Optimizing Case-Based Reasoning System for Functional Test Script Generation with Large Language Models

Siyuan Guo*
School of Artificial Intelligence,
International Center of Future Science
Jilin University
Changchun, China
guosyjlu@gmail.com

Huiwu Liu
Huawei Technologies Ltd.
Nanjing, China
liuhuiwu@huawei.com

Xiaolong Chen
Huawei Technologies Ltd.
Nanjing, China
chenxiaolong42@huawei.com

Yuming Xie
Huawei Technologies Ltd.
Nanjing, China
yuming.xie@huawei.com

Liang Zhang
Huawei Technologies Ltd.
Nanjing, China
zhangliang1@huawei.com

Tao Han
Huawei Technologies Ltd.
Nanjing, China
hantao@huawei.com

Hechang Chen†
School of Artificial Intelligence
Jilin University
Changchun, China
chenhc@jlu.edu.cn

Yi Chang†
School of Artificial Intelligence,
International Center of Future Science
Jilin University
Changchun, China
yichang@jlu.edu.cn

Jun Wang†
University College London
London, UK
jun.wang@cs.ucl.ac.uk

Abstract

In this work, we explore the potential of large language models (LLMs) for generating functional test scripts, which necessitates understanding the dynamically evolving code structure of the target software. To achieve this, we propose a case-based reasoning (CBR) system utilizing a 4R cycle (i.e., retrieve, reuse, revise, and retain), which maintains and leverages a case bank of test intent descriptions and corresponding test scripts to facilitate LLMs for test script generation. To improve user experience further, we introduce Re4, an optimization method for the CBR system, comprising reranking-based retrieval finetuning and reinforced reuse finetuning. Specifically, we first identify positive examples with high semantic and script similarity, providing reliable pseudo-labels for finetuning the retriever model without costly labeling. Then, we apply supervised finetuning, followed by a reinforcement learning finetuning stage, to align LLMs with our production scenarios, ensuring the faithful reuse of retrieved cases. Extensive experimental results on two product development units from Huawei Datacom demonstrate the superiority of the proposed CBR+Re4. Notably, we also show that the proposed Re4 method can help alleviate the repetitive generation issues with LLMs.

*This work was done during the internship at Huawei.
†Corresponding Authors.

KDD '25, August 3–7, 2025, Toronto, ON, Canada.

ACM ISBN 979-8-4007-1454-2/25/08
https://doi.org/10.1145/3711896.3737254

CCS Concepts

• **Computing methodologies** → **Natural language generation**.

Keywords

Case-Based Reasoning; Large Language Model; Test Script Generation; Functional Testing; Reinforcement Learning

ACM Reference Format:
Siyuan Guo, Huiwu Liu, Xiaolong Chen, Yuming Xie, Liang Zhang, Tao Han, Hechang Chen, Yi Chang, and Jun Wang. 2025. Optimizing Case-Based Reasoning System for Functional Test Script Generation with Large Language Models. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2 (KDD '25), August 3–7, 2025, Toronto, ON, Canada.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3711896.3737254

1 Introduction

Software testing is a critical phase in the software development lifecycle, ensuring the quality and reliability of software products. Functional testing, a key component of this process, verifies that the software's features and operations align with the specified requirements, thereby meeting user expectations. In Huawei Datacom, writing test scripts constitutes approximately 40% of the workload in functional testing. Even seasoned test engineers can produce only one or two test scripts per day, highlighting an urgent need to improve the efficiency of script writing.

Large language models (LLMs) have demonstrated remarkable success in complex code generation tasks, such as bug fixing [16, 32], automated data science [9, 10], and code translation [21, 45]. Yet, their application in software testing has primarily focused on unit test generation [3, 35], leaving the more complex domain of functional testing underexplored [33]. In this work, we investigate the potential of LLMs to assist test engineers in generating functional

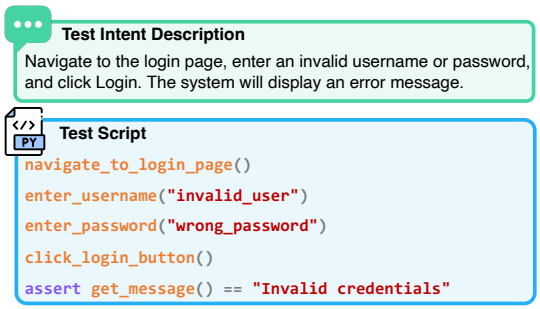

Figure 1: An example of the functional test script.

test scripts. Different from unit test generation, functional test scripts require invoking existing functions within the target software to build workflows based on the test intent description, as illustrated in Figure 1. This requirement goes beyond the static knowledge embedded in LLMs. Furthermore, as the software evolves with each version, the knowledge of the code structure should be also continuously updated. Unfortunately, such dynamic knowledge cannot be directly integrated into the context of LLMs due to the lengthy code structure of the target software, while performing continual finetuning is computationally expensive and inefficient.

To address the aforementioned challenge, we adopt a classic AI problem-solving paradigm, case-based reasoning (CBR) [1, 10, 18, 36], which maintains a structured case bank of past test intent descriptions and corresponding test scripts to enhance the capabilities of LLMs in test script generation. As shown in Figure 2(a), we employ the classic 4R cycle [1] to construct the CBR system, which consists of four steps: (1) **Retrieve** similar cases from the case bank based on the given test intent description; (2) **Reuse** the retrieved cases to generate the test script using LLMs; (3) **Revise** the generated test scripts by human test engineers; (4) **Retain** the revised test script and corresponding test intent description into the case bank for future use. Benefiting from the CBR system, LLMs can utilize the mapping between test intent descriptions and corresponding function calls from the retrieved cases to generate test scripts. Additionally, the CBR system offers a flexible learning mechanism by continuously retaining human-validated cases in the case bank during the deployment stage.

Thanks to the zero-shot capabilities of pretrained retriever models and LLMs, the CBR system has shown initial effectiveness in our production scenario. To further enhance the user experience, we aim to optimize the Retrieve and Reuse steps within the CBR system, as the other steps do not rely on machine learning models. For the Retrieve step, finetuning the retriever model can be challenging due to the lack of ground-truth labeled data. While previous works propose to utilize the feedback from LLMs to generate pseudo-labels [6, 29, 30], their applicability in our production scenario is limited by the high computational and time costs associated with intensive LLM interactions. In terms of the Reuse step, supervised finetuning (SFT) appears promising for enabling LLMs to generate test scripts by reusing the retrieved cases. However, ground-truth test scripts typically include function calls absent from the retrieved cases. As a result, the SFT objective may push LLMs to generate unseen functions, introducing noise during alignment and thereby exacerbating hallucination issues during inference, which would undermine the user experience.

To this end, we propose **Re4**, an optimization method for the CBR system with **re**ranking-based **re**trieval finetuning and **re**inforced **re**use finetuning. Given that test script generation can be viewed as a text-to-code translation task, similar test scripts are expected to share similar test intent descriptions. Building on this insight, we propose a reranking-based retrieval finetuning method, which identifies positive examples with both high semantic and script similarity. This can provide reliable pseudo-labels for contrastive learning, enabling the finetuning of the retriever model without the need for costly human labeling or feedback from LLMs. Moreover, we propose a reinforced reuse finetuning method to align the LLM with our production scenario, ensuring faithful reuse of the retrieved cases for test script generation. We first perform SFT as a warm-up stage to establish initial alignment. Following this, we introduce a reinforcement learning finetuning (RLFT) stage with a critic-free online reinforcement learning algorithm, REINFORCE [39]. In this stage, we leverage the similarity between the generated script and the ground-truth test script as the golden reward to further refine the alignment, which eliminates the undesired behavior patterns introduced during the SFT stage. We present extensive experimental results to demonstrate the effectiveness of the proposed Re4 optimization method. Empirically, CBR+Re4 outperforms other baselines on datasets collected from two product development units (PDUs) at Huawei Datacom. Notably, CBR+Re4 can also effectively alleviate the repetitive generation issues encountered with the previously deployed CBR+SFT method.

We summarize the contributions of our work as follows:

- To the best of our knowledge, this is the first attempt to utilize LLMs to assist test engineers in functional testing.
- We propose a CBR system to enhance the capabilities of LLMs for functional test script generation.
- We introduce the Re4 method to finetune both the retriever model and the LLM within the CBR system, ensuring better alignment with our production scenario.
- We conduct extensive experiments on real-world datasets from Huawei Datacom to demonstrate the superiority of the proposed Re4 method. Meanwhile, our findings show that CBR+Re4 alleviates the repetitive generation issues of LLMs, further improving the user experience.

2 Preliminaries

To clearly explain the problem under investigation, we first introduce the overall case-based reasoning framework for functional test script generation with large language models, and then present the business metric in our production scenario.

