Direct Repair Optimization: Training Small Language Models For Educational Program Repair Improves Feedback

Charles Koutcheme, Nicola Dainese and **Arto Hellas**
Aalto University, Espoo, Finland
first.last@aalto.fi

Abstract

Locally deployed Small Language Models (SLMs) offer a promising solution for providing timely and effective programming feedback to students learning to code. However, SLMs often produce misleading or hallucinated feedback, limiting their reliability in educational settings. Current approaches for improving SLM feedback rely on existing human annotations or LLM-generated feedback. This paper addresses a fundamental challenge: Can we improve SLMs' feedback capabilities without relying on human or LLM-generated annotations? We demonstrate that training SLMs on the proxy task of program repair is sufficient to enhance their ability to generate high-quality feedback. To this end, we introduce Direct Repair Optimization (DRO), a self-supervised online reinforcement learning strategy that trains language models to reason about how to efficiently fix students' programs. Our experiments, using DRO to fine-tune LLaMA-3.1–3B and Qwen-2.5–3B on a large-scale dataset of Python submissions from real students, show substantial improvements on downstream feedback tasks. We release our code to support further research in educational feedback and highlight promising directions for future work.

Code: github.com/KoutchemeCharles/rlpf

1 Introduction

Learning to program remains challenging for many students, despite advances in teaching methodologies (Luxton-Reilly et al., 2018; Vihavainen et al., 2014). A key part of addressing these challenges is providing timely and accurate feedback (Jeuring et al., 2022), which is crucial for learning (Hattie and Timperley, 2007). Large Language Models (LLMs) such as GPT-4 have shown exceptional success in that task (Lohr et al., 2025), leading to their growing adoption in classrooms (Ahmed et al., 2025; Wang et al., 2024; Vadaparty et al., 2024; Liu et al., 2024a; Liffiton et al., 2024).

However, reliance on vendor-hosted LLMs raises substantial privacy concerns and potential ethical issues related to institutional control (Das et al., 2025). The privacy issues, jointly with scalability issues and associated costs, are driving a growing shift towards leveraging smaller open-source models (SLMs), which can be deployed and run locally within educational institutions or on students' computers and browsers (Yu et al., 2025b; Liu et al., 2024b).

However, smaller language models tend to produce misleading or hallucinated feedback, potentially confusing learners and negatively impacting learning (Koutcheme et al., 2025). Current methods for enhancing SLMs typically rely on supervised learning (Kotalwar et al., 2024)or reinforcement learning from either human annotations (Woodrow et al., 2025) or synthetic data generated by larger models (Ashok Kumar and Lan, 2024). These strategies come with limitations: human annotations are difficult to scale and replicate, while synthetic data inherits biases and capabilities from general-purpose LLMs.

Addressing these limitations raises a broader challenge for the field of programming education: *can we improve small language models' abilities to generate meaningful programming feedback without relying on external LLMs or human annotations?* Exploring this question opens a promising research direction focused on assessing how effectively small models can perform when trained exclusively on educational data. Understanding this can yield insights into the inherent capabilities of these models, free from external biases (DeepSeek-AI, 2025). This question matters not just technically, but also pedagogically; training models exclusively grounded in student work and learning contexts could facilitate more adaptable classroom deployment, enhance institutional control over model behaviour, and mitigate privacy concerns (Das et al., 2025).

Recent reinforcement learning techniques such as Group Relative Preference Optimization (GRPO) (Shao et al., 2024) have shown great promise in improving language models' reasoning capabilities, allowing relatively small models to reach the performance of significantly larger ones with minimal LLM supervision (DeepSeek-AI, 2025). In parallel, prior work reveals a strong correlation between language models' abilities to generate programming feedback and their capacity for fixing students' programs (Koutcheme et al., 2024a): models proficient in program repair are independently good at generating feedback.

Building on this insight, we hypothesise that improving a small language model's repair capabilities through reasoning could be sufficient to enhance the model's feedback-generation abilities. We argue that the reasoning needed for generating a repair involves a thinking process useful to provide students with feedback, much like how teaching assistants reason about students' mistakes before giving advice (Koutcheme, 2022).

Our paper thus aims to answer the following research question:

(RQ) How effective is training small language models to reason about educational program repair for improving their ability to generate high-quality programming feedback, and how does this technique compare to training models with LLM supervision?

To address this question, we introduce Direct Repair Optimization (DRO). This reinforcement learning training pipeline, based on Group Relative Preference Optimization, leverages historical datasets of student submissions to fine-tune small language models for program repair, using unit test results and syntactic/semantic distance measures as rewards to guide learning.

We apply DRO to fine-tune LLaMA-3.1-3B (Dubey et al., 2024) and Qwen-2.5-3B (Hui et al., 2024) on a large-scale dataset of student code from introductory programming courses (de Freitas et al., 2023). We evaluate the resulting model on multiple feedback tasks — including bug explanations, patch descriptions, and next-step hints — and show consistent improvements over all feedback criteria. Our contributions are as follows:

- We introduce a new LLM-free training pipeline for feedback generation based on program repair and reinforcement learning.
- We show that reasoning about program repair transfers to various forms of feedback, including explanations, fixes, and hints.
- We further demonstrate that reasoning about program repair improves the performance of LLM-distilled models.
- We release our code and data processing pipeline to support future research in aligning language models for educational feedback[1].

2 Related work

2.1 Programming Feedback

Program repair. Program repair has long been a cornerstone of AI-driven programming education, serving as a foundation for generating actionable feedback, such as next-step hints, through Intelligent Tutoring Systems (Rivers and Koedinger, 2017). Before the rise of instruction-tuned and chat language models, much of the work in this domain focused on leveraging closed pre-trained models, such as OpenAI Codex, to generate repairs through zero- or few-shot prompting. These approaches often relied on historical student submissions, automated unit tests, and other contextual information to guide the repair generation process (Zhang et al., 2022; Joshi et al., 2023).

In parallel, open-source language models were also explored for program repair tasks. For example, prior efforts have fine-tuned such models using datasets derived from student submissions (Koutcheme et al., 2023b) and automated repair tools (Koutcheme, 2023). These works demonstrated the viability of repair-focused training but did not directly explore its implications for improving natural language feedback.

Using program repair for improving feedback. Even with the advent of chat models, several works proposed leveraging the quality of repairs generated alongside feedback as a validation mechanism to ensure only relevant suggestions reach learners (Phung et al., 2024; Sahai et al., 2023). In parallel, other studies propose generating a high-quality candidate repair program as a reasoning step to generate higher-quality feedback. (Phung et al., 2023; Sahai et al., 2023). This strategy has been extended with success to distil LLM-generated repair-induced feedback to small language models via Supervised Fine-tuning (Kotalwar et al., 2024).

[1] github.com/KoutchemeCharles/rlpf

Reinforcement learning from human and AI feedback. Supervised Fine-Tuning (SFT) methods are limited in their ability to align language models with nuanced human objectives (Ouyang et al., 2022). Several works have explored reinforcement learning techniques, such as Direct Preference Optimization (DPO) (Rafailov et al., 2024), to train small language models to generate higher-quality feedback. For example, this has been done by leveraging teaching assistants' (TAs) edits of their responses to student forum questions (Hicke et al., 2023), collecting live TA preferences over several model-generated feedback (Woodrow et al., 2025), or combining existing high-quality human annotations with AI-generated alternatives (Ashok Kumar and Lan, 2024).

However, these methods typically rely on human supervision or the existence of labeled human feedback. This reliance poses challenges in contexts where such annotations are scarce or unavailable. While full Reinforcement Learning with AI Feedback (RLAIF) (Lee et al., 2024) approaches have been theorized to work well in the programming domain (Scarlatos et al., 2024), in this paper, we explore whether we can bootstrap feedback capabilities in small language models without requiring any human or LLM involvement.

2.2 Improving SLM Reasoning Without LLMs

Recent work has proposed leveraging automatically evaluable tasks to define preference pairs for DPO optimization (Pang et al., 2024), replacing the need for human or LLM judgments. In domains like programming and math, where correctness can be verified programmatically, this strategy has shown promising results. Combined with large-scale sampling and chain-of-thought prompting (Wei et al., 2022), such methods have yielded substantial improvements with minimal supervision (Pang et al., 2024).

Most recently, a new line of alignment techniques (Shao et al., 2024; Liu et al., 2025; Yu et al., 2025a) takes the idea back to a fully online optimization paradigm, showing major improvements. Inspired by the success of small models such as DeepScaleR (Luo et al., 2025), we explore whether these reinforcement learning techniques can be adapted to improve small language models in programming education.

3 Methodology

In this work, we hypothesise that training small language models for program repair will improve their feedback ability. To validate this hypothesis, we propose Direct Repair Optimization.

3.1 Background

Before presenting our approach, we first describe the setup and assumptions underlying our work.

