Fine-Tuning LLMs to Analyze Multiple Dimensions of Code Review: A Maximum Entropy Regulated Long Chain-of-Thought Approach

GUOHAO SHI, Nanjing University, China XIANWEI WU, Nanjing University, China HAOCHUAN HE, Nanjing University, China XUEMING GU, University of Waterloo, Canada QIANQIAN ZHAO, Northeastern University, China KUI LIU, Huawei Technologies Co., Ltd., China QIUSHI WANG, Huawei Technologies Co., Ltd., China ZHAO TIAN, Huawei Technologies Co., Ltd., China HAIFENG SHEN, Southern Cross University, Australia GUOPING RONG*, Nanjing University, China

YONGDA YU, Nanjing University, China

Large Language Models (LLMs) have shown great potential in supporting automated code review due to their impressive capabilities in context understanding and reasoning. However, these capabilities are still limited compared to human-level cognition because they are heavily influenced by the training data. Recent research has demonstrated significantly improved performance through fine-tuning LLMs with code review data. However, compared to human reviewers who often simultaneously analyze multiple dimensions of code review to better identify issues, the full potential of these methods is hampered by the limited or vague information used to fine-tune the models. This paper contributes MelcotCR, a chain-of-thought (COT) finetuning approach that trains LLMs with an impressive reasoning ability to analyze multiple dimensions of code review by harnessing long COT techniques to provide rich structured information. To address context loss and reasoning logic loss issues that frequently occur when LLMs process long COT prompts, we propose a solution that combines the Maximum Entropy (ME) modeling principle with pre-defined reasoning pathways in MelcotCR to enable more effective utilization of in-context knowledge within long COT prompts while strengthening the logical tightness of the reasoning process. Empirical evaluations on our curated MelcotCR dataset and the public CodeReviewer dataset reveal that a low-parameter base model, such as 14B Qwen2.5, fine-tuned with MelcotCR can surpass state-of-the-art methods in terms of the accuracy of detecting and describing code issues, with its performance remarkably on par with that of the 671B DeepSeek-R1 model.

*Guoping Rong is the corresponding author.

Authors' Contact Information: Yongda Yu, Nanjing University, Nanjing, China, yuyongda@smail.nju.edu.cn; Guohao Shi, Nanjing University, Nanjing, China, 335933870@qq.com; Xianwei Wu, Nanjing University, Nanjing, China, 652024320006@smail.nju.edu.cn; Haochuan He, Nanjing University, Nanjing, China, 1516998446@qq.com; XueMing Gu, University of Waterloo, Avenue, Canada, x7gu@uwaterloo.ca; Qianqian Zhao, Northeastern University, Shenyang, China, zhaoqianqian2025@163.com; Kui Liu, Huawei Technologies Co., Ltd., Xi'an, China, kui.liu@huawei.com; Qiushi Wang, Huawei Technologies Co., Ltd., Xi'an, China, wangqiushi6@huawei.com; Zhao Tian, Huawei Technologies Co., Ltd., Xi'an, China, tianzhao@huawei.com; Haifeng Shen, Southern Cross University, Gold Coast, Australia, haifeng.shen@scu.edu.au; Guoping Rong, ronggp@nju.edu.cn, Nanjing University, Nanjing, China.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

 $\@ifnextchar[{\@model{O}}{@}$ 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM 1557-735X/2018/8-ART111

111:2 Yu et al.

CCS Concepts: • Software and its engineering → Software creation and management.

Additional Key Words and Phrases: Code Review, Large Language Model, Long Chain-of-Thought

ACM Reference Format:

1 Introduction

As a cornerstone practice in software quality assurance, code review plays an irreplaceable role in modern software development [11]. Traditional manual review processes, which heavily rely on line-by-line scrutiny of code by developers, are not only time-consuming and labor-intensive but also heavily dependent on individual reviewers' expertise, a limitation that has driven the innovation and advancement of Automated Code Review (ACR) technologies [14, 29, 31, 48, 59]. Recent breakthroughs in AI-driven code analysis based on Large Language Models (LLMs) have naturally attracted significant research attention to ACR, resulting in multiple LLM-based ACR systems [31, 59].

As an LLM's capability in context understanding and reasoning is highly influenced by its training data, a common strategy is to fine-tune base LLMs with code-review-specific data so that they can perform better in ACR tasks. In particular, Liu et al. [31] fine-tuned Llama to create Llama-Reviewer, which is able to generate review comments through the direct learning of a raw code review dataset from a previous study [30] to enhance the LLM's commenting capability. It should be noted that the information used in fine-tuning is only a direct code-to-comment mapping that lacks analytical depth. In contrast, Yu et al. developed Carllm [59] that used chain-of-thought (COT) to fine-tune the base LLMs by decomposing the review process into three progressive stages: defect localization, review comment generation, and solution proposal. The COT approach provides richer information for fine-tuning, resulting in improved ACR performance compared to Llama-Reviewer.

However, compared to human reviewers who often simultaneously consider multiple dimensions of code review, e.g., code intent, boundary conditions, and invocation relationships, when analyzing code issues [2, 22], the full potential of LLMs for ACR is hampered by the limited or vague information used to fine-tune them. Specifically, Llama-Reviewer [31] merely instructs the LLM to perform ACR via vague prompts, while in Carllm [59], each reasoning step is constrained in such a way that it only requires the LLM to identify issues without providing much guiding information. Some studies have revealed a trend that employing COT to prompt LLMs can significantly enhance their performance in specific tasks [9, 52, 63], while there exists a positive correlation between COT length and model performance gain [5, 55].

In this paper, we contribute *MelcotCR*, a fine-tuning approach that trains LLMs with an impressive structured reasoning ability to analyze multiple dimensions of code review by harnessing long COT techniques [5, 54, 57]. It systematically decomposes a code review task into multiple fine-grained sub-tasks including code functionality summarization, core logic analysis, change impact analysis, and inspection of multiple concrete issues, thereby stimulating an LLM to generate coherent reasoning chains. However, a known challenge with long COT is its inherent context loss and reasoning logic loss issues, as the length of COT increases [5, 55].

To address this challenge, we propose a solution that combines the Maximum Entropy (ME) modeling principle with pre-defined reasoning pathways in *MelcotCR* to enable more effective utilization of in-context knowledge within long COT prompts while strengthening the logical tightness of the reasoning process. This solution expands each ground truth answer into multiple

semantically equivalent but syntactically distinct expressions, enabling the model to learn bias-invariant essential knowledge.

The main contributions of this work are summarized as follows:

- We propose *MelcotCR*, a novel fine-tuning approach that trains LLMs with the ability to analyze multiple dimensions in code review by combining the maximum entropy principle with long COT techniques to cultivate diversified problem-solving capabilities.
- We conduct empirical evaluations to demonstrate that a 14B low-parameter base model fine-tuned with *MelcotCR* can outperform state-of-the-art methods, with its ACR performance remarkably on par with that of Deepseek-R1 671B.
- We perform experiments to investigate the impacts of fine-tuning methods and reasoning steps on *MelcotCR*'s ACR performance. The findings not only validate the proposed approach but also suggest future research directions.

The rest of the paper is organized as follows. Section 2 introduces some related work. Section 3 describes the research methodology, followed by the evaluation process and results in Section 4. Section 5 discusses the implications and validity risks. Section 6 concludes the paper with a summary of contributions and future work.

2 Related work

In this section, we review the related work that forms the foundation of this work, including automated code review, chain of thought, and maximum entropy.

2.1 Automated Code Review (ACR)

ACR is an essential aspect of improving software development efficiency, which aims to reduce the manual effort and time required for code assessment. The main focus of an ACR system is to detect potential code defects and suggest or generate relevant review comments. It typically comprises two components: defect detection and review comment recommendation/generation.

