Parallel-SFT: Improving Zero-Shot Cross-Programming-Language Transfer for Code RL

Zhaofeng Wu[♋] **Shiqi Wang**[♒] **Emma Peng**[♒] **Anuj Goyal**[♒] **Shruti Bhosale**[♒]
Melanie Kambadur[♒] **Sebastian Ruder**[♒] **Yoon Kim**[♋] **Chloe Bi**[♒]
[♒]Meta Superintelligence Labs [♋]MIT

Abstract

Modern language models demonstrate impressive coding capabilities in common programming languages (PLs), such as C++ and Python, but their performance in lower-resource PLs is often limited by training data availability. In principle, however, most programming skills are universal across PLs, so the capability acquired in one PL should transfer to others. In this work, we propose the task of zero-shot cross-programming-language transfer for code RL. We find that, for Llama-3.1, RL training for code generation in a source PL fails to improve, and sometimes even degrades, the performance on other target PLs. To address this, we hypothesize that effective RL transfer requires a generalizable SFT initialization before RL. We thus propose **Parallel-SFT**, an SFT strategy that incorporates "parallel programs"—functionally equivalent code implemented in multiple PLs—into the data mixture. We demonstrate that this improves transferability: when we subsequently perform RL on our Parallel-SFT model, we observe better generalization to unseen PLs. Analysis of the model internal representations reveals that Parallel-SFT leads to a more functionality-centric latent space, where equivalent programs across PLs are more tightly clustered, which we hypothesize to contribute to the improved transferability.

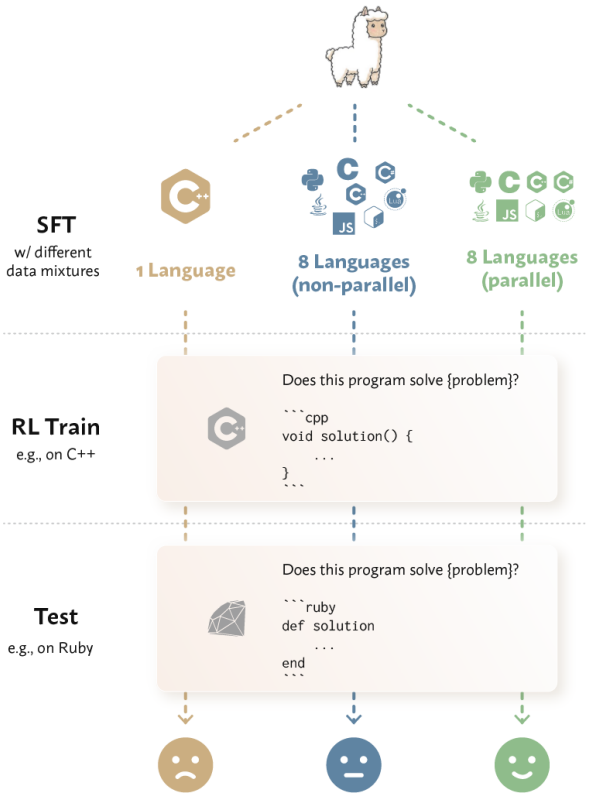


Figure 1: We propose zero-shot cross-PL transfer for code RL, where we perform RL training and testing on different PLs. Llama-3.1 fails to transfer effectively (§3; not represented in this figure). We find that, compared to single-PL SFT, multi-PL SFT improves transferability in downstream RL, which is further enhanced when the SFT code programs are parallel (§4). The code validation task is shown here as an example.

1 Introduction

As language models (LMs) are increasingly integrated into real-world coding workflows, their ability to handle the long tail of programming languages (PLs) becomes critical. Yet, LMs often show vastly disparate performance across PLs depending on their resource availability (Cassano et al., 2023, 2024; Blinn et al., 2024; Xu et al., 2025; Chai et al., 2025). This disparity signals a fundamental inefficiency in leveraging universal code reasoning. Analogous to human programmers who can learn algorithms in a certain source PL and reproduce them in other target PLs, LMs ideally should also generalize their programming capabilities across PLs.

In this work, we focus on this PL generalization problem, specifically within the RL stage of code post-training, where task-specific training usually takes place. We define the task of **zero-shot cross-programming-language transfer**, optimizing a task policy via RL in a source PL and zero-shot

Findings of the Association for Computational Linguistics: ACL 2026, pages 26583–26598

evaluating its performance on another target PL. Empirically, we find that Llama-3.1 (Grattafiori et al., 2024) exhibits limited transferability: while RL yields consistent benefits within the source PL, these gains fail to transfer to different target PLs, sometimes even degrading the performance.

To overcome this limitation, we hypothesize that successful RL transfer requires a generalizable SFT initialization. We propose **Parallel-SFT**, which incorporates synthetically generated "parallel programs"—functionally-equivalent programs across PLs—into the SFT data mixture (Figure 1). Rather than merely modeling the surface form of code tokens, Parallel-SFT semantically grounds programs with execution equivalence. Across two coding tasks, Parallel-SFT outperforms both a single-source-PL baseline and a standard multi-PL baseline (without parallel programs) in downstream RL transfer. Remarkably, Parallel-SFT sometimes even outperforms direct training on the target PL, surpassing the standard "oracle" data-rich setting, highlighting the effectiveness of our method.

In the multilingual NLP literature, successful LM generalizability between natural languages is often attributed to language-general representations, where multilingual models develop representations that capture the underlying semantics of texts (Pires et al., 2019; Wu and Dredze, 2019; Conneau et al., 2020b; *i.a.*). Similarly, we hypothesize that Parallel-SFT aligns the PL representation space, leading to more semantic code representations. This enables the policy improvements learned during source-PL RL to better propagate to target PLs. Our analysis supports this hypothesis: Parallel-SFT induces higher similarity between parallel programs even in *unseen* PLs. We hope our method inspires future work on training generalizable models not only across PLs, but also across natural languages, domains, and tasks.

2 Background

In this section, we outline the coding tasks considered in this work and the standard post-training pipeline to solve them. We then review the paradigm of zero-shot cross-lingual transfer in NLP, which motivates our experimental setup.

2.1 Coding Tasks

We focus on two coding tasks: code generation and code validation.

In **code generation**, the model takes a natural language description of a coding question, q, and generates a solution code program c. c is evaluated against a suite of instance-specific test cases, and is considered correct iff it passes all. Performance is typically measured using the pass@k metric: for each question, the model samples k solutions, and the instance is considered solved if at least one sample passes all tests. Broadly, the field distinguishes between two difficulty levels: competition coding that consists of complex Olympiad-level questions (Li et al., 2022b), and more general coding questions that are familiar to software engineers (Chen et al., 2021; Austin et al., 2021; Hendrycks et al., 2021; *i.a.*). In this work, we evaluate on the former, but train on the latter too.

In **code validation**, the model takes as input a tuple (q, c) containing a question and a candidate code solution, and it outputs whether c is correct, i.e., whether c solves q. Performance is measured by boolean accuracy.[1]

2.2 Training for Coding Tasks

The standard recipe for training coding tasks includes three stages: pretraining, supervised fine-tuning (SFT), and RL. First, a model is pretrained on massive corpora of text and code. Second, the model undergoes SFT on a dataset of instruction and ground-truth response pairs, $(x^{\text{SFT}}, y^{\text{SFT}})$. Like in pretraining, the SFT data also usually includes both natural language tasks and coding tasks.

Finally, we perform RL to optimize for a specific coding task. It requires a dataset of prompts x^{RL} and a verifier.[2] For code generation, $x^{\text{RL}} = q$, the coding question, and the verifier is a code executor with test cases. For code validation, $x^{\text{RL}} = (q, c)$,[3] and the verifier extracts the boolean prediction from the code solution and compares it against the ground truth. During training, the policy model samples a response, which is scored by the verifier. This binary signal is used to update the policy using RL algorithms such as GRPO (Shao et al., 2024).

2.3 Zero-Shot Cross-Lingual Transfer

Zero-shot cross-lingual transfer is a core paradigm in multilingual NLP. In this setup, a multilingual

[1] Code validation models are useful in RL post-training as reward models to estimate the correctness of generated programs, circumventing the need for tests and the sometimes high execution cost.

[2] Or a reward model for non-verifiable tasks, which we do not consider.

[3] Precisely, the x^{RL} in both cases also requires a template.

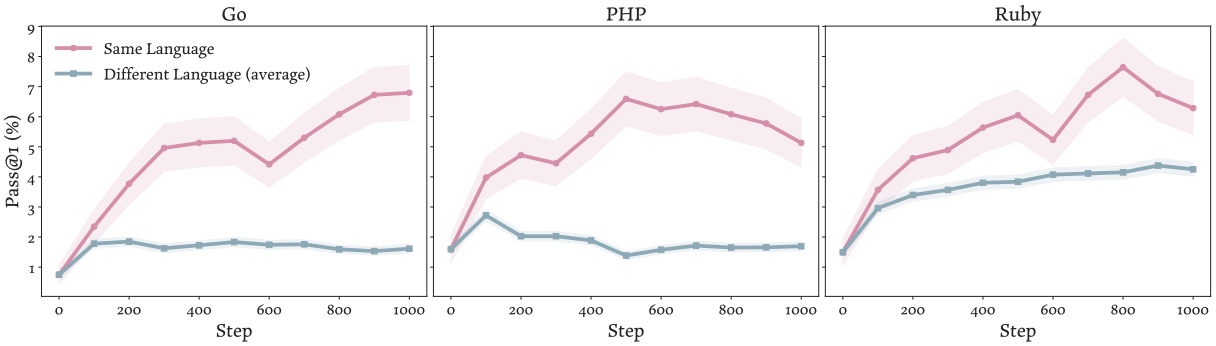


Figure 2: **Failure of Naive Cross-Lingual Transfer.** We visualize the pass@1 performance on a target language (e.g., Go) during RL training on the same vs. different source languages (averaged over 10 source PLs). Shaded regions indicate 95% confidence intervals. While RL on the target language itself leads to consistent gains, cross-PL transfer from a different source language leads to limited improvements or even performance degradation.

model is finetuned on task data in a *source* language and then evaluated zero-shot on the same task in a different *target* language. Despite never seeing any task data in the target language, models often achieve surprisingly high performance (Huang et al., 2019; Conneau and Lample, 2019; Conneau et al., 2020a), provided that they were exposed to unsupervised target-language data during pretraining. This transferability is often attributed to language-general representations (Conneau et al., 2020b). This is most intuitive when an LM is finetuned by training a task-specific head while freezing the pretrained parameters (Peters et al., 2018, 2019; Artetxe and Schwenk, 2019; *i.a.*), where language-general representations allow the task head's decision boundary to naturally generalize across languages (Conneau et al., 2018). While modern models typically update the pretrained parameters too, a language-general initialization still enables a general learning process.

While this paradigm is successful for natural languages, its application to programming languages has yielded mixed results. Early work focused on pretraining encoder models on code-only corpora (Feng et al., 2020; Wang et al., 2021b). These studies typically involved non-generative tasks (e.g., code search; Chen et al., 2022; Baltaji et al., 2025) or training a separate decoder from scratch to pair with the pretrained encoder (Ahmed and Devanbu, 2021; Chen et al., 2022). Despite mostly positive transfer effects, these settings are far removed from the current pretraining paradigm. Recently, Baltaji et al. (2025) investigated cross-PL code generation in modern LLMs via in-context learning, but found that cross-PL transfer often underperforms zero-shot prompting. Furthermore, interpretability studies have revealed low cross-PL representation generality in code models (Utpala et al., 2024), potentially limiting transferability.

3 Cross-Programming-Language Generalization Evaluation

We quantify the zero-shot cross-lingual transferability of current models for code RL. Starting from an LM, we perform RL for a coding task in a *source* PL, and then zero-shot evaluate the resulting model on the same task in another *target* PL.

Model & Task. We use Llama-3.1-8B-Instruct (Grattafiori et al., 2024) as our initial model.[4,5] We consider the task of competition-level code generation and use the CodeForces dataset for training and evaluation (Penedo et al., 2025). See dataset statistics in §B.

Language Selection. Since our goal is to improve performance on medium- and low-resource PLs, we select 3 diverse target PLs: Go, PHP, and Ruby. Respectively, they rank 15th, 16th, and 25th on the TIOBE popularity index (as of December 2025), even lower than PLs like Ada and MATLAB.[6] This categorization aligns with past work on medium-/low-resource code modeling (Ahmed and Devanbu, 2022; Du et al., 2024; Cassano et al.,

[4]It is not an SFT model, which typically precedes the RL stage, but it has also gone through preference tuning with DPO. We do not believe this affects our results, especially since it is common for current LMs to undergo multiple stages of policy optimization (Yang et al., 2025a; Bercovich et al., 2025; *i.a.*). In §A, we validate this with our own custom-trained SFT models and observe qualitatively similar results.

[5]While Qwen models (Qwen et al., 2025; Yang et al., 2025a) have higher raw coding scores, past work has suggested likely substantial data contamination on coding benchmarks, and Llama models to a much less extent (Wu et al., 2025a). We thus choose to use Llama-3.1 as a cleaner testbed.

[6]https://www.tiobe.com/tiobe-index/

2023). We also consider 8 additional languages that we only treat as source: Python, C, C++, Java, C#, JavaScript, Bash, and Lua.

Results. Figure 2 shows the transfer performance throughout RL training, averaged over 10 source languages (11 PLs in total, minus the target language itself). We also show the in-PL performance, where the same target language is used for RL training. We see that while training using the target language yields consistent improvements, transferring from another source language leads to limited, or sometimes degraded effects (negative transfer). This is consistent with the observation from Baltaji et al. (2025) that was based on in-context learning rather than RL. In §A, we show similar results initializing not from the instruct model but the base pretrained model and performing our own source-PL SFT. This confirms that this is a fundamental issue and not due to model variations.

4 Method: Parallel-SFT

Observing unsatisfactory cross-PL generalization, in contrast to prior success for natural languages (§2.3), we reflect on differences between natural languages and programming languages. Past work on multilingual NLP has attributed multilingual capabilities to organic parallel texts (i.e., translations) in pretraining corpora. They function like a "Rosetta Stone" to help align the model representations of different languages. Balashov (2025) distinguishes between two kinds of such parallel texts, "local" (parallel texts within a single pretraining instance) and "global" (parallel texts across different pretraining instances). **We argue that both signals are lacking for programming languages.**

First, natural language corpora benefit from "incidental bilingualism," where a single document contains parallel text. For example, Briakou et al. (2023) found that 0.34% of PaLM's (Chowdhery et al., 2022) pretraining data contains parallel sentences, totaling over 30 million pairs. In contrast, code files are almost exclusively monolingual, let alone containing parallel programs.

Globally, across pretraining instances, parallel sentences abound in training corpora: people discuss similar topics in all languages. Fundamentally, this is because natural languages are grounded in the same physical world and model the same "underlying reality" (Huh et al., 2024). This creates a shared semantic structure that models leverage, for example as their generalizability benefits from domain similarity (Lample et al., 2018a; Conneau et al., 2020b). Programming languages, however, are often domain-specialized. It is rare for an LM training script in Python to have a direct counterpart in Ruby or C++, as those languages are typically applied to different domains (e.g., web development or systems programming). This domain mismatch limits the organic emergence of parallel signals, and in turn leads to empirically fragmented representations across PLs (Utpala et al., 2024) and weak semantic understanding in lower-resource PLs (Gu et al., 2025). This contributes to the negative transfer effects observed in §3.

To bridge this gap, we propose **Parallel-SFT**: a method to improve cross-PL capabilities by explicitly constructing "parallel programs"—functionally equivalent code programs across PLs—anchored by (near-)identical natural language instructions. This is reminiscent of second-order co-occurrences in word models, where models can infer from a sentence pair "I have an adorable pet Corgi." and "I have an adorable pet Samoyed." the relationship between the words "Corgi" and "Samoyed". Similarly, we hypothesize we can improve the model's cross-PL representation with training instances such as "Write a Python implementation of binary search. `def binary_search(...`" and "Write a Java implementation of binary search. `public int binarySearch(...`". Intuitively, this semantically grounds code models with execution equivalence.

Formally, our SFT data follows the format $\left\{\left((q_i, c_i^{\text{Python}}), (q_i, c_i^{\text{C++}}), \cdots\right)\right\}_i$, where the c^{lang} are mutual translations that all solve q. By forcing the model to map the same question q to different language-specific realizations c^{lang}, Parallel-SFT encourages a functionality-centric, rather than syntax-centric, representation space. We will show that Parallel-SFT enables a more generalizable SFT model: the subsequent RL stage in a source language transfers better to target languages.

5 Experimental Setup

5.1 Parallel Data Construction

We construct a high-quality parallel program dataset. Following our examples in §4 and common practices (Rozière et al., 2024; Lambert et al., 2025; *i.a.*), we use instances of the *code generation* task for SFT training. We utilize a combination of the APPS (Hendrycks et al., 2021) and CodeContests (Li et al., 2022a) datasets, which contain

diverse code generation instances. To ensure rigorous evaluation, we filter the data to exclude any CodeForces data which we reserve for RL training and evaluation (§5.3). We also filter out questions that have visual inputs.

These datasets contain mostly solutions in Python. To generate parallel programs in other languages, we perform synthetic code translation: for each instance (q, c^{Python}), we translate c^{Python} into $c^{\text{C++}}$, c^{Java}, etc., by prompting Llama-4-Maverick, a 400B-parameter mixture-of-expert model (Meta AI, 2025). We include our translation prompt in §C. This model generally has good translation accuracy: it achieves 57.3% pass@1 when validated on the tests in the datasets. We further filter out all translations that fail the tests to ensure translation quality.[7]

These datasets contain multiple Python solutions per question. We sample 8 generations per source c^{Python} solution to ensure an abundance of candidate programs. Our final synthetic dataset contains 3,111 unique questions and, on average, 181 instances per language per question, totaling ≈6.2 million instances. Formally, the dataset is $\{(q_i, (C_i^{\text{Python}}, C_i^{\text{C++}}, \cdots))\}_{i=1}^{N}$ where $N = 3,111$. Each set $C_i^{\text{lang}} = \{c_{ij}^{\text{lang}}\}_{j=1}^{N_i^{\text{lang}}}$ is the set of verified code solutions in PL "lang", with N_i^{lang} averaging to 181. This amplifies the parallel program signal: if a question is only seen once per language, it could easily be forgotten in the course of training.

5.2 SFT Dataset Mixtures

We construct a realistic general-purpose SFT dataset by utilizing the Tulu 3 SFT mixture (939k instances; Lambert et al., 2025). We only retain its natural language data on instruction following, safety, etc. To be clean, we remove its coding instances (N=142k) and replace them with an equal-sized sample from our synthetic dataset in §5.1.

For this sampled subset (of N=142k), we compare three mixtures:

1. **1 Language (source)**: This is the monolingual baseline where the SFT data contains code exclusively in the source language of downstream RL training. This represents the standard setting where one relies solely on the high-resource language available for both SFT and RL. We ensure that all 3,111 questions in our dataset are included, uniformly sampling solutions within each question.
2. **8 Languages (parallel)**: This is our proposed method. We distribute the 142k instances equally across the 8 languages and all 3,111 questions. *All* questions appear in *all* languages. We use Python, C, C++, Java, C#, JavaScript, Bash, and Lua, the same as in §3.
3. **8 Languages (non-parallel)**: To isolate the effect of parallel programs from simply seeing more PLs during SFT, we construct a PL-disjoint mixture. We ensure that no question appears in more than one language, removing the parallel alignment signal. The solution distribution across languages is still uniform.

These mixtures have the same diversity as they cover the same number of questions (N=3,111) and the same number of solutions (N=142k). They differ in the number of solutions per question and per language.[8]

To put the transfer performance into perspective, we also consider an oracle: **1 Language (target)**. It is analogous to 1 Language (source) but the data is exclusively in the target language of the downstream transfer. In the subsequent RL stage, this SFT model is always paired with target-language RL training data, representing the ideal "data-rich" scenario where transfer is unnecessary.

5.3 Training Setup

We evaluate the transfer effect from a source PL to another target PL. For the source, we consider C++ and Python, two high-resource languages. For the target PL, we consider Go, PHP, and Ruby, the same as in §3, all of which are unseen during SFT. We also consider the other SFT languages as target languages whenever appropriate.

We initialize from the Llama-3.1-8B base model[9] and train 3 SFT models on the mixtures in §5.2,[10] as well as the oracle. For the oracle, we continue from the 1 Language (target) SFT model and further RL-train it with target PL data to get the "data-rich" in-language training performance.

We consider the two coding tasks in §2.1. For code generation, we use CodeForces, following §3. For code validation, we use the CodeForces submissions dataset[11] that consists of human sub-

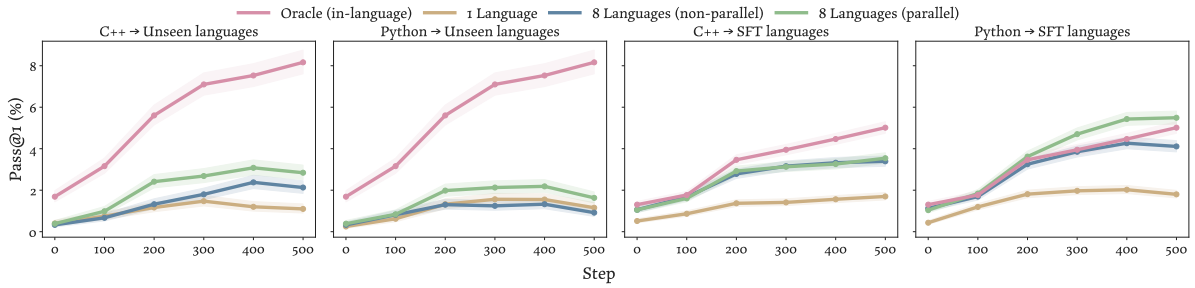
[7] As a sanity check, we also filter out original Python solutions that fail the tests in our execution environment.

[8] It may also be tempting to compare to the original Tulu coding mixture, but this would be confounded by distinct dataset properties, e.g., data complexity, the PLs covered, etc.

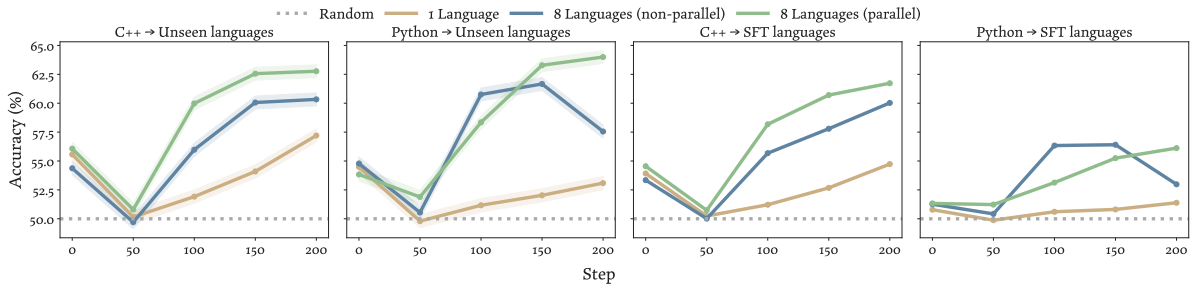
[9] See Footnote 5 for our rationale.

[10] Note that we have two 1 Language (source) models corresponding to the two source languages.

[11] https://huggingface.co/datasets/open-r1/codeforces-submissions



(a) Code generation.



(b) Code validation.

Figure 3: **Parallel-SFT improves cross-PL RL transfer.** We report performance across two source languages and two coding tasks. We report the (average) transfer effect to 3 unseen languages (left) and the SFT languages (right; 7 PLs for code generation *excluding the source*; 5 for code validation (see Footnote 15)). Shaded regions denote 95% confidence intervals. Parallel-SFT yields the best transferability, outperforming the baselines and, in some cases, surpassing the target-language oracle. Non-parallel multi-PL training sometimes is beneficial too.

missions to the CodeForces website. It contains submissions in multiple PLs, including all of our 3 target languages. See dataset statistics and additional training details in §B.

During evaluation, we sample 8 times per question for both tasks. Pass@8 for code generation requires all 8 samples. For pass@1 as well as boolean accuracy for code validation,[12] we treat them as independent samples to reduce estimation variance.

6 Results

Figure 3a presents the transfer performance for code generation. On the left, we visualize the zero-shot transfer results to the 3 target PLs unseen during SFT. When C++ is the source language, simple (non-parallel) multi-PL training already offers generalization benefits over the monolingual baseline. Theoretically, this is consistent with prior literature that posited multi-task training as a form of meta-learning that facilitates better downstream adaptation (Wang et al., 2021a). Empirically, this aligns with the multilingual literature where more training languages enable better generalization (Arivazhagan et al., 2019; Aharoni et al., 2019; *i.a.*). Parallel-SFT enables further improvements and leads to the best transfer performance throughout RL.

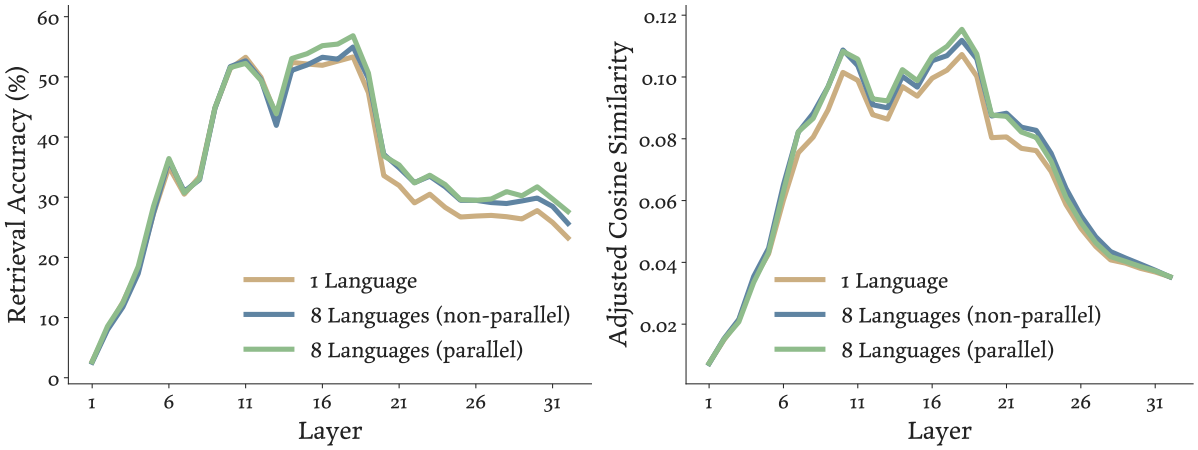
On the right, we show the performance when the target PL is in the 8-Language SFT mixture.[13] Remarkably, when Python is the source PL, Parallel-SFT not only outperforms the baselines but even the in-language oracle.[14] This supports our hypothesis that Parallel-SFT creates a high-quality "generalist" initialization that allows models to leverage abundant high-quality source data effectively, compared to SFT- and RL-training on the target PL alone, which may suffer from noise. Additional results measured in pass@8 (§E) show similar trends.

While the code generation results validate the effectiveness of Parallel-SFT, we acknowledge that zero-shot transfer is primarily a scientific question for this task, but not a practical challenge. This is because its training instances are PL-agnostic, with only a natural language prompt and stdin-stdout tests. When we "train in a source language," we

[12] Though accuracy is the metric, we do not train a classification model because LMs can develop chains-of-thought.

[13] Here, comparing the 8 Languages models with the 1 Language model is no longer fair because the latter does not see these languages during SFT, unlike the former.

[14] We conjecture that this may be due to models learning more effectively on higher-resource PLs, analogous to multilingual models learning more effectively in English. We leave further studies on this phenomenon to future work.



(a) Retrieval accuracy

(b) Adjusted cosine similarity

Figure 4: **Parallel-SFT improves representation generality.** We measure the cross-PL alignment of program representations for unseen languages across model layers. Non-parallel multi-PL training induces more aligned parallel program representations, while Parallel-SFT further improves them. The inverted U-shape suggests that alignment peaks in the middle layers, where semantic reasoning occurs.

instruct the model to generate programs in that language and execute the output in that language. Therefore, while the improved transferability from Parallel-SFT is still scientifically meaningful, we also consider the code validation task. In code validation, the code program in the input *is* PL-dependent, making zero-shot transfer a real practical problem. Figure 3b summarizes the results.[15] Overall, we observe similar trends to code generation, corroborating the utility of Parallel-SFT.

Furthermore, in §D, we also confirm that Parallel-SFT, while achieving improved cross-language generalizability, does not sacrifice in-language performance.

7 Analysis

Inspired by the multilingual NLP literature (§2.3), we investigate whether the improved transferability correlates with more PL-general representations. Specifically, we check if Parallel-SFT promotes the representation similarity between standalone parallel programs (i.e., without contexts), as is standard in multilingual research (Wu et al., 2025b; Shen et al., 2025; Gao et al., 2025; *i.a.*).

We construct a held-out dataset with 312 code programs from our SFT distribution, each from a different question. None of the questions has been observed in SFT. We consider translations of these programs in 3 languages unseen during SFT—Go, PHP, and Ruby—obtained from the synthetic translation procedure in §5.1. This yields a list of parallel programs $\{(c_i^{\text{Go}}, c_i^{\text{PHP}}, c_i^{\text{Ruby}})\}_{i=1}^{N}$ where $N = 312$.

For each SFT model, we derive program representations, separately at each layer ℓ, e.g. $\mathbf{r}_{i,\ell}^{\text{Go}}$. We compute the program representations via echo embedding (Springer et al., 2025) that repeats the program twice in a lightweight template which empirically produces strong sequence representations.[16]

Representation similarity is most straightforwardly computable by cosine similarity. It, however, can be misleading due to the *anisotropy* of the representation space (Ethayarajh, 2019). That is, it is possible that a model simply utilizes a small subset of its representation space for code, embedding *all* programs closer together, not just parallel programs. We account for this in two robust metrics:

1. **Retrieval accuracy** measures whether a program and its translation have closer representations than all other non-corresponding programs (Artetxe and Schwenk, 2019; Feng et al., 2022; Shen et al., 2025; *i.a.*). Formally, e.g., we measure the accuracy $\cos(\mathbf{r}_{i,\ell}^{\text{Go}}, \mathbf{r}_{i,\ell}^{\text{PHP}}) > \max_{j \neq i} \cos(\mathbf{r}_{i,\ell}^{\text{Go}}, \mathbf{r}_{j,\ell}^{\text{PHP}})$.

[15] Because of the realistic data constraint, we no longer have sufficient target-PL RL data for the oracle results for code validation. Additionally, the RL dataset lacks data in Bash and Lua, so when we consider the SFT languages as the target PLs, we only average over 5 languages (8 training languages minus the source PL itself as well as Bash and Lua).

[16] The intuition is that autoregressive models do not attend to the future, but future information is helpful for a good representation. By repeating the sequence twice, the early tokens in the second occurrence can attend to future tokens in the first occurrence. We lightly adapt the template in Springer et al. (2025): "Rewrite the following code: {x}. The rewritten code: {x}" where "{x}" is substituted with the code program. We mean-pool the token representations in the second "{x}".

2. **Adjusted cosine similarity.** Following Wu et al. (2025b), we correct for baseline anisotropy by subtracting the cosine similarities between non-parallel pairs. Formally, e.g., we measure $\cos(\mathbf{r}^{\text{Go}}_{i,\ell}, \mathbf{r}^{\text{PHP}}_{i,\ell}) - \mathbb{E}_{j\neq i}[\cos(\mathbf{r}^{\text{Go}}_{i,\ell}, \mathbf{r}^{\text{PHP}}_{j,\ell})]$. We estimate the second term with a single random sample.

We show the results in Figure 4. We observe that simple (non-parallel) multi-PL training already improves program alignment over the monolingual baseline, and explicit parallel program training further promotes it. This confirms that Parallel-SFT induces more functionality-centric representations.

Furthermore, interestingly, the similarity measures both follow an inverted U-shape, higher in intermediate layers and lower on the two ends. This is a familiar trend in multilingual studies (Pires et al., 2019; Wu and Dredze, 2019; Conneau et al., 2020b; Wu et al., 2025b; *i.a.*). The initial/final layers need to map from/to the surface form of PLs and are thus more specialized to the syntax of individual PLs. The middle layers, on the other hand, serve as a "semantic hub" and perform more abstract reasoning (Wendler et al., 2024; Wu et al., 2025b). Parallel-SFT accentuates this trend: for example, its adjusted cosine similarities are similar to the baseline models on the two ends, but notably higher in the middle layers, where more PL-independent reasoning occurs.

8 Discussion and Related Work

Multilingual representation learning is a foundational problem in NLP, from traditional word embeddings (Mikolov et al., 2013; Al-Rfou' et al., 2013; Ammar et al., 2016; *i.a.*) to later sequence models (Schwenk and Douze, 2017; España-Bonet et al., 2017; Eriguchi et al., 2018; *i.a.*). These models achieve remarkable zero-shot cross-lingual generalizability, both on early classification tasks such as entailment (Huang et al., 2019; Conneau and Lample, 2019; Conneau et al., 2020a; Hu et al., 2020; *i.a.*) and more recently on tasks in the LM post-training pipeline, such as SFT and reward modeling (Wu et al., 2024; Shaham et al., 2024; Chen et al., 2024; Shimabucoro et al., 2025; *i.a.*).

This success is typically driven by two factors: **structural isomorphism** and **explicit alignment signals**. Early approaches rely on the latter, providing explicit supervised signals such as bilingual lexicons (Mikolov et al., 2013; Kočiský et al., 2014; Faruqui and Dyer, 2014; Ammar et al., 2016; *i.a.*) or parallel texts (Schwenk and Douze, 2017; Artetxe and Schwenk, 2019; Conneau and Lample, 2019; Huang et al., 2019; *i.a.*). Later work showed that representation alignment could also emerge from purely unsupervised signals, due to structural isomorphism alone (Lample et al., 2018a,b), although explicit parallel data is still helpful (Reid and Artetxe, 2023; Shen et al., 2025). However, we argue that these factors are often absent for programming languages.

First, natural languages have distributionally similar lexicons, where most words have translations in other languages. However, PLs' lexical structures usually differ due to heterogeneous paradigms. For instance, non-object-oriented languages lack keywords related to classes; some PLs rely on specific control flow structures like `switch` statements or lazy returns (`yield`) that others lack; and mechanisms such as garbage collection and concurrency can be explicitly specified via certain keywords in some PLs but are implicit in others. Even for shared concepts, usage distributions diverge vastly: while the `boolean` type is universal, it may be syntactically required in Java (`boolean x = true;`) but implicitly typed in Python (`x = True`).

Beyond structure similarities, parallel natural language sentences provide explicit alignment signals, whether deliberately leveraged in pretrained objectives or organically emerging in unsupervised corpora. However, as we discussed in §4, they are often lacking for programming languages too. Without these explicit signals or the implicit structural isomorphism of natural languages, zero-shot transfer in code becomes challenging.

Prior work has attempted to bridge this gap through either data-centric or structure-centric methods. Data-centric approaches attempt to scale up multi-PL datasets, but existing resources are typically scarce beyond small-scale evaluation benchmarks (Khan et al., 2024; Chai et al., 2025; Xu et al., 2025; *i.a.*). Others have synthetically generated data in low-resource PLs via translation (Cassano et al., 2024), a practice termed "translate-train" in the multilingual NLP nomenclature (Conneau et al., 2018). Alternatively, structure-centric approaches enforce alignment by leveraging language-agnostic intermediate forms such as Abstract Syntax Trees (Guo et al., 2022; Chen et al., 2023; Gong et al., 2024; *i.a.*) and Intermediate Representations (Ma et al., 2023; Szafraniec et al., 2023; Paul et al., 2024).

Our approach diverges from these methods. We augment data via synthetic translation, similar to

translate-train methods, but with more flexibility afforded by zero-shot cross-PL transfer rather than requiring task- and language-specific translations. And unlike structure-centric methods, we do not need parsers or compilers, allowing the model flexibility to learn necessary structures from raw text—analogous to the shift in NLP away from explicit syntactic and semantic structures. Concurrent work (Yang et al., 2025b) also leverages parallel programs, but focuses on pretraining scaling laws.

9 Conclusion

In this work, we identified unsatisfactory zero-shot cross-PL transferability for Llama-3.1. To address this, we proposed Parallel-SFT that uses parallel programs to train a more generalizable SFT model prior to RL. We demonstrated that our method improves transferability across PLs, and our analysis confirms that Parallel-SFT induces a more PL-general representation space through grounding code with execution equivalence. This kind of semantic understanding of code is crucial for many application including coding agents and code retrieval (Gu et al., 2025), and we hope that Parallel-SFT inspires future work to train more general and transferrable code models.

Limitations

While Parallel-SFT demonstrates improvements in cross-PL transfer, our exploration was not exhaustive. We did not perform a search over many design decisions, including SFT data formatting, curricula, or mixing ratios. Further iterations on these could yield further gains, perhaps additionally leveraging typological similarities between PLs, which would better inform transfer. Additionally, our experiments focused on fundamental coding tasks. We anticipate that the benefits of Parallel-SFT extend to more complex settings, such as reasoning models and coding agents, but verifying this remains an open question. We leave these explorations to future work.

Acknowledgments

We thank Alex Gu, Naman Jain, Yuning Mao, and colleagues at the Meta Superintelligence Labs for discussions and help at various stages of this project.

References

Roee Aharoni, Melvin Johnson, and Orhan Firat. 2019. Massively multilingual neural machine translation. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 3874–3884, Minneapolis, