

000 001 *Coder-R³: RECOGNIZE, REVIEW AND REPAIR DEFEC- 002 TIVE CODE WITH FINETUNED LLMs IN PRACTICE* 003 004

005 **Anonymous authors**
006 Paper under double-blind review
007
008
009

ABSTRACT

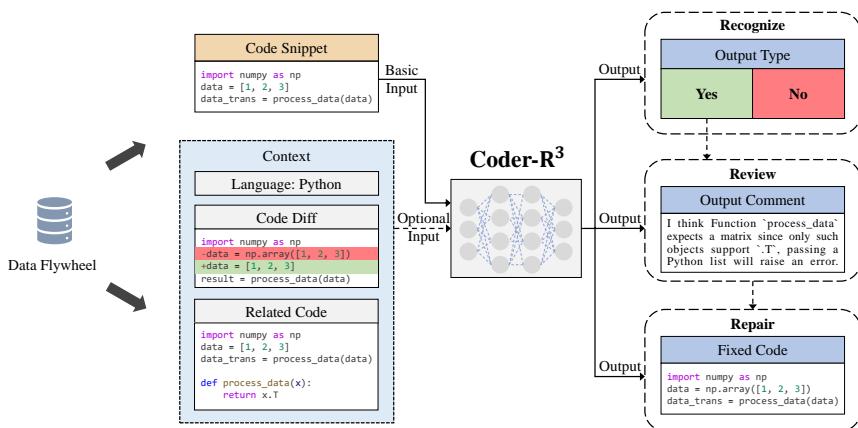
011 Large language models have emerged as powerful tools for software engineering
012 tasks, demonstrating particular promise in code review activities. However, ex-
013 isting research on code review LLMs typically decomposes the review process
014 into discrete subtasks, collecting data and fine-tuning separate models for each
015 individual component. This fragmented approach overlooks the synergistic re-
016 lationships between different tasks, necessitates multiple models with complex
017 multi-stage invocations, and consequently exhibits limited practical applicabil-
018 ity in real-world deployment scenarios. In this work, we advance beyond previ-
019 ous code review research by proposing a unified and comprehensive code review
020 problem modeling approach. We focus on the complete code review process, in-
021 cluding Recognize, Review, and Repair defective code fragments consecutively,
022 and propose *Coder-R³*, an approach that enables a single LLM to handle all code
023 review-related subtasks uniformly. Additionally, we establish a practically fea-
024 sible closed-loop iterative process for industrial scenarios, encompassing data con-
025 struction, model evaluation, and operational feedback integration. We rigorously
026 evaluate the effectiveness of various strategies, including input context selection,
027 output format, and training methodologies. *Coder-R³* achieves state-of-the-art
028 performance on the CodeReviewer benchmark, and demonstrates superior effec-
029 tiveness in real enterprise scenarios. Our work provides valuable insights for en-
030 terprises seeking to leverage LLMs to enhance code review efficiency.
031

1 INTRODUCTION

034 The code review activity (Li et al., 2022; Tufano et al., 2022; Lu et al., 2023a; Huq et al., 2022) is an
035 integral component in modern software engineering workflow. Having been widely adopted across
036 development platforms like GitHub, the code review process significantly contributes to software
037 quality assurance and long-term maintainability. However, code review has always been a difficult
038 and time-consuming activity both for reviewers and developers. The previous research (Bosu &
039 Carver, 2013) show that developers spend 6.4 hours on average weekly reviewing other developers'
040 code. With the advance of information technology, software systems have become increasingly
041 complex, making the code review process more challenging.

042 To address this challenge, recent research has increasingly focused on automating code review
043 tasks (Tufano et al., 2024). Large language models (LLMs) (Radford et al., 2018; 2019; Brown
044 et al., 2020; Achiam et al., 2023), with their strong linguistic capabilities, have shown considerable
045 potential in handling various programming-related tasks. In an effort to alleviate the workload of
046 software developers, many studies have explored the application of LLMs to automate code review
047 processes (Hossain et al., 2024; Caumartin et al., 2025; Xue et al., 2024), achieving notable advan-
048 tages in this area. Generally, the automated code review process is divided into three subtasks, in-
049 cluding code change quality estimation, review comment generation, and code refinement (Li et al.,
050 2022). However, due to the limitations of methodology and data, plenty of studies still focus on
051 a single subtask (Dai et al., 2025; Sobania et al., 2023; Lopes et al., 2024; Cai et al., 2023), such
052 as review comment generation or code refinement, resulting in a lack of overall code review ca-
053 pability. Some other studies work on the whole process, but perform these subtasks by individual
models (Guo et al., 2024; Begolli et al., 2025), making the whole process complex and computa-
tionally intensive. Additionally, code review data, while relatively easy to collect, often suffers from

054
055
056
057
058
059
060
061
062
063
064
065
066
067
068



069 **Figure 1: The framework of our proposed Coder-R³.** Data are collected through a data flywheel
070 mechanism. The code snippet is the code that is being reviewed. The context includes additional
071 information helps the model to understand the code, such as programming language, code diff,
072 and related code. Taking the code snippet and context as input, the *Coder-R³* model perform the
073 **Recognize**, **Review** and **Repair** tasks in a unified manner.

074
075 inconsistent quality, which in turn constrains the real-world deployment of models. Overall, the
076 current automated code review is not practical enough.

077 In this work, we focus on the comprehensive code review task, *i.e.* given a static code fragment,
078 the model firstly recognizes the defect in the code, then writes a review explaining the defect and
079 generates repaired code if the code has deflection. This is a unified form of the three code review
080 subtasks, emphasizing the connection among them. We train a single LLM to accomplish the
081 comprehensive task, which is far more practical and efficient in real-world scenarios. Specifically, we
082 introduce *Coder-R³*, an approach to finetune LLMs in practical code review scenarios. As shown
083 in Fig. 1, we fine-tune LLMs via Supervised Fine-Tuning (SFT) with uniformly formatted datasets
084 from data flywheel, improving the overall ability for code review. Through extensive experimen-
085 tation, we investigated the impact of various input configurations, output representations, and training
086 strategies on the code review task, ultimately identifying the optimal approach. Our method achieves
087 state-of-the-art results on the CodeReviewer benchmark.

