PRELIMINARY PREPRINT VERSION – DO NOT CITE
This is not the version of record. The final version will appear in the Proceedings of IJCAI-ECAI 2026.

Improving Code Translation with Syntax-Guided and Semantic-aware Preference Optimization

Yuhan Wu[1], **Huan Zhang**[1], **Wei Cheng**[1], **Chen Shen**[1], **Jingyue Yang**[1] and **Wei Hu**[1,2,*]

[1]State Key Laboratory for Novel Software Technology, Nanjing University, China
[2]National Institute of Healthcare Data Science, Nanjing University, China
{yhwu, zhanghuan, wchengcs, cshen, jyyang}.nju@gmail.com, whu@nju.edu.cn

Abstract

LLMs have shown immense potential for code translation, yet they often struggle to ensure both syntactic correctness and semantic consistency. While preference-based learning offers a promising alignment strategy, it is hindered by unreliable semantic rewards derived from sparse test cases or restrictive reference translations. We argue that a robust semantic reward for code translation must be derived directly from the source code. In this paper, we propose CTO to improve code translation with syntax-guided and semantic-aware preference optimization. Through contrastive learning, we train a cross-lingual semantic model to directly assess functional equivalence between source and translated code. By formulating code translation as a multi-objective optimization problem, this robust semantic signal is seamlessly unified with compiler-based syntactic feedback within the direct preference optimization framework. Extensive experiments on C++, Java, and Python translations demonstrate that CTO significantly outperforms existing baselines and alternative preference optimization strategies.

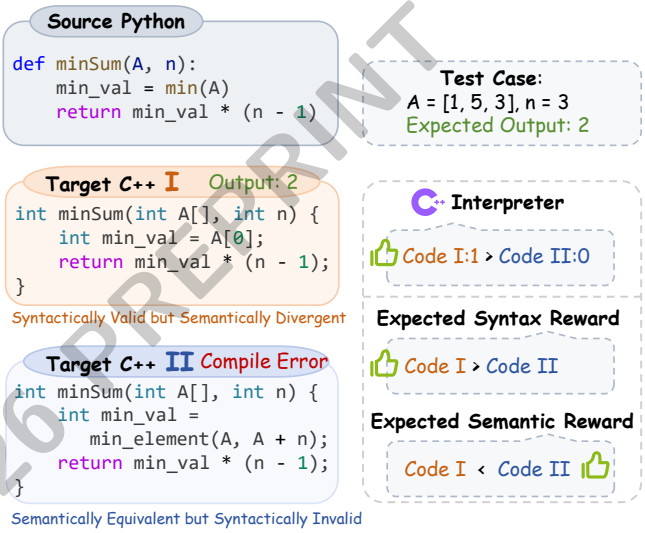


Figure 1: A motivating example illustrating the entanglement of syntax and semantics in execution-based reward. **Target C++ I** is syntactically valid but semantically flawed; however, it exploits sparse test cases (where `A[0]` happens to be the minimum) to achieve a false positive pass (*reward hacking*). Conversely, **Target C++ II** maintains semantic equivalence (using `min_element`) but receives a zero reward due to a compilation error.

1 Introduction

Code translation, the process of migrating functionality from a source programming language to a target language, is a crucial task in modern software engineering. It promotes code reuse, facilitates legacy system modernization, and enables cross-language interoperability in an increasingly heterogeneous software ecosystem [Nguyen *et al.*, 2014; Zhu *et al.*, 2022a; Yan *et al.*, 2023]. While recent advances in pre-trained language models have shown immense potential [Lu *et al.*, 2021; Zhu *et al.*, 2022b; Zheng *et al.*, 2023; Roziere *et al.*, 2020], their practical application is frequently hindered by a key challenge: ensuring that the translated code is not only syntactically correct but also semantically equivalent to the source. Even state-of-the-art large language models (LLMs) often make errors due to inadequate understanding of the syntactic and semantic nuances across different programming languages [Pan *et al.*, 2024; Yang *et al.*, 2024].

To address this, a natural evolution is to move beyond standard supervised finetuning and adopt preference-based learning. By leveraging reward signals derived from user judgments or task-specific criteria, reinforcement learning from human feedback (RLHF) [Ouyang *et al.*, 2022] has been widely employed to align model outputs with human preferences. However, applying this paradigm to code translation raises a fundamental question: **How can we define and obtain reliable reward signals for both syntactic correctness and semantic consistency?** For syntax, compiler feedback serves as an infallible oracle, providing a deterministic and binary signal of syntactic correctness [Shojaee *et al.*, 2023]. For semantics, however, the path is fraught with challenges.

As illustrated in Figure 1, prevailing strategies for semantic rewards are built on flawed foundations. The most common strategy is to derive rewards from test cases, treating test out-

*Corresponding author

IJCAI-ECAI 2026 PREPRINT

comes as a proxy for functional correctness [Le *et al.*, 2022; Gee *et al.*, 2025; Zhang *et al.*, 2025]. However, real-world test suites are often sparse and exhibit low coverage, which can lead to reward hacking [Ma *et al.*, 2025], where the model may overfit to these limited test cases without preserving the intended semantics of the source code. In the context of code translation, relying on test cases leads to a critical entanglement of syntax and semantics. A single syntactical divergence in the target language often invalidates the entire execution, yielding a zero pass rate regardless of semantic accuracy. Consequently, the model receives a binary reward that fails to decouple semantic correctness from syntactic validity, making it impossible to quantify the magnitude of semantic deviation. An alternative is to use the reference translation as a semantic anchor. However, this approach gives rise to two flawed strategies. The first strategy compares the generated code to the reference using text-based similarity metrics such as CodeBLEU [Ren *et al.*, 2020]. This strategy is fundamentally superficial, as it rewards lexical similarity over true functional equivalence. The second, more sophisticated strategy introduces a monolingual semantic model to evaluate the similarity between candidate translations and the reference. However, it implicitly assumes that the reference translation is a perfect and complete representation of the source semantics. This assumption introduces a *semantic bottleneck* that restricts output diversity and may even propagate errors in the reference. We argue that a robust semantic reward should be disentangled from such flawed proxies and instead be derived directly from the source code itself. We propose CTO, a novel approach that improves code translation via syntax-guided and semantic-aware preference optimization.

We reformulate code translation as a multi-objective preference optimization problem, explicitly modeling both syntax and semantics as distinct objectives. It leverages compiler feedback to construct syntax-aware preference pairs. For semantic alignment, it trains a cross-lingual semantic model via contrastive learning to assess functional equivalence directly between the source and translated candidates. It injects the semantic reward difference of a preference pair directly into the learning process, biasing the preference strength to unify syntactic and semantic objectives within the direct preference optimization (DPO) [Rafailov *et al.*, 2023] framework.