Defect detection aims to identify potential issues within code snippets under review. For instance, CodeT5 [50] adopts a unified framework that supports both code understanding and generation tasks, thus facilitating multi-task learning. CodeBert [10] is a bimodal pre-training model tailored for programming languages and natural language, excelling in tasks such as natural language-based code search and code documentation generation. DACE [44] employs CNN and LSTM techniques to extract Diff features from the code, enabling the prediction of code quality in Diff patches. LogiCode [62] leverages LLMs to detect logical anomalies, automatically generating Python code to identify issues such as incorrect component quantities or missing elements.

Review comment recommendation/generation produces review comments through retrieval or generation methods. For example, LLaMA-Reviewer [31] uses low-parameter fine-tuning techniques to enhance LLaMA, leading to impressive results in generating review comments. CodeReviewer [30] achieves notable success in detecting code defects, generating review comments, and performing code repair tasks by developing pre-training tasks specifically designed for code review in an end-to-end manner. Notably, both studies utilize the same dataset [30] for training and validating their models, assuming that the existence of review comments represents the ground truth, without evaluating whether these comments are genuinely related to the code fixes. DCR [14] learns the similarity between code commit 'diffs' and review comments to retrieve comments relevant to specific code commits. CommentFinder [17] employs deep learning techniques to retrieve pertinent code review comments, thus minimizing the time reviewers spend crafting these comments. The BitsAI-CR [33] framework enhances ACR through a two-stage approach that combines a rule-based

111:4 Yu et al.

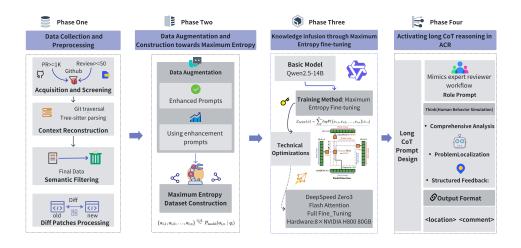


Fig. 1. A schematic overview of the MelcotCR approach

initial issue detection with a model called Reviewfilter for verification. This system, implemented with a taxonomy of review rules, achieves a precision rate of 75.0% in review comment generation.

2.2 Chain of Thought (COT)

COT is commonly applied in LLMs to assist in better problem-solving or decision-making [52]. This process helps an LLM generate more accurate, context-aware, and logical output, particularly in complex tasks like code review, debugging, or code generation [27, 34, 59]. COT requires breaking down a problem into smaller and manageable components and guiding the model to reason through each part step by step. For example, when analyzing code to identify potential issues, the model will first pinpoint the location of a potential problem, then describe the specific issue at that location, and finally suggest a repair solution [59]. This sequential reasoning process helps improve the overall accuracy of the model's output and ensures that it understands the underlying issue, not just generating a quick response. In addition, Yu Nong et al. [34] investigated the use of LLMs and the COT prompting to address security vulnerabilities in software, which achieved impressive results.

Long COT extends this idea to more complex, multi-step tasks in which the model is asked to engage in deeper reasoning, considering multiple stages or layers of a problem before reaching a conclusion [5, 51, 55]. Edward Yeo et al. [58] explored the mechanics of long COT reasoning in LLMs and found that long COT enhanced reasoning capabilities by enabling strategies such as backtracking and error correction. The researchers systematically investigated the conditions under which long COT emerged, highlighting the role of reinforcement learning (RL) in developing these capabilities. Not only in the field of coding, but also in many other non-coding fields such as translation, long COT has demonstrated extremely strong capabilities. Jiaan Wang et al. [49] introduced DRT, a new method that used long COT reasoning to enhance neural machine translation (MT). Ruohong Zhang et al. [61] introduced a COT reasoning approach to improve visual-language models, which significantly enhanced performance in visual tasks by augmenting training data and incorporating reinforcement learning. It has been found that a longer reasoning process enables the model to handle tasks with more intricate dependencies, ultimately providing better solutions.

2.3 Maximum Entropy Methods

Traditional RL typically aims to maximize cumulative reward [20]. However, pursuing only high rewards can lead to overly "deterministic" policies, sacrificing exploration capability [1]. To address this issue, researchers began to incorporate the idea of ME into RL [15, 16, 43]. Soft Actor-Critic (SAC), proposed by Haarnoja et al. [16], is currently one of the most representative ME-based RL algorithms. It is an off-policy deep RL algorithm that maximizes both the expected cumulative reward and the entropy of the policy.

The principle of Maximum Entropy (ME) originated in information theory [19]. Its core idea advocates that, given known constraints, the probability distribution with the greatest entropy (informational uncertainty) should be chosen to avoid inherent bias and local optima. In short, ME fundamentally shifts the objective of policy optimization in RL to "maximize reward + maximize entropy" [24]. This strategy shows impressive efficacy in RL, e.g., in text classification and machine translation, ME classifiers avoid overfitting and effectively integrate diverse features by maximizing the entropy of the conditional probability distribution [8, 18, 32, 35]. These early applications have laid the groundwork for extending the ME principle to complex decision-making systems.

A long COT prompt can also be viewed as a continuous and complex decision-making process. Furthermore, during an ACR process—as previously discussed—there exists an inherent need to simultaneously consider multidimensional information and perform comprehensive analysis. These characteristics demonstrate substantial comparability with features already observed in current ME-based solutions.

3 Methodology

Figure 1 depicts the *MelcotCR* approach, which comprises four phases: (1) collection and filtration of code review data from the open source community, (2) construction of ME-regulated fine-tuning datasets, (3) knowledge infusion through ME-regulated fine-tuning, and (4) activation of long COT reasoning capabilities through custom-crafted prompts.

3.1 Data Collection and Preprocessing

This phase includes the main steps such as raw data acquisition and initial screening, historical context reconstruction, semantic filtering, and long code 'diff' truncation.

Step 1: Raw data acquisition and screening. We collected project development data from the GitHub Archive¹, which captures timeline data of open-source projects including code submissions, review requests, and merge operations. Our data retrieval focused on code review records spanning from January 1, 2022 to November 1, 2024, with explicit filtering of review comments that triggered actual code modifications later on. This filtering strategy aligns with the established research consensus [22, 41, 53] that reviews leading to substantive fixes tend to contain valuable information for code review. As a result, the initial dataset comprised 12,152,191 raw review comments. To select active projects that can provide more abundant information for the dataset, we established the selection criteria that require a minimum of 1000 pull requests and 50 review comments. This selection process yielded 11,324 qualified projects containing 6,735,961 filtered review comments.

Step 2: Historical context reconstruction. As illustrated in Figure 2, we reconstructed the historical project states by traversing version-controlled files, as reviewers typically inspect code within its necessary development context. A critical challenge arises from the inherent limitations of using isolated 'diff' patch fragments referenced in review comments in current studies [31, 59]: a single 'diff' fragment typically contains only incomplete partial code logic. However, we observed that

¹https://www.gharchive.org/

111:6 Yu et al.

when feeding isolated 'diff' fragments to LLMs for code review, LLMs tend to disproportionately focus review attention on the 'diff' fragment, identifying issues located in or related to that 'diff' segment. Yet, code issues do not always reside within 'diff' fragments.

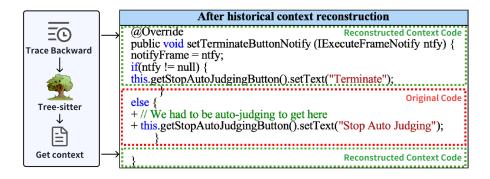


Fig. 2. The historical context reconstruction

To address this validity concern, we cloned 11,324 open-source repositories and systematically reconstructed comprehensive 'diffs' through git commit analysis. Our method consists of two substeps. First, we traced backward through commit histories associated with each review, matching the review's referenced 'diff' fragments against historical commits to identify the precise pre-commit state. Subsequently, we employed tree-sitter ² to establish semantic context for each 'diff': for intra-function modifications, we extracted the encompassing function body as context; for extra-function changes, we captured the syntactic unit containing the 'diff' (e.g., class definitions, struct declarations, or module-level scopes). It should be noted that our parsing process specifically targeted mainstream programming languages (Java, Python, C/C++, JavaScript/TypeScript, and Go) [59] to ensure practical relevance, resulting in 211,868 valid data entries. Notable data attrition occurred during this process, primarily due to two factors. The first was that historical commit truncation may occur to manage repository size, preventing recovery of early development information. The second was our systematic exclusion of documentation-related commits during code parsing, despite their prevalence in open-source reviews.