088 In terms of practical application, we integrate the model into the company’s actual code review
089 system and establish a data flywheel mechanism. Within this system, the model generates review
090 suggestions, which are then validated and annotated by human reviewers. The resulting data is
091 systematically collected, filtered, and preprocessed to form a high-quality code review dataset. By
092 iteratively training on this continuously updated dataset and redeploying the improved model, we
093 create a closed-loop pipeline for continuous enhancement. Our experiments validate the method’s
094 strong effectiveness on company internal data and demonstrate its promising performance in real-
095 world deployment.

096 Overall, our work is distinguished by the following aspects: (1) We unify the modeling of the code
097 review task and propose a method *Coder-R³* designed for completing the whole code review process
098 with a single LLM. (2) We perform extensive experiments to evaluate the effectiveness of various
099 strategies, demonstrating the practicality and effectiveness of our method. (3) We deploy an iterative,
100 data-centric approach that combines public datasets with real-world scenarios to continuously
101 improve model performance. (4) Our approach achieves state-of-the-art results on the Codereviewer
102 dataset, and also reveals strong performance in real-world scenarios.

103 2 RELATED WORK

104 CodeReviewer (Li et al., 2022) is a pioneering work that divides the code review process into three
105 sub-tasks: code change quality estimation, review comment generation and code refinement. It

108 introduces a large-scale dataset for these tasks and employs pre-training and fine-tuning strategies
 109 to solve each task effectively.
 110

111 Recent research has addressed code review sub-tasks individually. For example, CRScore++ (Ka-
 112 padnis et al., 2025) integrates reinforcement learning with analysis tools and LLM feedback to op-
 113 timize review comment generation, while ThinkRepair (Yin et al., 2024) uses thought-chain knowl-
 114 edge bases to improve vulnerability reasoning and code repair. However, these approaches focus on
 115 single tasks and lack comprehensive review capabilities. Some newer studies explore connections
 116 between tasks. DISCOREV (Ben Sghaier & Sahraoui, 2024) distills repair models to train review
 117 models, and CORAL (Sghaier et al., 2025a) enhances this with reinforcement learning and diversi-
 118 fied feedback. Although these methods show promising results, they still rely on separate models for
 119 different tasks, increasing complexity and resource requirements. CFT (Kumar & Chimalakonda,
 120 2024) attempts to train LLM using federated multi-task learning, but gains unsatisfactory results.
 121

122 Additionally, existing code review datasets have several limitations. The CodeReviewer dataset,
 123 for instance, collects data for its three tasks separately from inconsistent sources, leading to in-
 124 formational gaps that complicate unified processing. Moreover, as it was crawled from GitHub
 125 without strict filtering, the dataset contains a significant amount of low-quality text. A recent study,
 126 CuREV (Sghaier et al., 2025b), used LLMs to clean the CodeReviewer data and found a large vol-
 127 ume of irrelevant, unclear, or verbose content, underscoring the dataset’s inherent weaknesses.
 128

129 With respect to the context, most code review studies have overlooked its significance, relying solely
 130 on code snippets or code changes as model input. A recent research (Guo et al., 2025) demonstrates
 131 that inputting the review line number improves code repairing significantly, but it lacks practicability
 132 in real-world scenarios. Compared with code review tasks, the SWE-Bench (Jimenez et al., 2024)
 133 task emphasizes code context, as it requires inspecting vulnerabilities across the entire repository.
 134 These two types of tasks share certain similarities, suggesting that code review could also benefit
 135 from contextual information.
 136

137 3 APPROACH

138 3.1 TASK FORMULATION

139 We formulate the code review task as: leveraging the relevant context, perform categorization
 140 (**R**ecognize), explanation (**R**eview), and fixing (**R**epair) for the input code snippet. We will de-
 141 velop a unified model capable of performing these three tasks simultaneously, addressing these three
 142 highly interrelated tasks in a single framework.
 143

144 As shown in Fig. 1, the input of this task is $X = \langle \text{code_snippet}, \text{context} \rangle$. Here, the code snippet
 145 (*code_snippet*) is extracted from the modified location, and concatenated with the preceding and
 146 succeeding N lines. Compared with using code changes as input, adopting code snippets is a more
 147 general approach. The relevant context (*context*) may include various forms such as function calls,
 148 documentation, or code diffs. When the context is empty, the model is expected to review only the
 149 *code_snippet* itself. The output is represented as $Y = \langle \text{type}, \text{comment}, \text{fix_code} \rangle$. Each field is
 150 explained as follows: **type**: This indicates the classification categories. Similar to CodeReviewer,
 151 ‘yes’ signifies that a review is needed, while ‘no’ means no review is necessary. **comment**: For
 152 defective code, the model should generate comments that explain the issue and propose possible
 153 solutions. **fix_code**: For defective code, the output should include the corrected version of the code
 154 snippet based on the comment.
 155

156 In our experiments, we employed various input-output configurations and training strategies to ex-
 157 plore best practices for the code review task. For the input, the contextual information of the code
 158 included one or more of the following elements: programming language, code change diff, and ex-
 159 tracted contextual code segments. For the output of the repair task, the result could be represented
 160 either as a modified code snippet or in the form of a diff. Regarding the modeling approach, one
 161 method involved a single model performing all three tasks simultaneously, directly generating a
 162 structured output containing $\langle \text{type}, \text{comment}, \text{fix_code} \rangle$. This integrated approach offered faster
 163 inference but posed greater learning challenges. Alternatively, a more conventional automatic code
 164 review pipeline was implemented by invoking the model three separate times, each dedicated to one
 165

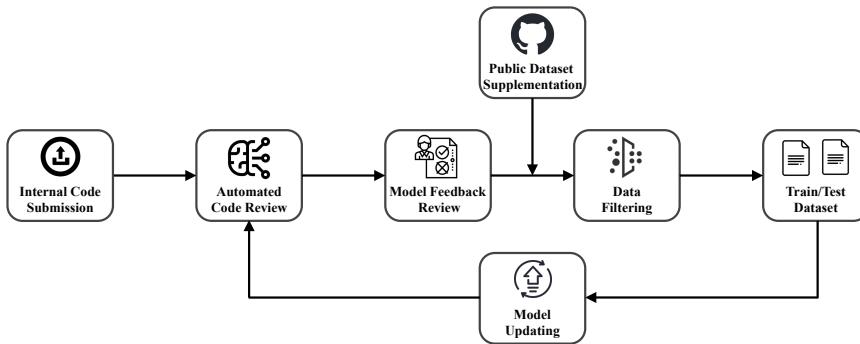


Figure 2: **Data flywheel.** Within the data flywheel, company employees submit code, which is automatically reviewed by the model. The review outcomes, combined with publicly available code data, undergo human verification and systematic data filtering. The resulting high-quality data are then used to iteratively retrain the model, establishing a self-reinforcing cycle that progressively enhances both model performance and dataset quality.

specific task. Both approaches utilize only a single model. We refer to these two methodologies as joint training and hybrid training, respectively.

3.2 CODE CONTEXT CONSTRUCTION

Language. The programming language of the code has a significant impact on defect analysis. For the majority of the collected data, language labels are already available. For the small portion of data without such annotations, we employed the open-source tool GuessLang¹ along with the LLMs Qwen2.5-Coder-32B (Hui et al., 2024) and Qwen3-Coder-30B-A3B (Team, 2025) to infer the programming language. The final label for each instance was determined through majority voting among the three systems.

Code diff. The code diff, which refers to the modifications made to the code compared to its previous version, serves as additional information to highlight potential defect regions for the model. During dataset construction, the code changes submitted by users are extracted and utilized as the code diff.

Related code. The functions invoked within a code snippet may not be included in the given snippet. Consequently, incorporating related code can provide complementary contextual information, thereby enabling a more comprehensive analysis of potential code defects. We parse source code files into abstract syntax trees using Tree-sitter² and adopt functions as the slicing unit. For each function snippet, we additionally capture its internal call relations, which are the set of functions invoked within its body. This approach allows each snippet to be represented not only by its own implementation but also by the contextual information regarding dependencies on other functions.

3.3 DATA FLYWHEEL

Manual annotation of code review data demands expert-level programmers and remains highly time-consuming. Even with model-assisted pre-annotation, each instance still takes an average of 3 ~ 5 minutes. In practice, we designed a relatively low-cost strategy as shown in Fig. 2: deploying the system internally to collect human feedback, cleaning publicly available datasets, and introducing a data filtering process to ensure quality.

Internal Data Collection. We collect internal data by integrating open-source models (e.g., Qwen2.5-Coder) and commercial APIs (e.g., GPT-5) into a company’s code management platform. When an engineer submits a merge request, the system reviews the code snippets and contextual information. Approved code is integrated directly, while flagged segments receive model-generated annotations for issues and fixes. Reviewers adjust these annotations and provide feedback by accepting or rejecting suggestions. Accepted recommendations are paired with the final merged code to create high-quality labeled data, enhancing the training dataset and improving the model iteratively.

¹<https://github.com/yoeo/guesslang>

²<https://github.com/tree-sitter/tree-sitter>

216 **Public Dataset Supplementation.** Additional data are collected from public datasets and GitHub,
 217 then mapped to a standardized format. These data are used for training models, increasing the size
 218 of the dataset and enhancing the generalizability of the model.

219 **Data Filtering.** For the collected data, we conducted multiple rounds of human-AI collaborative
 220 screening to ensure high quality. An initial consistency check was performed using multi-round
 221 model voting. Data with low consensus was subsequently reviewed and corrected by human an-
 222 notators. We developed a lightweight annotation system that allows code experts to examine and
 223 modify defect categories, locations, review comments, and repair suggestions. In the first round of
 224 annotation, we engaged 21 code experts to inspect and label 968 instances.

225 **Iterative Process.** The process follows a regular cyclic schedule, typically every one to two weeks,
 226 during which new data is collected to train an updated model. This model is then deployed to the
 227 production environment, facilitating a subsequent round of data collection and enabling continuous
 228 and scheduled model iteration.

230 3.4 MODEL TRAINING AND EVALUATION

232 We utilize Supervised Fine-Tuning to train *Coder-R*³. The base pre-trained model is Qwen2.5-
 233 Coder-7B (Hui et al., 2024). The training of the model *Coder-R*³_{SFT} requires the raw
 234 dataset to be in the following format: $D_{SFT} = (X_i = \langle \text{code_snippet}, \text{context} \rangle, Y_i =$
 235 $\langle \text{type}, \text{comment}, \text{fix_code} \rangle)$. Instruction fine-tuning data in the form of question–answer (QA)
 236 pairs is generated through embedding D_{SFT} into the corresponding template $Prompt_{SFT}$. Q de-
 237 notes the question serving as the input prompt, while $A = (A_1, A_2, \dots, A_T)$ represents the reference
 238 answer consisting of a sequence of tokens. This data is then used for SFT training, in which tokens
 239 of the model’s responses are compared with the ground-truth answers using cross-entropy loss to
 240 optimize the model.

$$241 \quad \mathcal{L}_{SFT}(\theta) = - \sum_{t=1}^T \log P_{\theta}(A_t | Q, A_{<t}) \quad (1)$$

244 At each time step t , the model with parameters θ predicts the probability of the next token A_t con-
 245 ditioned on the question Q and the previously generated tokens $A_{<t}$. The objective is to maximize
 246 the likelihood of the reference answer sequence.

247 4 EXPERIMENTS ON PUBLIC DATASET

250 4.1 THE CODEREVIEWER DATASET

252 We choose Codereviewer (Li et al., 2022) dataset from several public datasets because it’s the
 253 only large-scale multilingual public dataset possessing data on the three code review tasks. Despite
 254 some quality issues, Codereviewer dataset is a commonly used dataset in code review tasks. Other
 255 datasets like Tufano (Tufano et al., 2022) and CodeXGLUE (Lu et al., 2021) lack some information
 256 or collect data for each task from completely different sources, which are unsuitable for our code
 257 review setting. We performed a series of preprocessing steps on the dataset to align it with the
 258 specific requirements of our task setup. The details of the dataset and the preprocessing procedure
 259 are described in Appendix A.1.

260 4.2 EXPERIMENTAL SETTING

263 We implemented our approach using the LLaMA-Factory (Zheng et al., 2024) framework, employ-
 264 ing LoRA (Hu et al., 2022) technique during SFT to enable efficient fine-tuning. The training is
 265 finished on four H100 80G GPUs with the learning rate of $2e - 5$, and the batch size for each GPU
 266 is 1 because of the long context.

268 4.3 EVALUATION METRICS

269 We evaluate the three code review tasks following Codereviewer’s setting.

270
 271 **Table 1: Ablations on the Codereviewer dataset.** The configuration utilizing language identifiers
 272 and code diffs as input context, code snippets as output, and a hybrid training strategy yielded the
 273 best overall performance.

274 275 Language	276 Context Diff	277 Task	278 Output	279 Recognize				280 Review		281 Repair			
				Precision	282 Recall	F1-Score	283 Accuracy	284 Accuracy	285 BLEU	286 Accuracy	287 BLEU-review	288 BLEU-code	289 EM
	\times	Joint	Snippet	56.73	59.28	57.98	57.14	55.36	5.47	79.15	8.07	85.42	9.27
	✓	Joint	Snippet	57.00	58.25	57.62	57.21	54.52	5.32	77.65	8.25	85.43	9.51
	✓	Joint	Snippet	99.90	13.80	24.25	56.94	4.57	5.37	99.72	9.76	85.53	16.12
	\times	Recognize	Snippet	75.34	61.39	67.65	70.66	-	-	-	-	-	-
	\times	Review	Snippet	-	-	-	-	-	5.39	-	-	-	-
	\times	Repair	Snippet	-	-	-	-	-	-	-	-	85.58	41.05
	\times	Hybrid	Snippet	79.33	58.77	67.52	71.74	-	5.33	-	-	85.65	40.48
	✓	Hybrid	Snippet	76.25	63.47	69.28	71.86	-	5.37	-	-	85.58	41.15
	✓	Hybrid	Snippet	79.68	71.60	75.43	76.68	-	5.61	-	-	86.97	45.47
	✓	Hybrid	Diff	81.14	60.85	69.55	73.36	-	5.68	-	-	85.54	40.89

283
 284 **Recognize Task.** This is a binary classification task aimed at determining whether a given code
 285 snippet requires review. We use precision, recall, F1-score, and accuracy to evaluate the model’s
 286 predictions. For the calculation of precision, recall, and F1-score, code snippets that need review are
 287 treated as the positive class. This helps in clearly quantifying how well the model identifies snippets
 288 that actually require attention from reviewers.

289 **Review Task.** For this sequence generation task, we adopt the BLEU (Bilingual Evaluation Under-
 290 study) (Papineni et al., 2002) score to assess the quality of the generated review comments.

$$292 \quad 293 \quad BLEU = BP \cdot \exp \left(\frac{1}{N} \sum_{n=1}^N \log p_n \right) \quad (2)$$

$$294$$

$$296 \quad 297 \quad BP = \begin{cases} 1 & \text{if } c > r \\ \exp(1 - \frac{r}{c}) & \text{if } c \leq r \end{cases} \quad (3)$$

$$298$$

299 where p_n denotes the proportion of n-grams in the candidate text that also appear in the reference
 300 text. c and r denote the lengths of the candidate text and the reference text. N is the maximum
 301 n-gram order used in the calculation, which is set to 4 in practice. And the brevity penalty BP is
 302 introduced to penalize candidate texts that are too short. When computing BLEU, all whitespace
 303 is normalized to a single space and punctuation is separated from words as independent tokens,
 304 providing a consistent tokenization and spacing scheme for n-gram statistics.

305 **Repair Task.** To evaluate the performance of this task, we calculate the BLEU score between the
 306 generated code and the target code, as well as the exact match (EM) rate. When computing the
 307 BLEU score for code snippets, we preprocess the data by removing leading and trailing spaces from
 308 each instance and compressing consecutive spaces into a single space, consistent with the procedure
 309 in Codereviewer(Li et al., 2022) to ensure fairness. While BLEU reflects the similarity between the
 310 generated code and the target code, the EM rate is more critical here because code is highly sensitive
 311 to changes—even minor modifications can lead to compilation errors or runtime exceptions. Only
 312 when the generated code is completely identical to the target code is it considered a successful
 313 refinement.

314 4.4 ABLATION STUDY

315 We investigate the contribution of each variable via extensive ablations, and shown in Tab. 1.

316 **Modeling Approaches.** For the joint training approach, the calculation of BLEU and Exact Match
 317 (EM) scores excluded samples that were incorrectly classified as “requiring no review”, consider-
 318 ing only those correctly identified for further tasks. Across all experimental settings, joint training
 319 consistently underperformed compared to hybrid training. This may be attributed to the excessive
 320 information the model must process simultaneously in the joint setup, hindering its ability to effec-
 321 tively address each subtask sequentially. The results suggest that integrating all three tasks into a
 322 single model remains a challenging objective.

324
 325 **Table 2: Evaluation on the Codereviewer dataset.** *Coder-R³* achieves state-of-the-art results on
 326 the Recognize and Repair tasks, surpassing all baseline pre-trained LLMs and most of prior works.
 327 (- indicates that the model did not perform this task or did not test this metric. * indicates the result
 328 of using DeepSeek-7B, as it matches *Coder-R³* in parameter scale.)

Methods	Recognize				Review	Repair	
	Precision	Recall	F1-Score	Accuracy	BLEU	BLEU	EM
Qwen-2.5-Coder-7B (Hui et al., 2024)	47.05	13.65	21.17	49.16	0.63	74.96	22.38
Qwen-2.5-Coder-14B (Hui et al., 2024)	56.32	8.64	14.99	50.98	0.54	74.91	26.22
Qwen-2.5-Coder-32B (Hui et al., 2024)	54.93	39.87	46.21	53.59	0.51	72.69	26.36
OpenAI GPT-5 (OpenAI, 2025)	51.06	75.00	60.75	51.30	0.42	76.25	31.63
Codereviewer (Li et al., 2022)	78.60	65.63	71.53	73.89	5.32	82.61	30.32
LLaMA-Reviewer (Lu et al., 2023b)	60.99	83.50	70.49	-	5.70	82.27	-
CFT-reg(Kumar & Chimalakonda, 2024)	49.14	62.90	55.17	-	0.67	76.10	-
DISCOREV (Ben Sghaier & Sahaoui, 2024)	-	-	-	-	7.33	85.49	-
CoRAL (Sghaier et al., 2025a)	-	-	-	-	8.67	-	-
Toggle(Hossain et al., 2024)	-	-	-	-	-	-	25.59
Intention is All You Need*(Guo et al., 2025)	-	-	-	-	-	-	50.04
<i>Coder-R³</i>	79.68	71.60	75.43	76.68	5.61	86.97	45.47

343 **Input Features.** The inclusion of programming language information led to minor improvements
 344 across most metrics, whereas adding diff information yielded substantially greater gains. For ex-
 345 ample, accuracy improved by 0.12% with language metadata and by 4.82% with diff information.
 346 Similar trends were observed in other metrics, with the exception of the Repair task, where adding
 347 language metadata caused a very slight decrease in BLEU score (-0.07). These findings confirm
 348 that both types of contextual features contribute positively to model performance on this dataset.

349 Notably, under the joint training setup, incorporating diff inputs led to severe overfitting—precision
 350 reached 99.90%, while recall dropped to 13.80%. We hypothesize that this may be due to truncated
 351 diff segments in the CodeReviewer code refinement data, creating a discernible distribution gap
 352 between data types.

353 **Output Format.** We also experimented with using diff-style outputs under the best-performing
 354 training configuration, introducing line numbers of every code line into the input code and asking the
 355 model to generate a diff rather than the full repaired code. During evaluation, the generated diff was
 356 applied to the original code to recover the repaired version for assessment. However, this approach
 357 underperformed compared to directly generating the full code snippet, indicating that accurately
 358 producing diff-structured edits remains a difficult task for LLMs.

360 4.5 COMPARED TO OTHER METHODS

362 For the CodeReviewer dataset, we employed code snippets, programming language identifiers, and
 363 code diffs as input, with the repaired code snippet as the output. The model was trained using
 364 the hybrid training approach. The same input and output configuration was applied to the baseline
 365 models for a fair comparison. The evaluation result is shown in Tab. 2.

366 Our proposed *Coder-R³* model significantly outperformed all baseline pre-trained LLMs across
 367 all three tasks, demonstrating the effectiveness of our training methodology. It is worth noting
 368 that Qwen-2.5-Coder-7B, without any fine-tuning, achieved an accuracy below 50% in the binary
 369 classification task. This suggests a potential discrepancy between the labeling conventions in the
 370 CodeReviewer dataset and common understanding, leading to the pre-trained model’s performance
 371 being worse than random guessing in this specific context. Interestingly, we observed that the per-
 372 formance on the Review task decreased as the model size increased, with even GPT-5 delivering the
 373 lowest results. This counterintuitive finding may indicate potential limitations in the data quality.

374 In comparisons with prior work, our model achieved state-of-the-art results on the Recognize and
 375 Repair tasks, though a performance gap remains on the Review task. For the Recognize task, our
 376 model surpassed previous methods on all metrics except Recall, with a 2.8% higher accuracy than
 377 CodeReviewer. On the Review task, both DISCOREV and CoRAL leveraged the relationship be-
 378 tween review and repair tasks to achieve superior results. For the Repair task, our model attained

378
 379 **Table 3: Ablations on the internal dataset.** The optimal performance was achieved by the model
 380 configuration that used a hybrid training approach, with code diff as input context, and generated a
 381 full code snippet as output.

Language	Context			Task	Output	Recognize				Review	Repair	
	Diff	Related Code	Task			Precision	Recall	F1-Score	Accuracy		BLEU	EM
✗	✗	✗	Joint	Snippet	Snippet	98.69	23.15	37.51	70.06	31.04	56.60	11.58
✓	✗	✗	Joint	Snippet	Snippet	98.32	22.52	36.65	69.77	30.83	55.42	10.88
✓	✓	✗	Joint	Snippet	Snippet	98.71	23.60	38.09	70.22	31.63	56.79	11.68
✓	✓	✓	Joint	Snippet	Snippet	98.30	22.22	36.25	69.66	32.43	54.50	10.68
✗	✗	✗	Hybrid	Snippet	Snippet	75.77	46.74	57.81	73.52	32.84	77.53	18.16
✓	✗	✗	Hybrid	Snippet	Snippet	75.03	46.28	57.25	73.17	32.28	77.51	18.70
✗	✓	✗	Hybrid	Snippet	Snippet	77.28	47.96	59.19	74.33	33.06	78.01	18.23
✗	✗	✓	Hybrid	Snippet	Snippet	75.55	47.35	58.21	73.61	33.25	77.69	18.36

391
 392 the highest BLEU score, while slightly trailing behind the study by Guo et al. in EM. This minor
 393 discrepancy can be attributed to their use of an Intention Framework augmented with RAG and the
 394 inclusion of associated line numbers as input, which creates a different experimental setting and
 395 precludes a direct comparison.

5 CODE REVIEW IN PRACTICE

399 Beyond utilizing public datasets, we have concentrated on ensuring the method’s effectiveness and
 400 feasibility in real-world code review environments. To achieve this, we gathered internal data from a
 401 company (with over 1,000 software engineers), trained models based on this data, and assessed the
 402 method’s performance within the company’s actual code review system. The experimental settings
 403 employed are consistent with those detailed in Sec. 4.2.

5.1 INTERNAL DATASET CURATION

404 This data is sourced from real-world development processes within the company. Due to the com-
 405 pany’s rigorous programming guidelines and the fact that code contributors and reviewers tend to
 406 be more diligent and responsible compared to those in open-source projects, the data is of higher
 407 quality than that typically found in public datasets. The details of the dataset and the preprocessing
 408 procedure are described in Appendix A.2.

5.2 EVALUATION METRICS

414 The metrics of three code review tasks are consistent with the Codereviewer dataset, as described
 415 in Sec. 4.3. Additionally, we introduced real-world production metrics based on the company’s
 416 code review system to evaluate the model’s practicability: Online Accept Rate(OAR), Replay Pre-
 417 cision(RP) and Replay Recall(RR). OAR means the accept rate of suggestion made by the model,
 418 while RP and RR means the classification metrics of online data. More details in Appendix B.1.

5.3 EXPERIMENT RESULTS

421 **Ablation Study.** Similar to Sec. 4.4, we conduct ablations on the internal dataset, the results are
 422 shown in Tab. 3. On the internal dataset, Joint Training again underperformed compared to Hybrid
 423 Training. The Joint Training approach resulted in a model with high precision but low recall for the
 424 classification task, although the overall accuracy did not decrease significantly. This phenomenon
 425 may be attributed to the excessive number of tokens involved in the joint training process, which
 426 could have hindered effective optimization of the tokens specifically responsible for classification.