We conduct extensive experiments on pairwise code translation between C++, Java, and Python. Results demonstrate that CTO outperforms existing approaches and alternative preference optimization strategies, achieving a superior alignment with the intended functionality of the source code.

To summarize, our main contributions are as follows:

- We propose CTO, a novel approach that improves code translation via syntax-guided and semantic-aware preference optimization. It introduces a reward-biasing mechanism within DPO, enabling unified optimization of both syntactic correctness and semantic consistency.
- We train a cross-lingual semantic model as a robust reward source. It directly evaluates functional equivalence between the source code and translation candidates, overcoming the fundamental limitations of reward signals derived from other proxies.
- Our experiments show that CTO improves translation accuracy by up to 3.66% on the TransCoder-Test dataset and 4.27% on HumanEval-X with the finetuned CodeT5 model. With a larger finetuned CodeLlama-7B model, it yields accuracy gains of 5.60% and 6.70%, respectively.

2 Related Work

Code Translation. Early works [Roziere *et al.*, 2020; Roziere *et al.*, 2022] on code translation primarily focus on self-supervised learning. Building upon this, several works [Szafraniec *et al.*, 2023; Huang *et al.*, 2023; Liu *et al.*, 2023] explore incorporating code structure information to enhance code representations for unsupervised code translation. These methods demonstrate the feasibility of learning cross-language representations without parallel data. Despite their effectiveness, unsupervised approaches often struggle with collecting enormous amounts of code corpora and high computational resource consumption.

With the availability of high-quality parallel datasets [Zhu *et al.*, 2022a; Yan *et al.*, 2023; Khan *et al.*, 2024; Yan *et al.*, 2024] and pre-trained code models [Wang *et al.*, 2021; Guo *et al.*, 2021; Zheng *et al.*, 2023; Feng *et al.*, 2020], supervised finetuning has become a paradigm for code translation. Building on this foundation, recent studies explore incorporating reinforcement learning (RL) techniques [Shojaee *et al.*, 2023] and variational inference techniques [Du *et al.*, 2024] to further enhance the performance. Besides, a few recent works [Ibrahimzada *et al.*, 2024; Wang *et al.*, 2024] focus on repository-level code translation, aiming to facilitate complex codebase migration. Our work follows the supervised finetuning paradigm, further advancing code translation performance without relying on any auxiliary datasets.

Reinforcement Learning from Human Feedback. RLHF has significantly improved the performance of downstream tasks by aligning models with human preferences. While the reward-based methods like proximal policy optimization [Schulman *et al.*, 2017] and group relative policy optimization [Shao *et al.*, 2024] demonstrate success, they suffer from increased memory overhead and substantial computational burdens due to their requirement for a large number of samples during each policy update cycle.

Reward-free methods address these limitations from a different perspective. Direct preference optimization [Rafailov *et al.*, 2023] eliminates the need for an explicit reward model by implicitly estimating reward signals from preference data. Identity preference optimization [Azar *et al.*, 2024] introduces a general non-decreasing bounded function to mitigate overfitting, while simple preference optimization [Meng *et al.*, 2024] further simplifies the process by removing the reference model altogether. Additionally, some studies [Park *et al.*, 2024; Meng *et al.*, 2024] explore the regularization terms as a margin to improve reward modeling. Despite these advancements, directly applying these techniques to code translation remains suboptimal due to inadequate syntax representation and incomplete semantic modeling. To address this challenge, CTO enhances preference optimization by jointly incorporating syntax and semantics, leading to improved accuracy and robustness in code translation.

IJCAI-ECAI 2026 PREPRINT

3 Methodology

3.1 Reward Model Training

The overall reward comprises both syntactic and semantic components. The syntactic reward is readily accessible, as it can be directly derived from the compiler feedback. This binary and rule-based signal indicates whether the translation code adheres to the target language's grammar, following standard practice in prior work [Shojaee *et al.*, 2023].

In contrast, designing an effective semantic reward presents a more challenging problem, as code semantics encompass not only functional equivalence, but also logical fidelity to the source code's intent. Prior studies have approached this from different perspectives. Execution-based reward like CodeRL [Le *et al.*, 2022] utilizes discrete, rule-based heuristics to evaluate semantic correctness, primarily relying on unit tests. However, test cases are often scarce or even unavailable in function- or program-level translation training data, resulting in unstable execution-based reward signals. Furthermore, such reward is inherently binary and sparse, making it difficult to quantify the proximity of an incorrect translation to the correct logic. Reference-based reward like PPOCoder [Shojaee *et al.*, 2023] employs reward signals derived from static program representations, such as abstract syntax trees and dataflow graphs, which often depend on comparisons with ground-truth reference implementations. The reliance on reference translations restricts their applicability in real-world scenarios, where such references are often incomplete or entirely absent. Moreover, minor code variations, such as variable renaming, can lead to significant runtime or functional errors, which are often not adequately penalized by textual metrics.

Semantic Reward Model. To address these issues, we propose to assess semantic correctness by measuring semantic distance within a high-dimensional latent space. Formally, our objective is to learn a mapping function $f : \mathcal{X} \cup \mathcal{Y} \to \mathcal{Z}$ such that the geometric proximity in $\mathcal{Z}$ reflects the functional equivalence between the source code x and the target code y. To shape this latent space, we construct a source-anchored triplet $\mathcal{T} = (x, y^+, \mathcal{Y}^-)$ for each training instance. We designate the source code x as the fixed anchor point in the latent space. It serves as the semantic oracle which all translation candidates are measured. The reference translation y^+ is defined as the positive sample and $\mathcal{Y}^- = \{y_1^-, \ldots, y_K^-\}$ is semantically divergent negatives. The optimization goal is to pull the embedding $f(y^+)$ into the immediate ϵ-neighborhood of the anchor $f(x)$, while explicitly repelling the hard negatives in $\mathcal{Y}^-$ beyond this neighborhood boundary, thereby establishing a robust semantic margin against subtle deviations.