3.1.1 Environment

We assume a typical educational programming setting, where student submissions are regularly collected and evaluated using automated assessment tools, such as unit tests, to assign scores and provide feedback (Paiva et al., 2022). Leveraging this infrastructure, we make two key assumptions: first, we assume access to a training dataset $\mathcal{D} = \{(d^i, s^i, c^i)\}_{i=1}^N$, comprising N tuples, where each tuple consists of a problem description d^i, a corresponding student program s^i, and a correctness label c^i, with $c^i = 0$ indicating an incorrect program and $c^i = 1$ indicating a correct one; second, we assume the availability of a grading function $u(s)$ that assigns a normalized score $c^i \in [0, 1]$ to each program s^i, reflecting its functional correctness based on unit test results.

3.1.2 Definition: high-quality repair

To support the rest of this article, we formalize the concept of a high-quality repair. A repair is typically considered high-quality if it meets two key criteria: functional correctness and closeness to the student's original incorrect program (Koutcheme et al., 2024c; Phung et al., 2023; Joshi et al., 2023; Zhang et al., 2022). Functional correctness ensures that the repair successfully resolves the intended issues, while closeness ensures the repair preserves the student's original approach, making the solution more interpretable and educationally meaningful (Price et al., 2017). Given a candidate repair $\mathcal{R}$ (generated by an LM) for an incorrect program s_i, we assess its quality using automated evaluation methods. Functional correctness is measured through unit test results provided by the grading function u. For closeness, we use ROUGE (Lin, 2004), as it has been shown to be an effective and efficient measure for selecting high-quality repairs (Koutcheme et al., 2023a).

3.2 Direct Repair Optimization

In this section, we introduce Direct Repair Optimization (DRO), our approach to improve language models' ability to generate educationally meaningful program repairs. DRO is an online reinforcement learning training method based on variants of Group Relative Preference Optimization (Shao et al., 2024). Figure 1 shows an overview of the method. At each iteration, given an incorrect program s^i, we (1) generate several completions, (2) compute individual reward scores, based on such generations and (3) update the model parameters based both on such rewards and a divergence score. Below we detail each step.

3.2.1 Sampling answers

Given an incorrect program, we sample G generations from our language model $\mathcal{G}_1^i, \mathcal{G}_2^i, \ldots, \mathcal{G}_g^i \sim \pi_\theta(s^i, d^i)$. Our prompt asks our model to fix the student's program but to reflect thoroughly before providing an answer (see Figure 2, Appendix C). Each generation contains a thought $\mathcal{T}_l^i$ and a final repair $\mathcal{R}_g^i$: $\mathcal{G}^i = (\mathcal{T}_g^i, \mathcal{R}_g^i)$. Following (Shao et al., 2024), our prompt imposes the language model to structure its response using a set of predefined tags: the thought pattern should be generated within `<think> . . . </think>` and the repair within `<answer>. . . </answer>` tags.

3.2.2 Computing rewards

For each generation $\mathcal{G}_g^i$, we compute a reward r_g^i, reflecting the two criteria that define a high-quality repair (see Section 3.1.2): functional correctness and closeness (or "proximity") to the student's original incorrect program. The total reward r_g^i is thus the sum of two separate components: $r_g^i = f_g^i + p_g^i$.

The functional reward is an outcome-based reward (Luo et al., 2025) computed by extracting $\mathcal{R}_l^i$ and passing it through the available grading function (i.e., unit tests):

$$f_g^i = \begin{cases} +1.0 & \text{if } u(\mathcal{R}_g^i) = 1 \\ +0.5 & \text{if } u(\mathcal{R}_g^i) - u(s^i) > 0 \\ -1.0 & \text{if } \mathcal{R}_g^i \text{ not compiling} \\ 0 & \text{otherwise} \end{cases}$$

We reward fully correct repairs (+1.0), give partial credit (+0.5) if the repair improves upon the student's original program (i.e., it passes more tests than s^i), and penalize repairs that fail to compile or generations that do not follow the expected format (–1.0). Our reward encourages the model to make meaningful progress toward correctness, even when it cannot fully solve the task. We believe that partially correct repairs that are better than the student's original work could also benefit feedback generation.

The closeness reward evaluates how well the generated repair aligns with the student's original code:

$$p_g^i = \begin{cases} \text{ROUGE}(s^i, \mathcal{R}_l^i) & \text{if } u(\mathcal{R}_l^i) = 1 \\ 0 & \text{otherwise} \end{cases}$$

By integrating this reward, we encourage the model not only to solve the programming task but to do so by building on the student's own approach, implicitly forcing reasoning about what the student is currently doing and trying to achieve. This makes the repair (and the resulting feedback) more pedagogically aligned. We provide this reward only when the repair is fully correct. Since repairing a student program inherently requires changes, correctness and closeness can become competing objectives. Rewarding both simultaneously for partial outputs would risk destabilising training.

3.2.3 Updating the model using the rewards

We update the model using the computed rewards with the dr.GRPO loss function (Liu et al., 2025), a recent reinforcement learning loss function designed for training stability and efficiency:

$$\mathcal{J}_{\text{dr.GRPO}} = -\frac{1}{LG} \sum_{g=1}^{G} \sum_{t=1}^{|o_g|} l_{m,t} \tag{1}$$

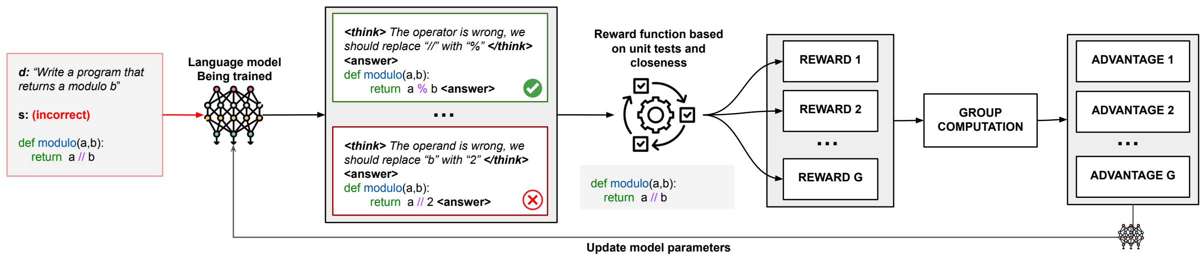
where

$$l_{g,t} = \frac{\pi_\theta(o_{g,t} \mid d^i, s^i, o_{g,<t})}{\pi_{\theta_{\text{old}}}(o_{g,t} \mid d^i, s^i, o_{g,<t})} \hat{A}_g$$

and

$$\hat{A}_g = (r_g^i - \bar{r})$$

Here, LG is the maximum allowed completion length, $\pi_{\theta_{\text{old}}}$ is the model before the current update, and $\hat{A}_g$ is the advantage, computed as the reward deviation from the batch mean $\bar{r}$. In practice, we use a clipped surrogate version of this objective that accounts for multiple updates per generation. For clarity and space, we provide the full objective and implementation details in section A.1 (Appendix A, where we also show how this formulation differs from the original GRPO loss introduced in the DeepSeek paper (DeepSeek-AI, 2025) and how it is better adapted to our task.



d: "Write a program that returns a modulo b"

s: (incorrect)

```
def modulo(a,b):
    return a // b
```

```
<think> The operator is wrong, we should replace "//" with "%" </think>
<answer>
def modulo(a,b):
    return a % b <answer>
```

...

```
<think> The operand is wrong, we should replace "b" with "2" </think>
<answer>
def modulo(a,b):
    return a // 2 <answer>
```

```
def modulo(a,b):
    return a // b
```

Figure 1: **Overview of Direct Repair Optimization.**

4 Experiments

In this section, we present the experiments conducted to answer our research question.

4.1 Falconcode: A Real-life Dataset

To answer our research question, we train language models using FalconCode (de Freitas et al., 2023), a publicly available dataset containing real-life CS1 students' solutions to many Python programming exercises. This dataset distinguishes itself through the presence of free-form assignments, enabling a broader evaluation of feedback abilities.

We follow the preprocessing approach of (Koutcheme et al., 2024a) by selecting the incorrect programs from all students' last submitted solutions (as these specific solutions often reflect students who need the most help) for each assignment that can be automatically evaluated with unit tests. Following those steps results in training, validation, and testing splits with 690, 826, and 711 incorrect programs from 44, 62 and 62 assignments.

4.2 Feedback Tasks

We train our model to generate better program repairs and aim to evaluate whether this improvement transfers to feedback generation. Specifically, we assess our models performance on three feedback types widely studied in prior work (Koutcheme et al., 2025; Kotalwar et al., 2024; Phung et al., 2024; Hellas et al., 2023): explanations ($\mathcal{E}$), code patches ($\mathcal{P}$), and hints ($\mathcal{H}$).