427 Regarding input features, the inclusion of programming language had almost no effect on the inter-
 428 nal dataset, suggesting that the longer code segments provided sufficient context for the model to
 429 infer the language. The addition of diff information still provide a measurable advantage. While the
 430 inclusion of related code leads to minor improvements, particularly in the review task, the substan-
 431 tial increase in token length made its inclusion cost-ineffective. If more concise and relevant code
 contexts could be extracted, it would likely yield greater benefits.

Table 4: **Evaluation on the internal dataset.** *Coder-R³* demonstrates strong efficacy on the internal dataset, achieving significant improvements compared to baseline models.

Methods	Recognize				Review	Repair		
	Precision	Recall	F1-Score	Accuracy	BLEU	BLEU	EM	
Qwen-2.5-Coder-7B (Hui et al., 2024)	51.56	16.39	24.88	61.57	5.11	75.54	14.48	
Qwen-2.5-Coder-14B (Hui et al., 2024)	54.94	22.14	31.56	62.73	5.86	75.83	13.95	
Qwen-2.5-Coder-32B (Hui et al., 2024)	42.18	82.29	55.77	49.34	6.45	75.79	14.74	
<i>Coder-R³</i>		77.28	47.96	59.19	74.33	33.06	78.01	18.23

Compared to Other Methods. On the internal dataset, we adopted code snippets and code diffs as input, with the complete repaired code as the output, and trained the model using a multi-stage approach. The same input-output configuration was applied to the baseline models for comparison.

As shown in Tab. 4, our model significantly outperformed the non-fine-tuned baselines across all metrics, achieving an 11.6% improvement in accuracy, a 26.61-point increase in BLEU for the Review task, and gains of 2.18 and 3.49 in BLEU and EM respectively for the Repair task.

The baseline models' initial performance on our dataset was notably higher on the Recognize and Review tasks compared to their results on the CodeReviewer dataset, suggesting superior label and annotation quality in our internal data. In contrast, the overall lower scores on the Repair task may be attributed to the longer code snippets and more extensive modifications required, indicating a potentially higher difficulty level than that of the CodeReviewer data.

However, we observed that the 32B model exhibited a distinct tendency to classify samples as “requiring review”, a behavior not seen in the two smaller models. This bias resulted in a high recall score but a corresponding decrease in overall accuracy. This phenomenon presents an interesting direction for further investigation.

5.4 PERFORMANCE ON REAL-WORLD DEPLOYMENT

We deployed a trained Qwen2.5-Coder-32B model into the company’s internal code review system. On a weekly basis, we collected data on the model’s performance in real-world usage. This weekly human feedback data was then incorporated into the training data, allowing us to iteratively update the model through rolling training iterations. The actual results are shown in Fig. 3. Model performance was evaluated weekly, with ‘Baseline’ representing the results of the untrained baseline model on the preceding week’s data. A consistent upward trend was observed across all three metrics as training progressed, demonstrating the effectiveness of our training methodology and data iteration strategy. The most significant improvement occurred after the initial fine-tuning round, with subsequent gains gradually diminishing over time before the metrics eventually approached a stable asymptote.

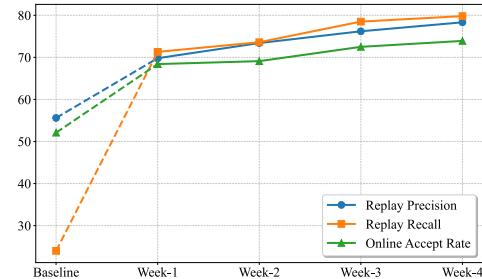


Figure 3: **Performance on real deployment.** We iteratively train and deploy *Coder-R*³ by collecting human feedback data. All metrics showed a consistent increase over time, validating the efficacy of our approach.

6 CONCLUSION

In this work, we present *Coder-R*³, a novel approach that enables large language models to simultaneously perform three core code review tasks: **Recognize**, **Review**, and **Repair**. By leveraging a data flywheel mechanism, we continuously collect internal corporate data to train our model, deploy it for practical use, and gather new feedback—forming a closed-loop iterative pipeline. Our method achieves state-of-the-art performance on the CodeReviewer benchmark and demonstrates strong effectiveness in real-world industrial scenarios, contributing significantly to the practical implementation of automated code review systems.

486 7 STATEMENTS
487488 7.1 ETHICS STATEMENT
489490 This research strictly adheres to academic ethical principles and is dedicated to promoting social
491 well-being and human progress. We are committed to conducting research responsibly, striving to
492 minimize potential negative consequences. Our work upholds the highest standards of scientific
493 excellence, ensuring methodological transparency and reproducible results.494 All data usage complies with ethical review approvals. The Codereviewer dataset is an open-sourced
495 dataset with CC BY 4.0 License. The internal data we collected was obtained with the company's
496 permission, and due to proprietary information, it will not be publicly available at this time. Whether
497 the data will be shared in the future is contingent on company decisions.498 Regarding the application of LLMs in research, we only utilize LLMs as tools for assisting and
499 polishing writing. The core method development in this research does not involve LLMs as any
500 important, original, or non-standard components.
501502 7.2 REPRODUCIBILITY STATEMENT
503504 We guarantee that our research is reproducible. Our data processing procedures are consistent with
505 the descriptions in Sec. 3.3 and Sec. 5.1. The data processing and training code will be open-sourced.
506 Our training process can be reproduced by downloading the open-source LLM Qwen-Coder-7B and
507 fine-tuning it using the code.
508509 REFERENCES
510511 Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Ale-
512 man, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical
513 report. *arXiv preprint arXiv:2303.08774*, 2023.514 Igli Begolli, Meltem Aksoy, and Daniel Neider. Fine-tuning multilingual language models for code
515 review: An empirical study on industrial c# projects. *arXiv preprint arXiv:2507.19271*, 2025.516 Oussama Ben Sghaier and Houari Sahraoui. Improving the learning of code review successive
517 tasks with cross-task knowledge distillation. *Proceedings of the ACM on Software Engineering*,
518 1(FSE):1086–1106, 2024.519 Amiangshu Bosu and Jeffrey C Carver. Impact of peer code review on peer impression formation:
520 A survey. In *2013 ACM/IEEE International Symposium on Empirical Software Engineering and*
521 *Measurement*, pp. 133–142. IEEE, 2013.522 Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal,
523 Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are
524 few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.525 Kaiwei Cai, Junsheng Zhou, Li Kong, Dandan Liang, and Xianzhuo Li. Automated comment gener-
526 ation based on the large language model. In *International Conference on Computer Science and*
527 *Education*, pp. 283–294. Springer, 2023.528 Genevieve Caumartin, Qiaolin Qin, Sharon Chatragadda, Janmitsinh Panjrolia, Heng Li, and
529 Diego Elias Costa. Exploring the potential of llama models in automated code refinement: A
530 replication study. In *2025 IEEE International Conference on Software Analysis, Evolution and*
531 *Reengineering (SANER)*, pp. 681–692. IEEE, 2025.532 Zhenlong Dai, Bingrui Chen, Zhuoluo Zhao, Xiu Tang, Sai Wu, Chang Yao, Zhipeng Gao, and
533 Jingyuan Chen. Less is more: Adaptive program repair with bug localization and preference
534 learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 128–
535 136, 2025.536 Qi Guo, Junming Cao, Xiaofei Xie, Shangqing Liu, Xiaohong Li, Bihuan Chen, and Xin Peng. Ex-
537 ploring the potential of chatgpt in automated code refinement: An empirical study. In *Proceedings*
538 *of the 46th IEEE/ACM International Conference on Software Engineering*, pp. 1–13, 2024.