Training Objective. Our semantic reward model adopts a dual-encoder architecture, which contains a source code encoder and a target code encoder, sharing the same parameters. To effectively learn semantic similarity for cross-lingual code representation, we use the InfoNCE loss [van den Oord *et al.*, 2018] as the training objective:

$$\mathcal{L}_f = -\log \frac{\exp\left(\cos(\frac{f(\mathbf{x}), f(\mathbf{y}^+)}{\tau}\right)}{\exp\left(\cos(\frac{f(\mathbf{x}), f(\mathbf{y}^+)}{\tau}\right) + \sum_{i=1}^{K} \exp\left(\cos(\frac{f(\mathbf{x}), f(\mathbf{y}_i^-)}{\tau}\right)}, \tag{1}$$

where $\mathbf{x}$ denotes the source code, $\mathbf{y}^+$ and $\mathbf{y}_i^-$ denotes reference translation and negative translations respectively. $f(\cdot)$ is the encoder that maps code snippets to semantic representations, $\cos(\cdot, \cdot)$ denotes cosine similarity, and τ is the temperature parameter.

Dataset for Training. The positive samples come from the reference code in the supervised dataset. To construct negative samples for training the semantic reward model, we employ an LLM as a perturbation generator. Specifically, the LLM rewrites the reference code by introducing subtle perturbations that alter the intended semantics. These perturbed variants, which often remain syntactically valid but semantically flawed, making them ideal negative examples. Together with the original reference translations, they form the training dataset for learning cross-lingual semantic alignment.

Reward Score Function. Based on the semantic model, we encode both the source and candidate target code into embeddings within a shared vector space. For a set of candidate translations $y = \{y_1, \ldots, y_n\}$ in the target language, we compute the cosine similarity between each candidate y_i and the source code x. The similarity scores are transformed into logit values and standardized via z-score normalization to produce the final semantic reward:

$$\begin{aligned} r_s(y_i) &= \frac{s_i - \mu(s_1, \ldots, s_n)}{\sigma(s_1, \ldots, s_n)}, \\ s_i &= \log \frac{\cos(f(x), f(y_i))}{1 - \cos(f(x), f(y_i))}, \end{aligned} \tag{2}$$

where μ and σ represent the mean and standard deviation of the scores $s_1, \ldots, s_n$, respectively. This function enables listwise scoring by capturing the semantic similarity between the source code and each candidate translation.

3.2 Preference Optimization

Problem Formulation. We formulate code translation as a multi-objective optimization problem over syntax and semantics, and apply a linear scalarization strategy to integrate the multiple objectives into a unified optimization goal. We denote the syntax objective as g, the semantics objective as s, and the preference dataset for code translation as $\mathcal{D} = \{\mathcal{D}_g, \mathcal{D}_s\}$, where $\mathcal{D}_g$ and $\mathcal{D}_s$ denote the syntactic preference dataset and semantic preference dataset, respectively. Given the dataset $\mathcal{D}$, we define the oracle reward model as a weighted combination of syntax and semantic rewards:

$$\mathbf{r}^*(\mathbf{x}, \mathbf{y}) = w \cdot r_g^*(\mathbf{x}, \mathbf{y}) + (1 - w) \cdot r_s^*(\mathbf{x}, \mathbf{y}), \tag{3}$$

where $r_g^*(\mathbf{x}, \mathbf{y})$ and $r_s^*(\mathbf{x}, \mathbf{y})$ evaluate the syntax and semantic rewards of the translated target code, respectively. $w \in [0, 1]$ is a weight parameter to balance syntactic and semantic preferences. In contrast to the classical Pareto optimization,

IJCAI-ECAI 2026 PREPRINT

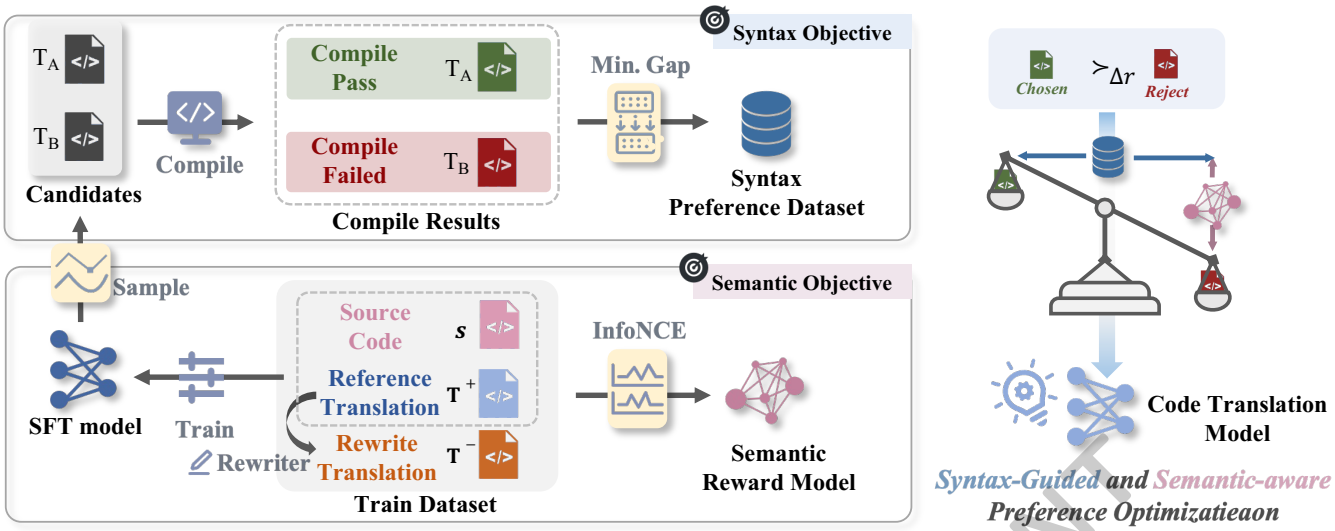


Figure 2: Overview of our CTO.

which seeks to approximate the Pareto front of multiple objectives, we focus on optimizing the code translation model under a specific preference weighting $w = 0.5$. This choice represents an equal prioritization of syntactic and semantic objectives.

This leads to the following training objective for the translation model π_θ:

$$\max_{\pi_\theta} \mathbb{E}_{\mathbf{x},\mathbf{y}\sim\pi(\cdot\,|\,\mathbf{x})}\left[\mathbf{r}^*(\mathbf{x},\mathbf{y}) - \beta \log \frac{\pi_\theta(\mathbf{y}\,|\,\mathbf{x})}{\pi_{\text{sft}}(\mathbf{y}\,|\,\mathbf{x})}\right], \quad (4)$$

where π_{sft} is the supervised finetuned model serving as the reference policy, β is a regularization coefficient controlling deviation from the reference model.

While previous works [Shojaee *et al.*, 2023] have leveraged RL for reward-based finetuning, such approaches tend to be unstable and resource-intensive. We propose an RL-free variant of DPO, designed to integrate syntax and semantics into a unified training process efficiently.

Derivation of CTO. The optimal policy under the RLHF objective (Eq. (4)) takes the form:

$$\pi^*_\theta(y\,|\,x) = \frac{1}{Z(x)}\pi_{\text{sft}}(y\,|\,x)\exp\left(\frac{\mathbf{r}^*(x,y)}{\beta}\right), \quad (5)$$

where the partition function is defined as: $Z(x) = \sum_y \pi_{\text{sft}}(y\,|\,x)\exp\left(\frac{1}{\beta}\left(w\cdot r^*_g(x,y) + (1-w)\cdot r^*_s(x,y)\right)\right)$.

Let $\mathcal{D}_k = \{(x, y_w, y_l)\}$ denote the pairwise preference dataset under objective $k \in \{g, s\}$, where $y_w \succ y_l$ indicates that y_w is preferred over y_l for input x. The pairwise preference likelihood under the oracle reward model is given by:

$$p_{\mathcal{D}_k}(y_w \succ y_l \mid x) = \sigma\left(\mathbf{r}^*(x,y_w) - \mathbf{r}^*(x,y_l)\right), \quad (6)$$

where $\sigma(\cdot)$ denotes the sigmoid function.

Substituting Eq. (5) into Eq. (6) and approximating the ground-truth reward $r^*_{-k}(x,y)$ from the complementary objective $-k$ using a learned reward model $r_{-k}(x,y)$, we derive the final training objective of CTO:

$$\mathcal{L}_{\text{CTO}} = -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}_k}\left[\log\sigma\left(\frac{\beta}{w}\left(\log\frac{\pi_\theta(y_w\,|\,x)}{\pi_{\text{sft}}(y_w\,|\,x)} - \log\frac{\pi_\theta(y_l\,|\,x)}{\pi_{\text{sft}}(y_l\,|\,x)}\right) - \frac{1-w}{w}\left(r_{-k}(x,y_w) - r_{-k}(x,y_l)\right)\right)\right], \quad (7)$$

where $-k$ denotes the objective complementary to k (i.e., if $k = g$, then $-k = s$, and vice versa).

This formulation reveals the core mechanism of CTO: by leveraging preference data from one objective (captured in $\mathcal{D}_k$) and integrating the explicit reward difference from the other (r_{-k}) as a rectification term, we realize the simultaneous modeling of both syntactic and semantic objectives within a unified optimization process.

Preference Dataset Construction. Constructing a high-quality preference dataset is essential for effective model training. Syntactic feedback is generally more stable and verifiable, as grammar correctness can be deterministically checked via compilation. Such feedback serves as an oracle-like signal that enables the reliable classification of candidate translations into preferred and less preferred sets. In contrast, semantic feedback is inherently more ambiguous. Although extended unit tests and semantic reward models offer approximate semantic judgments, determining the absolute correctness of a translation remains an undecidable problem. Given this discrepancy in reliability, we build our preference dataset primarily based on syntactic correctness, where preferences between candidate translations are derived from deterministic compilation results.

As shown in Figure 2, we begin with a supervised dataset comprising aligned source and target code pairs. A base code LLM is finetuned on this dataset to improve its code translation capabilities. For a source code snippet x, we generate a set of candidate translations $\{y_1, y_2, \ldots, y_n\}$ using the finetuned model. Each candidate y_i is passed through a compiler to obtain a binary signal indicating whether the code

IJCAI-ECAI 2026 PREPRINT

Methods	TransCoder-Test						HumanEval-X					
	C→J	C→P	J→C	J→P	P→C	P→J	C→J	C→P	J→C	J→P	P→C	P→J
TransCoder	66.59	43.75	80.51	49.57	34.05	36.51	48.17	38.41	38.41	42.68	20.73	26.83
TransCoder-ST	68.26	65.95	**82.23**	72.41	57.38	56.43	46.34	40.85	40.85	64.63	21.95	23.17
CodeT5-SFT	73.65	70.91	80.73	72.41	65.95	66.39	47.56	43.90	45.73	65.24	27.44	28.66
PPOCoder	72.20	57.11	65.74	52.15	38.97	51.24	42.68	40.85	42.68	59.76	26.83	**29.27**
CTO (CodeT5)	**75.73**	**74.57**	**82.23**	**75.43**	**67.66**	**68.26**	**50.00**	**45.73**	**49.39**	**69.51**	**29.27**	**29.27**
CodeLlama-7B	62.86	68.32	56.71	68.97	65.52	49.37	75.61	62.80	50.00	78.66	42.68	51.83
CodeLlama-7B-SFT	80.91	78.23	79.01	81.47	80.73	77.39	82.93	68.29	59.15	82.32	50.00	56.71
CTO (CodeLlama-7B)	**86.51**	**81.90**	**82.87**	**83.83**	**82.23**	**79.46**	**84.15**	**74.39**	**65.85**	**83.54**	**50.61**	**59.15**
Qwen2.5-Coder-7B	71.37	78.23	42.40	79.31	60.38	27.19	77.44	74.39	70.12	81.71	56.71	68.29
Qwen2.5-Coder-7B-SFT	87.55	82.97	91.65	87.07	86.93	84.85	84.76	77.44	72.56	86.59	62.80	70.12
CTO (Qwen2.5-Coder-7B)	**90.87**	**87.93**	**93.36**	**90.95**	**89.29**	**85.89**	**87.19**	**82.31**	**73.78**	**87.19**	**64.02**	**71.95**

Table 1: CA@1 scores on the TransCoder-Test and HumanEval-X datasets. "C", "J", and "P" denote C++, Java, and Python, respectively. The best scores are marked in **bold**.

compiles successfully. Based on the compilation results, the candidates are partitioned into a pass set and a fail set, from which we construct a preference dataset. Instead of selecting random pairs, we employ `git-diff`[1] to identify the closest matching pair (y_w, y_l), ensuring that low-gap pairs capture fine-grained preferences, where y_w comes from the pass set and y_l comes from the failed set. We then incorporate scores from the semantic reward model as soft guidance signals. Specifically, the semantic scores of y_w and y_l are used during optimization to enrich the syntactic preference with semantic discrimination.

This strategy leverages the determinism of syntactic preference labels while still benefiting from the fine-grained semantic discrimination offered by the semantic reward model. Such integration facilitates the simultaneous optimization of syntactic accuracy and semantic alignment.

4 Experiments and Results

4.1 Experiment Settings

Dataset Construction. Our supervised finetuning training set and preference dataset construction are based on the XLCoST dataset [Zhu *et al.*, 2022a], which contains parallel snippet-level and program-level code for commonly used programming languages, with each sample accompanied by an example test case to verify functionality. We collect the parallel program-level data from the training and validation splits. The supervised finetuning training set consists of 6,884 Java↔C++, 6,419 C++↔Python, and 7,278 Java↔Python samples, with a validation set of 346, 323, and 376 samples for each pair, respectively. After finetuning, we generate 10 candidate responses for each input using a sampling temperature of 0.9.

For the semantic model training dataset, we employ the Qwen3-8B model to rewrite the reference code of XLCoST, thereby generating corresponding negative samples. Because the inference is conducted on candidates sampled from the supervised finetuning model at the semantic model's test time, rather than on the reference data itself, this avoids data leakage and ensures benchmarks remain entirely unseen by the semantic model.

[1]https://git-scm.com/docs/git-diff

Benchmarks. To assess the performance of our CTO, we employ two benchmark datasets:

- **TransCoder-Test** [Roziere *et al.*, 2020] is a standard benchmark for unsupervised code translation, with tasks in Java, C++, and Python (482, 467, and 464 tasks, respectively) and an average of 10 test cases per task.
- **HumanEval-X** [Zheng *et al.*, 2023] is extended from the HumanEval benchmark [Chen *et al.*, 2021] and designed for cross-lingual code correctness evaluation using automated test cases. It includes 164 tasks, with 6.9 test cases on average.

Evaluation Metrics. Computational Accuracy at top-K (CA@K) measures the proportion of tasks where at least one of the top-K translations generated by the model passes all test cases. We use CA@1, showing the model's effectiveness in generating functionally equivalent code to the reference.

Baseline Methods. We compare CTO to two categories of code translation methods:

- **Unsupervised translation** does not require parallel corpora for training. Instead, it leverages self-supervised learning techniques: (1) **TransCoder** [Roziere *et al.*, 2020] is an unsupervised model with approximately 100M parameters that leverages masked language modeling, denoising autoencoding, and back translation as its core pre-training tasks. (2) **TransCoder-ST** [Roziere *et al.*, 2022] extends TransCoder by incorporating automated unit tests to reduce noise in back translation, thereby improving model performance.
- **Supervised translation** relies on parallel corpora with aligned source and target code pairs to learn a transformation between different programming languages: (1) **CodeT5-SFT** is a supervised finetuned variant of CodeT5 [Wang *et al.*, 2021]. We choose CodeT5 as the base model due to its comparable scale (770M) and architecture to TransCoder and TransCoder-ST, ensuring a balanced comparison. (2) **PPOCoder** [Shojaee *et al.*, 2023] is built upon CodeT5-SFT with proximal policy optimization (PPO). It incorporates compiler feedback and both syntactic and semantic match scores as reward signals to improve translation quality. (3) **CodeLlama-**

IJCAI-ECAI 2026 PREPRINT

Methods	TransCoder-Test						HumanEval-X					
	C→J	C→P	J→C	J→P	P→C	P→J	C→J	C→P	J→C	J→P	P→C	P→J
CTO (CodeLlama-7B)	**86.51**	**81.90**	**82.87**	**83.83**	**82.23**	**79.46**	**84.15**	**74.39**	**65.85**	**83.54**	**50.61**	**59.15**
w/o semantic	82.31	81.47	80.08	81.90	81.80	79.25	82.78	73.78	60.98	83.54	50.00	58.54
w/o syntax	82.16	79.09	80.30	82.33	**82.23**	**79.46**	82.32	65.85	60.98	80.49	48.78	**59.15**
IPO (CodeLlama-7B)	83.82	75.43	82.65	83.62	80.09	44.19	40.24	66.46	63.41	81.10	48.78	23.78
SimPO (CodeLlama-7B)	83.40	73.71	81.58	83.40	80.30	77.80	41.46	65.85	63.41	81.70	48.78	26.83
CTO (Qwen2.5-Coder-7B)	**90.87**	**87.93**	**93.36**	**90.95**	**89.29**	**85.89**	**87.19**	**82.31**	**73.78**	**87.19**	**64.02**	**71.95**
w/o semantic	89.63	86.21	92.51	87.93	89.08	85.06	85.98	78.66	73.17	86.59	62.20	**71.95**
w/o syntax	89.00	83.19	93.15	86.85	86.51	85.06	85.36	75.00	71.95	86.59	63.41	70.73
IPO (Qwen2.5-Coder-7B)	89.00	87.07	92.93	89.87	88.44	85.27	44.51	79.87	**73.78**	85.98	60.98	34.14
SimPO (Qwen2.5-Coder-7B)	90.04	87.28	**93.36**	89.01	87.79	85.27	44.51	79.87	73.17	85.98	62.20	34.75

Table 2: CA@1 scores of ablation study and alternative methods on the TransCoder-Test and HumanEval-X datasets.

7B-SFT is a supervised finetuned variant of CodeLlama-7B [Rozière *et al.*, 2023], which serves as a representative foundational model for code-related tasks. (4) **Qwen2.5-Coder-7B-SFT** is a supervised finetuned variant of Qwen2.5-Coder-7B [Hui *et al.*, 2024].

Implementation. For supervised finetuning, CodeT5 uses a learning rate of 5e-5 and a batch size of 128, while CodeLlama-7B and Qwen2.5-Coder-7B use 3e-5 and 32, respectively, with gradient accumulation for memory constraints. For the semantic reward model, we employ Qwen3-Embedding 0.6B as the base model. The training is conducted with a learning rate of 6e-6 and a batch size of 32 for 5 epochs. For preference optimization, we set the learning rate to 5e-7 for CodeT5-SFT, and 5e-5 for both CodeLlama-SFT and Qwen2.5-Coder-7B. For CodeLlama-7B and Qwen2.5-Coder-7B, we apply LoRA [Hu *et al.*, 2022] for all finetuning processes, including both supervised finetuning and preference optimization phases.

All experiments are conducted on one NVIDIA RTX A800 GPU and Ubuntu 20.04 LTS. See our source code[2] for other implementation details.

4.2 Code Translation Results

Table 1 shows the experimental results on the TransCoder-Test and HumanEval-X datasets. Our CTO consistently outperforms existing methods across six translation tasks.

Compared with the unsupervised code translation methods, CTO (CodeT5) surpasses both TransCoder and TransCoder-ST across all evaluated scenarios. This performance gap underscores the efficacy of supervised finetuning on parallel corpora, which facilitates the direct acquisition of cross-lingual mappings and ensures higher translation reliability. In contrast, unsupervised methods rely on back translation, which can introduce noise and misalignment. For PPOCoder, we follow its reward function and parameter settings in [Shojaee *et al.*, 2023] and apply PPO on our finetuned CodeT5 model. Contrary to our expectations, PPOCoder exhibits performance degradation in most cases. This implies that PPOCoder does not always benefit from its reward function and may even introduce instability to the finetuned model.

Since the preference dataset is derived from supervised finetuned models, its quality depends on the model's ability to generate valid compilable samples. However, CodeT5-SFT generates fewer valid samples in translation tasks from Python, making preference data collection more difficult. This observation is consistent with previous studies [Feng *et al.*, 2024], which have noted a strong correlation between the performance of supervised finetuned models and the effectiveness of preference optimization.

To assess the scalability of CTO on larger code models, we conduct experiments using decoder-only LLMs, CodeLlama-7B and Qwen2.5-Coder-7B. Across six language pairs and two benchmarks, CTO consistently outperforms their supervised finetuned counterparts (CodeLlama-7B-SFT and Qwen2.5-Coder-7B-SFT). These improvements validate the generalization of CTO across different model architectures and parameter sizes.

4.3 Ablation Study and Alternative Methods

To investigate the effectiveness of multi-objective preference optimization in our proposed method, we conduct an ablation study by systematically removing key components and analyzing their impact. To measure the contribution of multi-objective preference optimization, we design two variants:

- **CTO w/o semantic** removes the semantic preference from the preference optimization stage, meaning the model is optimized solely based on the syntax preference data. Consequently, CTO degenerates into DPO, which helps isolate the impact of semantic reward.
- **CTO w/o syntax** removes the syntax preference from the preference optimization stage, i.e., the model is optimized solely based on the semantic preference data. In this setting, CTO degenerates to vanilla DPO, where the reference translation is treated as the chosen sample and the rejected samples are standard outputs that do not match the reference.

As shown in Table 2, the results clearly indicate that each preference contributes significantly to the final performance. Notably, in the w/o syntax variant, we rely on the example unit test shipped with the original dataset as auxiliary signals to guide optimization. But the performance still falls short compared to CTO. These results confirm that multi-objective preference plays a crucial role in maximizing performance.

Additionally, to further validate the effectiveness of our preference optimization strategy, we compare our method

[2]https://github.com/nju-websoft/CTO

IJCAI-ECAI 2026 PREPRINT

Methods	TransCoder-Test						HumanEval-X					
	C→J	C→P	J→C	J→P	P→C	P→J	C→J	C→P	J→C	J→P	P→C	P→J
CTO (CodeLlama-7B)	**86.51**	**81.90**	**82.87**	**83.83**	**82.23**	**79.46**	**84.15**	74.39	**65.85**	**83.54**	**50.61**	**59.15**
w/ CodeBLEU	83.20	81.47	82.01	82.76	**82.23**	73.86	82.93	**75.61**	62.80	81.71	**50.61**	57.93
w/ CodeBERTScore	83.61	81.25	82.23	82.54	82.01	73.44	82.93	**75.61**	62.80	82.32	**50.61**	57.32
CTO (Qwen2.5-Coder-7B)	**90.87**	**87.93**	**93.36**	**90.95**	**89.29**	**85.89**	**87.19**	**82.31**	**73.78**	**87.19**	**64.02**	71.95
w/ CodeBLEU	88.80	87.28	92.72	88.58	88.44	84.65	86.59	79.27	73.17	86.59	62.80	72.56
w/ CodeBERTScore	89.63	87.07	92.29	88.58	88.87	85.27	86.59	79.88	73.17	85.98	63.41	**73.17**

Table 3: Comparison of CA@1 scores between reference-based metrics and the semantic reward model.

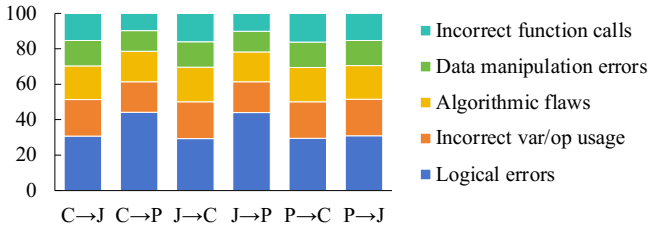
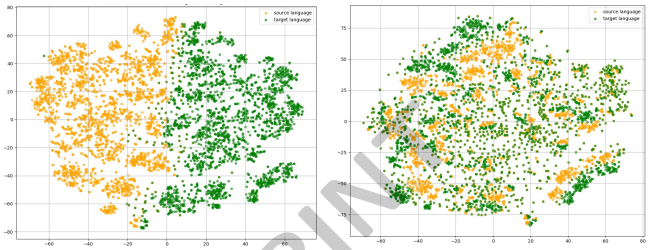


Figure 3: Distribution of negative sample types.



(a) Qwen3-Embedding 0.6B **(b) Semantic Model**

Figure 4: t-SNE representation of source language code (yellow) and target language code (green).

with two typical reward-free preference optimization techniques: identity preference optimization (IPO) [Azar *et al.*, 2024] and simple preference optimization (SimPO) [Meng *et al.*, 2024]. These baselines perform preference optimization on supervised finetuned models, enabling us to assess whether CTO provides a tangible advantage over existing techniques. As also presented in Table 2, CTO achieves higher performance across most translation tasks on both TransCoder-Test and HumanEval-X benchmarks. While IPO and SimPO fail to match the performance of our method in certain scenarios, they even cause performance degradation when applied to supervised finetuned models.

Overall, these results demonstrate that our CTO effectively integrates syntactic and semantic modeling within a preference optimization framework, leading to superior performance in code translation tasks. These findings reinforce the importance of multi-objective preference optimization as a robust strategy for enhancing code translation models.

4.4 Reward Model Evaluation

Negative Sample Types. To evaluate the quality and diversity of our semantic reward model's training data, we analyze the distribution of various negative sample types, following the semantic error classification outlined in [Pan *et al.*, 2024]. As shown in Figure 3, the categorization reveals that logical errors constitute the largest portion of the dataset. The extensive coverage of error categories, coupled with their relatively balanced representation, prevents the model from being biased toward specific patterns and ensures robust detection of various semantic discrepancies.

