On-Policy Optimization with Group Equivalent Preference for Multi-Programming Language Understanding

Haoyuan Wu[1*] **Rui Ming**[1*] **Jilong Gao**[1*] **Hangyu Zhao**[1] **Xueyi Chen**[1]
Yikai Yang[1] **Haisheng Zheng**[2] **Zhuolun He**[1,2] **Bei Yu**[1]
[1]The Chinese University of Hong Kong [2]ChatEDA Tech
{hywu24, byu}@cse.cuhk.edu.hk

Abstract

Large language models (LLMs) achieve remarkable performance in code generation tasks. However, a significant performance disparity persists between popular programming languages (e.g., Python, C++) and others. To address this capability gap, we leverage the code translation task to train LLMs, thereby facilitating the transfer of coding proficiency across diverse programming languages. Moreover, we introduce OORL for training, a novel reinforcement learning (RL) framework that integrates on-policy and off-policy strategies. Within OORL, on-policy RL is applied during code translation, guided by a rule-based reward signal derived from unit tests. Complementing this coarse-grained rule-based reward, we propose Group Equivalent Preference Optimization (GEPO), a novel preference optimization method. Specifically, GEPO trains the LLM using intermediate representations (IRs) groups. LLMs can be guided to discern IRs equivalent to the source code from inequivalent ones, while also utilizing signals about the mutual equivalence between IRs within the group. This process allows LLMs to capture nuanced aspects of code functionality. By employing OORL for training with code translation tasks, LLMs improve their recognition of code functionality and their understanding of the relationships between code implemented in different languages. Extensive experiments demonstrate that our OORL for LLMs training with code translation tasks achieves significant performance improvements on code benchmarks across multiple programming languages.

1 Introduction

With the rapid advancement of LLMs [2], code-specific language models have garnered significant attention within the research community. Building upon pre-trained LLMs, code LLMs such as the StarCoder series [12, 16], CodeLlama series [23], DeepSeekCoder series [27], QwenCoder [20] series, and CodeStral [18] have demonstrated superior performance in code generation tasks. However, although current state-of-the-art code generation LLMs excel at generating code using popular programming languages (e.g., Python, C++), their performance diminishes when applied to solving the same problems using other programming languages [19]. This capability gap limits the universal applicability of these powerful LLMs and hinders developers working in diverse programming language ecosystems.

Consequently, it is crucial to enhance LLMs' proficiency across a wider range of programming languages to match their performance in widely used ones [26, 19]. We address this challenge by training LLMs on code translation tasks, requiring them to translate code from one programming

[*]Equal Contribution

38th Conference on Neural Information Processing Systems (NeurIPS 2025).

language to another accurately. Intuitively, if LLMs can accurately translate Python code into other programming languages, they can achieve comparable performance in Python code generation tasks in those programming languages. Moreover, focusing on code translation facilitates the LLM's understanding of inter-language similarities and differences and allows it to learn from exemplary code structures.

Recently, on-policy RL algorithms [24, 1, 25, 14, 9] utilizing rule-based rewards have demonstrated significant performance in code generation tasks [7, 6]. However, the inherent coarse-grained nature of these rule-based rewards, lacking process-level supervision, limits their effectiveness in guiding LLMs to recognize the functional equivalence of intermediate code components during training. Implementing fine-grained process-level rewards is challenging, particularly given that multiple valid implementations may exist for the same code functionality. Given that leveraging preference data can guide LLM behavior towards desired outcomes, preference optimization [21, 3] offers an opportunity to incorporate process-level constraints during training.

Functional equivalence is crucial during the training process for code translation tasks. High-level programming languages can be too abstract for effective functional understanding by LLMs, especially for less common ones [26]. Consequently, interlingual intermediate representations (IRs) can be leveraged to facilitate cross-language transfer from high- to low(er)-resource programming languages [19, 10, 26]. Compiler IRs are agnostic to the source programming language and target execution platform, providing a method to align constructs from different programming languages semantically [5]. Therefore, we construct groups of IRs for the preference optimization process. By guiding the LLM to distinguish IRs equivalent to the source code from inequivalent ones and utilizing signals about the mutual equivalence between IRs within the group, the LLM can capture fine-grained information regarding code functionality.

This study introduces OORL, a novel RL framework integrating on-policy and off-policy strategies for training. First, we employ on-policy RL with a binary, rule-based reward signal. This component ensures the fundamental correctness of the code translation, verifying that the generated code adheres to formatting requirements, compiles successfully, and passes predefined unit tests. This provides a strong, unambiguous signal for task completion. Second, we introduce group equivalent preference optimization (GEPO) to incorporate finer-grained quality distinctions and address the unique nature of code generation, where multiple equivalences can exist. GEPO is a novel preference optimization method that extends beyond traditional pairwise comparisons. It leverages preference data comparing groups of "winner" (equivalent IRs) and "loser" (inequivalent IRs) responses. Crucially, GEPO explicitly models the concept of functional equivalence within the winner group, guiding the model to recognize that multiple distinct IRs can be equally valid. This allows the model to learn nuanced aspects of code functionality across multiple programming languages.

Our contributions can be summarized as follows:

- Introduce OORL, a novel RL framework for training with code translation tasks to enhance LLMs' understanding across multiple programming languages.
- Develop GEPO, a novel preference optimization method that extends beyond pairwise comparisons by explicitly modeling equivalence within IRs groups.
- Conduct extensive evaluations demonstrating the superior performance of our methods across various code benchmarks using multiple programming languages.

2 Preliminaries

2.1 RL with an Explicit Reward Function

RL [24, 7] has been widely used in preference optimization for LLMs. Let π_{ref} denote an LLM obtained after supervised fine-tuning (SFT). RL optimizes the policy model π_θ, typically initialized from π_{ref} with parameters θ, by maximizing an expected reward signal. This requires defining a reward function $R(x, y)$ that assigns a scalar score to a complete response y given the prompt x. This score reflects the accuracy, quality, or alignment of the response with human preferences.