2.1 The Case-Based Reasoning System

In this work, we focus on investigation of functional test script generation with large language models (LLM). Given a test intent description q, LLMs are tasked with generating the corresponding test script y, which can be framed as a text-to-code generation problem. However, different from general text-to-code generation problems and other test script generation tasks (e.g., unit test generation), functional test script generation requires LLMs to understand

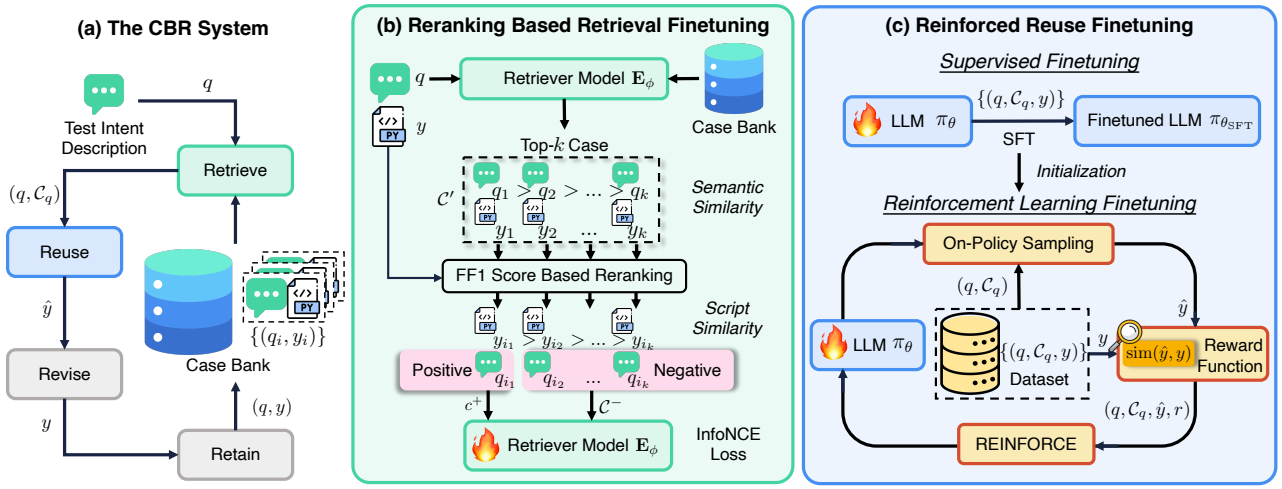

Figure 2: The overall paradigm of the proposed CBR+Re4. (a) The CBR system, which follows a 4R cycle of Retrieve, Reuse, Revise and Retain; (b) Reranking-based retrieval finetuning, which identifies cases with both high semantic and script similarity as positive examples and applies contrastive learning for finetuning; (c) Reinforced reuse finetuning, which consists of supervised finetuning and function f1 score based reinforcement learning finetuning for LLMs.

detailed code structures of the target software. Given that the size of industrial software typically exceeds the token capacity of LLMs, this task still remains underexplored [33].

In our production scenario, there exist a large collection of functional test scripts written by human test engineers for each software, containing rich implicit knowledge that maps test intent description to test scripts. To fully harness this wealth of knowledge, we adopt a classic problem-solving paradigm, case-based reasoning (CBR) [1, 18, 36], to enhance the capabilities of LLMs for functional test script generation. Specifically, we follow the classic 4R CBR cycle [1], which contains four steps: **(1) Retrieve** similar cases from the case bank; **(2) Reuse** the retrieved cases to solve the current task; **(3) Revise** the solution to guarantee the correctness for the current task; **(4) Retain** the task and the solution into the case bank. The overall workflow of CBR system is demonstrated in Figure 2(a).

We now provide a detailed description of the CBR system for functional test script generation with LLMs. Given a test intent description q, the CBR system first retrieves top-M similar cases from the case bank $C = \{c_i\}_{i=1}^{N}$ with each case $c_i = (q_i, y_i)$. To achieve this, we utilize a pretrained embedding model $\mathbf{E}_\phi(\cdot)$ as the retriever model to calculate the semantic similarity between two test intent descriptions q and q' as:

$$\text{sim}_\phi(q, q') = \cos\langle \mathbf{E}_\phi(q), \mathbf{E}_\phi(q') \rangle. \tag{1}$$

Next, the LLM $\pi_\theta(\cdot)$ reuses the retrieved cases $C_q = \{c_i\}_{i=1}^{M}$ to generate the test script, i.e., $\hat{y} \sim \pi_\theta(\cdot|q, C_q)$. Human test engineers then verify the correctness of the generated test scripts and revise them as necessary. Finally, the test intent description q and the corresponding revised test script y are retained into the case bank, i.e., $C \leftarrow C \cup \{(q, y)\}$. We summarize the workflow of this CBR system in Algorithm 1 in Appendix A.2.

Note that the aforementioned Retrieve-Reuse process in the CBR system is similar to the retrieval-augmented generation (RAG) techniques [7, 25, 34]. The key difference is that RAG retrieves relevant document chunks to provide detailed context for LLM generation, whereas CBR retrieves similar cases to enable analogical reasoning for task-solving. Furthermore, the CBR system benefits from the Retain step to achieve a flexible learning mechanism during deployment, eliminating the need for resource-intensive finetuning.

2.2 Business Metric

In the context of our production scenario, the goal is to assist human test engineers in improving their efficiency in writing functional test scripts, rather than fully automating the process in current stage. This decision is driven by two main considerations: (1) the provided test intent descriptions are often not detailed enough to enable complete test script generation by LLMs; (2) the retrieved cases may not comprehensively cover the software structures required for functional test script generation. Therefore, the acceptance rate, the percentage of generated test scripts accepted by human test engineers, can be regarded as the online business metric. However, this metric is significantly biased in practice, as user preferences for accepting generated test scripts vary widely. Worse still, the limited number of users within each PDU prevents the mitigation of bias through the law of large numbers.

To solve this issue, we design offline business metrics by evaluating the script similarity between the generated test script $\hat{y}$ and the ground-truth test script y. This task is particularly challenging due to the vast space of functionally equivalent code. One notable characteristic of our production scenario is that functional test scripts focus on invoking functions to structure the desired workflow. Thus, the accuracy of functions within the generated test script is crucial for achieving high user satisfaction. Based on this observation, we propose to measure the script similarity with three offline business metrics: function precision, function recall

and function f1 score, defined as follows:

$$\text{FPrecision}(\hat{y}, y) = \frac{|\text{Func}(\hat{y}) \cap \text{Func}(y)|}{|\text{Func}(\hat{y})|}, \tag{2}$$

$$\text{FRecall}(\hat{y}, y) = \frac{|\text{Func}(\hat{y}) \cap \text{Func}(y)|}{|\text{Func}(y)|}, \tag{3}$$

$$\text{FF1}(\hat{y}, y) = \frac{2 \cdot \text{FPrecision}(\hat{y}, y) \cdot \text{FRecall}(\hat{y}, y)}{\text{FPrecision}(\hat{y}, y) + \text{FRecall}(\hat{y}, y)}, \tag{4}$$

where $\text{Func}(\cdot)$ returns the set of functions invoked in the test script. Among them, function precision measures the accuracy of the functions called in the generated test script, while function recall evaluates how well the generated test script covers the functions in the ground-truth test script. The function f1 score balances both precision and recall, providing a comprehensive measure of overall performance. Hence, the function f1 score is the most crucial business metric in our production scenario.

In addition to these three metrics, we also incorporate code similarity [8], measured using a normalized Levenshtein distance [19] score, as a complementary metric. We defer implementation details of the aforementioned offline business metrics to Appendix A.1.

3 Methodology

In this section, we aim to optimize the aforementioned CBR system for better alignment with our production scenario. As Revise and Retain steps do not involve machine learning models, we focus on optimizing the retriever model $\mathbf{E}_\phi(\cdot)$ in the Retrieve step, and the LLM $\pi_\theta(\cdot)$ in the Reuse step. We introduce Re4, an optimization method for our CBR system, comprising two key components: (1) a reranking-based retrieval finetuning method, which optimizes the retriever model to retrieve cases more similar to the query, and (2) a reinforced reuse finetuning method, which enables LLMs to faithfully reuse the retrieved cases for solving new tasks. The overall framework of Re4 is illustrated in Figure 2.

In this work, we adopt the offline training strategy to avoid high computational costs required in real-time online training. Let $\mathcal{D} = \{(q_i, y_i)\}_{i=1}^N$ be the training set with N samples, where q_i is the i-th test intent description, and y_i is the corresponding test script. For each sample $(q_i, y_i) \in \mathcal{D}$, we regard the other samples in the training set as the case bank, i.e., $C = \mathcal{D} \setminus \{(q_i, y_i)\}$.