Latent Space Visualization. To assess the effectiveness of our trained reward model, we conduct an evaluation of our semantic reward model and visualize the source and target code embeddings derived from the reward model in a shared vector space. Figure 4 shows that embeddings from different programming languages form coherent clusters instead of being strictly segregated by language. This verifies the model's ability to fuse cross-lingual code semantics into a unified embedding space, suggesting that it captures meaningful alignment between source and target language structures.

Effectiveness of the Semantic Reward Model. Lastly, we compare the performance of our semantic reward model and previous reference-based metrics such as CodeBLEU [Ren *et al.*, 2020] and CodeBERTScore [Zhou *et al.*, 2023] when applied to CTO. As presented in Table 3, our semantic reward model consistently outperforms both CodeBLEU and CodeBERTScore across most scenarios. This performance gap highlights the advantage of leveraging our learned semantic model over relying solely on reference-based metrics as the semantic reward.

Overall, these results confirm the cross-lingual semantic coherence captured by our semantic reward model and its superior effectiveness over prior reference-based metrics.

5 Conclusion

In this paper, we present CTO to improve code translation by syntax-guided and semantic-aware preference optimization. We formulate code translation as a multi-objective preference optimization problem. By leveraging compiler feedback and semantic reward modeling, CTO effectively refines translation quality, ensuring both syntactic correctness and semantic alignment. Experimental results across multiple benchmarks demonstrate that CTO significantly outperforms state-of-the-art methods in code translation accuracy and robustness. The proposed preference optimization approach is also generalizable, making it adaptable to various programming languages and model architectures. In future work, we will extend CTO to support a broader range of programming languages and explore its effectiveness in other software migration scenarios.

IJCAI-ECAI 2026 PREPRINT

Acknowledgments

This work was supported by the "111 Center" (No. B26023), the Fundamental and Interdisciplinary Disciplines Breakthrough Plan of the Ministry of Education of China (No. JYB2025XDXM118), and the Cooperation Fund of Huawei Cooperation Project (No. TC20230202021-2024-12).

References

[Azar *et al.*, 2024] Mohammad Gheshlaghi Azar, Zhaohan Daniel Guo, Bilal Piot, Remi Munos, Mark Rowland, Michal Valko, and Daniele Calandriello. A general theoretical paradigm to understand learning from human preferences. In *PMLR*, volume 238, pages 4447–4455, Palau de Congresos, VLC, Spain, 2024. PMLR.

[Chen *et al.*, 2021] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *CoRR*, 2107.03374:1–35, 2021.

[Du *et al.*, 2024] Yali Du, Hui Sun, and Ming Li. A joint learning model with variational interaction for multilingual program translation. In *ASE*, pages 1907–1918, Sacramento, CA, USA, 2024. ACM.

[Feng *et al.*, 2020] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. CodeBERT: A pre-trained model for programming and natural languages. In *EMNLP*, pages 1536–1547, Virtual, 2020. ACL.

[Feng *et al.*, 2024] Duanyu Feng, Bowen Qin, Chen Huang, Zheng Zhang, and Wenqiang Lei. Towards analyzing and understanding the limitations of DPO: A theoretical perspective. *CoRR*, 2404.04626:1–8, 2024.

[Gee *et al.*, 2025] Leonidas Gee, Milan Gritta, Gerasimos Lampouras, and Ignacio Iacobacci. Code-Optimise: Self-generated preference data for correctness and efficiency. In *Findings of NAACL*, pages 79–94, Albuquerque, NM, USA, 2025. ACL.

[Guo *et al.*, 2021] Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie Liu, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu, et al. GraphCodeBERT: Pre-training code representations with data flow. In *ICLR*, pages 1–18, Virtual, 2021. OpenReview.net.

[Hu *et al.*, 2022] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *ICLR*, pages 1–26, Virtual, 2022. OpenReview.net.

[Huang *et al.*, 2023] Yufan Huang, Mengnan Qi, Yongqiang Yao, Maoquan Wang, Bin Gu, Colin Clement, and Neel Sundaresan. Program translation via code distillation. In *EMNLP*, pages 10903–10914, Singapore, 2023. ACL.

[Hui *et al.*, 2024] Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, et al. Qwen2.5-coder technical report. *CoRR*, 2409.1218:1–32, 2024.

[Ibrahimzada *et al.*, 2024] Ali Reza Ibrahimzada, Kaiyao Ke, Mrigank Pawagi, Muhammad Salman Abid, Rangeet Pan, Saurabh Sinha, and Reyhaneh Jabbarvand. Repository-level compositional code translation and validation. *CoRR*, 2410.24117:1–22, 2024.

[Khan *et al.*, 2024] Mohammad Abdullah Matin Khan, M. Saiful Bari, Xuan Do Long, Weishi Wang, Md. Rizwan Parvez, and Shafiq Joty. XCodeEval: An execution-based large scale multilingual multitask benchmark for code understanding, generation, translation and retrieval. In *ACL*, pages 6766–6805, Bangkok, Thailand, 2024. ACL.

[Le *et al.*, 2022] Hung Le, Yue Wang, Akhilesh Deepak Gotmare, Silvio Savarese, and Steven Chu Hong Hoi. CodeRL: Mastering code generation through pretrained models and deep reinforcement learning. In *NeurIPS*, pages 21314–21328, New Orleans, LA, USA, 2022. Curran Associates Inc.

[Liu *et al.*, 2023] Fang Liu, Jia Li, and Li Zhang. Syntax and domain aware model for unsupervised program translation. In *ICSE*, pages 755–767, Melbourne, VIC, Australia, 2023. IEEE.

[Lu *et al.*, 2021] Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin B. Clement, Dawn Drain, Daxin Jiang, Duyu Tang, et al. CodeXGLUE: A machine learning benchmark dataset for code understanding and generation. In *NeurIPS*, pages 1–16, Virtual, 2021. Curran Associates, Inc.

[Ma *et al.*, 2025] Zeyao Ma, Xiaokang Zhang, Jing Zhang, Jifan Yu, Sijia Luo, and Jie Tang. Dynamic scaling of unit tests for code reward modeling. In *ACL*, Vienna, Austria, 2025. ACL.

[Meng *et al.*, 2024] Yu Meng, Mengzhou Xia, and Danqi Chen. SimPO: Simple preference optimization with a reference-free reward. In *NeurIPS*, pages 1–38, Vancouver, BC, Canada,, 2024. Curran Associates Inc.

[Nguyen *et al.*, 2014] Anh Tuan Nguyen, Tung Thanh Nguyen, and Tien N. Nguyen. Migrating code with statistical machine translation. In *ICSE Companion*, pages 544–547, Hyderabad, India, 2014. ACM.