Once the reward function $R(x, y)$ is established, an RL algorithm can be employed to optimize the policy model π_θ. The objective is typically to maximize the expected reward, often incorporating a KL divergence penalty term to prevent π_θ from deviating too far from the initial reference policy

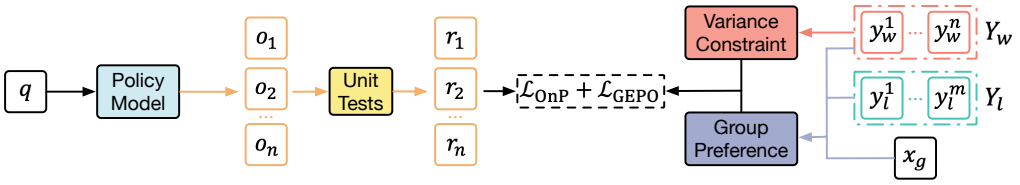

Figure 1: Overview of the OORL integrating on-policy RL with the rule-based reward and on-policy preference optimization (GEPO).

model π_{ref}, thereby maintaining generative capabilities and diversity:

$$\max_{\theta} \mathbb{E}_{x \sim D_{\text{prompt}}} \mathbb{E}_{y \sim \pi_\theta(\cdot|x)} [R(x, y) - \beta \mathbb{D}_{\text{KL}}(\pi_\theta(\cdot|x) || \pi_{\text{ref}}(\cdot|x))], \tag{1}$$

where D_{prompt} is a dataset of prompts and β is a hyperparameter controlling the KL penalty.

2.2 Direct Preference Optimization

Direct Preference Optimization (DPO) [21] is an alternative approach that leverages the preference dataset D_{pref} to directly optimize the policy model π_θ without the need for explicitly training a separate reward model. DPO establishes a direct link between preference probabilities and policy probabilities, deriving an objective equivalent to optimizing against a learned reward model within the RLHF framework. Specifically, DPO assumes an implicit reward function $r_\phi(x, y)$ exists such that the human preference data can be modeled via the Bradley-Terry model:

$$P(y_w \succ y_l|x) = \sigma(r_\phi(x, y_w) - r_\phi(x, y_l)), \tag{2}$$

where $\succ$ denotes "is preferred over". Theoretical derivation shows that the RL objective of aligning with this implicit reward (including the KL penalty) can be transformed into a maximum likelihood objective computed directly on the preference data. The DPO loss function is defined as:

$$\mathcal{L}_{\text{DPO}}(\theta; \pi_{\text{ref}}) = -\mathbb{E}_{(x, y_w, y_l) \sim D_{\text{pref}}} [\log \sigma(\beta \log \frac{\pi_\theta(y_w|x)}{\pi_{\text{ref}}(y_w|x)} - \beta \log \frac{\pi_\theta(y_l|x)}{\pi_{\text{ref}}(y_l|x)})], \tag{3}$$

where π_θ is the policy model being optimized, π_{ref} is the reference policy model, and β is a scaling factor for the implicit reward.

3 Methods

Integrating on-policy (Section 3.1) and off-policy (Section 3.2) strategies, as illustrated in Figure 1, we introduce OORL (Section 3.3), a novel RL framework for multi-programming language understanding.

3.1 On-policy RL for Code Translation

Although LLMs achieve remarkable performance in code generation tasks, a significant performance disparity persists between high-resource programming languages (e.g., Python, C++) and others. To address this capability gap, we leverage the code translation task to train LLMs, thereby facilitating the transfer of coding proficiency across diverse programming languages. For example, if LLMs can accurately translate Python code into other programming languages, they can achieve comparable performance in Python code generation tasks while using those programming languages.

RL with Binary Code Translation Reward. Recently, on-policy RL algorithms utilizing rule-based rewards have demonstrated significant performance in code generation tasks [7, 6]. Consequently, we apply on-policy RL to training with code translation tasks. Let's first define the RL setup for the code translation task: (1) **State** (s): The state typically consists of the input q with source code and potentially the sequence of tokens generated so far $o_{<t}$ with translated code; (2) **Action** (a): The action corresponds to the generation of the next token o_t by the policy model; (3) **Policy** (π_θ): The policy model $\pi_\theta(a|s)$ is the LLM being optimized, parameterized by θ. It's typically initialized from

Figure 2: Overview of the equivalent and equivalent IRs groups for the training process with GEPO. The equivalent IRs group is constructed from different LLVM optimization levels (e.g., -Oz, -O3) of the source code. The inequivalent IRs group is augmented from the equivalent IRs group.

the SFT reference model π_{ref}. During the on-policy RL process, given the query q and the trajectory o, we employ a binary reward function, $R_{\text{rule}}(q, o)$, assigned upon completion of the generation o:

$$R_{\text{rule}}(q, o) = \begin{cases} 1, & \text{if } o \text{ represents a successful code translation from } q, \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

Specifically, successful translation demonstrates that o must adhere to the standard format specified in q and the translated code extracted from o can be successfully compiled and pass unit tests. The policy model π_θ is optimized using an on-policy RL algorithm (e.g., PPO, GRPO) to optimize Equation (1). The on-policy RL algorithm operates by iteratively sampling trajectories (q, o) from the current policy model π_θ. For each action in each sampled trajectory, it estimates the advantage A_t, which quantifies how much better the received reward R_t is compared to an expected baseline. The policy parameters θ are then updated using a clipped objective function designed to maximize the expected advantage while limiting the change in the π_θ at each step. The on-policy RL loss can be defined as:

$$\mathcal{L}_{\text{OnP}} = -\mathbb{E}_{(q,o)\sim\pi_\theta}[\min(r_t(\pi_\theta)A_t, \text{clip}(r_t(\pi_\theta), 1-\epsilon, 1+\epsilon)A_t)], \tag{5}$$

where $r_t(\pi_\theta) = \frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}$ is the probability ratio between new and old policies. ϵ is a hyperparameter defining the clipping range. This rule-based RL component, focused on maximizing the binary success signal defined in Equation (4). It provides a strong, unambiguous signal for task completion, which is complemented by the finer-grained preference information incorporated through the GEPO loss, as described in the subsequent section.

3.2 Group Equivalent Preference Optimization with IRs

The inherent coarse-grained nature of rule-based rewards, which lack process-level supervision, limits their effectiveness in guiding LLMs to recognize the functional equivalence of intermediate code components. Implementing fine-grained process-level rewards is challenging, particularly as multiple valid implementations can exist for the same code functionality. Given that leveraging preference data can effectively guide LLM behavior towards desired outcomes, preference optimization [21, 3] offers a promising avenue to incorporate process-level constraints during training.

Functional equivalence is crucial in code generation scenarios. However, the functions implemented with high-level programming languages can be too abstract for LLMs to understand, especially

for less common ones. In contrast, compiler IRs expose detailed and low-level operators that are significantly easier for LLMs to understand [26]. Consequently, we propose performing preference optimization using IR translation tasks, which require LLMs to translate code written in high-level programming languages into IRs. This process facilitates the LLM's understanding of the code functionality expressed in the high-level languages. Furthermore, equivalent IRs resulting from different compiler optimization stages can guide LLMs in discerning the equivalent functionality of different code implementations.