Step 3: Semantic filtering. Previous studies [4, 41, 56, 60] have noted the presence of low-value reviews in raw peer review comments, which also occurred in our dataset. To improve data quality, we needed to filter out these review comments. Using similar strategies applied in a previous study [59], we performed semantic filtering on our dataset, resulting in 13,881 data entries. Table 1 lists typical types of low-value comments.

Step 4: Long code truncation. To control resource consumption during training, we performed truncation on excessively long code while strictly preserving complete 'diff' fragments. Using the QWen2.5 tokenizer [37], we calculated tokens and enforced a 1000-token limit. When the code exceeded 1000 tokens, we retained three lines of code preceding and three lines following the 'diff' fragment as its context, maintaining the integrity of the 'diff' itself. After this step, we obtained a curated dataset of 12,881 code review instances. We refer to this dataset as the *MelcotCR* dataset.

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

Categories of Low-Value Reviews	Descriptions		
	Developer acknowledgments of issue resolution,		
Confirmation	predominantly containing status updates rather than		
	technical insights		
	Automated bot-generated messages referencing		
Submission Notices	commit hashes, filtered through hash value patterns		
	via regular expressions		
Dell Demont Fronts	Notifications regarding PR merges or		
Pull Request Events	initiations devoid of technical problem descriptions		
	Comments containing unreachable web links or		
URL References	external resource pointers excluded due to		
	crawling limitations		
Mandana	@-based user notifications lacking technical problem		
Mentions	statements or contextual details		
	Recommendations for test additions without		
Test Suggestions	sufficient contextual justification, generally		
	considered non-critical		

Table 1. Typical low-value comments

3.2 Data Entry Augmentation and Construction

There are two major steps in this phase: (1) data entry augmentation, and (2) maximum entropy dataset construction.

Step 1: Data entry augmentation. Original comments retrieved from open-source projects mostly describe issues directly, but do not articulate the full analytical reasoning process that leads to the conclusion of those issues. This phenomenon is detrimental to model training as valuable information is missing, a limitation confirmed by previous research [59]. So we also performed logical augmentation on the original review comments. Specifically, we needed to enhance both the expression and logic of the original review comments, which should include clear problem locations, professional explanations and root cause analyses of the issues, an assessment of the potential impacts of the issues, and the suggested solutions. The specific enhancement prompts are shown in Figure 3.

Step 2: ME-regulated fine-tuning dataset construction. A conventional approach to constructing a fine-tuning dataset, as formulated in Equation 1, involves pairing each query q with a canonical answer a, thereby guiding the model to generate responses aligned with predefined standards during subsequent fine-tuning processes.

$$a_i \sim P_{\text{model}}(a_i \mid q_i)$$
 (1)

However, this conventional fine-tuning method leads the models to mimicking both the desired correct answers and their solution paths only from the training data. Although this limitation is less problematic in conventional COT, it may exert negative effects in long COT scenarios because the latter, as the term suggests, inherently involves multiple coherent reasoning steps demanding logical self-consistency. To address this limitation, we emphasize infusing essential knowledge about code review into the model while eliminating trivial elements such as expression formats or styles. To ensure content validity while maximizing response diversity, we formulate a ME-regulated fine-tuning data construction paradigm shown in Equation 2. For each query q, we generate n (set to 10 in our study) distinct answer instances $\{a_1, a_2, ..., a_n\}$ to comprehensively model the ME probability space.

$$\{a_{i,1}, a_{i,2}, \dots, a_{i,n}\} \stackrel{\text{i.i.d.}}{\sim} P_{\text{model}}(a_{i,n} \mid q_i)$$
 (2)

111:8 Yu et al.

You are an expert software engineer performing detailed code analysis. Expand the provided code review comment by strictly following this structure:

<location>

[Exact problematic code snippet (keep it minimal)]

<comment>

- 1. *Technical analysis*: Clearly state the specific technical issue in 1-2 sentences. Focus on code quality, performance, security, or maintainability aspects.
- 2. Root cause rationale: Explain exactly why the identified code location causes the problem in 2-3 sentences. Reference specific patterns, anti-patterns, or language-specific considerations.
- 3. *Impact assessment*: List the most critical potential consequences of not addressing this issue, prioritizing production system impacts.
- 4. Solution proposal: Provide a concrete, actionable fix. Include specific methods/patterns to implement, avoiding vague suggestions. Just give suggestions for changes, don't provide code.

</comment>

Requirements:

- Address ONLY the highest priority issue
- Keep explanations technical but accessible to intermediate developers
- Use bullet points for impact assessment
- Maintain strict XML formatting

Code: [Target Code Snippet]

Code Review Feedback: [Expended Code Review Comment]

Fig. 3. Prompt used to generate ME-regulated fine-tuning datasets

The data entry generation process used the QWQ 32B model [46] as the base model. All inference parameters strictly adhered to QWQ 32B's recommended configurations [46] to ensure generation quality and diversity. Following the answer generation phase, we proceeded to construct instruction tuning data specifically designed for ME-regulated fine-tuning. The instruction template, as illustrated in Figure 3, was populated with code commit information and raw code review data to formulate a complete instruction, which was later used to generate enhanced code review comments.

3.3 Knowledge infusion through ME-regulated fine-tuning (MEFT)

This section provides MEFT details, including base model selection, loss function definition, and hyperparameter configuration.

Base model selection. Given the requirements of long COT reasoning, which necessitate extended context lengths and robust foundational model capabilities, we selected Qwen2.5-14B [37] - one of the most powerful open-source models under 20B parameters in the open-source community - as

our base model. This choice was constrained by computational resource limitations that precluded training larger models.

MEFT loss function definition. During the fine-tuning phase, we employed full-parameter fine-tuning to maximize the model's knowledge capacity for comprehensive review expertise integration. Conventional fine-tuning methods typically optimize the following objective:

$$\mathcal{L}_{LLM}(x) = \sum_{i=1}^{n} log P(x_i | x_{< i})$$
(3)

while $P(x_i|x_{< i})$ means the model predicts the i-th token conditioned on preceding tokens. While this approach enhances the model's ability to generate standard answers, it risks overfitting through incidental learning of task-irrelevant priors present in reference answers, thereby constraining adaptability to complex tasks. To address this limitation, we implemented MEFT using diversified answer distributions:

$$\mathcal{L}_{MEFT}(x) = \sum_{i=1}^{n} log P(\{x_{i,1}, x_{i,2}, \dots, x_{i,n}\} | x_{< i})$$
(4)

In general, this method enables the model to learn multiple solution pathways per problem, enhancing its generalization capabilities.

Hyperparameter configuration. For computational efficiency, we applied Flash Attention [6, 7] in the attention layers to reduce memory overhead from extended token sequences. Further memory optimization was achieved through DeepSpeed Zero3 [39], which offloads the optimizer's states and partial model parameters to system memory. The complete training process was conducted on 8×NVIDIA H800 80GB GPUs. Detailed hyperparameter configurations are provided in Table 2.

Table 2. Training hyperparameters

epochs	batch	lr	cutoff	warmup
2	64	1e-7	5000	500

3.4 Activating long COT reasoning in ACR

Existing research suggests that long COT improves the performance of LLMs by enabling them to engage in long reasoning processes and explore a broader range of considerations [5, 51, 55]. In the context of ACR, long COT mirrors a typical cognitive workflow of a human reviewer. First, they need to understand the implemented functionality and primary execution paths of the code under review. Then, they systematically examine the modifications introduced in the new submission to assess whether and how the changes impact the intended functionality or outcomes. After that, they conduct thorough analyses of potential code quality issues such as examining error (or exception) handling and edge case coverage, checking resource management (memory, connections, etc.), evaluating the correctness of API/dependency usage, and inspecting common vulnerability patterns [22]. So we constructed a long COT code review prompt to emulate this manual process, as illustrated in Figure 4.