Explanations identify and describe *all* issues in a student's program, while code patches outline the necessary corrections. These two types of diagnostic feedback help students understand their mistakes after submitting an incorrect solution. Hints, in contrast, are more Socratic, guiding students toward resolving one of the issues without giving away the answer, and are most valuable while students are still actively working and may be stuck.

Quality attributes. We evaluate the generated feedback using quality criteria established in prior work. Explanations and code patches are assessed based on accuracy ($\mathcal{E}_A$, $\mathcal{P}_A$) and selectiveness ($\mathcal{E}_S$, $\mathcal{P}_S$) (Koutcheme et al., 2025). Hints, are evaluated along three dimensions: correctness ($\mathcal{H}_C$), informativeness ($\mathcal{H}_I$), and concealment ($\mathcal{H}_{Con}$) (Phung et al., 2024). Table 3 (Appendix A) provides detailed definitions for each attribute. We later detail our evaluation strategy.

Generation strategy. When generating feedback, we always prompt our models to generate the three types of feedback sequentially: first the explanations, then the code patches, and finally *a single hint* for the first identified issue $\mathcal{F} = (\mathcal{E}, \mathcal{P}, \mathcal{H})$. This ordering draws from prior work, treating the explanation as a form of chain-of-thought reasoning (Wei et al., 2022) that supports the generation of more accurate patches (Koutcheme et al., 2025) and a more helpful hint (Phung et al., 2023).

4.3 Models

We train Llama-3.2-3B (Dubey et al., 2024) and Qwen-2.5-3B (Hui et al., 2024), two language models with strong performance on programming tasks. Models in the 3B parameters range strike a practical balance: they are small enough for deployment on edge devices (Kotalwar et al., 2024) yet large enough to rapidly benefit from reinforcement learning optimization (Sui et al., 2025).

Parameter efficient finetuning. We train both models using QLoRa (Dettmers et al., 2023), a Parameter-Efficient Fine-Tuning technique (PEFT) (Houlsby et al., 2019) that quantises a language model to 4-bit and adds on top a set of trainable parameters called adapters, while the base model remains frozen. Using PEFT techniques has two benefits: quantized models allow easy edge device deployment, while adapters allow easy recovery of the base model functionalities.

4.4 Baseline and Oracle

Our objective is to assess (1) whether reasoning for repair improves programming feedback and (2) how close our DRO-trained SLMs' can approach the performance of models having access to LLM supervision. To help answer each sub-question, we consider two LLM-distillation training references:

- **Reason-Repair (RR)**: For each incorrect program s^i, we prompt an LLM to generate a corrected repair by reasoning, without producing feedback (see Figure 3, Appendix C). We then fine-tune our small models on this thought and repair $(\mathcal{T}^i_{LLM}, \mathcal{R}^i_{LLM})$ sequence. These models allow us to evaluate whether learning to repair through thoughts using supervised fine-tuning improves feedback performance.
- **Repair-First Feedback (RFF)**: We adopt the repair-before-feedback strategy from (Sahai et al., 2023; Phung et al., 2023), prompting the LLM (see Figure 4) to first generate a repair $\mathcal{R}^i_{LLM}$ for the incorrect program as an intermediate reasoning step to produce the full feedback $\mathcal{F}^i_{LLM}$. While this two-step prompting strategy introduces an *implicit* form of reasoning similar to chain of thought, we note that it differs from the more *explicit* form of reasoning used in models such as DeepSeek (Luo et al., 2025). To our knowledge, such explicit reasoning has not been explored in prior work for providing feedback. Our small models are fine-tuned on these repair-feedback sequences $(\mathcal{R}^i_{LLM}, \mathcal{F}^i_{LLM})$. Since this approach relies on direct access to feedback during training, we consider RFF-trained models as oracles, serving as a strong upper bound.

We use OpenAI GPT-4o-mini (OpenAI, 2024) as the LLM due to its strong feedback performance (Koutcheme et al., 2024b) and cost efficiency.

4.5 Prompting Strategy

In our main experiments, we prompt the DRO and Reason-Repair models to generate feedback $\mathcal{F}$ immediately (see Figure 3), without any intermediate repair step (Koutcheme et al., 2025). This prompting strategy aims to study whether repair fine-tuning transfers directly into feedback improvements. In contrast, following (Sahai et al., 2023), we prompt the Repair-First Feedback models to first generate a repair for the incorrect programs before generating the final feedback $\mathcal{F}^i$.

We present in Appendix B the full results of our experiments, prompting all models with both strategies. The second approach evaluates whether and to what extent generating repairs before feedback effectively enhances small language models' performance.

4.6 LLMs-as-feedback-judges

We leverage LLMs-as-judges (Zheng et al., 2023; Thakur et al., 2024) to evaluate our models. Following the jury-based approach proposed by (Verga et al., 2024), we use a panel of two language models: GPT-4o-mini and Gemini-2.0-flash (Google DeepMind, 2024). While GPT-4o-mini and Gemini-2.0-flash are lighter versions of their full-size counterparts, they remain strong judges for programming feedback. For instance, GPT-4o-mini has been shown to perform on par with GPT-4o for evaluating feedback quality (Koutcheme et al., 2025). Moreover, (Verga et al., 2024) demonstrate that ensembles of smaller LLMs of different families can outperform even single large models, particularly by mitigating individual model biases.

For each feedback $\mathcal{F}$ generated on the test set, we prompt both judges (see Figure 6, Appendix C) to provide binary decisions across all quality criteria (Table 3). We adopt a strict unanimity policy: a criterion is marked correct only if both judges agree. While this method does not provide absolute performance guarantees (see Section 6), it offers a consistent, scalable, and reliable strategy for comparing the relative effectiveness of different training approaches.

4.7 Experiment Details

We describe our experiment settings, including specific hyperparameters used for training and inference, in section A.3 (Appendix A).

5 Results

In this section, we present the results of our experiments. A more in-depth analysis, including results for the *repair-first* strategy, is provided in Appendix B. For DRO, we show results for training for one epoch (DRO-1) and two epochs (DRO-2).

5.1 Direct Repair Optimization Results

Table 1 shows the results of our experiments answering the question: *How effective is training for repair in improving feedback abilities?* We can make the following observations.

Table 1: **Feedback performance results.** We contrast DRO model performance at n training epoch (**DRO-n**) against LLM-distilled training variants RR (Reason-Repair) for two (BASE) language models, Llama-3.1-3B and Qwen-2.5-3B. We bold (resp. underline) the best (resp. second best) results.

Method	$\mathcal{E}_A$	$\mathcal{E}_S$	$\mathcal{P}_A$	$\mathcal{P}_S$	$\mathcal{H}_C$	$\mathcal{H}_I$	$\mathcal{H}_{Con}$
Llama-3.1-3B							
BASE	37.4	55.4	34.5	25.2	67.5	63.4	67.8
DRO-1	40.5	64.4	36.6	30.4	**73.6**	**69.2**	**72.0**
DRO-2	**43.5**	**64.7**	40.5	32.2	72.6	67.9	71.9
RR	42.9	62.9	**42.9**	**41.8**	52.7	49.4	56.3
Qwen-2.5-3B							
BASE	49.6	69.2	39.1	32.2	85.1	80.0	78.2
DRO-1	53.2	66.0	47.4	34.7	87.3	84.5	80.7
DRO-2	**59.8**	**74.1**	**51.6**	**38.7**	90.7	88.3	**88.3**
RR	59.1	70.0	51.1	**38.7**	**91.4**	**90.0**	85.9

Training for repair improves feedback abilities. DRO improves feedback quality across all criteria for both base models. We also observe gains when training a model via supervised fine-tuning on thoughts and repairs generated by a language model (*RR*), although hint performance declines for Llama.

We interpret this success as a form of transfer learning (Raffel et al., 2020), where training on one task improves performance on another related task. Butler et al. (Butler and Winne, 1995) characterize feedback as a two-step process: first, noticing mistakes, and second, communicating them to the learner. We view program repair as a "super-task" of the noticing stage: to fix a student's code, the model must identify what is wrong (analogous to generating an explanation) and then determine how to correct it (analogous to generating a patch). Manual inspection also suggests that most, if not all, reasoning traces produced during repair explicitly highlight both the underlying issues and their corresponding fixes.

Unlike full feedback, however, program repair does not involve pedagogical communication. In other words, it does not improve the second stage: the model's ability to convey information effectively to a learner. Still, we believe the use of low rank adapters allows the model to preserve its original pedagogical capabilities while refining its analytical skills through targeted repair training. Lastly, since a hint is a form of non-revealing explanation, stronger repair capabilities indirectly enhance Socratic feedback.

DRO improvements scale rapidly with base model performance. We observe that the speed and magnitude of the improvement at each training epoch n depend on the base model. Qwen, which is a stronger base model than Llama, is showing more substantial gains after just one epoch of training, and even faster gains after a second training epoch. These results align with findings from prior work (DeepSeek-AI, 2025; Luo et al., 2025), suggesting that performance improvements when training reasoning models scale faster as the quality (i.e. initial performance) of the base model increases.