3.1 Reranking-based Retrieval Finetuning

In the CBR system, the Retrieve step is crucial for retrieving similar cases from the case bank to support the subsequent Reuse step. While pretrained embedding models offer strong zero-shot retrieval performance, we aim to further finetune the models to better align with the specific corpus of our production scenario. Therefore, we utilize contrastive learning, a widely adopted approach for representation learning, to finetune the pretrained embedding model. Formally, given a test intent description q with an associated positive example $c^+ = (q^+, y^+)$ and a pool of negative examples C^-, the well-known InfoNCE loss [15, 23] is defined as

$$\mathcal{L}(\phi) = -\frac{\exp(\text{sim}_\phi(q, q^+)/\tau)}{\exp(\text{sim}_\phi(q, q^+)/\tau) + \sum_{q^- \in C^-} \exp(\text{sim}_\phi(q, q^-)/\tau)}, \tag{5}$$

where τ denotes the hyper-parameter for temperature. Different from general retrieval problems [17, 22, 41, 47], positive examples are not explicitly available in our setting, making the key challenge the effective mining of positive and negative examples for a given query. One promising approach is to leverage LLMs to provide pseudo-labels of positive and negative examples [6, 29, 30]. However, this requires multiple rounds of sampling from LLMs, resulting in significant computational and time costs.

To address this challenge, we aim to identify the positive example based on the nature of our production scenario. As the basic assumption of CBR [4, 14] — that similar problems have similar solutions — is invertible in text-to-code generation due to its translation nature, we can expect that similar test scripts should correspond to similar test intent descriptions. Therefore, we identify positive examples as those with both high semantic similarity and script similarity. Since functional testing emphasizes invoking functions to construct workflows, the script similarity can be effectively measured using the function f1 score, as defined in Eq. (4). Based on the aforementioned findings, we propose a reranking-based retrieval finetuning method, as shown in Figure 2(b).

Given a case (q, y), we first retrieve top-k cases from the case bank C based on the semantic similarity defined in Eq. (1), i.e.,

$$C' = \underset{(q', y') \in C}{\arg \text{top-}k}\ \text{sim}_\phi(q, q'). \tag{6}$$

Then, we rerank the retrieved cases C' based on the script similarity with function f1 score. The positive case c^+ is labeled as the case with most similar test script, and the remaining cases form the pool of negative cases C^-, as follows:

$$c^+ = \underset{(q', y') \in C'}{\arg\max}\ \text{FF1}(y', y), \tag{7}$$

$$C^- = C' \setminus \{c^+\}. \tag{8}$$

As such, we can follow the InfoNCE loss defined in Eq. (5) to finetune the pretrained embedding model. Following previous works [17, 47, 48], we also include in-batch negative examples for the InfoNCE loss, which has been shown to be an effective trick that boosts the number of training examples. We summarize the reranking-based retrieval finetuning method in Algorithm 2 in Appendix A.2.

3.2 Reinforced Reuse Finetuning

In the CBR system, the Reuse step focuses on adapting solutions of past similar cases to the current task. While modern LLMs demonstrate strong instruction-following capabilities, they may still suffer from misalignment issues [31, 37, 44] when confronted with unseen tasks. Therefore, further finetuning of LLMs is necessary to ensure alignment with the desired behavior in our production scenarios. To this end, we propose a reinforced reuse finetuning method that incorporates both supervised finetuning (SFT) and reinforcement learning finetuning (RLFT), as shown in Figure 2(c).

3.2.1 Supervised Finetuning. Given a training sample $(q, y) \in \mathcal{D}$ and its corresponding retrieved cases C_q, we can perform standard SFT by maximizing the log probability of each token in y. The loss function of SFT can be formulated as

$$\mathcal{L}_{\text{SFT}}(\theta) = -\log \pi_\theta(y|q, C_q). \tag{9}$$

Then, we can derive a finetuned LLM with parameters θ_{SFT}, which is also the starting point for RLFT.

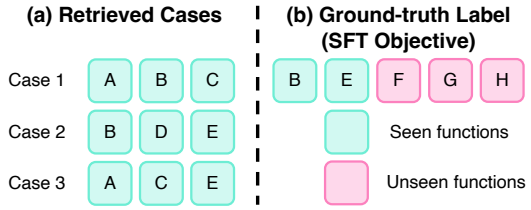

Figure 3: A motivating example for the limitation of SFT. SFT objective may contain unseen functions beyond the retrieved cases, resulting in noisy alignment.

3.2.2 Reinforcement Learning Finetuning. While SFT is efficient for alignment, it can be problematic in our production scenario, which requires LLMs precisely invoking functions contained in the retrieved cases to structure the testing workflow. Now, we present a motivating example. As illustrated in Figure 3, the retrieved cases include five functions (A, B, C, D, E), whereas the ground-truth label involves five different functions (B, E, F, G, H), three of which (F, G, H) are absent from the retrieved cases. Since the SFT objective enforces fitting the ground-truth label, it compels LLMs to generate unseen functions, introducing noise into alignment and exacerbating hallucination issues during inference. To address this issue, we introduce an online RLFT stage to further refine the alignment and mitigate these negative effects. Specifically, given a sample $(q, y) \in \mathcal{D}$ and its corresponding retrieved cases $C_q = \{(q_i, y_i)\}_{i=1}^{M}$, the optimization objective of RLFT can be formulated as:

$$\max_{\theta} \mathbb{E}_{\hat{y}\sim\pi_\theta(\cdot|q,C_q)}[r(\hat{y})], \tag{10}$$

where $r(\cdot)$ is the reward function that quantifies the quality of the generated script. Here, we utilize the script similarity between the generated script $\hat{y}$ and ground-truth label y as the golden reward, which can be effectively measured by the function f1 score defined in Eq. (4). Thus, we can reformulate the optimization objective as:

$$\max_{\theta} \mathbb{E}_{\hat{y}\sim\pi_\theta(\cdot|q,C_q)}[\mathrm{FF1}(\hat{y}, y) - \beta \cdot \mathbb{D}_{\mathrm{KL}}(\pi_\theta || \pi_{\theta_{\mathrm{SFT}}})], \tag{11}$$

where we follow previous works [2, 20, 28, 46] to incorporate a KL divergence penalty term to avoid too much deviation from the reference SFT policy model $\pi_{\theta_{\mathrm{SFT}}}$, and β denotes the coefficient for the KL divergence penalty. As such, we can encourage faithful reuse that invokes the correct functions, while penalizing the inclusion of incorrect functions resulting from the hallucination issue. Meanwhile, this objective does not push LLMs to fit unseen functions as SFT, thereby avoiding the introduction of new noise during RLFT.

To solve the problem defined in Eq. (11), proximal policy optimization (PPO) [27] is the most widely adopted algorithm; however, it is overly complicated for optimization and hyper-parameter tuning, especially for LLMs. To avoid unnecessary complicated designs, many previous works, such as RLOO [2], Remax [20] and GRPO [28], devote to the simplification of PPO algorithm. However, they typically require two or more times of on-policy sampling for variance reduction in the optimization process, leading to additional computational and time costs and limiting the application in our production scenario. Different from these algorithms designed for reinforcement learning from human feedback (RLHF) for open-ended text generation, our RLFT setting requires LLMs to focus on the reuse of the retrieved cases for test script generation, significantly narrowing the action space. As such, the randomness from the sampling of LLMs is far less compared to open-ended text generation, thereby weakening the high variance issue of the stochastic gradient. This enables us to omit all the variance reduction techniques and back to the most basic on-policy RL algorithm REINFORCE [39] for RLFT, which can be formulated as:

$$\mathcal{L}_{\mathrm{REINFORCE}}(\theta) = \mathbb{E}_{\hat{y}\sim\pi_\theta(\cdot|q,C_q)}[-r(\hat{y}) \cdot \log \pi_\theta(\hat{y}|q, C_q))], \tag{12}$$

where $r(\hat{y})$ denotes the reward function containing the function f1 score and the KL divergence penalty term as in Eq. (11). We provide a more detailed discussion on why REINFORCE works in our RLFT setting in Appendix B. We summarize the reinforced reuse finetuning method in Algorithm 3 in Appendix A.2.

4 Experiments

4.1 Experimental Setups

4.1.1 Dataset. We collect datasets from two product development units (PDU) in Huawei Datacom, referred to as Data Communication Network (DCN) and Software Platform (SP). The DCN dataset consists of 30,887 samples, while the SP dataset contains 58,429 samples. The collected datasets include noisy samples, such as those with poor coding practices and ambiguous or lengthy descriptions. To minimize human effort, we opt not to pre-process them further. Instead, we curate a clean and representative testing dataset to ensure reliable evaluation, comprising 366 samples for SP and 689 samples for DCN. We use consistent train-validation-test splits across all the methods.

4.1.2 Experiment Setting. The CBR system in this paper involves two backbone models, a pretrained embedding model for the Retrieve step and a LLM for the Reuse step. We utilize bge-m3 [5] as the embedding model, and an internal LLM with approximately 7B parameters as the base model across all the methods. Due to computational requirements, we utilize LoRA [13] for finetuning, enabling training to be conducted on a single Huawei Ascend 910B NPU with 64GB of memory.