4 Evaluation

To empirically validate the proposed *MelcotCR* approach, we formulate the following research questions:

111:10 Yu et al.

You are a Senior Software Engineer performing critical code review. You have 15+ years of experience in debugging complex systems and identifying subtle code defects. Approach this task with meticulous attention to detail and surgical precision.

Analyze the provided code to identify exact problem locations through systematic investigation. Follow this process:

1. Comprehensive Analysis

<think>

- Begin with code summary: Explain core functionality in 2-3 sentences
- Identify key code flows and critical execution paths
- Analyze recent diffs (if available) for regression patterns
- Examine error handling and edge case coverage
- Check resource management (memory, connections, etc.)
- Evaluate API/dependency usage correctness
- Cross-reference with common vulnerability patterns (OWASP, CWE)

</think>

2. Problem Localization

<location>

Paste EXACT problematic code snippet (1-5 lines maximum)

</location>

3. Structured Feedback

<comment>

- 3.1 Technical analysis:
 - Explain the nature of the defect
 - Reference specific language semantics/APIs involved
- 3.2 Root cause rationale:
 - Evidence supporting why this is problematic
 - Reference standards/violated best practices
- 3.3 Impact assessment:
 - Current consequences
 - Potential worst-case scenarios
- 3.4 Solution proposal:
 - Specific code-level fix
 - Alternative approaches (if applicable)
 - Recommended prevention patterns

</comment>

Example Output Structure: [ExampleOutputStruct]

Fig. 4. Prompt used to activate long COT reasoning in ACR

• RQ1: How effective is an LLM fine-tuned with *MelcotCR* in performing ACR tasks compared to existing methods?

- RQ1.1: How accurately does the fine-tuned model localize code issues?
- RQ1.2: How accurately do the review comments generated by the fine-tuned model describe the identified code issues?
- RQ2: What factors critically impact MelcotCR's effectiveness in fine-tuning LLMs for ACR tasks?

In particular, RQ1 aims to validate that fine-tuning an LLM with a unique reasoning ability to analyze multiple dimensions of code review by *MelcotCR* can lead to improved ACR performance in terms of its accuracy in both localizing code issues and describing the identified issues. RQ2 aims to provide a deep understanding of the factors in *MelcotCR* that play critical roles in improving ACR performance.

In this section, we first describe the experimental settings and then analyze the results to answer the research questions.

4.1 Experimental settings

Experimental settings include datasets, benchmark models, and evaluation metrics.

- 4.1.1 Datasets. The evaluation dataset is divided into two parts to evaluate the model performance in in-distribution and out-of-distribution review tasks, respectively. The in-distribution dataset consists of 1,000 randomly sampled test items from our curated MelcotCR dataset (the remaining data allocated to model training), assessing performance under the same distribution as the training data. The out-of-distribution dataset utilizes the CodeReviewer dataset [30]. As the collection time periods of CodeReviewer and MelcotCR datasets do not overlap, it is a reasonable benchmark dataset to evaluate the out-of-distribution performance of MelcotCR. As the CodeReviewer dataset lacks issue location annotations, it is only included in assessing the accuracy of review comments.
- 4.1.2 Benchmark models/approaches. Benchmark models/approaches are drawn from two primary sources. One category comprises models fine-tuned specifically for ACR tasks. To our knowledge, the only publicly known studies are Carllm [59], LLaMA-Reviewer [31], BitsAI-CR [45], and CodeReviewer [30]. However, we exclude CodeReviewer [30] and LLaMA-Reviewer [31] from our comparative analysis due to two fundamental limitations. First, their training protocol solely employs raw review comments without incorporating code location annotation capabilities. Second, Yu et al.'s work [59] revealed their substantial performance gaps compared to current state-of-the-art benchmarks. Meanwhile, although BitsAI-CR [45] is also designed to improve LLMs' ACR performance, it primarily concentrates on the data flywheel in enterprise environments. In addition, it utilizes internal corporate data and rules, making it impossible for us to conduct comparisons without a publicly available replication package.

The other category involves general models that are not fine-tuned for ACR tasks. Due to access restrictions, we cannot compare with advanced models like the GPT family. Instead, we selected accessible alternatives with comparable capability: Qwen 2.5 72B, QWQ 32B, and DeepSeek R1 671B. The benchmark models/methods are listed in Table 3.

Models/Approaches	Туре	Parameters	Research Questions
MelcotCR	Fine-tuned model	14B	RQ1, RQ2
Carllm	Fine-tuned model	14B	RQ1
Qwen 2.5 72B	General model	72B	RQ1
QWQ 32B	General model	32B	RQ1, RQ2
DeepSeek R1 (0120)	General model	671B	RQ1

Table 3. Models/Approaches in RQs

111:12 Yu et al.

4.1.3 Objective Metrics. We use the following objective metrics to evaluate the performance of various ACR approaches.

<u>Intersection over Union (IoU):</u> To evaluate code issue localization performance, we employ the *Intersection over Union (IoU)* metric [38, 40], which is calculated as follows:

$$IoU = \frac{label \cap predict}{label \cup predict}$$

where *label* denotes the set of code lines containing issues in the ground truth, and *predict* represents the set of code lines identified as problematic in the prediction. A higher *IoU* value signifies more precise issue localization, with the metric reaching its maximum value of 1 when the predicted and the ground-truth code segments completely overlap. This measurement effectively quantifies the spatial alignment between actual and detected code issues, providing an objective criterion for evaluating issue localization accuracy.

Hit Rate: This evaluation metric is calculated as follows:

$$Hit\ Rate = \frac{hit\ count}{total}$$

where *hit count* denotes the number of instances where the model-generated review comments align with the issues identified in the ground truth, and *total* represents the full test set size of 1,000 entries.

- 4.1.4 Human Evaluation Metrics. The human evaluation process involves a comprehensive examination of three elements for each entry: the original code submission, historical review comments, and LLM-generated feedback. This tripartite analysis aims to determine the practical value of the generated reviews. The assessment framework comprises two distinct tasks:
 - Accurate Identification of Historical Reviews: Requires the LLM-generated comments to describe identical issues to those specified in the ground truth.
 - *Provision of Valuable Review Feedback*: Mandates that the identified issues must objectively exist in the code samples without containing hallucinatory content.

Human evaluation was conducted by two senior students specializing in software engineering. From the complete test set, 540 entries were randomly sampled for manual assessment, ensuring a 95% confidence level with less than 3% margin of error. Each evaluator analyzed 300 entries, including 60 overlapping samples for consistency verification. The inter-rater reliability analysis yielded a Cohen's kappa [23] score of 0.8426, indicating substantial agreement. We also examined the consistency between LLM-as-judge [12, 26, 64] and human evaluation. The calculated kappa value was 0.5281, also indicating substantial agreement between human evaluation and LLM-as-judge evaluation.