540 Qi Guo, Xiaofei Xie, Shangqing Liu, Ming Hu, Xiaohong Li, and Lei Bu. Intention is all you need:
 541 Refining your code from your intention. In *2025 IEEE/ACM 47th International Conference on*
 542 *Software Engineering (ICSE)*, pp. 728–728. IEEE Computer Society, 2025.

543

544 Soneya Binta Hossain, Nan Jiang, Qiang Zhou, Xiaopeng Li, Wen-Hao Chiang, Yingjun Lyu, Hoan
 545 Nguyen, and Omer Tripp. A deep dive into large language models for automated bug localization
 546 and repair. *Proceedings of the ACM on Software Engineering*, 1(FSE):1471–1493, 2024.

547 Edward J Hu, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen,
 548 et al. Lora: Low-rank adaptation of large language models. In *International Conference on*
 549 *Learning Representations*, 2022.

550

551 Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang,
 552 Bowen Yu, Keming Lu, et al. Qwen2.5-coder technical report. *arXiv preprint arXiv:2409.12186*,
 553 2024.

554 Faria Huq, Masum Hasan, Md Mahim Anjum Haque, Sazan Mahbub, Anindya Iqbal, and Toufique
 555 Ahmed. Review4repair: Code review aided automatic program repairing. *Information and Soft-*
 556 *ware Technology*, 143:106765, 2022.

557

558 Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R
 559 Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth*
 560 *International Conference on Learning Representations*, 2024. URL <https://openreview.net/forum?id=VTF8yNQM66>.

561

562 Manav Nitin Kapadnis, Atharva Naik, and Carolyn Rose. Crscore++: Reinforcement learning with
 563 verifiable tool and ai feedback for code review. *arXiv preprint arXiv:2506.00296*, 2025.

564

565 Jahnavi Kumar and Sridhar Chimalakonda. Code review automation via multi-task federated llm—an
 566 empirical study. *arXiv preprint arXiv:2412.15676*, 2024.

567

568 Zhiyu Li, Shuai Lu, Daya Guo, Nan Duan, Shailesh Jannu, Grant Jenks, Deep Majumder, Jared
 569 Green, Alexey Svyatkovskiy, Shengyu Fu, et al. Automating code review activities by large-scale
 570 pre-training. In *Proceedings of the 30th ACM Joint European Software Engineering Conference*
 571 *and Symposium on the Foundations of Software Engineering*, pp. 1035–1047, 2022.

572

573 Cristina V Lopes, Vanessa I Klotzman, Iris Ma, and Iftekhar Ahmed. Commit messages in the age of
 574 large language models. *arXiv preprint arXiv:2401.17622*, 2024.

575

576 Jun Lu, Lei Yu, Xiaojia Li, Li Yang, and Chun Zuo. Llama-reviewer: Advancing code review
 577 automation with large language models through parameter-efficient fine-tuning. *2023 IEEE 34th*
 578 *International Symposium on Software Reliability Engineering (ISSRE)*, pp. 647–658, 2023a. URL
 579 <https://api.semanticscholar.org/CorpusID:261064667>.

580

581 Junyi Lu, Lei Yu, Xiaojia Li, Li Yang, and Chun Zuo. Llama-reviewer: Advancing code review
 582 automation with large language models through parameter-efficient fine-tuning. In *2023 IEEE*
 583 *34th International Symposium on Software Reliability Engineering (ISSRE)*, pp. 647–658. IEEE
 584 Computer Society, 2023b.

585

586 Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin
 587 Clement, Dawn Drain, Dixin Jiang, Duyu Tang, et al. Codexglue: A machine learning benchmark
 588 dataset for code understanding and generation. In *Thirty-fifth Conference on Neural Information*
 589 *Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021.

590

591 OpenAI. Gpt-5 system card. <https://openai.com/index/gpt-5-system-card>, 2025.

592

593 Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic
 594 evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association*
 595 *for Computational Linguistics*, pp. 311–318, 2002.

596

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language under-
 597 standing by generative pre-training. 2018.

594 Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language
 595 models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

596

597 Oussama Ben Sghaier, Rosalia Tufano, Gabriele Bavota, and Houari Sahraoui. Leveraging reward
 598 models for guiding code review comment generation. *arXiv preprint arXiv:2506.04464*, 2025a.

599 Oussama Ben Sghaier, Martin Weyssow, and Houari Sahraoui. Harnessing large language models
 600 for curated code reviews. In *2025 IEEE/ACM 22nd International Conference on Mining Software
 601 Repositories (MSR)*, pp. 187–198. IEEE, 2025b.

602

603 Dominik Sobania, Martin Briesch, Carol Hanna, and Justyna Petke. An analysis of the automatic
 604 bug fixing performance of chatgpt. In *2023 IEEE/ACM International Workshop on Automated
 605 Program Repair (APR)*, pp. 23–30. IEEE, 2023.

606

607 Qwen Team. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

608

609 Rosalia Tufano, Simone Masiero, Antonio Mastropaolo, Luca Pascarella, Denys Poshyvanyk, and
 610 Gabriele Bavota. Using pre-trained models to boost code review automation. In *2022 IEEE/ACM
 44th International Conference on Software Engineering (ICSE)*, pp. 2291–2302. IEEE, 2022.

611

612 Rosalia Tufano, Ozren Dabic, Antonio Mastropaolo, Matteo Ciniselli, and Gabriele Bavota. Code
 613 review automation: Strengths and weaknesses of the state of the art. *IEEE Transactions on
 614 Software Engineering*, 50(02):338–353, 2024.

615

616 Pengyu Xue, Linhao Wu, Zhongxing Yu, Zhi Jin, Zhen Yang, Xinyi Li, Zhenyu Yang, and Yue
 617 Tan. Automated commit message generation with large language models: An empirical study
 618 and beyond. *IEEE Transactions on Software Engineering*, 2024.

619

620 Xin Yin, Chao Ni, Shaohua Wang, Zhenhao Li, Limin Zeng, and Xiaohu Yang. Thinkrepair: Self-
 621 directed automated program repair. In *Proceedings of the 33rd ACM SIGSOFT International
 622 Symposium on Software Testing and Analysis*, pp. 1274–1286, 2024.

623

624 Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyuan Luo, Zhangchi Feng, and
 625 Yongqiang Ma. Llamafactory: Unified efficient fine-tuning of 100+ language models. In *Pro-
 626 ceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume
 627 3: System Demonstrations)*, Bangkok, Thailand, 2024. Association for Computational Linguis-
 628 tics. URL <http://arxiv.org/abs/2403.13372>.

629

630

631

632

633

634

635

636

637

638

639

640

641

642

643

644

645

646

647

648
649

A APPENDIX: DATASET DETAILS

650
651

A.1 THE CODEREVIEWER DATASET

652
653
654
655
656
657
658

Codereviewer dataset is collected from GitHub repositories in nine programming languages, including C, C++, C#, Go, Java, JavaScript, PHP, Python, and Ruby. It is composed of three subsets: code change quality estimation, review generation, and code refinement. The data formats for code change quality estimation and review generation are identical, and their datasets share a common subset of data. In contrast, the code repair set utilizes a separate and distinct dataset. The training sets for these three datasets contain 266k, 118k, and 150k instances respectively, while their corresponding test sets consist of 31k, 10k, and 13k instances.

659
660
661
662
663
664
665

Due to inconsistent data formats in the CodeReviewer dataset, we performed a series of preprocessing steps. Firstly, we excluded incomplete entries with missing values in certain fields. Secondly, using information such as original files, modified files, and diff content provided in the dataset, we reconstructed code snippets and standardized them into a unified format. What’s more, the training sets of the code change quality estimation and review generation datasets did not include programming language labels. We used the strategy described in Sec. 3.2 to label this data, which achieved an accuracy of over 97% on the test set.

666
667
668
669

For the joint training approach, each input data sample must possess labels for all three tasks to facilitate model training. We constructed a comprehensive dataset by combining samples from the code change quality estimation dataset that were flagged as ‘yes’ with all available samples from the code refinement dataset.

670
671

A.2 THE INTERNAL DATASET

672
673
674
675
676

We utilize internal data collected according to Sec. 3.3. In the company code review system, the review model will initially provide code review labels (category, explanation, and repaired code), and engineers can accept or reject the model’s suggestions and make modifications. We collect this human feedback data as high-quality labels for training and evaluating the model.

677
678
679
680
681
682
683
684
685
686
687
688

In the experiment, we collected human-annotated data over an eight-month period, focusing primarily on mainstream programming languages such as Python, Go, and C++. After preprocessing the data into a standardized format, we obtained 18,777 instances. To mitigate the risk of data overlap between the train and test sets, we used the data from the first six months as the train set and the data from last two months as the test set. The train set and test set were divided in a ratio of approximately 4.6 : 1. The distribution of the internal dataset by programming language is shown in Fig. 4.

689
690
691
692
693
694
695
696

Furthermore, the code segments in our internal dataset have an average length of 50.35 lines, which is notably longer than the average of 11.83 lines in the CodeReviewer dataset. Since longer code segments are more common in practical development, our dataset possesses greater practical relevance.

697
698
699
700

B METRIC DETAILS

701

B.1 PRODUCTION METRICS

To evaluate the performance in real-world scenarios, we deploy our model to the code review system of the company. By tracking whether human reviewers accept or reject the model’s suggestions,

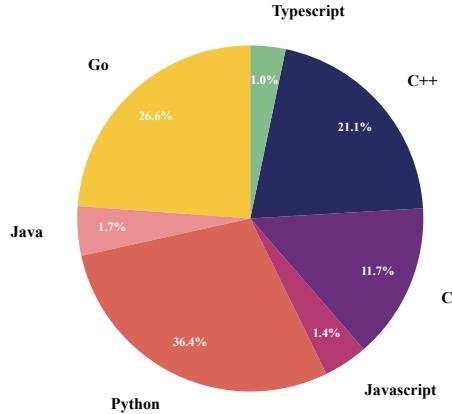


Figure 4: **Programming language distribution of the internal dataset.**

702 these metrics offer a more realistic reflection of its practical performance, particularly for open-
 703 ended generation tasks such as comment generation and code repair.
 704

705 We propose three production metrics: Online Accept Rate(OAR), Replay Precision(RP) and Replay
 706 Recall(RR). The OAR metric calculates the proportion of the model’s review suggestions that are
 707 accepted by human reviewers after deployment. Let N_{Accept} be the amount of accepted review
 708 instances and N_{Reject} be the amount of rejected review instances during the same period of time,
 709 OAR can be calculated as follows:

$$710 \quad 711 \quad 712 \quad OAR = \frac{N_{Accept}}{N_{Accept} + N_{Reject}} \quad (4)$$

713 For RP and RR metrics, we collected human-accepted or rejected data over a recent period after the
 714 model was deployed, which, after filtering and preprocessing, was constructed into a new test set.
 715 The model was then used to predict labels on this set and calculate precision and recall rates. It’s an
 716 offline method to quickly evaluate the model’s online effectiveness.
 717
 718
 719
 720
 721
 722
 723
 724
 725
 726
 727
 728
 729
 730
 731
 732
 733
 734
 735
 736
 737
 738
 739
 740
 741
 742
 743
 744
 745
 746
 747
 748
 749
 750
 751
 752
 753
 754
 755