However, traditional preference optimization methods [21, 3] typically rely on pairwise comparisons, which are insufficient in this context. Specifically, methods like DPO [21] do not account for the equivalence among IRs. Therefore, we introduce Group Equivalent Preference Optimization (GEPO) and construct groups D_{gpref} with equivalent and inequivalent IRs for the preference optimization process. GEPO compares groups of IR responses and specifically incorporates the concept of equivalence within the group of preferred ("winner") IR responses. By guiding the LLM to distinguish IRs equivalent to the source code from inequivalent ones and utilizing signals about the mutual equivalence between IRs within a group, GEPO enables the LLM to capture detailed information regarding code functionality.

For each prompt x_g with source code in D_{gpref} for GEPO, there exists a group of "winner" responses $Y_w = \{y_w^1, y_w^2, \cdots, y_w^n\}$, a group of "loser" responses $Y_l = \{y_l^1, y_l^2, \cdots, y_l^m\}$ and $y_g \in (Y_w \cup Y_l)$. Specifically, Y_w represents n functionally equivalent IRs while Y_l represents m inequivalent IRs. Following [21], we assume an implicit reward function $r_\phi(x_g, y_g)$ underlies the preferences. Instead of comparing the individual winner IR response and the loser IR response, GEPO compares the average reward of Y_w to the average reward of Y_l. Furthermore, it explicitly enforces equivalence within Y_w by constraining the variance of their rewards. The initial optimization problem for GEPO, formulated in terms of the implicit reward r_ϕ, can be defined as:

$$\begin{aligned}
&\min_\phi \mathcal{L}_{\text{GEPO}} = -\mathbb{E}_{(x_g, Y_w, Y_l)\sim D_{\text{gpref}}}[\log \sigma(\frac{1}{n}\sum_{i=1}^{n} r_\phi(x_g, y_w^i) - \frac{1}{m}\sum_{k=1}^{m} r_\phi(x_g, y_l^k))] \\
&\text{s.t. } \mathbb{E}_{(x_g, Y_w)\sim D_{\text{gpref}}}[\text{Var}(r_\phi(x_g, Y_w))] < \epsilon, \\
&\quad \text{Var}(r_\phi(x_g, Y_w)) = \frac{1}{n-1}\sum_{i=1}^{n}(r_\phi(x_g, y_w^i) - \frac{1}{n}\sum_{j=1}^{n} r_\phi(x_g, y_w^j))^2.
\end{aligned} \tag{6}$$

The primary objective term maximizes the probability that the average winner reward exceeds the average loser reward, modeled using the sigmoid function. The constraint bounds the variance of rewards within the winner group Y_w by a small threshold ϵ, encouraging $r_\phi(x_g, y_w^i) \approx r_\phi(x_g, y_w^j)$ for all $y_w^i, y_w^j \in Y_w$.

According to Appendix A, we can establish a direct relationship between the implicit reward function r_ϕ and the optimal policy π_θ being optimized, relative to the reference policy model π_{ref}. Accordingly, the optimal policy model π^* that maximizes Equation (1) can be given by $\pi^*(y_g|x_g) \propto \pi_{\text{ref}}(y_g|x_g)\exp(\frac{1}{\beta}r_\phi(x_g, y_g))$. This implies that the reward function can be expressed in terms of the policy probabilities, up to a partition function $Z(x_g)$, as follows:

$$r_\phi(x_g, y_g) = \beta \log \frac{\pi_\theta(y_g|x_g)}{\pi_{ref}(y_g|x_g)} + \beta \log Z(x_g), \tag{7}$$

where π_θ is the policy model to be optimized. Substituting this expression for $r_\phi(x_g, y_g)$ into Equation (6), the partition function terms $\beta \log Z(x_g)$ cancel out within the reward differences and the variance calculation. This yields an objective function that depends on the policy model π_θ, the reference policy model π_{ref}, and the preference data (x_g, Y_w, Y_l), which can be formulated as follows:

$$\begin{aligned}
&\min_\theta \mathcal{L}'_{\text{GEPO}} = -\mathbb{E}_{(x_g, Y_w, Y_l)\sim D_{\text{gpref}}}[\log \sigma(\frac{\beta}{n}\sum_{i=1}^{n} \hat{r}_\theta(x_g, y_w^i) - \frac{\beta}{m}\sum_{k=1}^{m} \hat{r}_\theta(x_g, y_l^k))] \\
&\text{s.t. } \mathbb{E}_{(x_g, Y_w)\sim D_{\text{gpref}}}[\frac{\beta^2}{n-1}\sum_{i=1}^{n}(\hat{r}_\theta(x_g, y_w^i) - \frac{1}{n}\sum_{j=1}^{n} \hat{r}_\theta(x_g, y_w^j))^2] < \epsilon', \\
&\quad \hat{r}_\theta(x_g, y_g) = \log \frac{\pi_\theta(y_g|x_g)}{\pi_{\text{ref}}(y_g|x_g)}.
\end{aligned} \tag{8}$$

Here, $\hat{r}_\theta(x_g, y_g)$ represents the log-probability ratio proportional to the implicit reward score. The constraint now applies to the variance of these log-probability ratios within the winner group. To solve this constrained optimization problem, we introduce a Lagrange multiplier $\lambda \geq 0$ for the variance constraint, transforming it into an unconstrained objective. The GEPO loss can be formulated as:

$$\mathcal{L}_{\text{GEPO}} = -\mathbb{E}_{(x_g, Y_w, Y_l)\sim D_{\text{gpref}}}[\log \sigma(\beta(\frac{1}{n}\sum_{i=1}^{n}\hat{r}_\theta(x_g, y_w^i) - \frac{1}{m}\sum_{k=1}^{m}\hat{r}_\theta(x_g, y_l^k)))] + \lambda\mathbb{E}_{(x_g, Y_w)\sim D_{\text{gpref}}}[\frac{\beta^2}{n-1}\sum_{i=1}^{n}(\hat{r}_\theta(x_g, y_w^i) - \frac{1}{n}\sum_{j=1}^{n}\hat{r}_\theta(x_g, y_w^j))^2]. \tag{9}$$

Finally, $\mathcal{L}_{\text{GEPO}}$ consists of two terms. The first term is analogous to the DPO loss but operates on the average log-probability ratios of the winner and loser groups, pushing the model to prefer winners over losers on average. The second term, weighted by the hyperparameter λ, explicitly penalizes the variance of the log-probability ratios within the winner group Y_w. Minimizing this term encourages $r_\phi(x_g, y_g)$ to be similar for all winners $y_w \in Y_w$, thereby enforcing the desired equivalence. The hyperparameter $\lambda = 1$ controls the strength of this equivalence regularization relative to the main preference objective. By minimizing $\mathcal{L}_{\text{GEPO}}$, we directly optimize the policy model π_θ to enhance the likelihood of high-quality, functionally equivalent code translation (Y_w) compared to undesired ones (Y_l), while simultaneously promoting diversity and consistency among the preferred solutions.