DRO is competitive with LLM-distillation. After two epochs of training (DRO-2), our models reach performance comparable to LLM-distilled variants. With Llama-3.1-3B, DRO-2 matches *RR* for generating explanations. While it underperforms on patches, it significantly outperforms *RR* on hint generation, where *RR* even falls behind the base model. With Qwen-2.5-3B, DRO-2 surpasses *RR* for generating both explanations and patches, and performs on par for writing hints.

5.2 Refining Distilled Models with Direct Repair Optimization

Table 2 shows the results of an experiment combining model distillation and reinforcement learning. Prior work highlights how SLM reasoning abilities can be bootstrapped by distilling chain-of-thoughts from an LLM, before being further enhanced through RL training (Sui et al., 2025). Following such an approach, we further fine-tuned Reason-Repair models using DRO for one epoch.

As we can observe, applying Direct Repair Optimization on top of the *RR* models consistently enhances feedback performance across almost all types of feedback for both models, allowing the RR-trained models to reach overall stronger performance (with the exception of a drop in Hint Concealment H_{Con}). Interestingly, we observe a stronger boost in diagnostic feedback performance for our Llama RR model than for our Qwen model. Looking more closely, both models reach similar overall performance after RL training, suggesting a performance trade-off might be happening between diagnostic and Socratic feedback abilities. We hypothesize that this plateau may stem from the limitations of LoRA adapters, which restrict how much new "knowledge" the model can acquire (Dettmers et al., 2023).

Table 2: **Feedback performance results.** We further fine-tune the Reasoning-Repair (RR) models with DRO and show performance benefits of the resulting **COMB** models. We bold (resp. underline) the best (resp. second best) results.

Method	$\mathcal{E}_A$	$\mathcal{E}_S$	$\mathcal{P}_A$	$\mathcal{P}_S$	$\mathcal{H}_C$	$\mathcal{H}_I$	$\mathcal{H}_{Con}$
Model: Llama-3.2-3B							
RR	42.9	62.9	42.9	41.8	52.7	49.4	56.3
DRO-2	43.5	64.7	40.5	32.2	72.6	67.9	71.9
RFF	43.5	70.9	36.3	40.4	**94.2**	**92.3**	**89.3**
COMB	**59.4**	**72.7**	**56.3**	**48.1**	53.9	48.9	51.9
Model: Qwen-2.5-3B							
RR	59.1	70.0	51.1	38.7	91.4	90.0	85.9
DRO-2	59.8	74.1	51.6	38.5	90.7	88.3	88.3
RFF	**72.4**	**82.4**	**62.7**	**50.1**	**94.4**	**91.7**	**89.5**
COMB	60.3	72.6	54.7	40.4	91.6	**91.7**	85.0

5.3 Comparison Against Oracles

Table 2 also contrasts the performance of models trained with Direct Repair Optimization against the Repair-Feedback-First (RFF) models. We consider RFF models as oracles as those were trained with privileged supervision in the form of both LLM-generated feedback and repairs used as implicit reasoning steps.

Despite not using any feedback supervision, our DRO-2 models perform competitively in several areas, particularly with Llama-3B, where they closely approach RFF performance on diagnostic feedback tasks. Moreover, our combined approach (COMB), which applies DRO fine-tuning on top of LLM-distilled models, surpasses RFF on several criteria for Llama, including both explanation and patch quality. However, for Qwen, both DRO and COMB consistently fall short of RFF performance across most evaluation metrics.

These findings suggest that while DRO can serve as a valuable complement to LLM supervision, and even outperform direct finetuning in some settings, it is not yet a reliable substitute for training models directly on feedback generated by LLMs. Further work is needed to understand when and how DRO can consistently match or exceed the performance of supervised approaches.

6 Discussion and Conclusion

In this paper, we explored whether training language models to reason about students' programs could improve feedback and provided insights into how this strategy compares to LLM supervision.

Summarizing answers to our research question. Our findings show that (1) reasoning to repair programs improves a model's ability to generate feedback, (2) DRO can further improve models fine-tuned on LLM-generated repairs, and (3) such refined models can, in some instances, match the performance of models trained directly on LLM-generated feedback.

Implications for programming education. Programming education is increasingly turning to open-source language models, particularly smaller ones, to support teaching at scale (Liu et al., 2024b). We are anticipating a shift from using proprietary LLMs (e.g., GPT-4o) to open-source alternatives (e.g., LLaMA-3.3-70B) for distillation (Kotalwar et al., 2024), as the latter can be hosted locally and offer more control (Denny et al., 2024). Our work takes a step further by exploring whether we can eliminate reliance on LLMs and train SLMs directly on educational data, avoiding both the costs of third-party APIs and the computational demands of hosting large open-source models.

Although our work focuses on programming data, we believe DRO could be adapted to provide feedback in all educational domains where the correctness of a student's work can be automatically evaluated. Methods such as Direct Repair Optimization can leverage much of the readily available data in educational platforms (i.e., student submissions and unit tests) without requiring extensive curation. Recent work has shown that combining such reinforcement learning methods with large-scale data can allow relatively small models to reach the performance of state-of-the-art proprietary models (Luo et al., 2025).

Extensions to other training approaches. Direct Repair Optimization can easily be combined with human and LLM-supervised training strategies. Models trained with DRO can be bootstrapped from a handful of high-quality LLM-generated examples (Hicke et al., 2023; Ashok Kumar and Lan, 2024; Muennighoff et al., 2025), and further refined using Reinforcement Learning from Human Feedback (RLHF) (Woodrow et al., 2025) to align with specific instructional goals. Such hybrid pipelines offer a practical path forward, starting from refined educational data, scaling up performance through large-scale reinforcement learning, and applying targeted human supervision as a final step to meet specific classroom needs.

Alleviating privacy concerns. Although third-party LLM hosting services and the use of proprietary APIs are becoming more affordable, institutional policies on sending student data to third-party services can restrict their use. Our experiments show that institutions with access to modest computational power (such as a single consumer-grade GPU)[2] can obtain powerful programming teaching assistant models tailored to their classes.

Such models can also be directly deployed on students' laptops (Liu et al., 2024b; Kotalwar et al., 2024; Ruan et al., 2024), enabling personalized, timely, and offline support.

Future work. Our future work will explore how Direct Repair Optimization performs compared to proprietary and open-source LLMs when trained on large-scale private educational programming data as well as public programming data from HuggingFace [3]. To this end, we plan to conduct human expert evaluations and perform A/B studies to evaluate how real students respond to such feedback (SLM vs LLMs). We will also investigate how human data and preferences can be integrated into the training pipeline to better align small models with specific institutional goals.

Looking ahead, we aim to move beyond training individual models on private institutional data and tackle the broader challenge of building foundation models for programming education (Bommasani et al., 2022). We believe such models could be pretrained from publicly open-source large-scale ethical data, and further refined with federated learning across multiple institutions.

Limitations

Our study is not without limitations. First, we conducted all experiments on a single dataset of Python programming submissions collected from one institution and did not explore whether our results hold in other contexts. Second, and perhaps more importantly, our evaluation lacks human annotations, expert assessment, or qualitative analysis. While prior work suggests LLMs can be used to assess programming feedback (Seo et al., 2025; Koutcheme et al., 2024b, 2025), such works also highlight that their judgments are not always perfect. Although we partly mitigate this by combining multiple LLMs-as-judges, our results must still be interpreted with caution.

We do not claim that DRO-trained models produce feedback that meets any absolute standard of quality (e.g., "nearly perfect feedback"). Rather, our findings establish DRO's relative performance: it improves feedback quality over a base untrained model and can match the performance of models trained via LLM distillation. Whether such feedback is ultimately pedagogically effective for students remains an open question until validated through human studies.

Additionally, our experiments were limited to two small models with around 3B parameters. While prior work suggests that performance improves with base model size (Sui et al., 2025), it remains to be seen whether the same trends hold when applying Direct Repair Optimization for improving other language models' programming feedback. Moreover, our experiments also did not include new state-of-the-art reasoning large language models such as OpenAI o3. Such models, which were effectively trained for reasoning, would probably act as better candidates for LLM-distillation and combined LLM-distillation and RL training.

Ethics Statement

This work has been conducted in accordance with national and institutional ethical guidelines. We recognize the growing importance of ethical considerations in AI research, particularly with respect to data use, model deployment, and societal impact.

The dataset used in this study is publicly available to the research community. Our primary goal is to advance the development and evaluation of open-source language models for feedback generation in programming education. By prioritizing open-source models, we aim to promote transparency, accessibility, and accountability, while mitigating privacy concerns associated with proprietary LLMs.