For the CBR system, we retrieve $M = 3$ cases from the case bank for both training and inference stage. For the optimization of the Retrieve step, we finetune the embedding model for five epochs, setting the temperature τ to 1.0, learning rate to 1e-6, and batch size to 64 for both datasets. In terms of the optimization for the Reuse step, we perform SFT for one epoch with the batch size of 32. As for RLFT, we finetune the LLM with one epoch and batch size of 64, using a KL divergence coefficient β of 0.1. For both SFT and RLFT, we set the learning rate as 3e-5 for DCN and 1e-5 for SP. Additionally, we apply a cosine scheduler with a 3% warmup for the learning rate.

During training, we set the LLM's sampling temperature to 0.9 to enhance the diversity of the generated test scripts. During inference, we adopt greedy decoding to exclude randomness.

4.1.3 Evaluation Metric. During evaluation, we regard the training set as the case bank. We adopt four evaluation metrics: code similarity (CS), function precision (FP), function recall (FR), and function F1 score (FF1), as detailed in Section 2.2. Among these, FF1 score is the most critical metric for evaluation.

Table 1: Performance comparison of different methods across two datasets. The best results are highlighted in bold. We also report the relative improvement of CBR+Re4 over the best baseline.

	DCN				SP			
	CS (↑)	FP (↑)	FR (↑)	FF1 (↑)	CS (↑)	FP (↑)	FR (↑)	FF1 (↑)
Naïve SFT	47.62	57.28	50.40	49.46	43.72	42.35	39.50	38.71
Naïve CBR	53.63	67.11	67.93	64.01	56.47	55.97	57.09	54.60
CBR	54.80	66.47	70.13	64.67	54.49	56.52	59.78	55.83
CBR+SFT	63.38	74.09	68.89	67.87	61.10	61.31	58.27	57.98
CBR+Re4 (Ours)	**64.60**	**75.50**	**71.33**	**70.11**	**62.38**	**64.89**	**62.43**	**61.92**
Improvement	**+1.92%**	**+1.90%**	**+3.54%**	**+3.30%**	**+2.09%**	**+5.84%**	**+7.14%**	**+6.80%**

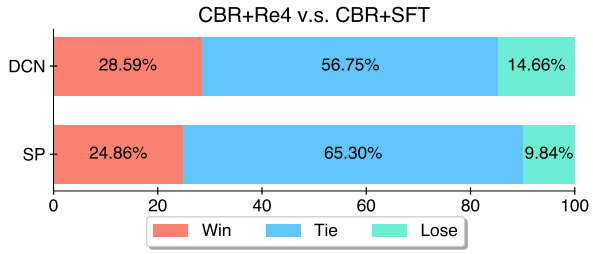

Figure 4: Comparison between CBR+Re4 and CBR+SFT. The win, tie and lose rates are evaluated by humans.

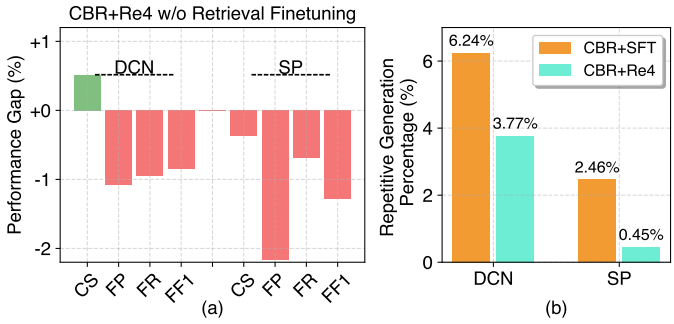

Figure 5: (a) Performance gap in an ablation study of CBR+Re4 w/o retrieval finetuning. (b) Repetitive generation percentage of different methods.

4.1.4 Baselines. To the best of our knowledge, this is the first paper that discusses the optimization of the full workflow in the CBR system with LLMs. Thus, there are not any other previous works suitable for comparison. To contextualize the performance of the proposed method, we carefully design four baselines:

- **Naïve SFT**: It directly performs supervised finetuning for test script generation without applying CBR techniques.
- **Naïve CBR**: It implements the simplest form of CBR by retrieving a single case from the case bank as the generated test script. Despite its simplicity, this approach is a strong baseline in our early attempts.
- **CBR**: It retrieves three similar cases from the case bank and uses them as context to prompt the LLM for generating test scripts. It does not perform any finetuning for both retriever model and LLM.
- **CBR+SFT**: Built on top of CBR, it further applies SFT for LLMs with three retrieved cases in the context, as described in Section 3.2.1. Note that it does not perform finetuning for the retriever model.

4.2 Main Results

4.2.1 Overall Comparison. We present the experimental results of the offline evaluation metrics for both datasets in Table 1. Note that we follow most LLM works to report the result of a single run due to the overly computational costs, which might be one limitation of this work. We can observe that Naïve SFT performs the worst among all methods, even underperforming the simplest baseline, Naïve CBR. This is expected, as functional test script generation requires the LLM to invoke functions beyond its knowledge, which cannot be directly injected by SFT. Consequently, this leads to significant hallucination issues during inference. In contrast, CBR outperforms Naïve CBR, thanks to the strong foundational capabilities of LLMs. These capabilities are further enhanced by incorporating SFT, making CBR+SFT the strongest baseline. Among all the methods, the proposed CBR+Re4 achieves the highest FF1 score for both datasets, showing improvement of 3.30% and 6.80% over the best baseline, CBR+SFT, for DCN and SP, respectively. This validates the effectiveness of both the reranking based retrieval finetuning method and the reinforced reuse finetuning method.

4.2.2 Human Evaluation. To comprehensively evaluate the quality of the generated test scripts, we perform pairwise human evaluations comparing the proposed CBR+Re4 with the best baseline, CBR+SFT, on two datasets. The win, tie, and loss rates are reported in Figure 4. The results show that CBR+Re4 achieves a higher win rate on both datasets compared to CBR+SFT, further validating the superiority of the proposed method.

4.3 In-Depth Analyses for Re4

In this subsection, we present in-depth analyses for the proposed finetuning methods for both retriever model and LLM.

4.3.1 Analyses for Reranking-based Retrieval Finetuning. As the key challenge for finetuning retriever models is lack of ground-truth labels, most advanced finetuning methods cannot be adapted to our setting. Thus, we analyze the proposed reranking-based retrieval finetuning method via an ablation study. The performance

Table 2: In-depth analyses of the proposed reinforced reuse finetuning method across two datasets. We highlight those results better than CBR+Re4 with .

	#On-policy samples	DCN				SP			
		CS (↑)	FP (↑)	FR (↑)	FF1 (↑)	CS (↑)	FP (↑)	FR (↑)	FF1 (↑)
CBR+Re4 (Ours)	1	64.60	75.50	71.33	70.11	62.38	64.89	62.43	61.92
Ablation study for reinforced reuse finetuning method									
CBR+Re4 w/o Finetuning	0	55.39	68.81	71.03	66.06	57.72	59.83	63.01	59.17
CBR+Re4 w/o SFT	1	55.24	69.87	70.94	66.60	58.78	61.02	63.12	60.02
CBR+Re4 w/o RLFT	0	64.10	74.8	69.44	68.64	60.92	61.99	59.29	59.06
Comparison with different RL algorithms in reinforced reuse finetuning method									
CBR+Re4 w/ Online DPO	2	63.99	74.76	69.31	68.58	60.71	61.98	59.56	59.09
CBR+Re4 w/ Remax	2	64.35	75.03	69.46	68.80	60.99	62.02	59.36	59.10
CBR+Re4 w/ RLOO	4	64.76	75.62	71.01	69.88	61.87	63.44	62.05	60.97
CBR+Re4 w/ GRPO	4	65.67	75.69	72.29	70.80	63.18	64.82	62.83	62.17

gap of CBR+Re4 without retrieval finetuning is reported for both datasets in Figure 5(a). The results show that the ablation causes a performance drop of approximately 1% in terms of FF1 score on both datasets. This highlights the effectiveness of the proposed reranking-based retrieval finetuning method.

4.3.2 Analyses for Reinforced Reuse Finetuning. Now, we present in-depth analyses for the proposed reinforced reuse finetuning method. First, we conduct an ablation study to validate its effectiveness. Next, we replace the proposed REINFORCE algorithm with several state-of-the-art RLHF algorithms to highlight the superiority of our approach. We then demonstrate an additional advantage of the RLFT stage in mitigating repetitive generation issues. Finally, we analyze the impact of the KL coefficient β through a hyper-parameter analysis.

Ablation study. We first conduct an ablation study of CBR+Re4 by evaluating the following ablation variants: **(1) w/o Finetuning**, which relies solely on the foundational capabilities of the LLM; **(2) w/o SFT**, which applies RLFT directly to the base LLM without the prior SFT stage; and **(3) w/o RLFT**, which involves only SFT stage without the subsequent RLFT stage.

The corresponding results are presented in the upper section of Table 2. Among them, CBR+Re4 w/o Finetuning demonstrates the poorest performance on both datasets, emphasizing the necessity of further aligning the LLM with our production scenario. Additionally, both CBR+Re4 w/o SFT and w/o RLFT underperform CBR+Re4. This outcome aligns with expectations: SFT may introduce undesired behaviors due to the noise in ground-truth labels, while RLFT may suffer from sample inefficiency. The combined SFT-RLFT paradigm, consistent with best practices in the RLHF community, strikes the desired trade-off between effectivenss and efficiency.

Comparison with advanced RLHF algorithms. We compare the proposed REINFORCE algorithm with several state-of-the-art RLHF algorithms. Unlike typical RLHF settings, our production scenario leverages a golden reward function instead of a trained reward model. Moreover, we focus on improving the case-based reasoning capabilities instead of general instruction following as in RLHF. Specifically, we evaluate the following algorithms: **(1) Online DPO** [11], which performs pairwise comparisons with the reward function and optimizes the LLM using DPO [24] in an online manner. **(2) Remax** [20], which incorporates a baseline via greedy decoding in the REINFORCE algorithm to reduce variance. **(3) RLOO** [2], which employs a variance-reduced multi-sample estimate for policy updates. **(4) GRPO** [28], which calculates the advantage in the PPO objective based on group-level relative rewards. Due to space limitations, implementation details are provided in Appendix C. Note that we exclude the PPO [27] algorithm from our comparison due to its high computational costs. Furthermore, prior RLHF works have demonstrated that Remax, RLOO, and GRPO achieve superior alignment compared to PPO.

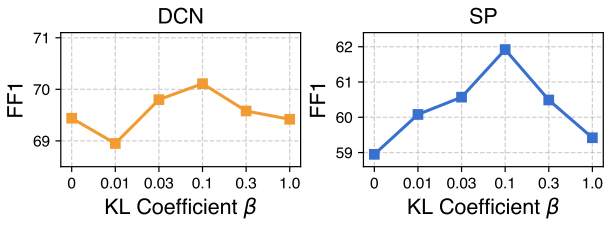

Figure 6: Hyper-parameter analyses of KL coefficient β on both datasets.

The experimental results are presented in the lower section of Table 2. These results show that all four RLHF algorithms further improve alignment compared to the SFT stage, confirming the necessity of the RLFT stage. Additionally, the proposed REINFORCE algorithm outperforms Online DPO, Remax, and RLOO, while delivering competitive performance relative to GRPO, further demonstrating its effectiveness. Notably, all these RLHF algorithms require two or more on-policy samples per query, significantly increasing time and computational costs. In contrast, the proposed REINFORCE algorithm requires only a single on-policy sample per query, striking the desired trade-off between the alignment performance and costs of on-policy sampling.

Analysis on repetitive generation issue. During prior online deployment of CBR+SFT, we observed that the generated test scripts

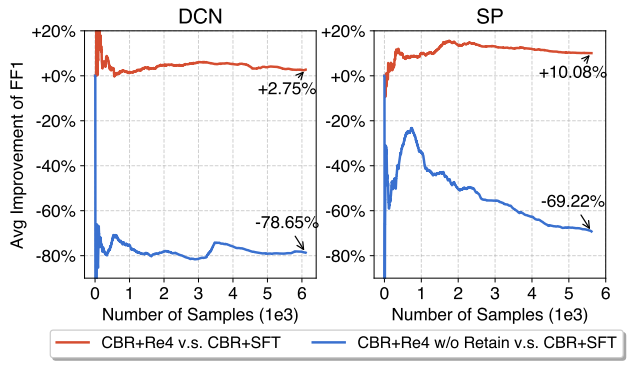

Figure 7: Online evaluation results of CBR+Re4 on two PDUs.

often exhibited the repetitive generation issue [12, 42]. An example of such repetitive generation is provided in Figure 10 in Appendix D. This issue significantly slows response times, increases inference costs, and adversely impacts user experience. As such patterns are challenging to detect using rule-based methods, we conduct a human evaluation to assess the repetitive generation issue in the test scripts generated by CBR+SFT and CBR+Re4.

The percentage of repetitive generation is presented in Figure 5(b). As observed, CBR+Re4 effectively mitigates the repetitive generation issue, reducing it by 2.47% for DCN and 2.41% for SP. Notably, CBR+Re4 exhibits only 0.45% repetitive generation in SP, highlighting the superiority of the RLFT stage. This improvement is understandable, as SFT may memorize poor coding practices from the training corpus, while the noise in the ground-truth labels further exacerbates these undesired behaviors. In contrast, during the RLFT stage, test scripts with repetitive generation patterns receive near-zero rewards due to repetitive invocation of the same functions. As such, these repetitive behaviors are penalized by the on-policy RL objective, thereby alleviating the issue.

Hyper-parameter analysis on KL coefficient β. Finally, we analyze an important hyperparameter in the reinforced reuse finetuning method: the KL divergence coefficient β. We evaluate β at values of $\{0, 0.01, 0.03, 0.1, 0.3, 1.0\}$ and report the FF1 score on both datasets in Figure 6. Notably, $\beta = 0$ serves as an ablation study of the KL divergence term in Eq. (12), resulting in a significant performance decline. This highlights the importance of preventing too much deviation from the finetuned LLM parameters. As expected, setting β too conservatively or too aggressively can negatively impact performance. As shown in Figure 6, $\beta = 0.1$ yields the best performance on both datasets.

4.4 Online Evaluation Results

In this subsection, we provide an online evaluation of the proposed CBR system. A natural approach to quantifying online performance is to estimate the acceptance rate through online A/B tests. However, this can be problematic in our production scenario. Specifically, users exhibit significant variability in their preferences for accepting the generated test scripts, resulting in biased evaluation outcomes. Worse still, the limited number of users within each PDU prevents the law of large numbers from eliminating this bias. In addition, the human evaluation method described in Section 4.2.2 is prohibitively expensive for assessing online performance. To address this challenge, we report the online performance with FF1 score, evaluated on approximately 6,000 recent online samples from the deployed system across two PDUs. The evaluation aligns with the workflow of the CBR system: sequentially processing the user requests of test intent description, generating test scripts with the Retrieve and Reuse step, and finally retaining the test intent description with the revised test script into the case bank.

We compare the previously deployed **CBR+SFT** system with two methods: **(1) CBR+Re4** and **(2) CBR+Re4 w/o Retain**, an ablation variant of the CBR system where the Retain step is removed and the case bank remains fixed. It is worth noting that CBR+Re4 w/o Retain can be viewed as a variant of RAG, as it only reserves the Retrieve-Reuse process in CBR. Figure 7 depicts the average improvement in FF1 score as the number of online samples increases across two PDUs, DCN and SP. We observe that CBR+Re4 w/o Retain significantly underperforms CBR+Re4. This performance gap can be attributed to the dynamic nature of our scenario, where the software is continuously updated, and new modules are regularly introduced. Consequently, updating the case bank in an online manner is crucial for maintaining the effectiveness of test script generation. The Retain step in the CBR system addresses this need by enabling seamless integration of new information into the case bank. Furthermore, the proposed CBR+Re4 consistently outperforms the previously deployed CBR+SFT, achieving improvements in FF1 score of 2.75% in DCN and 10.08% in SP. These results validate the effectiveness of the proposed Re4 optimization method.

5 Related Work

5.1 LLMs for Software Testing

LLMs have demonstrated remarkable success in the field of code generation [9, 10, 16, 21, 32, 45]. Recently, the software testing community has been exploring the potential of LLMs for automated software testing [33], with a primary focus on unit test generation. For instance, Wang et al. [35] introduce TestEval, a benchmark designed to evaluate the capabilities of LLMs in generating unit test cases. Additionally, Alshahwan et al. [3] develop an autonomous workflow at Meta, enabling LLMs to improve assured unit test cases without human intervention. Different from these prior works, we explore the application of LLMs for functional test script generation, which necessitates a deep understanding of the complex code structure of the target software.

5.2 Case-Based Reasoning

CBR [1, 18, 36] is a classic AI paradigm that operates by retrieving similar past cases, reusing their solution and continuously retaining new cases into the case bank. There are some recent works [10, 38, 40] that integrate LLMs with CBR to enhance the capabilities of LLMs. Unlike these works, we explore the optimization of the CBR system in this work by further finetuning the retriever model and the LLM to align with the production scenario. Notably, the Retrieve-Reuse process in the CBR system is similar to the well-established RAG techniques [7, 25, 34], offering valuable insights for optimizing the CBR system. However, prior approaches [6, 29] to finetuning the retriever model rely heavily on feedback from LLMs, which proves too costly for our production scenario. Moreover, previous RAG

works [26, 43, 49] focus on finetuning LLMs to generate responses with the retrieved document chunks in a robust manner, while the CBR system emphasizes adapting and reusing the retrieved cases for solving the new problem. As a result, these existing methods cannot be seamlessly leveraged for optimizing our CBR system.

6 Conclusion

In this paper, we explore the pioneering application of LLMs for functional test script generation. We introduce a CBR system that facilitates LLMs to effectively and flexibly utilize the mapping between test intent descriptions and function calls in the retrieved cases for test script generation. To further enhance the CBR system, we propose Re4, which consists of a reranking-based retrieval finetuning method for the retriever model and a reinforced reuse finetuning method for the LLM. Experimental results on two real-world datasets from Huawei Datacom demonstrate that the proposed CBR+Re4 approach significantly outperforms other baselines. Moreover, the Re4 method helps mitigate the issue of repetitive generation in LLMs, further enhancing the user experience.

7 Limitation and Future Work

Firstly, we utilize bge-m3 with cosine similarity as the retriever in this work, which can be further improved by hybrid retrieval methods, multiple-stage retrieval methods, etc. Secondly, more comprehensive offline evaluation metrics may help us better benchmark the performance of the algorithms, such as function parameter accuracy, execution success rate, etc. In this work, we focus on the evaluation of function correctness, as this is the most important business metric based on our preliminary internal investigation. In contrast, execution success rate is a much more reliable evaluation metric; however, it is hard to calculate due to the complicated execution environment required by the target commercial software. Thus, there is still a long way to go towards fully automated functional testing, and this work aims to serve as a good starting point. Furthermore, the proposed CBR system can also be extended to broader decision-making scenarios in fields such as law, medicine, and finance. We plan to further explore the potentials in these domains in future work. Lastly, this paper focuses on function correctness and does not perform comprehensive user-centric evaluation, such as user acceptance rate, user satisfaction rate, time savings, etc. Investigation on these metrics may help us understand the bottleneck of the algorithms better.

Acknowledgments

We truly thank the reviewers for their great effort in our submission. In this work, Siyuan Guo, Hechang Chen and Yi Chang are supported by National Natural Science Foundation of China through grants (624B2059), National Key R&D Program of China under Grant (No. 2023YFF0905400), National Natural Science Foundation of China through grants (U2341229, 61976102, U19A2065, 62476110), Key R&D Project of Jilin Province (No. 20240304200SF), and International Cooperation Project of Jilin Province (No. 20220402009GH).

References

[1] Agnar Aamodt and Enric Plaza. 1994. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI communications* 7, 1 (1994), 39–59.

[2] Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Üstün, and Sara Hooker. 2024. Back to Basics: Revisiting REINFORCE-Style Optimization for Learning from Human Feedback in LLMs. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 12248–12267.

[3] Nadia Alshahwan, Jubin Chheda, Anastasia Finogenova, Beliz Gokkaya, Mark Harman, Inna Harper, Alexandru Marginean, Shubho Sengupta, and Eddy Wang. 2024. Automated unit test improvement using large language models at meta. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*. 185–196.

[4] Leila Amgoud and Vivien Beuselinck. 2022. Towards a principle-based approach for case-based reasoning. In *International Conference on Scalable Uncertainty Management*. Springer, 37–46.

[5] Jianlv Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. 2024. BGE M3-Embedding: Multi-Lingual, Multi-Functionality, Multi-Granularity Text Embeddings Through Self-Knowledge Distillation. arXiv:2402.03216

[6] Guanting Dong, Yutao Zhu, Chenghao Zhang, Zechen Wang, Zhicheng Dou, and Ji-Rong Wen. 2024. Understand what LLM needs: Dual preference alignment for retrieval-augmented generation. *arXiv preprint arXiv:2406.18676* (2024).

[7] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997* (2023).

[8] Zenon Gniazdowski and Maciej Boniecki. 2019. Detection of a source code plagiarism in a student programming competition. *arXiv preprint arXiv:1912.08138* (2019).

[9] Antoine Grosnit, Alexandre Maraval, James Doran, Giuseppe Paolo, Albert Thomas, Refinath Shahul Hameed Nabeezath Beevi, Jonas Gonzalez, Khyati Khandelwal, Ignacio Iacobacci, Abdelhakim Benechehab, et al. 2024. Large Language Models Orchestrating Structured Reasoning Achieve Kaggle Grandmaster Level. *arXiv preprint arXiv:2411.03562* (2024).

[10] Siyuan Guo, Cheng Deng, Ying Wen, Hechang Chen, Yi Chang, and Jun Wang. 2024. DS-Agent: Automated Data Science by Empowering Large Language Models with Case-Based Reasoning. In *Proceedings of the 41st International Conference on Machine Learning*. 16813–16848.

[11] Shangmin Guo, Biao Zhang, Tianlin Liu, Tianqi Liu, Misha Khalman, Felipe Llinares, Alexandre Rame, Thomas Mesnard, Yao Zhao, Bilal Piot, et al. 2024. Direct language model alignment from online ai feedback. *arXiv preprint arXiv:2402.04792* (2024).

[12] Tatsuya Hiraoka and Kentaro Inui. 2024. Repetition Neurons: How Do Language Models Produce Repetitions? *arXiv preprint arXiv:2410.13497* (2024).

[13] 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 *International Conference on Learning Representations*.

[14] Eyke Hullermeier. 2007. Credible case-based inference using similarity profiles. *IEEE Transactions on Knowledge and Data Engineering* 19, 6 (2007), 847–858.

[15] Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. 2022. Unsupervised Dense Information Retrieval with Contrastive Learning. *Transactions on Machine Learning Research* (2022).

[16] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. 2024. SWE-bench: Can Language Models Resolve Real-world Github Issues?. In *The Twelfth International Conference on Learning Representations*.

[17] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense Passage Retrieval for Open-Domain Question Answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, 6769–6781.

[18] Janet L Kolodner. 1992. An introduction to case-based reasoning. *Artificial intelligence review* 6, 1 (1992), 3–34.

[19] VI Levenshtein. 1966. Binary codes capable of correcting deletions, insertions, and reversals. *Proceedings of the Soviet physics doklady* (1966).

[20] Ziniu Li, Tian Xu, Yushun Zhang, Zhihang Lin, Yang Yu, Ruoyu Sun, and Zhi-Quan Luo. 2024. ReMax: A Simple, Effective, and Efficient Reinforcement Learning Method for Aligning Large Language Models. In *Proceedings of the 41st International Conference on Machine Learning*. 29128–29163.

[21] Bingchang Liu, Chaoyu Chen, Zi Gong, Cong Liao, Huan Wang, Zhichao Lei, Ming Liang, Dajun Chen, Min Shen, Hailian Zhou, et al. 2024. Mftcoder: Boosting code llms with multitask fine-tuning. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 5430–5441.

[22] Gabriel de Souza P Moreira, Radek Osmulski, Mengyao Xu, Ronay Ak, Benedikt Schifferer, and Even Oldridge. 2024. NV-Retriever: Improving text embedding models with effective hard-negative mining. *arXiv preprint arXiv:2407.15831* (2024).

[23] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. 2018. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748* (2018).

[24] Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct preference optimization: Your language

model is secretly a reward model. *Advances in Neural Information Processing Systems* 36 (2023).
[25] Ohad Rubin, Jonathan Herzig, and Jonathan Berant. 2022. Learning To Retrieve Prompts for In-Context Learning. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 2655–2671.
[26] Tobias Schimanski, Jingwei Ni, Mathias Kraus, Elliott Ash, and Markus Leippold. 2024. Towards Faithful and Robust LLM Specialists for Evidence-Based Question-Answering. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 1913–1931.
[27] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).
[28] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Y Wu, et al. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300* (2024).
[29] Weijia Shi, Sewon Min, Michihiro Yasunaga, Minjoon Seo, Richard James, Mike Lewis, Luke Zettlemoyer, and Wen-tau Yih. 2024. REPLUG: Retrieval-Augmented Black-Box Language Models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*. 8364–8377.
[30] Weiwei Sun, Lingyong Yan, Xinyu Ma, Shuaiqiang Wang, Pengjie Ren, Zhumin Chen, Dawei Yin, and Zhaochun Ren. 2023. Is ChatGPT Good at Search? Investigating Large Language Models as Re-Ranking Agents. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. 14918–14937.
[31] Weihao Tan, Wentao Zhang, Shanqi Liu, Longtao Zheng, Xinrun Wang, and Bo An. 2024. True Knowledge Comes from Practice: Aligning Large Language Models with Embodied Environments via Reinforcement Learning. In *The Twelfth International Conference on Learning Representations*.
[32] Runchu Tian, Yining Ye, Yujia Qin, Xin Cong, Yankai Lin, Yinxu Pan, Yesai Wu, Haotian Hui, Weichuan Liu, Zhiyuan Liu, et al. 2024. Debugbench: Evaluating debugging capability of large language models. *arXiv preprint arXiv:2401.04621* (2024).
[33] Junjie Wang, Yuchao Huang, Chunyang Chen, Zhe Liu, Song Wang, and Qing Wang. 2024. Software Testing With Large Language Models: Survey, Landscape, and Vision. *IEEE Transactions on Software Engineering* 50, 4 (2024), 911–936. https://doi.org/10.1109/TSE.2024.3368208
[34] Liang Wang, Nan Yang, and Furu Wei. 2023. Learning to retrieve in-context examples for large language models. *arXiv preprint arXiv:2307.07164* (2023).
[35] Wenhan Wang, Chenyuan Yang, Zhijie Wang, Yuheng Huang, Zhaoyang Chu, Da Song, Lingming Zhang, An Ran Chen, and Lei Ma. 2024. TESTEVAL: Benchmarking Large Language Models for Test Case Generation. *arXiv preprint arXiv:2406.04531* (2024).
[36] Ian Watson and Farhi Marir. 1994. Case-based reasoning: A review. *The knowledge engineering review* 9, 4 (1994), 327–354.
[37] Muning Wen, Ziyu Wan, Jun Wang, Weinan Zhang, and Ying Wen. 2024. Reinforcing LLM Agents via Policy Optimization with Action Decomposition. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.
[38] Kaitlynne Wilkerson and David Leake. 2024. On Implementing Case-Based Reasoning with Large Language Models. In *International Conference on Case-Based Reasoning*. Springer, 404–417.
[39] Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8 (1992), 229–256.
[40] Nirmalie Wiratunga, Ramitha Abeyratne, Lasal Jayawardena, Kyle Martin, Stewart Massie, Ikechukwu Nkisi-Orji, Ruvan Weerasinghe, Anne Liret, and Bruno Fleisch. 2024. CBR-RAG: case-based reasoning for retrieval augmented generation in LLMs for legal question answering. In *International Conference on Case-Based Reasoning*. Springer, 445–460.
[41] Lee Xiong, Chenyan Xiong, Ye Li, Kwok-Fung Tang, Jialin Liu, Paul N. Bennett, Junaid Ahmed, and Arnold Overwijk. 2021. Approximate Nearest Neighbor Negative Contrastive Learning for Dense Text Retrieval. In *International Conference on Learning Representations*.
[42] Jin Xu, Xiaojiang Liu, Jianhao Yan, Deng Cai, Huayang Li, and Jian Li. 2022. Learning to break the loop: Analyzing and mitigating repetitions for neural text generation. *Advances in Neural Information Processing Systems* 35 (2022), 3082–3095.
[43] Ran Xu, Hui Liu, Sreyashi Nag, Zhenwei Dai, Yaochen Xie, Xianfeng Tang, Chen Luo, Yang Li, Joyce C Ho, Carl Yang, et al. 2024. SimRAG: Self-Improving Retrieval-Augmented Generation for Adapting Large Language Models to Specialized Domains. *arXiv preprint arXiv:2410.17952* (2024).
[44] Xue Yan, Yan Song, Xidong Feng, Mengyue Yang, Haifeng Zhang, Haitham Bou Ammar, and Jun Wang. 2025. Efficient Reinforcement Learning with Large Language Model Priors. In *The Thirteenth International Conference on Learning Representations*.
[45] Zhen Yang, Fang Liu, Zhongxing Yu, Jacky Wai Keung, Jia Li, Shuo Liu, Yifan Hong, Xiaoxue Ma, Zhi Jin, and Ge Li. 2024. Exploring and unleashing the power of large language models in automated code translation. *Proceedings of the ACM on Software Engineering* 1, FSE (2024), 1585–1608.
[46] Yongcheng Zeng, Guoqing Liu, Weiyu Ma, Ning Yang, Haifeng Zhang, and Jun Wang. 2024. Token-level Direct Preference Optimization. In *Proceedings of the 41st International Conference on Machine Learning*. 58348–58365.
[47] Jingtao Zhan, Jiaxin Mao, Yiqun Liu, Jiafeng Guo, Min Zhang, and Shaoping Ma. 2021. Optimizing dense retrieval model training with hard negatives. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 1503–1512.
[48] Jingtao Zhan, Jiaxin Mao, Yiqun Liu, Min Zhang, and Shaoping Ma. 2020. RepBERT: Contextualized text embeddings for first-stage retrieval. *arXiv preprint arXiv:2006.15498* (2020).
[49] Tianjun Zhang, Shishir G Patil, Naman Jain, Sheng Shen, Matei Zaharia, Ion Stoica, and Joseph E Gonzalez. 2024. Raft: Adapting language model to domain specific rag. *arXiv preprint arXiv:2403.10131* (2024).

Appendix

A Algorithmic Details

A.1 Business Metric Details

For the proposed function precision, function recall, and function f1 score, we first extract the list of the invoked functions in the test scripts with abstract syntax tree, and then calculate these metrics with Eq. (2), Eq. (3), and Eq. (4). To make it more clear, we provide an example of the calculation of these three metrics in Figure 8.

Now, we detail the definition of the code similarity [8] with the tool of Levenshtein distance [19]. Given the generated test script $\hat{y}$ and the ground-truth test script y, the code similarity is defined as:

$$\text{CodeSimilarity}(\hat{y}, y) = 1 - \frac{d_{\text{L}}(\hat{y}, y)}{\max(|\hat{y}|, |y|)}, \tag{13}$$

where d_{L} denotes the Levenshtein distance, measuring the minimum number of single-character edits (insertions, deletions, or substitutions) needed to change one string into another. As stated earlier, code similarity serves only as a complementary evaluation metric, and we take function f1 score as the most critical evaluation metric due to the characteristics of our production scenario.

A.2 Algorithm Details

In this subsection, we provide the pseudo-codes for our proposed method. We first summarize the workflow of the proposed CBR system during deployment in Algorithm 1. Our CBR system adopts the 4R cycle with Retrieve, Reuse, Revise and Retain. Then, we summarize the proposed reranking-based retrieval finetuning method in Algorithm 2, which first generates pseudo labels for positive and negative examples and then finetunes the retriever model with contrastive learning. Finally, we summarize the proposed reinforced reuse finetuning method in Algorithm 3, which consists of supervised finetuning and reinforcement learning finetuning, striking the desired trade-off between efficiency and effectiveness.

B Why Does REINFORCE Work in Our RLFT Setting?

REINFORCE [39] is the most basic on-policy RL algorithm. Despite its simplicity, it suffers from high variance in the stochastic gradient [2, 20]. As highlighted in [20], this high variance arises from two sources: (1) external randomness in the Markov Decision Process (MDP) and (2) internal randomness in the sampling of LLMs. In RLHF and RLFT settings, external randomness is eliminated due to the deterministic nature of the transition and reward functions in the MDP. However, in RLHF, external randomness still affects

Generated Test Script $\hat{y}$

```
navigate_to_login_page()
enter_password("wrong_password")
enter_username("invalid_user")
click_logged_button()
assert message == "Invalid credentials"
```

Ground-Truth Test Script y

```
navigate_to_login_page()
enter_username("invalid_user")
enter_password("wrong_password")
click_login_button()
assert get_message() == "Invalid credentials"
```

$\text{Func}(y)$: `navigate_to_login_page, enter_password, enter_username, click_logged_button`
$\text{Func}(\hat{y})$: `navigate_to_login_page, enter_username, enter_password, click_login_button, get_message`
$\text{Func}(\hat{y}) \cap \text{Func}(y)$: `navigate_to_login_page, enter_username, enter_password`
$\text{FPrecision} = |\text{Func}(\hat{y}) \cap \text{Func}(y)|/|\text{Func}(\hat{y})| = 3/4$
$\text{FRecall} = |\text{Func}(\hat{y}) \cap \text{Func}(y)|/|\text{Func}(y)| = 3/5$
$\text{FF1} = 2 \times \text{FPrecision} \times \text{FRecall}/(\text{FPrecision} + \text{FRecall}) = 2/3$

Figure 8: An example for the calculation of function precision, function recall and function f1 score.

Algorithm 1 The CBR System During Deployment

Input: Case Bank C, Finetuned embedding model $\mathbf{E}_\phi$, and Finetuned LLM π_θ.
for user request with test intent description q **do**
  ▷ Retrieve
  Retrieve top-M cases C_q based on Eq. (6)
  ▷ Reuse
  Sample a test script $\hat{y} \sim \pi_\theta(\cdot|q, C_q)$
  ▷ Revise
  Revise the generated test script $\hat{y}$ as y by test engineers if necessary
  ▷ Retain
  Retain the test intent description and the test script as a new case into the case bank, i.e., $C \leftarrow C \cup \{(q, y)\}$
end for

Algorithm 2 Reranking-based Retrieval Finetuning Method

Input: Training set $\mathcal{D}$, Pretrained embedding model $\mathbf{E}_\phi$.
Initialize labeled training dataset $\mathcal{D}_{\text{labeled}} = \{\}$
▷ Generate pseudo-labels for positive and negative examples
for sample (q, y) in $\mathcal{D}$ **do**
  Construct the case bank as $C = \mathcal{D} \setminus \{(q, y)\}$
  Retrieve top-k cases C_q based on Eq. (6)
  Rerank the retrieved cases C_q to generate pseudo-label for positive example c^+ and negative examples C^- as in Eq. (7) and Eq. (8)
  Store the labeled samples $\mathcal{D}_{\text{labeled}} \leftarrow \mathcal{D}_{\text{labeled}} \cup \{(q, c^+, C^-)\}$
end for
▷ Finetune the embedding model with labeled dataset
for labeled sample (q, c^+, C^-) in $\mathcal{D}_{\text{labeled}}$ **do**
  Update ϕ by minimizing $\mathcal{L}(\phi)$ in Eq. (5)
end for

Algorithm 3 Reinforced Reuse Finetuning Method

Input: Training set $\mathcal{D}$, Finetuned embedding model $\mathbf{E}_\phi$, Large language model π_θ.
Retrieve M cases C_q for each sample in $\mathcal{D}$ using $\mathbf{E}_\phi$
▷ Supervised finetuning
for sample (q, C_q, y) in $\mathcal{D}$ **do**
  Update θ by minimizing $\mathcal{L}_{\text{SFT}}(\theta)$ in Eq. (9)
end for
▷ Reinforcement learning finetuning
for sample (q, C_q, y) in $\mathcal{D}$ **do**
  Sample a test script $\hat{y} \sim \pi_\theta(\cdot|q, C_q)$
  Update θ by minimizing $\mathcal{L}_{\text{REINFORCE}}(\theta)$ in Eq. (12)
end for

performance, as the model receives varying reward scales due to the diverse prompts in the training set, leading to high variance in the stochastic gradient [20]. In contrast, our RLFT setting does not exhibit this issue, since it focuses on enabling LLMs to reuse retrieved cases for test script generation, significantly narrowing the action space. This results in more aligned reward scales and reduced variance.