To facilitate statistical measurement, analysis, and comparison, we define the following two metrics:

<u>Human Hit:</u> involves manually comparing the review comments generated by the model with the standard answers to determine whether the code issues identified by the model align with the ground-truth answers. The final result is calculated as the proportion of consistent cases out of the total number of evaluations.

$$Human\ Hit = \frac{human\ hit\ count}{total}$$

<u>Human Valuable</u>: involves manually reviewing the model-generated comments alongside the original code to assess whether the model has identified valuable code issues. Here, "valuable" is defined as the model pointing out genuine code problems, regardless of whether they match the

ground-truth answers. The final result is calculated as the proportion of valuable review comments out of the total number of evaluations.

$$Human\ Valuable = \frac{human\ valuable\ count}{total}$$

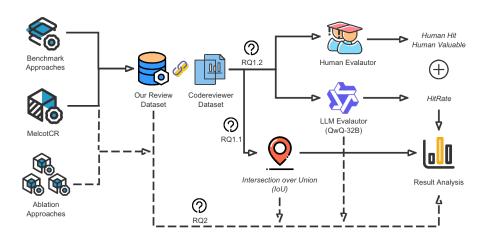


Fig. 5. The process for comparative evaluation of MelcotCR

4.1.5 The experimental evaluation process. Figure 5 depicts our comparative framework consisting of two primary components. For the first category concerning prior code review studies, we select the state-of-the-art baseline Carllm [59] as our benchmark. To ensure methodological parity, we replicate the experimental setup by maintaining identical model parameter scales, equivalent training data scope, and consistent implementation of full fine-tuning during model reproduction. The second comparison focuses on contemporary dominant LLMs utilizing prompt engineering for review generation. We choose QWQ-32B [46], Deepseek-R1 671B [13], and Qwen2.5 72B [37] as baseline references. During the evaluation process, we conduct iterative prompt optimisation to achieve optimal generation quality, with explicit instructions requiring models to prioritize the identification of critical issues to prevent low-quality review outputs.

RQ	Datasets	Methods	Evaluation methods	Metrics
1.1	MelcotCR	Carllm 14B MelcotCR 14B Qwen2.5 72B QWQ 32B Deepseek-R1 671B	automated	IoU
1.2	MelcotCR and CodeReviewer	Carllm 14B MelcotCR 14B Qwen2.5 72B QWQ 32B Deepseek-R1 671B	Human and automated	Hit Rate, Human Hit and Human Valuable
2	MelcotCR	MelcotCR 14B	automated	IoU and HitRate

Table 4. Experimental settings for RQs

The experiment for RQ1.1 Localizing code issues plays a crucial role in ACR as it helps developers pinpoint the exact locations of code issues, and improves both reviewing and fixing efficiencies. As

111:14 Yu et al.

shown in Table 4, the evaluation of code issue localization employs *IoU* for automated assessment. This metric provides a detailed evaluation of the distance between the positions of code issues identified by the model and the ground truth.

The experiment for RQ1.2 Code review comments exhibit diverse expression methods, which leads to significant bias when using similarity metrics such as BLEU [36] to evaluate the performance of review comments, as many studies have reported [21, 47]. As shown in Table 4, to avoid evaluation bias caused by expression differences, we employ a combination of LLM-as-Judge [12, 26, 64], commonly used for assessing complex semantic tasks, and manual evaluation to measure the accuracy of generated review comments. For LLM-as-Judge, we assess whether the generated review comments identify the same issues as the reference answers, using the prompt shown in Figure 6. When constructing the evaluation prompt, we compare the generated comments with the reference answers to determine the correctness of the review. By providing the correct answers, we guide the LLM to focus on the content of review comments rather than their expression, thereby significantly mitigating subjective bias in LLM evaluations.

Analyze the given review comment and the standard review comment. Determine if they highlight the same core issue(s), even if phrased differently. Focus on the substance of the critique, not exact wording. If they address identical problems (e.g., methodology flaws, data limitations, clarity issues), output Review Consistent. If they diverge in the issues raised (e.g., one critiques methodology while the other focuses on formatting), output Review Inconsistent.

Standard Review Comment: [StdRevCom]

Given Review Comment: [GivenRevCom]

Fig. 6. LLM-as-Judge prompt

The evaluation dataset has two parts: 1,000 samples randomly selected from the *MelcotCR* dataset to evaluate performance on in-distribution data and 1,000 entries randomly selected from the CodeReviewer dataset [30]. During data sampling, we select code languages consistent with our study to avoid potential bias from less common languages. As the CodeReviewer dataset cannot provide ground truth for code issue locations, we only conduct manual and automated evaluations of the accuracy of generated review comments when using this dataset. The automated evaluation employs the QWQ 32B model as the base model, which demonstrates performance comparable to Deepseek-R1 while being more computationally efficient.

The experiment for RQ2 The process of analyzing which factors in *MelcotCR* significantly impact its performance in fine-tuning LLMs for ACR tasks comprises two main parts. The first part examines the effect of ME. The second part explores the effect of the thought process. As shown in Table 4, we conduct ablation experiments on different factors through automated evaluation, focusing on two aspects: positional localization and accuracy of code review. When analyzing the impact of ME on ACR performance, we compare ME-regulated fine-tuning against conventional fine-tuning with the options of regular and long COT arrangements. To assess the influence of different thinking steps on the final ACR performance, we investigate the impact of each individual step on the overall task by removing specific step prompts.

4.2 Evaluation results

Following the evaluation settings and process elaborated in the previous section, this section analyzes the evaluation results.

	MalastCD Jataset			C-1	eReviewer	Jatanet	
		MelcotCR dataset					aataset
	IoU	Hit	Human	Human	Hit	Human	Human
	100	Rate	Hit	Valuable	Rate	Hit	Valuable
Carllm 14B	24.12	21.9	25.83	76.11	35.02	36.40	87.78
MelcotCR	07.16	05.4	00.70	01.77	26.56	20.17	00.00
(Qwen2.5 14B)	27.16	25.4	29.72	81.67	36.56	39.17	92.00
Qwen2.5 72B	10.26	18.8	15.56	73.05	27.62	29.17	75.28
QWQ 32B	11.46	22.7	20.28	78.33	25.94	25.28	71.39
Deepseek-R1 671B (0120)	12.17	25.2	26.11	83.61	38.38	40.56	86.11

Table 5. Performance of each method on the MelcotCR and CodeReviewer datasets

4.2.1 RQ1.1: Issue localization accuracy. Table 5 presents the IoU performance of different methods in positional localization on the MelcotCR and CodeReviewer datasets. The results confirm that the low-parameter Qwen2.5 14B base model, when fine-tuned with MelcotCR, significantly surpasses Carllm, the state-of-the-art model fine-tuned for ACR tasks so far, and all the other models, especially Qwen2.5 72B, the same series but with significantly more parameters and Deepseek-R1 671B, enabling developers to precisely identify problematic code segments. LLMs not specifically trained for ACR generally exhibit suboptimal performance. Manual investigation reveals distinct failure patterns between QWQ 32B and Deepseek-R1: QWQ 32B's lower score primarily stems from its inability to detect valid code issues, whereas Deepseek-R1 tends to over-expand localization scope, unnecessarily increasing code inspection workload for reviewers and consequently wasting their cognitive efforts.

4.2.2 RQ1.2: Issue description accuracy. Table 5 presents the results of both automated and human evaluations. For automated evaluation in terms of *Hit Rate*, *MelcotCR* outperforms all other methods, including Deepseek-R1 whose parameter size is an order of magnitude larger on the *MelcotCR* dataset. This advantage stems from the comprehensive knowledge infusion through MEFT and the analytical steps designed to emulate the human review process. As illustrated by a comparative example in Figure 7, *MelcotCR* exhibits stronger analytical depth and logical reasoning capabilities compared to Carllm using a conventional COT approach.

For human evaluation, *MelcotCR* outperforms all other methods, including Deepseek-R1, in accurately identifying code issues (in terms of *Human Hit*), which confirms its effectiveness. In terms of maintaining the factual grounding of review comments to ensure that identified issues truly exist in code snippets without hallucination (*Human Valuable*), *MelcotCR* surpasses all other methods including the state-of-the-art fine-tuned Carllm, confirming that combining ME-regulated fine-tuning with long COT reasoning leads to substantial improvement in code review capability. However, due to the significant parameter scale disparity between the base model and Deepseek-R1, *MelcotCR* slightly underperforms Deepseek-R1.

When comparing LLM-as-judge with human evaluation results, QwQ 32B exhibits significant evaluation bias when assessing its own responses, displaying a higher probability of considering its own answers correct. Our human and LLM-based evaluations serve as mutual cross-validation to more accurately reflect the performance of different methods in generating reviews. Comparing different approaches reveals that the long COT model demonstrates superior performance in the review task. This further underscores the necessity of applying long COT techniques to ACR.