[Ouyang *et al.*, 2022] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. In *NeurIPS*, pages 27730–27744, New Orleans, LA, USA, 2022. Curran Associates Inc.

[Pan *et al.*, 2024] Rangeet Pan, Ali Reza Ibrahimzada, Rahul Krishna, Divya Sankar, Lambert Pouguem Wassi, Michele Merler, Boris Sobolev, Raju Pavuluri, Saurabh Sinha, and Reyhaneh Jabbarvand. Lost in translation: A study of bugs introduced by large language models while translating code. In *ICSE*, pages 1–13, Lisbon, Portugal, 2024. Curran Associates Inc.

[Park *et al.*, 2024] Ryan Park, Rafael Rafailov, Stefano Ermon, and Chelsea Finn. Disentangling length from quality in direct preference optimization. In *ACL*, pages 4998–5017, Bangkok, Thailand, 2024. ACL.

IJCAI-ECAI 2026 PREPRINT

[Rafailov *et al.*, 2023] Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D. Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. In *NeurIPS*, volume 36, pages 53728–53741, New Orleans, LA, USA, 2023. Curran Associates, Inc.

[Ren *et al.*, 2020] Shuo Ren, Daya Guo, Shuai Lu, Long Zhou, Shujie Liu, Duyu Tang, Neel Sundaresan, Ming Zhou, Ambrosio Blanco, and Shuai Ma. CodeBLEU: a method for automatic evaluation of code synthesis. *CoRR*, 2009.10297:1–8, 2020.

[Roziere *et al.*, 2020] Baptiste Roziere, Marie-Anne Lachaux, Lowik Chanussot, and Guillaume Lample. Unsupervised translation of programming languages. In *NeurIPS*, pages 20601–20611, Vancouver, BC, Canada, 2020. Curran Associates Inc.

[Roziere *et al.*, 2022] Baptiste Roziere, Jie Zhang, Francois Charton, Mark Harman, Gabriel Synnaeve, and Guillaume Lample. Leveraging automated unit tests for unsupervised code translation. In *ICLR*, pages 1–20, Virtual, 2022. OpenReview.net.

[Rozière *et al.*, 2023] Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. Code Llama: Open foundation models for code. *CoRR*, 2308.12950:1–48, 2023.

[Schulman *et al.*, 2017] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, 1707.06347:1–12, 2017.

[Shao *et al.*, 2024] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Y Wu, et al. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *CoRR*, 2402.03300:1–30, 2024.

[Shojaee *et al.*, 2023] Parshin Shojaee, Aneesh Jain, Sindhu Tipirneni, and Chandan K. Reddy. Execution-based code generation using deep reinforcement learning. *Trans. Mach. Learn. Res.*, 2023:1–26, 2023.

[Szafraniec *et al.*, 2023] Marc Szafraniec, Baptiste Roziere, Hugh Leather Francois Charton, Patrick Labatut, and Gabriel Synnaeve. Code translation with compiler representations. In *ICLR*, pages 1–20, Kigali, Rwanda, 2023. OpenReview.net.

[van den Oord *et al.*, 2018] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *CoRR*, 1807.03748:1–13, 2018.

[Wang *et al.*, 2021] Yue Wang, Weishi Wang, Shafiq Joty, and Steven C.H. Hoi. CodeT5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. In *EMNLP*, pages 8696–8708, Punta Cana, Dominican Republic, 2021. ACL.

[Wang *et al.*, 2024] Yanli Wang, Yanlin Wang, Suiquan Wang, Daya Guo, Jiachi Chen, John Grundy, Xilin Liu, Yuchi Ma, Mingzhi Mao, Hongyu Zhang, et al. RepoTransBench: A real-world benchmark for repository-level code translation. *CoRR*, 2412.17744:1–23, 2024.

[Yan *et al.*, 2023] Weixiang Yan, Yuchen Tian, Yunzhe Li, Qian Chen, and Wen Wang. CodeTransOcean: A comprehensive multilingual benchmark for code translation. In *EMNLP*, pages 5067–5089, Singapore, 2023. ACL.

[Yan *et al.*, 2024] Weixiang Yan, Haitian Liu, Yunkun Wang, Yunzhe Li, Qian Chen, Wen Wang, Tingyu Lin, Weishan Zhao, Li Zhu, Hari Sundaram, et al. CodeScope: An execution-based multilingual multitask multidimensional benchmark for evaluating LLMs on code understanding and generation. In *ACL*, pages 5511–5558, Bangkok, Thailand, 2024. ACL.

[Yang *et al.*, 2024] Zhen Yang, Fang Liu, Zhongxing Yu, Jacky Wai Keung, Jia Li, Shuo Liu, Yifan Hong, Xiaoxue Ma, Zhi Jin, and Ge Li. Exploring and unleashing the power of large language models in automated code translation. *Proc. ACM Softw. Eng.*, 1:1585–1608, 2024.

[Zhang *et al.*, 2025] Kechi Zhang, Ge Li, Yihong Dong, Jingjing Xu, Jun Zhang, Jing Su, Yongfei Liu, and Zhi Jin. CodeDPO: Aligning code models with self generated and verified source code. In *ACL*, Vienna, Austria, 2025. ACL.

[Zheng *et al.*, 2023] Qinkai Zheng, Xiao Xia, Xu Zou, Yuxiao Dong, Shan Wang, Yufei Xue, Lei Shen, Zihan Wang, Andi Wang, Yang Li, et al. CodeGeeX: A pre-trained model for code generation with multilingual benchmarking on HumanEval-X. In *KDD*, pages 5673–5684, Long Beach, CA, USA, 2023. ACM.

[Zhou *et al.*, 2023] Shuyan Zhou, Uri Alon, Sumit Agarwal, and Graham Neubig. CodeBERTScore: Evaluating code generation with pretrained models of code. In *EMNLP*, pages 13921–13937, Singapore, 2023. ACL.

[Zhu *et al.*, 2022a] Ming Zhu, Aneesh Jain, Karthik Suresh, Roshan Ravindran, Sindhu Tipirneni, and Chandan K. Reddy. XLCoST: A benchmark dataset for cross-lingual code intelligence. *CoRR*, 2206.08474:1–20, 2022.

[Zhu *et al.*, 2022b] Ming Zhu, Karthik Suresh, and Chandan K. Reddy. Multilingual code snippets training for program translation. In *AAAI*, volume 36, pages 11783–11790, Virtual, 2022. AAAI.

IJCAI-ECAI 2026 PREPRINT