We also acknowledge broader ethical dimensions of our work. These include questions of fairness and equity in access to high-quality feedback, the risk that language models may favor certain interaction styles or learner backgrounds, and the potential for such technologies to either reduce or exacerbate global disparities in education. As the use of LLMs in learning environments grows, we believe it is essential to continuously assess and address these challenges in collaboration with educators, institutions, and affected communities.

[2]We trained our models on a single 32GB VRAM GPU.
[3]https://huggingface.co/datasets

References

Umair Z. Ahmed, Shubham Sahai, Ben Leong, and Amey Karkare. 2025. Feasibility study of augmenting teaching assistants with ai for cs1 programming feedback. In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1*, SIGCSETS 2025, page 11–17, New York, NY, USA. Association for Computing Machinery.

Nischal Ashok Kumar and Andrew Lan. 2024. Improving socratic question generation using data augmentation and preference optimization. In *Proceedings of the 19th Workshop on Innovative Use of NLP for Building Educational Applications (BEA 2024)*, pages 108–118, Mexico City, Mexico. Association for Computational Linguistics.

Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, et al. 2022. On the opportunities and risks of foundation models.

Deborah L. Butler and Philip H. Winne. 1995. Feedback and self-regulated learning: A theoretical synthesis. *Review of Educational Research*, 65(3):245–281.

Badhan Chandra Das, M. Hadi Amini, and Yanzhao Wu. 2025. Security and privacy challenges of large language models: A survey. *ACM Comput. Surv.*, 57(6).

Adrian de Freitas, Joel Coffman, Michelle de Freitas, Justin Wilson, and Troy Weingart. 2023. Falconcode: A multiyear dataset of python code samples from an introductory computer science course. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, SIGCSE 2023, page 938–944, New York, NY, USA. Association for Computing Machinery.

DeepSeek-AI. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning.

Paul Denny, James Prather, Brett A. Becker, James Finnie-Ansley, Arto Hellas, Juho Leinonen, Andrew Luxton-Reilly, Brent N. Reeves, Eddie Antonio Santos, and Sami Sarsa. 2024. Computing education in the era of generative ai. *Commun. ACM*, 67(2):56–67.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. Qlora: Efficient finetuning of quantized llms. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, page 441, Red Hook, NY, USA. Curran Associates Inc.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, and Aiesha Letman et al. 2024. The llama 3 herd of models.

Google DeepMind. 2024. Gemini 2.0: Our largest and most capable AI model. `https://blog.google/technology/google-deepmind/google-gemini-ai-update-december-2024/`. Accessed: 2025-04-24.

John Hattie and Helen Timperley. 2007. The power of feedback. *Review of educational research*, 77(1):81–112.

Arto Hellas, Juho Leinonen, Sami Sarsa, Charles Koutcheme, Lilja Kujanpää, and Juha Sorva. 2023. Exploring the responses of large language models to beginner programmers' help requests. In *Proceedings of the 2023 ACM Conference on International Computing Education Research - Volume 1*, ICER '23, page 93–105, New York, NY, USA. Association for Computing Machinery.

Yann Hicke, Anmol Agarwal, Qianou Ma, and Paul Denny. 2023. Ai-ta: Towards an intelligent question-answer teaching assistant using open-source llms.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for NLP. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2790–2799. PMLR.

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

Johan Jeuring, Hieke Keuning, Samiha Marwan, Dennis Bouvier, Cruz Izu, Natalie Kiesler, Teemu Lehtinen, Dominic Lohr, Andrew Peterson, and Sami Sarsa. 2022. Towards giving timely formative feedback and hints to novice programmers. In *Proceedings of the 2022 Working Group Reports on Innovation and Technology in Computer Science Education*, ITiCSE-WGR '22, page 95–115, New York, NY, USA. Association for Computing Machinery.

Harshit Joshi, José Pablo Cambronero Sánchez, Sumit Gulwani, Vu Le, Gust Verbruggen, and Ivan Radicek. 2023. Repair is nearly generation: Multilingual program repair with llms. In *Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023, Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence, IAAI 2023, Thirteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2023, Washington, DC, USA, February 7-14, 2023*, pages 5131–5140. AAAI Press.

Nachiket Kotalwar, Alkis Gotovos, and Adish Singla. 2024. Hints-in-browser: Benchmarking language models for programming feedback generation. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

Charles Koutcheme. 2022. Towards open natural language feedback generation for novice programmers using large language models. In *Proceedings of the 22nd Koli Calling International Conference on Computing Education Research*, Koli Calling '22, New

York, NY, USA. Association for Computing Machinery.

Charles Koutcheme. 2023. Training Language Models for Programming Feedback Using Automated Repair Tools. In *Artificial Intelligence in Education*, pages 830–835, Cham. Springer Nature Switzerland.

Charles Koutcheme, Nicola Dainese, and Arto Hellas. 2024a. Using program repair as a proxy for language models' feedback ability in programming education. In *Proceedings of the 19th Workshop on Innovative Use of NLP for Building Educational Applications (BEA 2024)*, pages 165–181, Mexico City, Mexico. Association for Computational Linguistics.

Charles Koutcheme, Nicola Dainese, Sami Sarsa, Arto Hellas, Juho Leinonen, Syed Ashraf, and Paul Denny. 2025. Evaluating language models for generating and judging programming feedback. In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1*, SIGCSETS 2025, page 624–630, New York, NY, USA. Association for Computing Machinery.

Charles Koutcheme, Nicola Dainese, Sami Sarsa, Arto Hellas, Juho Leinonen, and Paul Denny. 2024b. Open source language models can provide feedback: Evaluating llms' ability to help students using gpt-4-as-a-judge. In *Proceedings of the 2024 Innovation and Technology in Computer Science Education, Volume 1*, ITICSE '24.

Charles Koutcheme, Nicola Dainese, Sami Sarsa, Juho Leinonen, Arto Hellas, and Paul Denny. 2024c. Benchmarking educational program repair.

Charles Koutcheme, Sami Sarsa, Juho Leinonen, Lassi Haaranen, and Arto Hellas. 2023a. Evaluating distance measures for program repair. In *Proceedings of the 2023 ACM Conference on International Computing Education Research - Volume 1*, ICER '23, page 495–507, New York, NY, USA. Association for Computing Machinery.

Charles Koutcheme, Sami Sarsa, Juho Leinonen, Arto Hellas, and Paul Denny. 2023b. Automated Program Repair Using Generative Models for Code Infilling. In *Artificial Intelligence in Education*, pages 798–803, Cham. Springer Nature Switzerland.

Harrison Lee, Samrat Phatale, Hassan Mansoor, Thomas Mesnard, Johan Ferret, Kellie Lu, Colton Bishop, Ethan Hall, Victor Carbune, Abhinav Rastogi, and Sushant Prakash. 2024. RLAIF vs. RLHF: scaling reinforcement learning from human feedback with AI feedback. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net.

Mark Liffiton, Brad E Sheese, Jaromir Savelka, and Paul Denny. 2024. Codehelp: Using large language models with guardrails for scalable support in programming classes. In *Proceedings of the 23rd Koli Calling International Conference on Computing Education Research*, Koli Calling '23, New York, NY, USA. Association for Computing Machinery.

Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain. Association for Computational Linguistics.

Rongxin Liu, Carter Zenke, Charlie Liu, Andrew Holmes, Patrick Thornton, and David J. Malan. 2024a. Teaching cs50 with ai: Leveraging generative artificial intelligence in computer science education. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 2*, SIGCSE 2024, page 1927, New York, NY, USA. Association for Computing Machinery.

Suqing Liu, Zezhu Yu, Feiran Huang, Yousef Bulbulia, Andreas Bergen, and Michael Liut. 2024b. Can small language models with retrieval-augmented generation replace large language models when learning computer science? In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1*, ITiCSE 2024, page 388–393, New York, NY, USA. Association for Computing Machinery.

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. 2025. Understanding r1-zero-like training: A critical perspective.

Dominic Lohr, Hieke Keuning, and Natalie Kiesler. 2025. You're (not) my type—can llms generate feedback of specific types for introductory programming tasks? *Journal of Computer Assisted Learning*, 41(1):2025.

Michael Luo, Sijun Tan, Justin Wong, Xiaoxiang Shi, William Y. Tang, Manan Roongta, Colin Cai, Jeffrey Luo, Li Erran Li, Raluca Ada Popa, and Ion Stoica. 2025. Deepscaler: Surpassing o1-preview with a 1.5b model by scaling rl. Notion Blog.

Andrew Luxton-Reilly, Simon, Ibrahim Albluwi, Brett A. Becker, Michail Giannakos, Amruth N. Kumar, Linda Ott, James Paterson, Michael James Scott, Judy Sheard, and Claudia Szabo. 2018. Introductory programming: a systematic literature review. In *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE 2018 Companion, page 55–106, New York, NY, USA. Association for Computing Machinery.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. 2025. s1: Simple test-time scaling.

OpenAI. 2024. Gpt-4o system card.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al.

2022. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744.

José Carlos Paiva, José Paulo Leal, and Álvaro Figueira. 2022. Automated assessment in computer science education: A state-of-the-art review. *ACM Trans. Comput. Educ.*, 22(3).

Richard Yuanzhe Pang, Weizhe Yuan, He He, Kyunghyun Cho, Sainbayar Sukhbaatar, and Jason Weston. 2024. Iterative reasoning preference optimization. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

Tung Phung, JosÃ© Cambronero, Sumit Gulwani, Tobias Kohn, Rupak Majumdar, Adish Singla, and Gustavo Soares. 2023. Generating high-precision feedback for programming syntax errors using large language models. In *Proceedings of the 16th International Conference on Educational Data Mining*, pages 370–377, Bengaluru, India. International Educational Data Mining Society.

Tung Phung, Victor-Alexandru Pădurean, Anjali Singh, Christopher Brooks, José Cambronero, Sumit Gulwani, Adish Singla, and Gustavo Soares. 2024. Automating human tutor-style programming feedback: Leveraging gpt-4 tutor model for hint generation and gpt-3.5 student model for hint validation. In *Proceedings of the 14th Learning Analytics and Knowledge Conference*, LAK '24, page 12–23, New York, NY, USA. Association for Computing Machinery.

Thomas W. Price, Rui Zhi, and Tiffany Barnes. 2017. Evaluation of a data-driven feedback algorithm for open-ended programming. In *Proceedings of the 10th International Conference on Educational Data Mining, EDM 2017, Wuhan, Hubei, China, June 25-28, 2017*. International Educational Data Mining Society (IEDMS).

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2024. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(1).

Kelly Rivers and Kenneth R Koedinger. 2017. Data-driven hint generation in vast solution spaces: a self-improving python programming tutor. *International Journal of Artificial Intelligence in Education*, 27:37–64.

Charlie F Ruan, Yucheng Qin, Xun Zhou, Ruihang Lai, Hongyi Jin, Yixin Dong, Bohan Hou, Meng-Shiun Yu, Yiyan Zhai, Sudeep Agarwal, et al. 2024. Webllm: A high-performance in-browser llm inference engine. *arXiv preprint arXiv:2412.15803*.

Shubham Sahai, Umair Z. Ahmed, and Ben Leong. 2023. Improving the coverage of gpt for automated feedback on high school programming assignments.

Alexander Scarlatos, Digory Smith, Simon Woodhead, and Andrew Lan. 2024. *Improving the Validity of Automatically Generated Feedback via Reinforcement Learning*, page 280–294. Springer Nature Switzerland.

Hyein Seo, Taewook Hwang, Jeesu Jung, Hyeonseok Kang, Hyuk Namgoong, Yohan Lee, and Sangkeun Jung. 2025. Large language models as evaluators in education: Verification of feedback consistency and accuracy. *Applied Sciences*, 15:671.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models.

Yang Sui, Yu-Neng Chuang, Guanchu Wang, Jiamu Zhang, Tianyi Zhang, Jiayi Yuan, Hongyi Liu, Andrew Wen, Shaochen Zhong, Hanjie Chen, and Xia Hu. 2025. Stop overthinking: A survey on efficient reasoning for large language models.

Aman Singh Thakur, Kartik Choudhary, Venkat Srinik Ramayapally, Sankaran Vaidyanathan, and Dieuwke Hupkes. 2024. Judging the judges: Evaluating alignment and vulnerabilities in llms-as-judges.

Annapurna Vadaparty, Daniel Zingaro, David H. Smith IV, Mounika Padala, Christine Alvarado, Jamie Gorson Benario, and Leo Porter. 2024. Cs1-llm: Integrating llms into cs1 instruction. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1*, ITiCSE 2024, page 297–303, New York, NY, USA. Association for Computing Machinery.

Pat Verga, Sebastian Hofstatter, Sophia Althammer, Yixuan Su, Aleksandra Piktus, Arkady Arkhangorodsky, Minjie Xu, Naomi White, and Patrick Lewis. 2024. Replacing judges with juries: Evaluating llm generations with a panel of diverse models.

Arto Vihavainen, Jonne Airaksinen, and Christopher Watson. 2014. A systematic review of approaches for teaching introductory programming and their influence on success. In *Proceedings of the Tenth Annual Conference on International Computing Education Research*, ICER '14, page 19–26, New York, NY, USA. Association for Computing Machinery.

Sierra Wang, John Mitchell, and Chris Piech. 2024. A large scale rct on effective error messages in cs1. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*, SIGCSE 2024, page 1395–1401, New York, NY, USA. Association for Computing Machinery.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Juliette Woodrow, Sanmi Koyejo, and Chris Piech. 2025. Improving generative ai student feedback: Direct preference optimization with teachers in the loop. `https://juliettewoodrow.github.io/paper-hosting/dpo_feedback.pdf`. Accessed: 2025-04-12.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, et al. 2025a. Dapo: An open-source llm reinforcement learning system at scale.

Zezhu Yu, Suqing Liu, Paul Denny, Andreas Bergen, and Michael Liut. 2025b. Integrating small language models with retrieval-augmented generation in computing education: Key takeaways, setup, and practical insights. In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1*, SIGCSETS 2025, page 1302–1308, New York, NY, USA. Association for Computing Machinery.

Jialu Zhang, José Cambronero, Sumit Gulwani, Vu Le, Ruzica Piskac, Gustavo Soares, and Gust Verbruggen. 2022. Repairing bugs in python assignments using language models.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA. Curran Associates Inc.

A Methodological and Experimental Details

A.1 Training loss

The original GRPO loss function is given by:

$$\mathcal{J}_{\text{GRPO}} = -\frac{1}{\sum_{g=1}^{G} |o_g|} \sum_{g=1}^{G} \sum_{t=1}^{|o_g|} l_{g,t} - \beta \text{KL}(\pi_\theta, \pi_{\text{ref}})$$

where

$$l_{g,t} = \frac{\pi_\theta(o_{g,t} \mid d^i, s^i, o_{g,<t})}{\pi_{\theta_{\text{old}}}(o_{g,t} \mid d^i, s^i, o_{g,<t})} \hat{A}_{g,t}$$

and $\hat{A}^i$ is a value called the advantage. Intuitively, the advantage tells each generation how much better it is than the $g-1$ other generations. Compared to popular offline preference methods such as DPO (Rafailov et al., 2024), which use explicit preference pairs, the advantage function helps "ranking" which of the multiple generations, without relying on pairwise comparisons.

$$\hat{A}_g = \frac{(r_g^i - \bar{r})}{std(r)}$$

$\text{KL}(\pi_\theta, \pi_{\text{ref}})$ is a value called the KL divergence. This value essentially tells how much the model responses are diverging from the model prior to the start of training. We omit the definition of this term for simplicity and refer the reader to the DeepSeek paper (DeepSeek-AI, 2025). In essence, $\mathcal{J}_{\text{GRPO}}$ is the weighted average of the advantage of all completions and a β scaled approximation of the KL divergence.

The following works have found a few issues with the original formulation.

Removing the KL term. (Yu et al., 2025a) finds that the $\text{KL}(\pi_\theta, \pi_{\text{ref}})$ term can slow down training, as in practice we want to allow the trained model to diverge from the original policy.

Length response bias. (Liu et al., 2025) show that the term $-\frac{1}{\sum_{g=1}^{G}|o_g|}$ introduces a *response length bias* favourizing longer generations. To address this issue, the authors propose dividing by a constant length, being the maximum allowed size of each generation (LG).

Program difficulty bias. Moreover, they also show that the standard deviation in the advantage computation $\hat{A}_g = \frac{(r_g^i - \bar{r})}{std(r)}$ introduces *a problem difficulty bias*, where overly hard or overly easy questions are weighted more heavily in the loss. In our situation, the "problem" is the student program to solve, and this bias would lead to scenarios where student programs which are too easy to fix or student programs which are too hard to solve would be given more attention. Removing the standard deviation addresses this issue. Taking these two changes into account yields the proposed loss function (see equation 1).

Taking into account multiple updates. Because sampling generations is computationally and time-intensive, in practice, we use a version of this loss function which takes into account multiple updates per generation, as proposed by (Yu et al., 2025a):

$$\mathcal{J}_{\text{dr.GRPO}} = -\frac{1}{\sum_{g=1}^{G}|o_g|} \sum_{g=1}^{G} \sum_{t=1}^{|o_g|} \left[min\left(l_{g,t}, \hat{C}_g \hat{A}_g)\right)\right]$$

where

$$\hat{C}_g = clip(l_{g,t}, 1-\epsilon_{low}, 1+\epsilon_{high})$$

constrains the subsequent updates to stay within a reasonable range of the original policy.

A.2 Feedback Quality Attributes.

Table 3 shows the definitions of the feedback quality criteria used in our work. These definitions are taken from prior work in programming feedback.

A.3 Experimental Details

We outline training and inference-specific details.

A.3.1 Training

We train our models using the HuggingFace TRL library. Unless explicitly outlined below, all hyperparameters were left at default values. We train all models with QLoRa (Dettmers et al., 2023) using an alpha $\alpha = 128$ and a rank $r = 128$. All models are trained on a single NVIDIA V100 GPU using our institution's research cluster. Training for one epoch on such compute takes approximately 8 hours. Training on an A100 takes less than 5 hours.

dr.GRPO specific hyperparameters. Table 4 shows the hyperparameters used to train our DRO model. These parameters follow prior work (DeepSeek-AI, 2025; Luo et al., 2025; Yu et al., 2025a). We train all models for two epochs on the training set of FalconCode. For each incorrect program, we generate four ($G = 4$) candidate reasoning and repairs $\mathcal{G}^i = (\mathcal{T}_l^i, \mathcal{R}_l^i)$. We highlight that we designed our method to run on an entry-level GPU with 32GB of RAM. Prior work (DeepSeek-AI, 2025) suggests that a higher number can substantially improve results, however, more generations require more GPU RAM.

Supervised Fine-tuning. Table 5 shows the hyperparameters used to train our distilled models via supervised fine-tuning on the training set of FalconCode. We train all models for three epochs.

A.3.2 Inference

For generating feedback at inference time, for all models, we generate both repair and feedback using greedy decoding. For judging, we query proprietary models GPT-4o-mini and Gemini-2.0-flash using the OpenAI Python API, also using greedy decoding.

B Results Details

Table 6 shows the full results of all our experiments, including the repair-first prompting strategy. We additionally report the performance of our models on additional clarity criteria.

B.1 Prompting for Repair

Prompting for repair often decreases base model performance. Prompting the *BASE* models to generate a repair before producing feedback decreases diagnostic feedback performance. This observation aligns with findings from (Koutcheme et al., 2024a), who showed that a model's ability to provide diagnostic feedback scales independently from its ability to perform program repair. In our case, using SLMs, a poor-quality greedy repair may degrade feedback quality more than providing no repair. This does not contradict (Sahai et al., 2023), as the authors use the Repair-First strategy with LLMs, which are strong at both repair and feedback. (Phung et al., 2023, 2024) extend the single-repair strategy with multiple repairs, but whether an SLM benefits from this is unknown. While it might alleviate this issue, it remains computationally expensive when running models locally.

Prompting to repair before feedback also decreases Socratic feedback performance for Qwen but increases it for Llama.

Prompting for repair brings benefits in diagnostic feedback performance for strong base models. Prompting to repair before feedback decreases the performance of the Llama DRO-trained models for generating diagnostic feedback but increases diagnostic feedback performance for Qwen models. We hypothesize that this effect is due to the base model overfitting on low-quality, greedy-generated repairs. We observe the same phenomenon for the RFF models, which were trained to repair before feedback: a decrease for Llama, but an increase for Qwen. For RR and their extended version with DRO (**COMB**), the effect is unclear.

B.2 Generations Clarity

We also studied how language models perform in terms of the clarity (Cle) of the generations. However, we mostly observe that there does not seem to be a clear correlation with base model performance, training method, or prompting strategy.

Name	Notation	Definition	Used in
Accuracy	$\mathcal{E}_A, \mathcal{P}_A$	All issues in the student's code (or all required fixes) are correctly identified.	(Koutcheme et al., 2024b, 2025)
Selectiveness	$\mathcal{E}_S, \mathcal{P}_S$	No non-existent or irrelevant issues are mentioned; no unnecessary changes are proposed.	(Koutcheme et al., 2024b, 2025)
Clarity	$\mathcal{E}_C, \mathcal{P}_C$	The explanation or patch is easy to understand, well-formatted, and concise.	(Koutcheme et al., 2025)
Correctness	$\mathcal{H}_C$	The hint provides correct information that would help fix the student's code.	(Phung et al., 2024; Kotalwar et al., 2024)
Informativeness	$\mathcal{H}_I$	The hint contains useful information that helps the student understand or resolve the issue.	(Phung et al., 2024; Kotalwar et al., 2024)
Concealment	$\mathcal{H}_{Con}$	The hint avoids revealing the full solution and encourages reasoning.	(Phung et al., 2024; Kotalwar et al., 2024)
Clarity	$\mathcal{H}_{Cle}$	The hint is clearly written, easy to read, and free of unnecessary complexity.	(Phung et al., 2024; Kotalwar et al., 2024)

Table 3: Feedback quality attributes used in this study, taken from prior work.

Table 4: **GRPO training hyperparameters.**

Hyperparameter	**Value**
Learning rate	1e-6
Epochs	2
Warmup ratio	0.1
Max gradient norm	0.2
Scheduler type	constant_with_warmup
Optimizer	paged_adamw_8bit
Gradient checkpointing	True
Batch size	2
Max prompt length	512
Max completion length (LG)	1512
GRPO-specific	
Num generations	4
Num iterations	2
Epsilon	0.2
Epsilon high	0.28
Top-p	0.95
Temperature	0.7
Model settings	
Precision	fp16
LoRA config	
LoRA rank (r)	128
LoRA alpha	128

Table 5: **Supervised Fine-Tuning hyperparameters.**

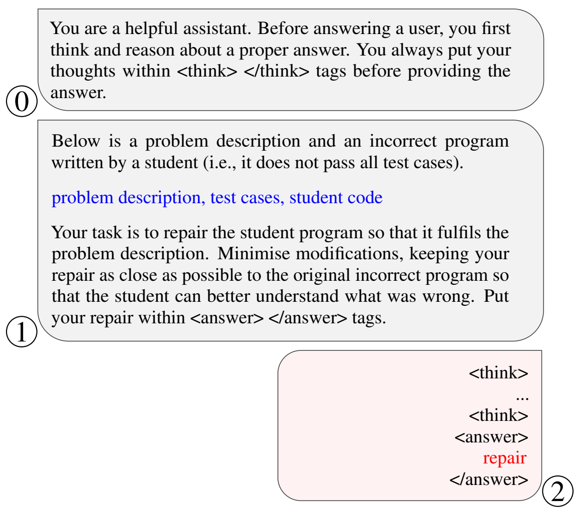
Hyperparameter	**Value**
Learning rate	1e-4
Epochs	3
Warmup ratio	0.1
Scheduler type	cosine
Batch size	8
Model settings	
Precision	fp16
LoRA config	
LoRA rank (r)	128
LoRA alpha	128

C Prompts

This section shows all the prompts used in our study.

Table 6: **Feedback performance results.** We contrast DRO model performance at n training epoch (**DRO-n**) against LLM-distilled training variants RR (Reason-Repair) and RFF (Repair First then Feedback), as well as the RR further fine-tuned with DRO(COMB), for two (BASE) language models, Llama-3.1-3B and Qwen-2.5-3B, for two prompting strategies: Direct Feedback and Repair First.

	Llama-3.1-3B										Qwen-2.5-3B									
Method	$\mathcal{E}_A$	$\mathcal{E}_S$	$\mathcal{E}_{Cle}$	$\mathcal{P}_A$	$\mathcal{P}_S$	$\mathcal{P}_{Cle}$	$\mathcal{H}_C$	$\mathcal{H}_I$	$\mathcal{H}_{Con}$	$\mathcal{H}_{Cle}$	$\mathcal{E}_A$	$\mathcal{E}_S$	$\mathcal{E}_{Cle}$	$\mathcal{P}_A$	$\mathcal{P}_S$	$\mathcal{P}_{Cle}$	$\mathcal{H}_C$	$\mathcal{H}_I$	$\mathcal{H}_{Con}$	$\mathcal{H}_{Cle}$
Prompting Strategy: Direct Feedback																				
BASE	37.4	55.4	61.9	34.5	25.2	74.7	67.5	63.4	67.8	72.7	49.6	69.2	60.2	39.1	32.2	68.4	85.1	80.0	78.2	80.7
DRO-1	40.5	64.4	62.7	36.6	30.4	75.5	73.6	69.2	72.0	77.6	53.2	66.0	65.3	47.4	34.7	71.4	87.3	84.5	80.7	83.1
DRO-2	43.5	64.7	58.2	40.5	32.2	72.7	72.6	67.9	71.9	75.8	*59.8*	*74.1*	59.5	*51.6*	38.7	68.6	90.7	88.3	88.3	80.3
RR	42.9	62.9	65.7	42.9	41.8	72.4	52.7	49.4	56.3	59.6	59.1	70.0	59.5	51.1	38.7	75.8	91.4	90.0	85.9	86.6
RFF	54.9	76.9	56.8	44.6	40.8	66.1	90.3	86.1	87.2	84.8	52.9	65.8	66.2	44.9	30.8	68.6	91.6	89.3	88.5	81.0
COMB	59.4	72.7	64.4	56.3	48.1	70.5	53.9	48.9	51.9	54.9	60.3	72.6	62.6	54.7	40.4	74.7	91.6	91.7	85.0	86.5
Prompting Strategy: Repair First																				
BASE	29.0	56.0	58.5	18.7	19.5	51.5	83.0	79.0	82.3	77.8	34.7	57.7	65.8	26.6	24.5	61.6	79.5	74.1	73.1	81.9
DRO-1	38.0	71.7	60.1	26.9	31.9	49.8	86.6	82.3	85.9	78.6	61.2	76.5	60.5	51.1	41.6	61.7	90.0	84.1	84.4	78.3
DRO-2	33.5	69.6	66.0	24.8	32.9	51.5	88.2	83.8	89.2	82.8	63.9	82.4	63.7	49.6	45.9	57.7	89.7	83.0	86.8	79.0
RR	47.7	76.1	60.3	39.1	41.6	56.4	93.1	88.7	90.3	82.0	46.7	66.2	58.8	40.6	35.6	66.0	88.5	82.3	80.7	82.1
RFF	43.5	70.9	67.4	36.3	40.4	56.0	94.2	92.3	89.3	88.0	72.4	82.4	68.2	62.7	50.1	63.4	94.4	91.7	89.5	79.7
COMB	50.9	79.3	57.9	44.3	46.3	50.6	94.0	91.7	90.2	82.8	56.8	73.4	58.5	47.7	40.1	64.0	89.0	87.2	81.9	84.2


⓪ You are a helpful assistant. Before answering a user, you first think and reason about a proper answer. You always put your thoughts within <think> </think> tags before providing the answer.

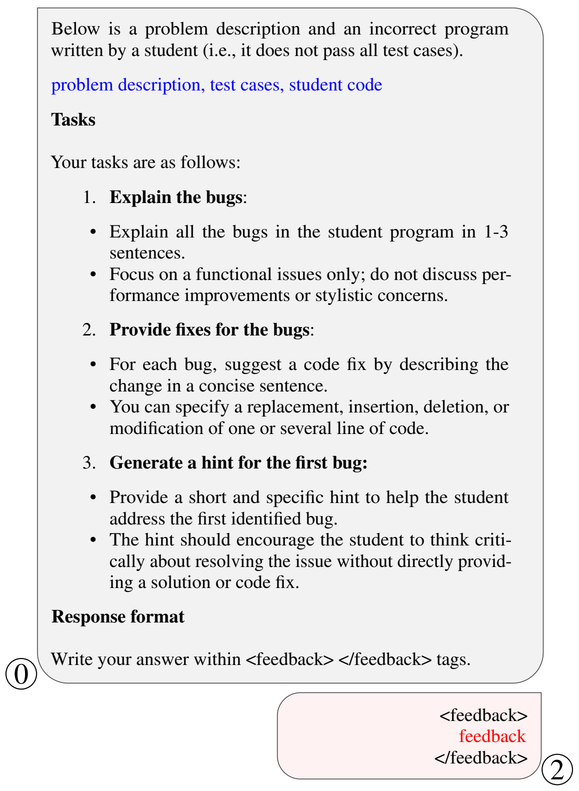
① Below is a problem description and an incorrect program written by a student (i.e., it does not pass all test cases).

problem description, test cases, student code

Your task is to repair the student program so that it fulfils the problem description. Minimise modifications, keeping your repair as close as possible to the original incorrect program so that the student can better understand what was wrong. Put your repair within <answer> </answer> tags.

② <think>
...
<think>
<answer>
repair
</answer>

Figure 2: **Training prompt.** We provide: ⓪ a system prompt that asks the language model to reason before answering, ① a description of the repair task. This prompt is used to obtain training data for Reason-Repair models ② and is used during training for our DRO models. Importantly, we prefill the model generation with a <think> tag to force the generation of the thinking content.


⓪ Below is a problem description and an incorrect program written by a student (i.e., it does not pass all test cases).

problem description, test cases, student code

Tasks

Your tasks are as follows:

1. **Explain the bugs**:
   - Explain all the bugs in the student program in 1-3 sentences.
   - Focus on a functional issues only; do not discuss performance improvements or stylistic concerns.
2. **Provide fixes for the bugs**:
   - For each bug, suggest a code fix by describing the change in a concise sentence.
   - You can specify a replacement, insertion, deletion, or modification of one or several line of code.
3. **Generate a hint for the first bug:**
   - Provide a short and specific hint to help the student address the first identified bug.
   - The hint should encourage the student to think critically about resolving the issue without directly providing a solution or code fix.

Response format

Write your answer within <feedback> </feedback> tags.

② <feedback>
feedback
</feedback>

Figure 3: **Generation prompt: Providing direct feedback.** We provide: ⓪ a description of the repair task, and ask the language models to generate feedback ①.

Below is a problem description and an incorrect program written by a student (i.e., it does not pass all test cases).

problem description, test cases, student code

Tasks

Your tasks are as follows:

1. **Repair the student program**
   - Minimise modifications, keeping your repair as close as possible to the original incorrect program so that the student can better understand what was wrong.
2. **Explain the bugs**:
   - Explain all the bugs in the student program in 1-3 sentences.
   - Focus on a functional issues only; do not discuss performance improvements or stylistic concerns.
3. **Provide fixes for the bugs**:
   - For each bug, suggest a code fix by describing the change in a concise sentence.
   - You can specify a replacement, insertion, deletion, or modification of one or several line of code.
4. **Generate a Hint for the first bug:**
   - Provide a short and specific hint to help the student address the first identified bug.
   - The hint should encourage the student to think critically about resolving the issue without directly providing a solution or code fix.

Response format

Write your repair between <repair> and </repair> tags and your feedback within <feedback> </feedback> tags.

⓪

<repair>
repair
</repair>
<feedback>
feedback
</feedback>

②

Figure 4: **Generation and training prompt for supervised finetuning.** We provide: ⓪ a description of the repair task, and ① ask the LLM models to generate feedback ②. The full completion is then learned by the Repair-First-Feedback models using supervised finetuning.

You are a helpful assistant. Before answering a user, you first think and reason about a proper answer. You always put your thoughts within <think> </think> tags before providing the answer.

⓪

Below is a problem description and an incorrect program written by a student (i.e., it does not pass all test cases).

problem description, student code

Your task is to repair the student program so that it fulfils the problem description. Minimise modifications, keeping your repair as close as possible to the original incorrect program so that the student can better understand what was wrong. Put your repair within <answer> </answer> tags.

①

<think>
...
<think>
<answer>
repair
</answer>

②

Use your thought process and your repair to provide feedback to the student.

Tasks

Your tasks are as follows:

1. **Explain the bugs**:
   - Explain all the bugs in the student program in 1-3 sentences.
   - Focus on a functional issues only; do not discuss performance improvements or stylistic concerns.
2. **Provide fixes for the bugs**:
   - For each bug, suggest a code fix by describing the change in a concise sentence.
   - You can specify a replacement, insertion, deletion, or modification of one or several line of code.
3. **Generate a Hint for the first bug:**
   - Provide a short and specific hint to help the student address the first identified bug.
   - The hint should encourage the student to think critically about resolving the issue without directly providing a solution or code fix.

Response format

Write your answer within <feedback> </feedback> tags.

③

Figure 5: **Inference prompt: Repair before feedback.** We provide: ⓪ a system prompt that asks the language model to reason before answering (only provided for DRO and Reason-Repair models), ① a description of the repair task. We obtain the model generation ②, and then in the following turn (③) ask the model to generate feedback.

You are a computer science professor teaching introductory programming using Python. You are an expert at evaluating programming feedback tailored to novices.

⓪

Below is a problem description and an incorrect program written by a student (i.e., it does not pass all test cases).

problem description>, student code

Below is the feedback written by a teaching assistant (TA), which includes an explain and fixes for the bugs in the program. As well as a hint for the first bug.

feedback

Your task is to evaluate the quality of the TA's feedback according to the grading criteria outlined below.

grading criteria

This evaluation will be conducted in two parts

1. Reasoning: Reflect on the quality of the TA's feedback.

- Reflect on the quality of the feedback, using the grading criteria as a guide.
- Discuss strengths and weaknesses in the explanation and hint.

2. Grading List: Conclude with your final assessment for each criterion.

- If the criterion is fully met, respond with "true"; otherwise, respond with "false".

Please provide your answer using a JSON format with two keys:

- "reasoning": your detailed written analysis
- "grading": a dictionary with each criterion as a key and your final answer (true or false) as the value.

Use only `true` or `false` (no other qualifiers) for each grading criterion in the JSON output.

①

Figure 6: **Judging prompt.** We provide our three LLM judges with a ⓪ system description describing their role, ① a description of the judging task, and the specification of the response format in json.