111:16 Yu et al.

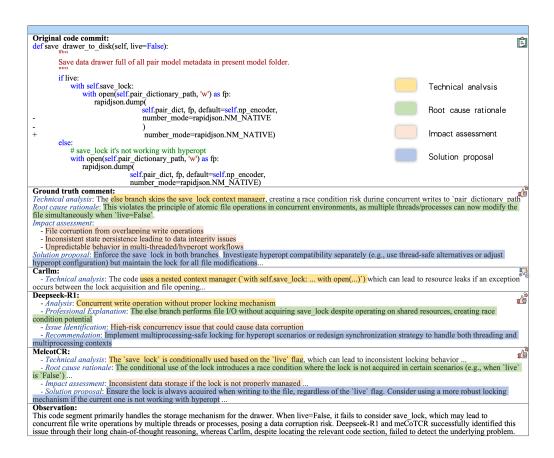


Fig. 7. The example comments generated by Carllm, MelcotCR, and Deepseek-R1

On the CodeReviewer dataset, *MelcotCR* performs comparably to Deepseek R1 on out-of-distribution data while significantly outperforming other methods. However, *MelcotCR* surpasses Deepseek R1 in generating useful review comments (*Human Valuable*).

4.2.3 RQ2: impacting factors. We examine factors influencing the performance of MelcotCR from two aspects of fine-tuning: fine-tuning methods and fine-tuning process steps.

Table 6. Effects of different fine-tuning methods and inferencing methods in MelcotCR

Fine-tuning Method	Inferencing Method	IoU	Hit Rate
Conventional	Regular COT	24.12	20.0
Conventional	Long COT	26.88	22.9
MEFT	Regular COT	26.60	20.3
WILL	Long COT	27.16	24.3

The influence of fine-tuning methods. As shown in Table 6, MEFT exhibits distinct effects in localizing code issues and generating review comments. For the task of localizing code issues, MEFT with regular COT can achieve the same performance as the conventional fine-tuning that adopts long COT. When MEFT adopts long COT, its performance is elevated significantly. Even with the conventional fine-tuning method, adopting long COT makes a significant difference. For the

task of generating precise review comments, MEFT does not yield significant improvement over conventional fine-tuning when using regular COT. In contrast, when employing long COT, MEFT demonstrates a notable advantage. As long COT incorporates additional outputs such as code analysis and change descriptions, it suggests that MEFT holds greater potential in enabling models to acquire deeper knowledge.

The influence of reasoning steps. As shown in Table 7, the experimental results indicate that 'diff analyze' has the greatest overall impact on correctly identifying code issue locations and generating accurate review comments, which aligns with the intuition that most code issues are triggered by code modifications [3, 42]. 'Summary' has the least influence, probably because summarization tasks are relatively simple, and high-level functional descriptions lack the granularity required to establish concrete connections with specific code defects, thus failing to provide sufficient contextual clues to improve accuracy in review tasks. 'Key code flows' significantly affect the correct generation of review comments, suggesting that effective code review generation requires not only localized inspection but also a comprehensive understanding of how components interact within the broader context.

Steps	IoU	Ratio	Hit Rate	Ratio
Full	27.16	-	24.3	-
- Summary	26.94	-0.81%	23.9	-1.64%
- Key code flows	26.67	-1.80%	22.3	-8.23%
- Diff anlyze	25.38	-6.55%	22.7	-6.58%
 Issue check 	26.32	-3.09%	23.1	-4.93%

Table 7. Effects of different reasoning steps in MelcotCR

5 Discussion

In this section, we discuss this work's implications for researchers and practitioners as well as its validity risks.

5.1 Methodological significance of MelcotCR

This study establishes three pivotal research directions for LLM-based ACR.

First, MEFT alleviates an issue in traditional supervised learning, that is, various biases are inadvertently introduced while acquiring knowledge. MEFT provides a new perspective: a shift from solely pursuing the best answer fitting to learning unbiased knowledge behind the answers.

Second, the long COT architecture establishes an extensible analytical framework for code review tasks. Unlike Yu et al.'s constrained three-phase decomposition of "localization-description-resolution" [59], *MelcotCR* enables dynamic integration of new analytical dimensions including code summarization, core logic analysis, diff change analysis, and potential issue identification. This innovation not only expands the cognitive depth of LLMs in code review scenarios but also reveals a new research paradigm: enhancing code review performance through systematic integration of diverse perspectives to both understand and analyze source code.

Finally, this work empirically demonstrates that algorithmic innovations can enable smaller-scale models, e.g., 14B parameters, to match or surpass the performance of significantly larger counterparts, e.g., 671B-parameter Deepseek-R1. This discovery illuminates a promising direction for further research – rather than continuously scaling up models, focusing on cognitive architecture enhancements could unlock the latent potential of existing models, thereby providing theoretical foundations for developing lightweight ACR systems.

111:18 Yu et al.

5.2 Implications for practiontioners

The superior performance of *MelcotCR* for ACR tasks demonstrates that incorporating multiple dimensions of information to facilitate the identification of issues in code by LLMs can yield positive gains. Through ME-regulated fine-tuning, we stimulate the model's long COT capability to effectively integrate multiple dimensions of information. This not only paves the way for the future evolution of ACR models but also leaves ample exploration space, for example, investigating whether introducing more dimensions of information or expanding the number of paraphrased review comments (this study used 10 semantically equivalent variants) could further enhance the model's deep reasoning capacity. We invite interested researchers to reproduce our work and explore this direction by making our replication package available at https://anonymous.4open.science/r/MelcotCR.

It is also worth noting that the reasoning process of *MelcotCR* is inspired by the human review process and hence the *MelcotCR* approach is inherently friendly to human-AI collaboration in such a way that the tool's COT reasoning process provides auxiliary support for human reviewers to rapidly comprehend code functionality, core logic, critical modifications, and so on. This structured analytical approach assists reviewers in efficiently parsing code implementations, thereby creating additional efficiency gains in the code review workflow.

5.3 Threats to Validity

We are conscious that the following factors may introduce risks to the validity of the study's findings.

<u>Limited LLMs.</u> Due to computational resource constraints, we experimented only with representative LLMs under restricted parameter configurations (less than 20B), specifically Qwen2.5-14B. Given the rapid evolution of LLM research, the broader community continues to explore the performance of diverse LLMs. Consequently, newer or larger models may outperform those evaluated in this work. In fact, existing studies have confirmed that long COT yields positive effects in certain scenarios only when the base model possesses sufficient parameters [28]. However, the timing and environmental factors of model training limited our initial exploration to Qwen2.5-14B. Future work could investigate newer iterations such as Qwen3. We are open to experimenting with newer models.

<u>Noisy Data.</u> Noise in the dataset may arise from multiple sources. First, while low-quality review comments in open-source communities were largely filtered out through semantic analysis, residual low-value data could still persist, potentially affecting model training. To mitigate this, we implemented manual screening and rule-based filtering to eliminate invalid reviews. Empirical results demonstrate that our approach maintains robust performance, suggesting minimal impact from residual noise.

<u>LLM Evaluation Bias.</u> Prior studies have identified inherent biases in LLM-based evaluation, particularly in scoring tasks where models tend to favor content stylistically similar to their own outputs [25]. In this work, we adopt a contrastive approach to directly compare reference answers with model-generated outputs. By establishing explicit evaluation rules and objectives, our method significantly reduces model-specific biases.

<u>Human Evaluation Bias.</u> Despite adopting standardized protocols and consistency checks, human evaluation can still introduce errors. To address this, we recruited evaluators with a software engineering background to ensure domain expertise in assessing the alignment between source code and review comments. We also conducted Cohen's kappa coefficient analysis to measure inter-rater reliability and permitted moderated discussions among evaluators to resolve contentious cases, further minimizing this risk.