3.3 OORL

Building upon the on-policy RL strategy and GEPO, we develop OORL, a novel RL framework for training with code translation tasks, as shown in Figure 1. Specifically, following the on-policy nature used for the RL part, we periodically sample trajectories (q, o) using the current policy model π_θ and compute the binary reward $R_{\text{rule}}(q, o)$. This allows the estimation of advantages A_t and the computation of the on-policy RL objective $\mathcal{L}_{\text{OnP}}(\theta)$, which drives the policy model towards successful task completion according to the defined rules. In parallel, we utilize the static preference dataset $D_{\text{grefs}} = \{(x_g, Y_w, Y_l)\}$ containing groups of winner and loser responses to calculate $\mathcal{L}_{\text{GEPO}}(\pi_\theta; \pi_{\text{ref}})$, which guides the policy model to align with fine-grained function equivalences among desirable IRs.

The parameters θ of π_θ are updated using a combined signal derived from both objectives. A conceptual representation of the combined objective to be minimized during policy updates can be expressed as:

$$\mathcal{L} = w_{\text{rl}} \cdot \mathcal{L}_{\text{OnP}} + w_{\text{gepo}} \cdot \mathcal{L}_{\text{GEPO}}, \tag{10}$$

where the non-negative w_{rl} and w_{gepo} serve as weights that balance the contribution of the rule-based task completion signal from RL against the nuanced quality and equivalence signal from GEPO. Specifically, we set $w_{\text{rl}} = 1$ and $w_{\text{gepo}} = 0.01$ for the training process with OORL.

4 Experiments

4.1 Experiment Settings

Training Data. The training dataset for OORL is constructed separately. For on-policy RL, we construct 2400 code translation problems with unit tests from the SYNTHETIC-1 [17] and APPS [8] datasets. Specifically, this batch of training data includes translation from Python and C++ as source codes to Python, C++, Java, PHP, Bash, and JavaScript as target codes. Meanwhile, GEPO is performed with 9600 curated C-to-IR groups from the SLTrans dataset [19].

Implementation Details. In this paper, we implement our OORL framework based on Qwen3-8B [20]. Specifically, we employ REINFORECE++ [1, 9] as the on-policy RL algorithm with OORL. The training process entailed using a cosine learning rate schedule with a warm-up ratio of 0.03, and the AdamW [15] optimizer with a learning rate of 5×10^{-7}, no weight decay, a batch size of 8, and a sequence length of 8192 tokens. The models underwent instruction tuning for one epoch using DeepSpeed-Zero stage2 with offload [22] on 4 A100 GPUs, each with 80G memory.

Baselines. Our Qwen3-8B-OORL, trained with the OORL framework, is compared against existing LLMs with comparable parameters. These include DS-Coder-V2-Lite-Inst [27], Qwen2.5-Coder-7B-

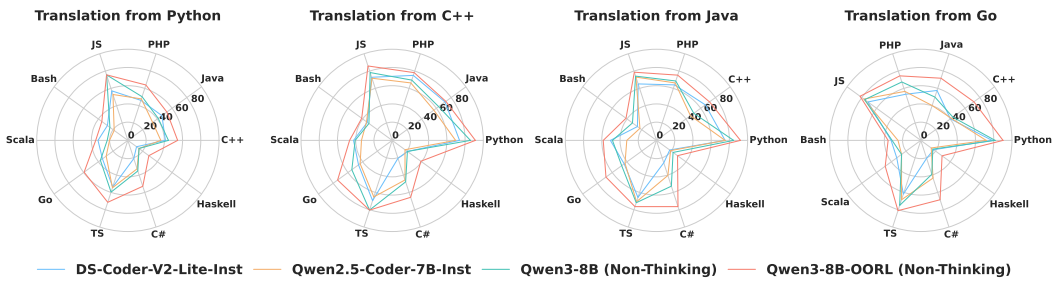

Figure 3: Code Translation Performance of LLMs on CrossPLEval.

Table 1: Statistics in the CrossPLEval Benchmark.

Programming Language	# Solution Files	Avg. Lines of Code (LoC)	Proportion of Files (%)
MultiPL-E(Python)	–	7.81	–
Python	119×10	19.94	30.36
C++	92×10	35.49	23.47
Go	92×10	47.60	23.47
Java	89×10	39.72	22.70
Total / Weighted Avg.	392×10	34.57	100.00

Avg. LoC for the total is a weighted average. Proportions are based on the total of 3920 files.

Inst [11], and Qwen3-8B [20]. Specifically, the experiments with Qwen3-8B are conducted using its non-thinking mode.

Evaluation Benchmarks. To comprehensively evaluate proficiency in multi-programming languages understanding, we utilize the MultiPL-E [4] benchmark and select eight mainstream languages for evaluation, including Python, C++, Java, PHP, TypeScript, C#, Bash, and JavaScript. In addition, we develop CrossPLEval, a new multilingual code translation benchmark, for further evaluation. CrossPLEval comprises 119 programming problems carefully selected from the TACO dataset [13], with details illustrated in Section 4.2. For evaluation, we use the pass@1 rate as the metric for MultiPL-E and CrossPLEval benchmarks.

4.2 CrossPLEval

CrossPLEval is designed to assess the ability of models to translate a canonical solution of a problem from a source language to a target language. Specifically, the model receives the function declaration and the canonical solution in the source language (e.g., Python) as input and is tasked with generating the corresponding solution in the target language. To facilitate evaluation, the target language function declaration is also provided, which helps constrain function names and variable types.

The 119 core problems in CrossPLEval originate from the TACO dataset [13], initially defined in Python. Recognizing the non-trivial nature of developing high-quality, semantically aligned test data across languages, we undertook meticulous manual processing for these problems to ensure data diversity and quality. This process involved two main efforts. First, concerning canonical solutions, while reference Python solutions were available for all 119 problems, we manually rewrote these into equivalent implementations in other mainstream programming languages. This yielded 89 solutions for Java, 92 for Go, and 92 for C++, resulting in 273 high-quality translation task instances from Python to these three target languages. Second, to ensure rigorous evaluation of functional correctness, we manually rewrote the original test cases for each problem into ten different programming languages, including C++, Java, PHP, JavaScript, Bash, Scala, Go, TypeScript, C#, and Haskell. These multilingual test cases enable verification of translation accuracy across a broad linguistic spectrum. Through this construction process, the CrossPLEval benchmark comprises 3920 independent coding problems, each of which includes at least 5 unit tests, thereby providing a solid foundation for evaluation. The details are described in Table 1.

