked it to choose the best summary outlining the primary behavior of the code change. In this anonymous evaluation, we did not consider ties. If GPT-4 could not choose the best summary, we deemed all techniques to fail and discarded the pair. We presented all 181 Java data pairs in MCMDP for anonymous evaluation.

Results. Tables 2 and 3 present the results, with the former including FIRA. Overall, after finetuning by HQCM, the SLM CodeLlama-7b achieved the highest BRSA among all compared techniques. All other SLMs also demonstrated impressive performance. Among baselines, MCMD-finetuned CCT5 ranked the highest, closely following our CodeLlama. Notably, all finetuned SLMs significantly outperformed traditional machine- or deep-learning techniques. After careful inspection, we realized that: For NNGen, this could be attributed to its reliance on an existing summary as the final summary without generating new summaries; FIRA is in the face of the out-of-vocabulary problem, leading to many generated summaries containing an "<unk>" token. We also found that our finetuned SLMs

[6] We did not include human experts this time as previous work [12, 54] and our previous anonymous evaluation have confirmed GPT-4's capability in this task.

Table 3: Comparison results between HQCM-finetuned SLMs and baselines (excluding FIRA) for the change summarization task. Llama2 represents Llama2-7b; CLlama stands for CodeLlama-7b; CCT5 is CCT5-220m.

Cat.	Metric%	Baselines		SLMs		
		NNGen	CCT5	CLlama	Llama2	CCT5
feat	BLEU	0.00	**12.92**	8.11	6.02	5.69
	ROUGE	1.89	**16.79**	11.50	10.00	10.31
	SEMSIM	14.79	49.44	**55.56**	53.28	53.98
refa.	BLEU	5.68	8.44	**11.61**	9.21	8.03
	ROUGE	5.20	9.26	**12.65**	9.96	7.81
	SEMSIM	23.41	42.81	45.39	**48.21**	40.81
styl.	BLEU	3.90	9.85	**9.96**	8.06	6.45
	ROUGE	3.25	11.65	**15.27**	14.00	11.81
	SEMSIM	20.36	34.49	**44.13**	43.06	42.68
doc.	BLEU	2.86	**12.32**	9.68	8.02	7.76
	ROUGE	3.41	**16.29**	14.52	11.96	12.40
	SEMSIM	24.40	51.57	**52.63**	47.63	46.41
buil.	BLEU	28.52	20.48	31.10	**40.03**	28.66
	ROUGE	32.62	28.61	42.02	**45.98**	38.26
	SEMSIM	49.63	66.19	**71.93**	71.51	63.45
fix	BLEU	6.08	8.92	7.87	8.50	**9.66**
	ROUGE	5.82	**10.67**	10.17	9.18	10.07
	SEMSIM	18.99	**49.26**	47.87	47.43	45.43
test	BLEU	1.95	**12.82**	10.04	3.52	7.23
	ROUGE	3.38	**16.09**	14.63	9.54	11.13
	SEMSIM	26.87	**56.21**	49.53	43.27	44.57
cicd	BLEU	0.00	**12.63**	9.18	6.41	8.37
	ROUGE	3.53	**13.20**	12.04	12.57	11.63
	SEMSIM	25.50	50.58	**54.43**	54.01	49.54
Ave.	BLEU	8.14	12.76	**13.62**	13.58	11.17
	ROUGE	7.64	15.55	**16.85**	15.65	14.44
	SEMSIM	25.49	50.05	**52.61**	50.92	48.31
BRSA		16.69	32.10	**33.92**	32.77	30.55

```
  import static org.hamcrest.Matchers.equalTo;
- public class ShapeRalationTests extends ESTestCase {
+ public class ShapeRelationTests extends ESTestCase {
    public void testValidOrdinals() {
      assertThat(...);
```

○ **NNGen**: "Refactor XPackTestCase to abstract class"

○ **FIRA**: "fix the build"

○ **CCT5**: "Add test for shape relation"

★ **Ours**: "Corrected incorrect class name in testing"

Figure 6: A *fix* example where all baselines fail to identify the typo (Rala to Rela). Our CodeLlama captured it successfully.

typically performed better in categories such as *feat*, *refactor*, *style*, *docs*, and *build*.