<u>Prompt Limitations</u>. While iterative experimentation was carried out to optimize prompt design, there may exist untested prompts capable of delivering superior performance. We encourage researchers to experiment with alternative prompts to explore further improvements.

Reasoning Steps. It is generally believed that increasing COT length has a positive correlation with improved model performance. In this work, we enhanced COT reasoning length and achieved better performance through MEFT and explicitly specifying the model's reasoning path. It is worth validating whether increasing the reasoning length further can lead to even better performance.

10 Variations for Maximum Entropy. Due to computational constraints, we provide only 10 variants for review comments, attempting to capture knowledge embedded in different pathways leading to the final answer – correct review comments. However, the value 10 may not be necessarily optimal and it is worth exploring different values. We conjecture that increasing the number of variants would likely yield greater benefits, although an upper limit should exist to balance between the cost of resources and the performance gain. This awaits verification in subsequent research.

6 Conclusions

In this work, we propose *MelcotCR*, an approach that combines maximum-entropy-regulated fine-tuning and long chain-of-thought reasoning to optimize LLMs for ACR tasks by emulating the cognitive process of human code reviewers. Experimental results demonstrate that this method outperforms previous state-of-the-art approaches in both position localization and review comment generation in ACR tasks. With 14B parameters, it attains performance on par with Deepseek-R1 671B, one of the current strongest open-source long-chain reasoning models. Our future work includes leveraging online reinforcement learning to further enhance the review capabilities of LLMs, and developing context exploration methods for ACR systems to advance intelligent code review capabilities.

References

- [1] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. 2017. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine* 34, 6 (2017), 26–38.
- [2] Amiangshu Bosu and Jeffrey C Carver. 2013. Impact of peer code review on peer impression formation: A survey. In 2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. IEEE, 133–142.
- [3] Amiangshu Bosu, Jeffrey C Carver, Munawar Hafiz, Patrick Hilley, and Derek Janni. 2014. Identifying the characteristics of vulnerable code changes: An empirical study. In *Proceedings of the 22nd ACM SIGSOFT international symposium on foundations of software engineering*. 257–268.
- [4] Junkai Chen, Zhenhao Li, Qiheng Mao, Xing Hu, Kui Liu, and Xin Xia. 2025. Understanding Practitioners' Expectations on Clear Code Review Comments. *Proceedings of the ACM on Software Engineering* 2, ISSTA (2025), 1257–1279.
- [5] Qiguang Chen, Libo Qin, Jinhao Liu, Dengyun Peng, Jiannan Guan, Peng Wang, Mengkang Hu, Yuhang Zhou, Te Gao, and Wanxiang Che. 2025. Towards reasoning era: A survey of long chain-of-thought for reasoning large language models. arXiv preprint arXiv:2503.09567 (2025).
- [6] Tri Dao. 2024. FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning. In *International Conference on Learning Representations (ICLR)*.
- [7] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. In Advances in Neural Information Processing Systems (NeurIPS).
- [8] Alaa M El-Halees. 2015. Arabic text classification using maximum entropy. IUG Journal of Natural Studies 15, 1 (2015).
- [9] Guhao Feng, Bohang Zhang, Yuntian Gu, Haotian Ye, Di He, and Liwei Wang. 2023. Towards revealing the mystery behind chain of thought: a theoretical perspective. Advances in Neural Information Processing Systems 36 (2023), 70757–70798.
- [10] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. 2020. Codebert: A pre-trained model for programming and natural languages. arXiv preprint arXiv:2002.08155 (2020).

111:20 Yu et al.

[11] Georgios Gousios, Martin Pinzger, and Arie van Deursen. 2014. An exploratory study of the pull-based software development model. In *Proceedings of the 36th international conference on software engineering*. 345–355.

- [12] Jiawei Gu, Xuhui Jiang, Zhichao Shi, Hexiang Tan, Xuehao Zhai, Chengjin Xu, Wei Li, Yinghan Shen, Shengjie Ma, Honghao Liu, et al. 2024. A survey on llm-as-a-judge. arXiv preprint arXiv:2411.15594 (2024).
- [13] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. arXiv preprint arXiv:2501.12948 (2025).
- [14] Anshul Gupta and Neel Sundaresan. 2018. Intelligent code reviews using deep learning. In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'18) Deep Learning Day.
- [15] Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. 2017. Reinforcement learning with deep energy-based policies. In *International conference on machine learning*. PMLR, 1352–1361.
- [16] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*. Pmlr, 1861–1870.
- [17] Yang Hong, Chakkrit Tantithamthavorn, Patanamon Thongtanunam, and Aldeida Aleti. 2022. Commentfinder: a simpler, faster, more accurate code review comments recommendation. In Proceedings of the 30th ACM joint European software engineering conference and symposium on the foundations of software engineering. 507–519.
- [18] Abraham Ittycheriah and Salim Roukos. 2005. A maximum entropy word aligner for arabic-english machine translation. In Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing. 89–96.
- [19] Edwin T Jaynes. 1957. Information theory and statistical mechanics. Physical review 106, 4 (1957), 620.
- [20] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. 1996. Reinforcement learning: A survey. Journal of artificial intelligence research 4 (1996), 237–285.
- [21] Tom Kocmi, Vilém Zouhar, Christian Federmann, and Matt Post. 2024. Navigating the metrics maze: Reconciling score magnitudes and accuracies. arXiv preprint arXiv:2401.06760 (2024).
- [22] Oleksii Kononenko, Olga Baysal, and Michael W Godfrey. 2016. Code review quality: How developers see it. In *Proceedings of the 38th international conference on software engineering*. 1028–1038.
- [23] Tarald O Kvålseth. 1989. Note on Cohen's kappa. Psychological reports 65, 1 (1989), 223-226.
- [24] Sergey Levine. 2018. Reinforcement learning and control as probabilistic inference: Tutorial and review. arXiv preprint arXiv:1805.00909 (2018).
- [25] Dawei Li, Renliang Sun, Yue Huang, Ming Zhong, Bohan Jiang, Jiawei Han, Xiangliang Zhang, Wei Wang, and Huan Liu. 2025. Preference Leakage: A Contamination Problem in LLM-as-a-judge. (2025). arXiv:2502.01534 [cs.LG] https://arxiv.org/abs/2502.01534
- [26] Haitao Li, Qian Dong, Junjie Chen, Huixue Su, Yujia Zhou, Qingyao Ai, Ziyi Ye, and Yiqun Liu. 2024. Llms-as-judges: a comprehensive survey on llm-based evaluation methods. arXiv preprint arXiv:2412.05579 (2024).
- [27] Jia Li, Ge Li, Yongmin Li, and Zhi Jin. 2025. Structured Chain-of-Thought Prompting for Code Generation. ACM Trans. Softw. Eng. Methodol. 34, 2, Article 37 (Jan. 2025), 23 pages. doi:10.1145/3690635
- [28] Yuetai Li, Xiang Yue, Zhangchen Xu, Fengqing Jiang, Luyao Niu, Bill Yuchen Lin, Bhaskar Ramasubramanian, and Radha Poovendran. 2025. Small models struggle to learn from strong reasoners. arXiv preprint arXiv:2502.12143 (2025).
- [29] Zhiyu Li, Shuai Lu, Daya Guo, Nan Duan, Shailesh Jannu, Grant Jenks, Deep Majumder, Jared Green, Alexey Svy-atkovskiy, Shengyu Fu, et al. 2022. Automating code review activities by large-scale pre-training. In Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 1035–1047.
- [30] Zhiyu Li, Shuai Lu, Daya Guo, Nan Duan, Shailesh Jannu, Grant Jenks, Deep Majumder, Jared Green, Alexey Svyatkovskiy, Shengyu Fu, et al. 2022. Automating code review activities by large-scale pre-training. In Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 1035–1047.
- [31] Junyi Lu, Lei Yu, Xiaojia Li, Li Yang, and Chun Zuo. 2023. LLaMA-Reviewer: Advancing code review automation with large language models through parameter-efficient fine-tuning. In 2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE). IEEE, 647–658.
- [32] Kamal Nigam, John Lafferty, and Andrew McCallum. 1999. Using maximum entropy for text classification. In IJCAI-99 workshop on machine learning for information filtering, Vol. 1. Stockholom, Sweden, 61–67.
- [33] Kaiwen Ning, Jiachi Chen, Jingwen Zhang, Wei Li, Zexu Wang, Yuming Feng, Weizhe Zhang, and Zibin Zheng. 2024. Defining and Detecting the Defects of the Large Language Model-based Autonomous Agents. arXiv preprint arXiv:2412.18371 (2024).
- [34] Yu Nong, Mohammed Aldeen, Long Cheng, Hongxin Hu, Feng Chen, and Haipeng Cai. 2024. Chain-of-thought prompting of large language models for discovering and fixing software vulnerabilities. arXiv preprint arXiv:2402.17230 (2024).