Table 2: Performance of different LLMs on MultiPL-E [4].

	Python	C++	Java	PHP	Bash	JS	TS	C#	Avg.
DS-Coder-V2-Lite-Inst	81.10	32.91	68.35	72.67	19.62	81.36	83.33	41.13	60.06
Qwen2.5-Coder-7B-Inst	88.40	57.14	70.88	73.91	43.67	78.88	84.27	52.53	68.71
Qwen3-8B	82.60	71.42	70.25	50.31	36.07	83.22	84.27	42.41	65.07
Qwen3-8B-OORL	**90.06**	**83.23**	**83.54**	**78.26**	**46.20**	**89.44**	**85.53**	**54.22**	**76.31**

Table 3: Results of code translation compared with other LLMs on CrossPLEval.

	Model	Target Language											
		Python	C++	Java	PHP	JS	Bash	Scala	Go	TS	C#	Haskell	Avg.
Py	DS-Coder-V2-Lite-Inst	-	41.17	46.21	46.21	57.14	27.73	32.77	34.45	53.78	21.00	11.76	37.22
	Qwen2.5-Coder-7B-Inst	-	36.13	33.61	47.89	52.94	18.54	20.16	29.41	54.62	32.77	15.12	34.12
	Qwen3-8B	-	44.53	41.17	50.42	75.54	24.36	22.68	37.25	59.66	35.29	15.13	40.60
	Qwen3-8B-OORL	-	**54.62**	**52.94**	**63.94**	**75.63**	**35.29**	**35.29**	**59.32**	**71.42**	**52.94**	**28.48**	**52.99**
C++	DS-Coder-V2-Lite-Inst	73.91	-	71.73	75.00	72.82	33.69	42.39	45.65	69.56	20.65	20.65	52.61
	Qwen2.5-Coder-7B-Inst	69.56	-	56.52	66.30	71.73	34.78	33.69	43.47	65.21	45.65	17.39	50.43
	Qwen3-8B	85.86	-	63.04	69.56	78.26	31.52	40.65	54.71	80.43	47.82	20.65	57.25
	Qwen3-8B-OORL	**91.30**	-	**72.82**	**78.15**	**85.86**	**41.30**	**46.73**	**74.02**	**80.43**	**65.65**	**38.91**	**67.52**
Java	DS-Coder-V2-Lite-Inst	75.28	66.29	-	64.04	65.16	25.84	50.56	43.82	65.82	25.84	17.97	50.06
	Qwen2.5-Coder-7B-Inst	75.28	46.06	-	66.29	73.03	22.47	32.58	42.69	70.78	40.44	19.10	48.87
	Qwen3-8B	84.64	50.56	-	68.53	74.15	32.58	48.31	47.19	71.91	52.80	22.47	55.31
	Qwen3-8B-OORL	**92.13**	**71.91**	-	**75.28**	**78.65**	**41.34**	**58.42**	**68.53**	**76.40**	**76.40**	**28.53**	**66.76**
Go	DS-Coder-V2-Lite-Inst	78.26	40.21	57.60	53.26	71.73	32.60	32.60	-	61.95	20.65	17.39	46.63
	Qwen2.5-Coder-7B-Inst	71.73	41.30	40.21	56.52	77.17	25.00	26.08	-	68.47	43.47	14.13	46.41
	Qwen3-8B	81.52	43.47	50.00	67.39	76.08	31.52	26.08	-	75.00	39.13	16.30	50.65
	Qwen3-8B-OORL	**90.21**	**72.06**	**71.73**	**74.45**	**82.39**	**35.86**	**48.26**	-	**81.08**	**68.47**	**28.80**	**65.33**

To illustrate the difficulty of CrossPLEval across different languages, we visualize the code translation performance in Figure 3. Current SOTA code LLMs with around 8B parameters achieve a low score on average performance across multiple programming languages.

4.3 Excellent Multi-Programming Language Understanding

To evaluate proficiency in understanding and generating code across multiple programming languages, we conduct evaluations using the established MultiPL-E [4] benchmark and our newly introduced CrossPLEval benchmark. Besides Python, C++, Java, PHP, Bash, and JavaScript, we also include Scala, Go, TypeScript, C#, and Haskell in our evaluation, noting that the latter five languages are not included in the training dataset. Moreover, Bash, Scala, and Haskell are particularly noteworthy as low-resource programming languages.

Table 2 presents the code generation performance on MultiPL-E, while Table 3 and Figure 3 show code translation results on CrossPLEval. Across both benchmarks and all tested models, our Qwen3-8B-OORL model consistently demonstrates superior performance. On MultiPL-E, it achieves the highest average score of 76.31, significantly outperforming the second highest Qwen2.5-Coder-7B-Inst with a significant margin. This strong performance extends across all eight languages evaluated in MultiPL-E, which are generally well-represented in large code corpora. This highlights the excellent ability to generate correct code from natural language instructions in these programming languages.

Furthermore, the CrossPLEval results demonstrate the code translation capabilities and, critically, its generalization ability. Qwen3-8B-OORL achieves the highest average translation scores across all source languages evaluated. The CrossPLEval benchmark is particularly valuable for evaluating performance on languages potentially less represented in training data, such as Scala, Go, TypeScript, C#, and the low-resource Haskell. Although absolute translation scores into target languages like Bash, Scala, and Haskell are lower compared to more common targets, our Qwen3-8B-OORL exhibits significant relative improvements over the baselines on these more challenging targets. For instance, when translating from Python to Haskell, Qwen3-8B-OORL scores 28.48, representing a substantial

Table 4: Ablation studies on various benchmarks using OORL.

	On-Policy RL Strategy	Off-Policy RL Strategy	MultiPL-E	CrossPLEval				
				Python	C++	Java	Go	Avg.
Qwen3-8B	-	-	65.06	40.60	57.25	55.32	50.65	50.96
	REINFORCE++	-	73.60	47.56	63.26	62.13	60.87	58.46
	REINFORCE++	DPO	73.48	50.93	62.93	62.80	58.15	58.70
	REINFORCE++	GEPO	**76.31**	**52.99**	**67.52**	**66.76**	**65.33**	**63.15**

gain over the Qwen3-8B. These results underscore the effective generalization beyond its direct training data and its ability to provide significant performance boosts for tasks involving low-resource programming languages.