Figure 5 depicts the results of our anonymous evaluation, which include HQCM-finetuned CodeLlama-7b and all baselines. The results indicate that HQCM-finetuned CodeLlama garnered 217% more (134 vs 43) votes than all baselines from GPT-4. Also, consistent with the findings in the BRSA metrics, both finetuned SLMs—CCT5 (134 votes) and CodeLlama (41 votes)—significantly outperformed NNGen (4 votes) and FIRA (0 votes).

Figure 6 illustrates an example where our finetuned CodeLlama successfully identified the typo where Rela was misspelled as Rala. This is an example that would be challenging even for a human being. All baselines fail to provide an even close understanding.

Table 4: Comparison results for change classification. L70/L7 stands for Llama2-70b/Llama2-7b and G4 represents GPT-4.

Cat.	Precision%			Recall%			F1 Score%		
	L70	G4	L7	L70	G4	L7	L70	G4	L7
feat	64.29	62.50	**86.87**	39.47	61.40	**75.44**	48.91	61.95	**80.75**
refa.	67.69	60.33	**73.13**	29.73	49.32	**66.22**	41.31	54.28	**69.50**
styl.	17.74	26.19	**72.73**	**100.00**	100.00	72.73	30.14	41.51	**72.73**
doc.	61.29	**87.50**	84.62	47.50	70.00	**82.50**	53.52	77.78	**83.54**
buil.	75.00	64.29	**84.62**	54.55	81.82	**100.00**	63.16	72.00	**91.67**
fix	38.28	47.53	**64.42**	73.68	57.89	**78.95**	50.39	52.20	**70.95**
test	**100.00**	100.00	87.50	34.00	46.00	**84.00**	50.75	63.01	**85.71**
cicd	**100.00**	100.00	90.91	45.45	72.73	**90.91**	62.50	84.21	**90.91**
Macro	65.54	68.54	**80.60**	53.05	67.40	**81.34**	58.63	67.96	**80.97**
Micro	47.67	58.17	**75.87**	47.30	57.72	**75.87**	47.48	57.95	**75.87**

> ***Finding 2.*** After finetuning by HQCM, all SLMs demonstrated impressive basic understandings of code changes: They achieved competitive (comparable or superior) BRSA with baselines and were more favored in anonymous evaluations.

4.3 RQ3: Advanced Understandings

The final evaluation concerns the advanced understanding of code changes in terms of change classification and code refinement. The two experiments leveraged the HQCM dataset for both training (80%) and testing (20%). In these experiments, we selected the SLM Llama2-7b and compared it with GPT-4 (>175b) and Llama2-70b.

Change Classification. Table 4 presents the results. Despite having a considerably fewer number of parameters, the HQCM-finetuned Llama2-7b significantly outperformed the created baselines by 13.01% and 17.92% for macro and micro averages. Among the baselines, GPT-4 achieved better results than Llama2-70b by ∼10%.

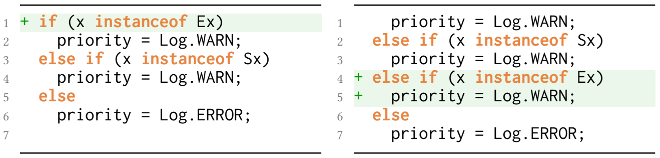
It is worth noting that the superiority of HQCM-finetuned Llama2 is demonstrated in every category considered in this paper. However, the finetuned Llama2 still overlooked approximately 30% of *style* changes compared to the two baselines and did not exhibit the same level of precision for *test* and *cicd* changes. In the *style* category, we found that SLMs exhibited weaker capabilities in detecting style changes such as whitespace, line wrapping, and breaking. As for the other two categories, our observation is that they often misclassified changes relating to testing or CI/CD, even if they were not the primary behavior. This suggests the necessity to incorporate more high-quality data in such categories or to further refine specific categories in the future, in order to enable more accurate understanding and to mitigate these issues.

Code Refinement. Table 5 describes the results. Similar to change classification, the HQCM-finetuned Llama2-7b significantly outperformed the created baselines for all the considered metrics even though with a considerably fewer number of parameters. GPT-4 also achieved better results than Llama2-70b.