Now, we provide empirical evidence supporting the above claim. Since a smaller gradient variance corresponds to a smaller gradient norm, we follow prior work [20] and plot the gradient norm of REINFORCE in our RLFT setting and RLHF setting in Figure 9. For RLHF, we follow [20] to train the LLM with the consistent training setups as ours, where we adopt three prompt dataset (*ultrafeedback*, *lmsys-chat-1m*, *sharegpt-en*) and the reward model (UltraRM-13B) for RLHF. The results reveal that the gradient norm of REINFORCE in our RLFT settings remains below 1 across both datasets, whereas this value fluctuates between 10 and 4000 in RLHF. Thus, REINFORCE does not suffer from the high variance issue and can achieve the desired performance in our RLFT setting. Note that the gradient norm of REINFORCE in RLHF remains 0 in the later stage of training due to training collapse.

C Discussion of the Compared RLHF Algorithms

In this section, we present a detailed discussion on how we tailor the state-of-the-art RLHF algorithms compared in the main body of the paper for our RLFT setting.

Online DPO [11] samples two test scripts $\hat{y}_1$ and $\hat{y}_2$ with the LLM in an on-policy manner. Then, we can label the preference $y_w \succ y_l$

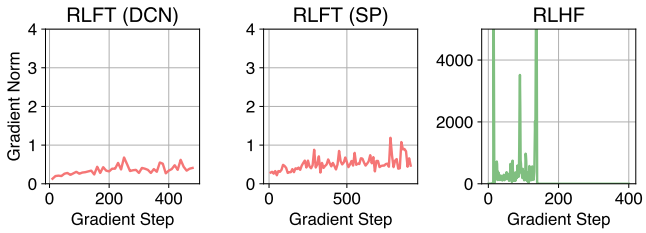

Figure 9: Comparison for the gradient norm of REINFORCE in RLFT and RLHF.

with the script similarity such that $\mathrm{FF1}(y_w, y) > \mathrm{FF1}(y_l, y)$. The optimization process aligns with the standard DPO loss [24] as:

$$\mathcal{L}_{\mathrm{DPO}}(\theta) = -\log \sigma \left(\beta \log \frac{\pi_\theta(y_w|q, C_q)}{\pi_{\theta_{\mathrm{SFT}}}(y_w|q, C_q)} - \beta \log \frac{\pi_\theta(y_l|q, C_q)}{\pi_{\theta_{\mathrm{SFT}}}(y_l|q, C_q)} \right). \tag{14}$$

However, Online DPO only reserves the preference relationship for alignment and ignores the fine-grained golden reward information, thereby decreasing the sample efficiency and resulting in inferior alignment performance. Moreover, it requires two on-policy samples per query, which brings 2× time and computational costs compared to REINFORCE.

Remax [20] samples two test scripts $\hat{y}_1$ and $\hat{y}_2$ with the LLM in an on-policy manner. Among them, $\hat{y}_2$ is sampled via greedy decoding to serve as a baseline for variance reduction. The optimization process merely utilizes $\hat{y}_1$, with the standard REINFORCE [39] loss function as:

$$\mathcal{L}_{\mathrm{Remax}}(\theta) = -\log \pi_\theta(\hat{y}_1|q, C_q)[r(\hat{y}_1) - r(\hat{y}_2)]. \tag{15}$$

However, the introduction of baseline value $r(\hat{y}_2)$ can be a redundant design in our context, and may even lead to a biased gradient estimator, despite being theoretically unbiased in expectation. Different from open-ended text generation, our setting only requires LLMs to reuse the retrieved cases for test script generation, thus significantly narrowing the action space. As such, the internal randomness mentioned in [20] is much less problematic in our setting. Similar to Online DPO, Remax also requires two on-policy samples per query, which brings 2× time and computational costs compared to REINFORCE.

RLOO [2] samples K test scripts per query with the LLM in an on-policy manner. The optimization process utilizes the standard REINFORCE [39] with leave-one-out estimator for variance reduction, which can be formulated as:

$$\mathcal{L}_{\mathrm{RLOO}}(\theta) = -\frac{1}{K} \sum_{i=1}^{K} \left[\log \pi_\theta \left(\hat{y}_i|q, C_q \right) \left(r\left(\hat{y}_i\right) - \frac{1}{K-1} \sum_{j \neq i} r\left(\hat{y}_k\right) \right) \right]. \tag{16}$$

Similarly, the variance reduction technique is also redundant for our setting. As suggested in [2], we set $K = 4$ in our implementation, which brings 4× time and computational costs compared to REINFORCE.

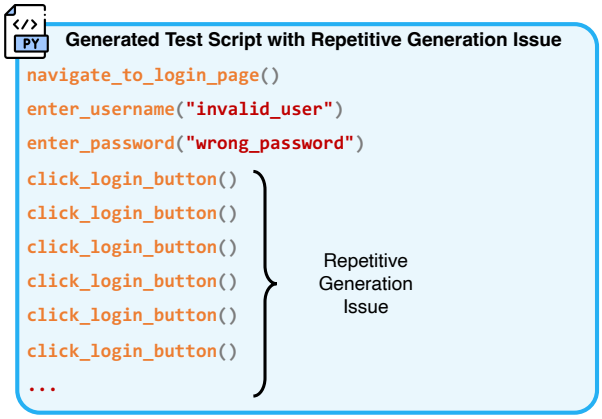

Figure 10: An example of generated test script with repetitive generation pattern.

GRPO [28] also samples K test scripts per query. In contrast to RLOO, GRPO adopts the loss function similar to PPO [27] for fine-tuning, which can be formulated as:

$$\mathcal{L}_{\mathrm{GRPO}}(\theta) = -\frac{1}{K} \sum_{i=1}^{K} \frac{1}{T_i} \sum_{t=1}^{T_i} \left[\min \left(\hat{\rho}_{i,t} \hat{A}_{i,t}, \mathrm{clip}(\hat{\rho}_{i,t} \hat{A}_{i,t}, 1-\epsilon, 1+\epsilon) \right) - \beta \mathbb{D}_{\mathrm{KL}}(\pi_\theta || \pi_{\theta_{\mathrm{SFT}}}) \right], \tag{17}$$

where T_i denotes the number of tokens in i-th generated script, $\hat{\rho}_{i,t} = \frac{\pi_\theta(\hat{y}_{i,t}|q, C_q, \hat{y}_{i,<t})}{\pi_{\theta_{\mathrm{old}}}(\hat{y}_{i,t}|q, C_q, \hat{y}_{i,<t})}$ denotes the importance rate of the t-th token in i-th generated test script, $\hat{A}_{i,t} = \frac{r(\hat{y}_i) - \mathrm{mean}(\{r(\hat{y}_1), \ldots, r(\hat{y}_K)\})}{\mathrm{std}(\{r(\hat{y}_1), \ldots, r(\hat{y}_K)\})}$ denotes the group-relative reward for advantage estimation, ϵ denotes the clipping parameter. GRPO achieves better performance than the proposed REINFORCE algorithm, benefiting from K× on-policy samples per query. Different from [28] that sets $K = 64$, we set $K = 4$ in our implementation to accommodate computational constraints, which brings 4× time and computational costs compared to REINFORCE.

D Example of Repetitive Generation Issue

Due to business considerations, we are unable to present a realistic example from our production scenario. We construct an illustrative example, as shown in Figure 10. Here are some possible reasons for this phenomenon: (1) Poor coding practices by human test engineers result in ground-truth test scripts with repetitive patterns, which is memorized by the LLMs through the SFT objective. (2) Noise within the SFT objective, as illustrated in Figure 3, contributes to hallucination issues, potentially worsening the repetitive generation problem.