The evaluation results demonstrate that Qwen3-8B-OORL not only excels in core code generation tasks on well-represented languages but also effectively translates and generalizes to improve performance on untrained and low-resource programming languages, marking a notable advancement in multilingual code understanding and generation.

4.4 Ablation Studies

To evaluate the individual contribution of each key component within our OORL framework, we conducted thorough ablation studies. The results of these studies are detailed in Table 4. The investigation systematically deconstructed OORL by starting with the Qwen3-8B model and incrementally adding the on-policy RL strategy (REINFORCE++), followed by a comparison of different off-policy preference optimization strategies, specifically DPO [21] and our proposed GEPO.

On-Policy RL Strategy. As illustrated in Table 4, training with code translation tasks using the REINFORCE++ [9, 1] algorithm increases the MultiPL-E score to 73.60 and the average CrossPLEval score to 58.46, demonstrating a significant performance improvement. This substantial gain demonstrates the effectiveness of on-policy RL, which directly refines multi-programming language understanding of LLMs through the rule-based reward derived from unit tests.

Off-Policy RL Strategy. As shown in Table 4, the combination of REINFORCE++ [9, 1] and DPO [21] only offers a marginal improvement on CrossPLEval. The effectiveness of DPO in code understanding is found to be limited. This limitation stems from the difficulty standard preference optimization methods face in fully understanding the functional equivalence between different code implementations. In contrast, our GEPO strategy is specifically designed to utilize functional equivalence information from IRs. It's worth noting that integrating GEPO with REINFORCE++ achieved significantly higher performance, reaching 76.31 on MultiPL-E and a notable average of 63.15 on CrossPLEval. This demonstrates the ability of our GEPO to leverage functional equivalence, providing substantial additional performance gains over standard off-policy methods in the context of multi-programming language understanding.

5 Related Works

5.1 RL for LLMs

RL [24, 25, 21, 3] has emerged as a prominent paradigm for aligning LLMs with desired behaviors and preferences. Investigations in this area have explored both on-policy and off-policy RL algorithms. On-policy methods [24, 1, 25, 14, 9] have received considerable attention due to their inherent stability and effectiveness in directly optimizing the policy using data collected under the current policy. For instance, PPO [24] has been widely adopted for fine-tuning LLMs based on human preference data. Similarly, REINFORCE-based methods, such as ReMax [14], RLOO [1], and GRPO [25], have been extensively employed in extending mathematical and general reasoning abilities, often utilizing rule-based rewards exclusively. Concurrently, off-policy algorithms offer potential advantages in terms of sample efficiency by allowing the agent to learn from prior experiences or data generated by different policies. DPO [21] and IPO [3] have been proposed as more stable and efficient RL techniques that implicitly learn a reward function from preference data and directly optimize the policy. These methods can be conceptualized as implicitly performing off-policy evaluation and

optimization. Building upon the successes of both on-policy and off-policy RL, we introduce OORL, a novel RL framework that integrates on-policy and off-policy strategies for training. We also incorporate GEPO, a novel off-policy method that leverages the concept of functional equivalence, to provide a more nuanced and effective approach for training LLMs in code translation tasks.

5.2 Code Understanding with IRs

Intermediate Representations (IRs) have been increasingly leveraged in recent work to enhance LLMs for code generation tasks. The language-agnostic nature of IRs allows LLMs to abstract away from specific syntax and focus more effectively on program logic. For instance, to improve multilingual code generation and cross-lingual transfer, IRCoder [19] employs continued pre-training of LLMs on SLTrans, a parallel dataset of source code and its corresponding LLVM IR, thereby aligning code and IR semantics. Similarly, Transcoder-IR[26] also utilizes LLVM IR, augmenting LLM training by exploring it as a pivot language. Acknowledging the potential complexities of standard compiler IRs for translation tasks, CoDist [10] proposes a custom, distilled code as a simplified intermediate representation and translation pivot. Furthermore, for compiler-specific applications, LLM Compiler [5] is specialized through extensive further pre-training of Code LLMs directly on large corpora of LLVM IR and assembly code. This specialization aims to enhance performance on tasks such as code optimization and disassembly. In this paper, we adopt a unique approach by leveraging LLVM IRs within our GEPO preference optimization process. We construct and compare groups of IRs derived from translated code. This method guides the LLM to discern fine-grained functional equivalence, which is crucial for effective cross-lingual code understanding.

6 Conclusion

In this paper, we introduce OORL, a novel RL framework that integrates both on-policy and off-policy strategies for training LLMs. Within the OORL framework, on-policy RL is applied during the code translation process, guided by a rule-based reward signal derived from unit tests. Complementing this coarse-grained rule-based reward, we propose Group Equivalent Preference Optimization (GEPO), a novel preference optimization method. GEPO specifically extends beyond traditional pairwise comparisons by explicitly modeling equivalence within groups of IRs. Extensive experiments demonstrate that our OORL framework achieves significant performance improvements on code benchmarks across multiple programming languages, highlighting its effectiveness for enhancing multilingual code understanding and generation. In general, this work presents a novel integrated RL approach that leverages functional equivalence and offers a new perspective for advancing LLMs' understanding capabilities across diverse programming languages.

Limitations

A primary limitation of OORL stems from the group-based optimization mechanism in GEPO, which processes grouped preference responses for each source prompt. Compared to previous preference optimization methods [21, 3] that handle a pair of responses per step, GEPO manages larger groups of preference data, necessitating more GPU memory resources. This increased memory can be particularly pronounced with larger group sizes or longer response sequences. One potential strategy to mitigate this issue is to serialize the processing of individual responses within each preference group, which can reduce the number of padding tokens and thereby lower memory overhead. Although the serialization is beneficial for memory efficiency, it introduces an increase in training latency due to the sequential handling of responses previously processed in parallel within the group.

Acknowledgement

This work is partially supported by The Research Grants Council of Hong Kong SAR (No. RFS2425-4S02 and No. CUHK14201624).

References

[1] A. Ahmadian, C. Cremer, M. Gallé, M. Fadaee, J. Kreutzer, O. Pietquin, A. Üstün, and S. Hooker. Back to basics: Revisiting reinforce style optimization for learning from human feedback in llms. *arXiv preprint arXiv:2402.14740*, 2024.

[2] Anthropic. Claude Sonnet. https://www.anthropic.com/claude/sonnet, 2025.

[3] M. G. Azar, Z. D. Guo, B. Piot, R. Munos, M. Rowland, M. Valko, and D. Calandriello. A general theoretical paradigm to understand learning from human preferences. In *International Conference on Artificial Intelligence and Statistics*, pages 4447–4455. PMLR, 2024.