It is worth mentioning that HQCM-finetuned Llama2 successfully generated exact chunks for over 18% of data pairs, with appropriately 23% of the generated chunks being equal in terms of their token sequence, excluding code comments. In contrast, Llama2-70b

Table 5: Comparison results for code refinement. L70/L7 stands for Llama2-70B/Llama2-7B and G4 represents GPT-4.

Cat.	ExactMatch			ExactCodeMatch			CrystalBLEU			CodeBLEU		
	L70	G4	L7	L70	G4	L7	L70	G4	L7	L70	G4	L7
feat	0.00	0.00	**8.77**	1.75	7.89	**12.28**	9.07	51.36	**57.14**	21.12	60.82	**65.79**
refa.	0.00	0.66	**28.48**	11.92	31.79	**35.10**	31.40	62.67	**76.99**	39.61	74.96	**81.49**
styl.	0.00	7.14	**57.14**	6.67	13.33	**26.67**	40.43	55.17	**87.94**	48.57	61.02	**87.15**
doc.	0.00	0.00	**12.20**	21.95	43.90	**46.34**	22.90	48.23	**62.20**	30.67	61.19	**72.70**
buil.	0.00	0.00	**45.45**	0.00	30.00	**48.33**	23.51	49.34	**68.11**	**0.00**	0.00	0.00
fix	0.00	3.76	**11.28**	9.02	**18.05**	14.29	34.30	55.80	**75.52**	42.96	69.92	**79.89**
test	0.00	0.00	**18.00**	10.00	20.00	**28.00**	34.94	57.03	**70.48**	38.85	68.82	**78.04**
cicd	0.00	0.00	**18.18**	4.17	4.17	**16.67**	24.34	**45.49**	41.72	**0.00**	0.00	0.00
Ave.	0.00	1.33	**18.48**	3.29	12.41	**23.30**	25.74	56.18	**69.27**	47.57	67.42	**80.43**



```
1 + if (x instanceof Ex)
2       priority = Log.WARN;
3   else if (x instanceof Sx)
4       priority = Log.WARN;
5   else
6       priority = Log.ERROR;
7
```

(a) GPT-4's Refinement

```
1       priority = Log.WARN;
2   else if (x instanceof Sx)
3       priority = Log.WARN;
4 + else if (x instanceof Ex)
5 +     priority = Log.WARN;
6   else
7       priority = Log.ERROR;
```

(b) Llama2-7B's Refinement

Figure 7: A *feat* example for code refinement where the chunk refined by GPT-4 alters the semantics of the chunk before changing but HQCM-finetuned Llama2-7B generated an exactly matched chunk given the refinement suggestion: "If NoClassDefFoundError occurs, set the log level to WARN". The code has been simplified for brevity. `Ex` is short for `NoClassDefFoundError` and `Sx` for `ActivityShare.ServerException`.

could not generate any exactly matched chunks, and GPT-4 worked for only 1.33% of data pairs. The ExactCodeMatch of GPT-4 is even 50% of that of our finetuned Llama2.

When considering metrics that account for code semantics (such as code patterns and dataflow relations), we observed that, even without finetuning, the current LLMs, particularly GPT-4, are capable of generating refined code chunks given a refinement suggestion with achieving >50 CrystalBLEU and CodeBLEU scores. This conforms to the findings of [12] and [52]. However, after finetuning through the HQCM dataset, the performance of even a smaller LM (Llama2-7B) could outperform them by approximately 13% or even up to 32% for Llama2-70B.

Figure 7 provides an example of this task. In this example, our Llama2-7B successfully generated an exactly matched code chunk following the refinement suggestion. Yet GPT-4, while having generated code for validating `NoClassDefFoundError`, the generated chunk breaks the semantics of the original chunk, leading to a missed exception (i.e., the exception checked preceding Line 1).

> ***Finding 3.*** After finetuning by HQCM, the SLM Llama2-7B demonstrated an advanced understanding of code changes: It achieved superior results to ≥70B baselines for both change classification and code refinement on our validated dataset.

4.4 Discussion

We developed a small yet high-quality dataset called HQCM for change understanding. Based on HQCM, we explored SLMs (7B and 220M) towards change summarization, change classification, and code refinement. Our evaluation indicated that HQCM-finetuned SLMs demonstrated competitive performance with state-of-the-art baselines in these tasks. We believe that this provides a positive answer to the open question that we raised in Section 1, and can supply support for those who plan to deploy LMs in environments with various constraints.

Experiences. Although HQCM-finetuned SLMs performed comparably to our baselines—traditional techniques and LLMs—we encountered several issues that may offer valuable insights for future research. Firstly, the performance of HQCM-finetuned SLMs does not always hold consistent across all categories (Sections 4.2 and 4.3). SLMs oftentimes struggle with recognizing *style* changes, such as code reformatting, when compared with non-finetuned LLMs. Consequently, they misidentify certain *style* changes as *fix*es, *refactor*s, or even *doc*s, leading to inaccurate change understandings. To alleviate this issue, one might employ specialized SLMs finetuned for classifying style changes and generating corresponding summaries, while relying on general SLMs for other types of changes. This approach, however, would necessitate a substantial amount of high-quality style data. The second issue involves code comments. We observed that SLMs sometimes rely on code comments to generate summaries, yet these comments pertain to the code chunk before changing, not the change itself. Our experience suggests that collecting more, high-quality data including code comments can alleviate the issue. The last issue exists in both SLMs and LLMs. We found that LMs, without further context, can identify what the code changes, while failing to recognizing why the code changes. It might underscore the need for context-search techniques, e.g., Retrieval-Augmented Generation (RAG), in this domain of change understanding. This is why we did not adopt the "Why/What" metric in our anonymous evaluation as existing work [11, 46] and the issue is also observed in [54].

Limitations. The HQCM dataset is currently not complete: It neither considers some commonly used industry categories such as *perf* and *chore* [1], nor includes code changes involving non-Java programming languages (except configuration or markup languages). Our evaluation also indicates that the HQCM dataset should be further revised to facilitate a more accurate understanding. Despite these limitations, our evaluation already confirmed our insight: A small yet high-quality dataset for change understanding significantly contributes to change-related tasks using SLMs than larger datasets with varying quality. In addition, the recent study [54] provides us with evidence that such capabilities could safely translate to other languages that we currently do not consider.

Threats to Validity. The first threat to validity pertains to human activities involved in creating HQCM and conducting anonymous evaluations. To address possible biases, we engaged multiple experts and required them to reach a consensus. Additionally, for anonymous evaluations, we requested experts to provide explicit reasons for their voting, which are included in our benchmark. Moreover, our current evaluation primarily focuses on Llama-series LMs, potentially introducing bias. However, we would like to argue that our results can be broadly applied to other LMs, as the performance of these LMs only varies marginally [4, 16].

We anticipate that our HQCM benchmark, which comprises the HQCM dataset, our SLMs, and the created baselines, will better support LMs in understanding code changes in the future. As part of our future work, we plan to conduct larger experiments involving more LMs and to rank them using our benchmark as well as introducing Elo rating systems [4, 10] for change understanding.

5 RELATED WORK

Recently, many works have been proposed for code change understanding. These works can generally be classified into two categories: The first category focuses on specialized understanding concerning specific downstream, change-related tasks, while the second targets general-purpose change understanding.

Specialized Understanding. Many of the works in this category have focused on change summarization or commit message generation. Early studies such as DeltaDoc and ChangeScribe extract specific information (e.g., path predicates) and utilize predefined rules or templates for summarization [3, 22, 26, 42]. Subsequent works like NNGen reuse summaries of the most similar code changes [15, 29]. While these studies presented promising results at the time, the mainstream approach today employs deep learning techniques, particularly neural machine translation following the encoder-decoder architecture. Some works translate code changes into summaries with [31] or without [17] code context using RNNs. CoDiSum [53] and PtrGNCMsg [27] attempt to resolve the out-of-vocabulary problem. FIRA represents code changes using fine-grained code graphs and generates summaries via a GNN encoder and a Transformer decoder [7]. Additionally, there are surveys studying these techniques [6, 29, 44, 54]. Work for other change-related tasks, such as just-in-time comment update [25], just-in-time defect prediction [18, 33, 37], and code refinement [12, 23, 49], have also been proposed recently.

Among them, the closely related to ours is [54], which specifically examined LM's change understanding capabilities for the change summarization task. Both our study and this work reached similar conclusions, finding that when prompted appropriately, GPT-4 demonstrated the best summarization capabilities among the studied LMs. However, our work specifically focuses on the practicality of deploying SLMs. We found that SLMs can demonstrate competitive change understanding capabilities in three change-related tasks after being finetuned by a small yet high-quality dataset like HQCM. This provides support for leveraging SLMs in environments with security, computational, and financial constraints. This distinction sets our work apart from theirs.

General-Purpose Understanding. CC2Vec devises a hierarchical attention network to extract features from a code change and transform the code change into a distributed vector [13]. Unlike CC2Vec, CCRep is self-supervised and was pre-trained with task-specific components [28]. Apart from change encoders, there are also pre-trained models. CodeReviewer was pre-trained for various code review activities typically involving code changes [23]. CCT5 is a recently proposed, SOTA model specifically for code changes. It was pre-trained with five carefully designed, change-related tasks and a large-scale dataset [24]. All these works are orthogonal to our work and we finetuned CCT5 for evaluation.

6 CONCLUSION

State-of-the-art techniques for change understanding suffer from "restricted code-change understanding" and "biased code-change datasets" problems, while techniques using LMs overly rely on large LMs. These hinder them from being deployed massively in practice. To mitigate these issues, we created a small yet high-quality dataset called HQCM and finetuned small LMs based on it. Our evaluation confirmed the effectiveness of HQCM in finetuning SLMs towards change-related tasks. More importantly, it suggested that SLMs, after finetuning by HQCM, can achieve competitive (comparable or superior) performance with state-of-the-art baselines and LLMs in change understanding for three change-related tasks: change summarization, change classification, and code refinement. We believe that this answers the open question positively.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their valuable feedback and discussions on future work. This work was supported by Ant Group and State Key Laboratory of Industrial Control Technology, China (Grant No. ICT2024C01). Cong Li is with Zhejiang University and Ant Group, where the former is the first institution of this work. Zhaogui Xu (zhengrong.xzg@antgroup.com) and Peng Di (dipeng.dp@antgroup.com) are the corresponding authors.

REFERENCES

[1] Angular. 2023. Angular. https://github.com/angular/angular

[2] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models Are Few-Shot Learners. In *Proceedings of the 2020 International Conference on Neural Information Processing Systems (NIPS '20)*.

[3] Raymond P.L. Buse and Westley R. Weimer. 2010. Automatically Documenting Program Changes. In *Proceedings of the 2010 IEEE/ACM International Conference on Automated Software Engineering (ASE '10)*. https://doi.org/10.1145/1858996.1859005

[4] Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Anastasios Nikolas Angelopoulos, Tianle Li, Dacheng Li, Hao Zhang, Banghua Zhu, Michael Jordan, Joseph E. Gonzalez, and Ion Stoica. 2024. Chatbot Arena: An Open Platform for Evaluating LLMs by Human Preference. arXiv:2403.04132 [cs.AI]

[5] Convential Commits. 2023. Conventional Commits: A Specification for Adding Human and Machine Readable Meaning to Commit Messages. https://www.conventionalcommits.org/en/v1.0.0

[6] Jinhao Dong, Yiling Lou, Dan Hao, and Lin Tan. 2023. Revisiting Learning-Based Commit Message Generation. In *Proceedings of the 2023 International Conference on Software Engineering (ICSE '23)*. https://doi.org/10.1109/ICSE48619.2023.00075

[7] Jinhao Dong, Yiling Lou, Qihao Zhu, Zeyu Sun, Zhilin Li, Wenjie Zhang, and Dan Hao. 2022. FIRA: Fine-Grained Graph-Based Code Change Representation for Automated Commit Message Generation. In *Proceedings of the 2022 International Conference on Software Engineering (ICSE '22)*. https://doi.org/10.1145/3510003.3510069

[8] Robert Dyer, Hoan Anh Nguyen, Hridesh Rajan, and Tien N. Nguyen. 2013. Boa: A Language and Infrastructure for Analyzing Ultra-Large-Scale Software Repositories. In *Proceedings of the 2013 International Conference on Software Engineering (ICSE '13)*.

[9] Aryaz Eghbali and Michael Pradel. 2022. CrystalBLEU: Precisely and Efficiently Measuring the Similarity of Code. In *Proceedings of the 2022 ACM/IEEE International Conference on Software Engineering: Companion Proceedings (ICSE '22)*. https://doi.org/10.1145/3510454.3528648

[10] A.E. Elo. 1966. *The USCF Rating System: Its Development, Theory, and Applications.* United States Chess Federation.

[11] Mingyang Geng, Shangwen Wang, Dezun Dong, Haotian Wang, Ge Li, Zhi Jin, Xiaoguang Mao, and Xiangke Liao. 2024. Large Language Models are Few-Shot Summarizers: Multi-Intent Comment Generation via In-Context Learning. In *Proceedings of the 2024 IEEE/ACM International Conference on Software Engineering (ICSE '24)*. https://doi.org/10.1145/3597503.3608134

[12] Qi Guo, Junming Cao, Xiaofei Xie, Shangqing Liu, Xiaohong Li, Bihuan Chen, and Xin Peng. 2024. Exploring the Potential of ChatGPT in Automated Code Refinement: An Empirical Study. In *Proceedings of the 2024 IEEE/ACM International Conference on Software Engineering (ICSE '24)*. https://doi.org/10.1145/3597503.3623306

[13] Thong Hoang, Hong Jin Kang, David Lo, and Julia Lawall. 2020. CC2Vec: Distributed Representations of Code Changes. In *Proceedings of the 2020 ACM/IEEE International Conference on Software Engineering (ICSE '20)*. https://doi.org/10.1145/3377811.3380361

[14] Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-Rank Adaptation of Large Language Models. In *Proceedings of the 2022 International Conference on Learning Representations*.

[15] Yuan Huang, Qiaoyang Zheng, Xiangping Chen, Yingfei Xiong, Zhiyong Liu, and Xiaonan Luo. 2017. Mining Version Control System for Automatically Generating Commit Comment. In *Proceedings of the 2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '17)*. https://doi.org/10.1109/ESEM.2017.56

[16] HuggingFace. 2023. Open LLM Leaderboard. https://huggingface.co/spaces/open-llm-leaderboard/open_llm_leaderboard

[17] Siyuan Jiang, Ameer Armaly, and Collin McMillan. 2017. Automatically Generating Commit Messages from Diffs using Neural Machine Translation. In *Proceedings of the 2017 IEEE/ACM International Conference on Automated Software Engineering (ASE '17)*.

[18] Yasutaka Kamei, Emad Shihab, Bram Adams, Ahmed E. Hassan, Audris Mockus, Anand Sinha, and Naoyasu Ubayashi. 2013. A Large-Scale Empirical Study of Just-in-Time Quality Assurance. *IEEE Transactions on Software Engineering* 39, 6 (2013), 757–773. https://doi.org/10.1109/TSE.2012.70

[19] Sungmin Kang, Juyeon Yoon, and Shin Yoo. 2023. Large Language Models Are Few-Shot Testers: Exploring LLM-Based General Bug Reproduction. In *Proceedings of the 2023 International Conference on Software Engineering (ICSE '23)*. https://doi.org/10.1109/ICSE48619.2023.00194

[20] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling Laws for Neural Language Models. arXiv:2001.08361 [cs.LG]

[21] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large Language Models are Zero-Shot Reasoners. In *Advances in Neural Information Processing Systems (NeurIPS '22)*.

[22] Tien-Duy B. Le, Jooyong Yi, David Lo, Ferdian Thung, and Abhik Roychoudhury. 2014. Dynamic Inference of Change Contracts. In *Proceedings of the 2014 IEEE International Conference on Software Maintenance and Evolution (ICSME '14)*. https://doi.org/10.1109/ICSME.2014.72

[23] Zhiyu Li, Shuai Lu, Daya Guo, Nan Duan, Shailesh Jannu, Grant Jenks, Deep Majumder, Jared Green, Alexey Svyatkovskiy, Shengyu Fu, and Neel Sundaresan. 2022. Automating Code Review Activities by Large-Scale Pre-Training. In *Proceedings of the 2022 ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '22)*. https://doi.org/10.1145/3540250.3549081

[24] Bo Lin, Shangwen Wang, Zhongxin Liu, Yepang Liu, Xin Xia, and Xiaoguang Mao. 2023. CCT5: A Code-Change-Oriented Pre-trained Model. In *Proceedings of the 2023 ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '23)*. https://doi.org/10.1145/3611643.3616339

[25] Bo Lin, Shangwen Wang, Zhongxin Liu, Xin Xia, and Xiaoguang Mao. 2023. Predictive Comment Updating With Heuristics and AST-Path-Based Neural Learning: A Two-Phase Approach. *IEEE Transactions on Software Engineering* 49, 4 (2023). https://doi.org/10.1109/TSE.2022.3185458

[26] Mario Linares-Vásquez, Luis Fernando Cortés-Coy, Jairo Aponte, and Denys Poshyvanyk. 2015. ChangeScribe: A Tool for Automatically Generating Commit Messages. In *Proceedings of the 2015 IEEE/ACM International Conference on Software Engineering (ICSE '15)*. https://doi.org/10.1109/ICSE.2015.229

[27] Qin Liu, Zihe Liu, Hongming Zhu, Hongfei Fan, Bowen Du, and Yu Qian. 2019. Generating Commit Messages from Diffs using Pointer-Generator Network. In *Proceedings of the 2019 International Conference on Mining Software Repositories (MSR '19)*. https://doi.org/10.1109/MSR.2019.00056

[28] Zhongxin Liu, Zhijie Tang, Xin Xia, and Xiaohu Yang. 2023. CCRep: Learning Code Change Representations via Pre-Trained Code Model and Query Back. In *Proceedings of the 2023 International Conference on Software Engineering (ICSE '23)*. https://doi.org/10.1109/ICSE48619.2023.00014

[29] Zhongxin Liu, Xin Xia, Ahmed E. Hassan, David Lo, Zhenchang Xing, and Xinyu Wang. 2018. Neural-Machine-Translation-Based Commit Message Generation: How Far Are We?. In *Proceedings of the 2018 ACM/IEEE International Conference on Automated Software Engineering (ASE '18)*. https://doi.org/10.1145/3238147.3238190

[30] Pablo Loyola, Edison Marrese-Taylor, and Yutaka Matsuo. 2017. A Neural Architecture for Generating Natural Language Descriptions from Source Code Changes. In *Proceedings of the 2017 Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers) (ACL '17)*. https://doi.org/10.18653/v1/P17-2045

[31] Pablo Loyola, Edison Marrese-Taylor, and Yutaka Matsuo. 2017. A Neural Architecture for Generating Natural Language Descriptions from Source Code Changes. In *Proceedings of the 2017 Annual Meeting of the Association for Computational Linguistics (Short Papers) (ACL '17)*. https://doi.org/10.18653/v1/P17-2045

[32] Wei Ma, Shangqing Liu, Zhihao Lin, Wenhan Wang, Qiang Hu, Ye Liu, Cen Zhang, Liming Nie, Li Li, and Yang Liu. 2024. LLMs: Understanding Code Syntax and Semantics for Code Analysis. arXiv:2305.12138 [cs.SE]

[33] Chao Ni, Wei Wang, Kaiwen Yang, Xin Xia, Kui Liu, and David Lo. 2022. The Best of Both Worlds: Integrating Semantic Features with Expert Features for Defect Prediction and Localization. In *Proceedings of the 2022 ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '22)*. https://doi.org/10.1145/3540250.3549165

[34] OpenAI. 2023. GPT-4 Technical Report. arXiv:2303.08774 [cs.CL]

[35] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training Language Models to Follow Instructions with Human Feedback. In *Advances in Neural Information Processing Systems (NeurIPS '22)*.

[36] Rangeet Pan, Ali Reza Ibrahimzada, Rahul Krishna, Divya Sankar, Lambert Pouguem Wassi, Michele Merler, Boris Sobolev, Raju Pavuluri, Saurabh Sinha, and Reyhaneh Jabbarvand. 2024. Lost in Translation: A Study of Bugs Introduced by Large Language Models while Translating Code. In *Proceedings of 2024 IEEE/ACM International Conference on Software Engineering (ICSE '24)*. https://doi.org/10.1145/3597503.3639226

[37] Chanathip Pornprasit and Chakkrit Tantithamthavorn. 2021. JITLine: A Simpler, Better, Faster, Finer-grained Just-In-Time Defect Prediction. arXiv:2103.07068 [cs.SE]

[38] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language Models are Unsupervised Multitask Learners. *OpenAI blog* 1, 8 (2019).

[39] Shuo Ren, Daya Guo, Shuai Lu, Long Zhou, Shujie Liu, Duyu Tang, Neel Sundaresan, Ming Zhou, Ambrosio Blanco, and Shuai Ma. 2020. CodeBLEU: a Method for Automatic Evaluation of Code Synthesis. arXiv:2009.10297 [cs.SE]

[40] Xiaoxia Ren, Fenil Shah, Frank Tip, Barbara G. Ryder, and Ophelia Chesley. 2004. Chianti: A Tool for Change Impact Analysis of Java Programs. In *Proceedings of the 2004 Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA '04)*. https://doi.org/10.1145/1028976.1029012

[41] Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2023. Code Llama: Open Foundation Models for Code. arXiv:2308.12950 [cs.CL]

[42] Jinfeng Shen, Xiaobing Sun, Bin Li, Hui Yang, and Jiajun Hu. 2016. On Automatic Summarization of What and Why Information in Source Code Changes. In *Proceedings of the 2016 IEEE Annual Computer Software and Applications Conference (COMPSAC '16)*. https://doi.org/10.1109/COMPSAC.2016.162

[43] Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. 2020. MPNet: Masked and Permuted Pre-Training for Language Understanding. In *Proceedings of the 2020 International Conference on Neural Information Processing Systems (NIPS '20)*.

[44] Wei Tao, Yanlin Wang, Ensheng Shi, Lun Du, Shi Han, Hongyu Zhang, Dongmei Zhang, and Wenqiang Zhang. 2022. A Large-Scale Empirical Study of Commit Message Generation: Models, Datasets and Evaluation. *Empirical Softw. Engg.* 27, 7 (2022). https://doi.org/10.1007/s10664-022-10219-1

[45] Yida Tao, Yingnong Dang, Tao Xie, Dongmei Zhang, and Sunghun Kim. 2012. How do Software Engineers Understand Code Changes? An Exploratory Study in Industry. In *Proceedings of the 2012 ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE '12)*. https://doi.org/10.1145/2393596.2393656

[46] Yingchen Tian, Yuxia Zhang, Klaas-Jan Stol, Lin Jiang, and Hui Liu. 2022. What Makes a Good Commit Message?. In *Proceedings of the 2022 International Conference on Software Engineering (ICSE '22)*. https://doi.org/10.1145/3510003.3510205

[47] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton,

Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models. arXiv:2307.09288 [cs.CL]

[48] Sentence Transformer. 2023. Pretrained Models. https://sbert.net/docs/sentence_transformer/pretrained_models.html#original-models

[49] Michele Tufano, Jevgenija Pantiuchina, Cody Watson, Gabriele Bavota, and Denys Poshyvanyk. 2019. On Learning Meaningful Code Changes via Neural Machine Translation. In *Proceedings of the 2019 International Conference on Software Engineering (ICSE '19)*. https://doi.org/10.1109/ICSE.2019.00021

[50] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. 2022. Emergent Abilities of Large Language Models. *Transactions on Machine Learning Research* (2022).

[51] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. 2022. Chain of Thought Prompting Elicits Reasoning in Large Language Models. In *Advances in Neural Information Processing Systems (NeurIPS '22)*.

[52] Chunqiu Steven Xia, Yuxiang Wei, and Lingming Zhang. 2023. Automated Program Repair in the Era of Large Pre-Trained Language Models. In *Proceedings of the 2023 International Conference on Software Engineering (ICSE '23)*. https://doi.org/10.1109/ICSE48619.2023.00129

[53] Shengbin Xu, Yuan Yao, Feng Xu, Tianxiao Gu, Hanghang Tong, and Jian Lu. 2019. Commit Message Generation for Source Code Changes. In *Proceedings of the 2019 International Joint Conference on Artificial Intelligence (IJCAI '19)*.

[54] Pengyu Xue, Linhao Wu, Zhongxing Yu, Zhi Jin, Zhen Yang, Xinyi Li, Zhenyu Yang, and Yue Tan. 2024. Automated Commit Message Generation with Large Language Models: An Empirical Study and Beyond. arXiv:2404.14824 [cs.SE]