[35] Franz Josef Och and Hermann Ney. 2002. Discriminative training and maximum entropy models for statistical machine translation. In *Proceedings of the 40th Annual meeting of the Association for Computational Linguistics*. 295–302.

- [36] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In Proceedings of the 40th annual meeting of the Association for Computational Linguistics. 311–318.
- [37] Qwen, ;, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. 2025. Qwen2.5 Technical Report. arXiv:2412.15115 [cs.CL] https://arxiv.org/abs/2412.15115
- [38] Md Atiqur Rahman and Yang Wang. 2016. Optimizing intersection-over-union in deep neural networks for image segmentation. In *International symposium on visual computing*. Springer, 234–244.
- [39] Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 2020. DeepSpeed: System Optimizations Enable Training Deep Learning Models with Over 100 Billion Parameters. In Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (Virtual Event, CA, USA) (KDD '20). Association for Computing Machinery, New York, NY, USA, 3505–3506. doi:10.1145/3394486.3406703
- [40] Hamid Rezatofighi, Nathan Tsoi, JunYoung Gwak, Amir Sadeghian, Ian Reid, and Silvio Savarese. 2019. Generalized intersection over union: A metric and a loss for bounding box regression. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 658–666.
- [41] Guoping Rong, Yongda Yu, Yifan Zhang, He Zhang, Haifeng Shen, Dong Shao, Hongyu Kuang, Min Wang, Zhao Wei, Yong Xu, et al. 2024. Distilling Quality Enhancing Comments from Code Reviews to Underpin Reviewer Recommendation. *IEEE Transactions on Software Engineering* (2024).
- [42] Caitlin Sadowski, Emma Söderberg, Luke Church, Michal Sipko, and Alberto Bacchelli. 2018. Modern code review: a case study at google. In *Proceedings of the 40th international conference on software engineering: Software engineering in practice*. 181–190.
- [43] John Schulman, Xi Chen, and Pieter Abbeel. 2017. Equivalence between policy gradients and soft q-learning. arXiv preprint arXiv:1704.06440 (2017).
- [44] Shu-Ting Shi, Ming Li, David Lo, Ferdian Thung, and Xuan Huo. 2019. Automatic code review by learning the revision of source code. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 4910–4917.
- [45] Tao Sun, Jian Xu, Yuanpeng Li, Zhao Yan, Ge Zhang, Lintao Xie, Lu Geng, Zheng Wang, Yueyan Chen, Qin Lin, et al. 2025. BitsAI-CR: Automated Code Review via LLM in Practice. arXiv preprint arXiv:2501.15134 (2025).
- [46] Qwen Team. 2025. QwQ-32B: Embracing the Power of Reinforcement Learning. https://qwenlm.github.io/blog/qwq-32b/
- [47] Ngoc Tran, Hieu Tran, Son Nguyen, Hoan Nguyen, and Tien Nguyen. 2019. Does BLEU score work for code migration?. In 2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC). IEEE, 165–176.
- [48] Rosalia Tufano, Luca Pascarella, Michele Tufano, Denys Poshyvanyk, and Gabriele Bavota. 2021. Towards automating code review activities. In 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE). IEEE, 163–174.
- [49] Jiaan Wang, Fandong Meng, Yunlong Liang, and Jie Zhou. 2024. Drt-o1: Optimized deep reasoning translation via long chain-of-thought. arXiv e-prints (2024), arXiv-2412.
- [50] Yue Wang, Weishi Wang, Shafiq Joty, and Steven CH Hoi. 2021. Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. arXiv preprint arXiv:2109.00859 (2021).
- [51] Yaoting Wang, Shengqiong Wu, Yuecheng Zhang, Shuicheng Yan, Ziwei Liu, Jiebo Luo, and Hao Fei. 2025. Multimodal chain-of-thought reasoning: A comprehensive survey. *arXiv preprint arXiv:2503.12605* (2025).
- [52] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. Advances in neural information processing systems 35 (2022), 24824–24837.
- [53] Ratnadira Widyasari, Ting Zhang, Abir Bouraffa, Walid Maalej, and David Lo. 2023. Explaining explanation: An empirical study on explanation in code reviews. arXiv preprint arXiv:2311.09020 (2023).
- [54] Yuyang Wu, Yifei Wang, Ziyu Ye, Tianqi Du, Stefanie Jegelka, and Yisen Wang. 2025. When more is less: Understanding chain-of-thought length in llms. arXiv preprint arXiv:2502.07266 (2025).
- [55] Yu Xia, Rui Wang, Xu Liu, Mingyan Li, Tong Yu, Xiang Chen, Julian McAuley, and Shuai Li. 2024. Beyond chain-of-thought: A survey of chain-of-x paradigms for llms. arXiv preprint arXiv:2404.15676 (2024).
- [56] Lanxin Yang, Jinwei Xu, Yifan Zhang, He Zhang, and Alberto Bacchelli. 2023. Evacro: Evaluating code review comments. In Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 275–287.
- [57] Edward Yeo, Yuxuan Tong, Morry Niu, Graham Neubig, and Xiang Yue. 2025. Demystifying long chain-of-thought reasoning in llms. arXiv preprint arXiv:2502.03373 (2025).

111:22 Yu et al.

[58] Edward Yeo, Yuxuan Tong, Morry Niu, Graham Neubig, and Xiang Yue. 2025. Demystifying long chain-of-thought reasoning in llms, 2025. *URL https://arxiv. org/abs/2502.03373* (2025).

- [59] Yongda Yu, Guoping Rong, Haifeng Shen, He Zhang, Dong Shao, Min Wang, Zhao Wei, Yong Xu, and Juhong Wang. 2024. Fine-tuning large language models to improve accuracy and comprehensibility of automated code review. ACM transactions on software engineering and methodology 34, 1 (2024), 1–26.
- [60] Yongda Yu, Lei Zhang, Guoping Rong, Haifeng Shen, Jiahao Zhang, Haoxiang Yan, Guohao Shi, Dong Shao, Ruiqi Pan, Yuan Li, et al. 2024. Distilling Desired Comments for Enhanced Code Review with Large Language Models. arXiv preprint arXiv:2412.20340 (2024).
- [61] Ruohong Zhang, Bowen Zhang, Yanghao Li, Haotian Zhang, Zhiqing Sun, Zhe Gan, Yinfei Yang, Ruoming Pang, and Yiming Yang. 2024. Improve vision language model chain-of-thought reasoning. arXiv preprint arXiv:2410.16198 (2024).
- [62] Yiheng Zhang, Yunkang Cao, Xiaohao Xu, and Weiming Shen. 2024. Logicode: an llm-driven framework for logical anomaly detection. *IEEE Transactions on Automation Science and Engineering* (2024).
- [63] Zhuosheng Zhang, Aston Zhang, Mu Li, and Alex Smola. 2022. Automatic chain of thought prompting in large language models. arXiv preprint arXiv:2210.03493 (2022).
- [64] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. Advances in neural information processing systems 36 (2023), 46595–46623.