[4] F. Cassano, J. Gouwar, D. Nguyen, S. Nguyen, L. Phipps-Costin, D. Pinckney, M.-H. Yee, Y. Zi, C. J. Anderson, M. Q. Feldman, et al. MultiPL-E: A scalable and extensible approach to benchmarking neural code generation. *arXiv preprint arXiv:2208.08227*, 2022.

[5] C. Cummins, V. Seeker, D. Grubisic, B. Roziere, J. Gehring, G. Synnaeve, and H. Leather. LLM Compiler: Foundation Language Models for Compiler Optimization. In *Proceedings of the 34th ACM SIGPLAN International Conference on Compiler Construction*, pages 141–153, 2025.

[6] A. Du, B. Gao, B. Xing, C. Jiang, C. Chen, C. Li, C. Xiao, C. Du, C. Liao, et al. Kimi L1. 5: Scaling reinforcement learning with LLMs. *arXiv preprint arXiv:2501.12599*, 2025.

[7] D. Guo, D. Yang, H. Zhang, J. Song, R. Zhang, R. Xu, Q. Zhu, S. Ma, P. Wang, X. Bi, et al. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

[8] D. Hendrycks, S. Basart, S. Kadavath, M. Mazeika, A. Arora, E. Guo, C. Burns, S. Puranik, H. He, D. Song, et al. Measuring coding challenge competence with apps. *arXiv preprint arXiv:2105.09938*, 2021.

[9] J. Hu. REINFORCE++: A Simple and Efficient Approach for Aligning Large Language Models. *arXiv preprint arXiv:2501.03262*, 2025.

[10] Y. Huang, M. Qi, Y. Yao, M. Wang, B. Gu, C. Clement, and N. Sundaresan. Program translation via code distillation. *arXiv preprint arXiv:2310.11476*, 2023.

[11] B. Hui, J. Yang, Z. Cui, J. Yang, D. Liu, L. Zhang, T. Liu, J. Zhang, B. Yu, K. Lu, et al. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*, 2024.

[12] R. Li, L. B. Allal, Y. Zi, N. Muennighoff, D. Kocetkov, C. Mou, M. Marone, C. Akiki, J. Li, J. Chim, et al. Starcoder: may the source be with you! *arXiv preprint arXiv:2305.06161*, 2023.

[13] R. Li, J. Fu, B.-W. Zhang, T. Huang, Z. Sun, C. Lyu, G. Liu, Z. Jin, and G. Li. TACO: Topics in algorithmic code generation dataset. *arXiv preprint arXiv:2312.14852*, 2023.

[14] Z. Li, T. Xu, Y. Zhang, Z. Lin, Y. Yu, R. Sun, and Z.-Q. Luo. ReMax: A simple, effective, and efficient reinforcement learning method for aligning large language models. *arXiv preprint arXiv:2310.10505*, 2023.

[15] I. Loshchilov and F. Hutter. Decoupled Weight Decay Regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[16] A. Lozhkov, R. Li, L. B. Allal, F. Cassano, J. Lamy-Poirier, N. Tazi, A. Tang, D. Pykhtar, J. Liu, Y. Wei, et al. Starcoder 2 and the stack v2: The next generation. *arXiv preprint arXiv:2402.19173*, 2024.

[17] J. Mattern, S. Jaghouar, M. Basra, J. Straube, M. D. Ferrante, F. Gabriel, J. M. Ong, V. Weisser, and J. Hagemann. SYNTHETIC-1: Two Million Collaboratively Generated Reasoning Traces from Deepseek-R1, 2025.

[18] MistralAI. Codestral. https://mistral.ai/news/codestral, 2024.

[19] I. Paul, G. Glavaš, and I. Gurevych. IRCoder: Intermediate representations make language models robust multilingual code generators. *arXiv preprint arXiv:2403.03894*, 2024.

[20] Qwen Team. Qwen3. https://qwenlm.github.io/blog/qwen3/, 2025.

[21] R. Rafailov, A. Sharma, E. Mitchell, C. D. Manning, S. Ermon, and C. Finn. Direct Preference Optimization: Your Language Model is Secretly a Reward Model. *Annual Conference on Neural Information Processing Systems (NIPS)*, 2023.

[22] S. Rajbhandari, J. Rasley, O. Ruwase, and Y. He. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–16. IEEE, 2020.

[23] B. Roziere, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, X. E. Tan, Y. Adi, J. Liu, R. Sauvestre, T. Remez, et al. Code Llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023.

[24] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[25] Z. Shao, P. Wang, Q. Zhu, R. Xu, J. Song, X. Bi, H. Zhang, M. Zhang, Y. Li, Y. Wu, et al. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

[26] M. Szafraniec, B. Roziere, H. Leather, F. Charton, P. Labatut, and G. Synnaeve. Code translation with compiler representations. *arXiv preprint arXiv:2207.03578*, 2022.

[27] Q. Zhu, D. Guo, Z. Shao, D. Yang, P. Wang, R. Xu, Y. Wu, Y. Li, H. Gao, S. Ma, et al. Deepseek-coder-v2: Breaking the barrier of closed-source models in code intelligence. *arXiv preprint arXiv:2406.11931*, 2024.

NeurIPS Paper Checklist

The checklist is designed to encourage best practices for responsible machine learning research, addressing issues of reproducibility, transparency, research ethics, and societal impact. Do not remove the checklist: **The papers not including the checklist will be desk rejected.** The checklist should follow the references and follow the (optional) supplemental material. The checklist does NOT count towards the page limit.

Please read the checklist guidelines carefully for information on how to answer these questions. For each question in the checklist:

- You should answer [Yes] , [No] , or [NA] .
- [NA] means either that the question is Not Applicable for that particular paper or the relevant information is Not Available.
- Please provide a short (1–2 sentence) justification right after your answer (even for NA).

The checklist answers are an integral part of your paper submission. They are visible to the reviewers, area chairs, senior area chairs, and ethics reviewers. You will be asked to also include it (after eventual revisions) with the final version of your paper, and its final version will be published with the paper.

The reviewers of your paper will be asked to use the checklist as one of the factors in their evaluation. While "[Yes] " is generally preferable to "[No] ", it is perfectly acceptable to answer "[No] " provided a proper justification is given (e.g., "error bars are not reported because it would be too computationally expensive" or "we were unable to find the license for the dataset we used"). In general, answering "[No] " or "[NA] " is not grounds for rejection. While the questions are phrased in a binary way, we acknowledge that the true answer is often more nuanced, so please just use your best judgment and write a justification to elaborate. All supporting evidence can appear either in the main paper or the supplemental material, provided in appendix. If you answer [Yes] to a question, in the justification please point to the section(s) where related material for the question can be found.

IMPORTANT, please:

- **Delete this instruction block, but keep the section heading "NeurIPS Paper Checklist"**,
- **Keep the checklist subsection headings, questions/answers and guidelines below.**
- **Do not modify the questions and only use the provided macros for your answers**.

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction reflect the paper's contributions and scope.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The paper discusses the limitations of the work.

Guidelines:

- The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
- The authors are encouraged to create a separate "Limitations" section in their paper.
- The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
- The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: For each theoretical result, the paper provides the full set of assumptions and a complete (and correct) proof.

   Guidelines:

   - The answer NA means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The paper discloses all the information needed to reproduce the main experimental results.

   Guidelines:

   - The answer NA means that the paper does not include experiments.

- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [No]

   Justification: We will open-source our code and data after publication.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (https://nips.cc/public/guides/CodeSubmissionPolicy) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://nips.cc/public/guides/CodeSubmissionPolicy) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).

- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: The paper specifies all the training and test details.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: The paper does not need to provide the experiment statistical significance.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
   - If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The paper provides sufficient information on the computation resources.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.

- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics https://neurips.cc/public/EthicsGuidelines?

   Answer: [Yes]

   Justification: The research is conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [NA]

    Justification: There is no societal impact of the work performed.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: The paper poses no such risks.

    Guidelines:

- The answer NA means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: The original owners of assets, used in the paper, are properly credited.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [NA]

    Justification: The paper does not release new assets.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

Justification: The paper does not involve crowdsourcing nor research with human subjects.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

    Answer: [NA]

    Justification: The core method development in this research does not involve LLMs as any important, original, or non-standard components.

    Guidelines:

    - The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy (https://neurips.cc/Conferences/2025/LLM) for what should or should not be described.

A Derivation of the Implicit Reward

This section details the derivation of the relationship between an implicit reward function $r_\phi(x_g, y_g)$ and the optimal policy $\pi_\theta(y_g|x_g)$, under the standard reinforcement learning objective with a KL-divergence penalty against a reference policy $\pi_{\text{ref}}(y_g|x_g)$. This formulation is foundational to methods like Direct Preference Optimization (DPO) and our proposed Group Equivalent Preference Optimization (GEPO).

Following [21], the objective is to find a policy π_θ that maximizes the expected reward $r_\phi(x_g, y_g)$ while remaining close to a reference policy π_{ref}, controlled by a coefficient β:

$$\begin{aligned}
&\max_{\pi_\theta} \quad \mathbb{E}_{x_g\sim D, y_g\sim\pi_\theta(y_g|x_g)}[r_\phi(x_g, y_g)] - \beta\mathbb{D}_{\text{KL}}[\pi_\theta(y_g|x_g)||\pi_{\text{ref}}(y_g|x_g)] \\
&= \min_{\pi_\theta} \quad \mathbb{E}_{x_g\sim D}\mathbb{E}_{y_g\sim\pi_\theta(y_g|x_g)}[\log \frac{\pi_\theta(y_g|x_g)}{\frac{1}{Z(x_g)}\pi_{\text{ref}}(y_g|x_g)\exp(\frac{1}{\beta}r_\phi(x_g, y_g))} - \log Z(x_g)] \\
&= \min_{\pi_\theta} \quad \mathbb{E}_{x_g\sim D}[\mathbb{D}_{\text{KL}}(\pi_\theta(y_g|x_g)||\pi^*(y_g|x_g)) - \log Z(x_g)]. \qquad (11)
\end{aligned}$$

Here, D represents the distribution of prompts x_g, and $\pi^*(y_g|x_g)$ is defined as the optimal policy distribution:

$$\pi^*(y_g|x_g) = \frac{1}{Z(x_g)}\pi_{\text{ref}}(y_g|x_g)\exp(\frac{1}{\beta}r_\phi(x_g, y_g)). \qquad (12)$$

The term $Z(x_g)$ is the partition function, ensuring that $\pi^*(y_g|x_g)$ normalizes to a valid probability distribution over all possible responses y_g:

$$Z(x_g) = \sum_{y_g}\pi_{\text{ref}}(y_g|x_g)\exp(\frac{1}{\beta}r_\phi(x_g, y_g)). \qquad (13)$$

The KL divergence term $\mathbb{D}_{\text{KL}}(\pi_\theta(y_g|x_g)||\pi^*(y_g|x_g))$ in Equation 11 is minimized (becomes zero) when the policy $\pi_\theta(y_g|x_g)$ is identical to $\pi^*(y_g|x_g)$. Therefore, the optimal policy π_θ that solves the optimization problem is:

$$\pi_\theta(y_g|x_g) = \pi^*(y_g|x_g) = \frac{1}{Z(x_g)}\pi_{\text{ref}}(y_g|x_g)\exp(\frac{1}{\beta}r_\phi(x_g, y_g)). \qquad (14)$$

To derive the expression for the implicit reward $r_\phi(x_g, y_g)$, we can rearrange Equation 14:

$$\begin{aligned}
\frac{\pi_\theta(y_g|x_g)Z(x_g)}{\pi_{\text{ref}}(y_g|x_g)} &= \exp(\frac{1}{\beta}r_\phi(x_g, y_g)) \\
\log(\frac{\pi_\theta(y_g|x_g)Z(x_g)}{\pi_{\text{ref}}(y_g|x_g)}) &= \frac{1}{\beta}r_\phi(x_g, y_g) \\
\beta(\log\frac{\pi_\theta(y_g|x_g)}{\pi_{\text{ref}}(y_g|x_g)} + \log Z(x_g)) &= r_\phi(x_g, y_g).
\end{aligned}$$

This gives us the final form for the implicit reward function:

$$r_\phi(x_g, y_g) = \beta\log\frac{\pi_\theta(y_g|x_g)}{\pi_{\text{ref}}(y_g|x_g)} + \beta\log Z(x_g). \qquad (15)$$

This relationship (Equation 15) demonstrates that the implicit reward $r_\phi(x_g, y_g)$ is proportional to the log-probability ratio of the policy $\pi_\theta(y_g|x_g)$ with respect to the reference policy $\pi_{\text{ref}}(y_g|x_g)$, plus a term related to the partition function. This expression is leveraged in Section 3.2 (specifically, Equation 7) to transform the GEPO objective, which is initially defined in terms of $r_\phi(x_g, y_g)$, into an objective that directly depends on the policy probabilities $\pi_\theta(y_g|x_g)$ and $\pi_{\text{ref}}(y_g|x_g)$.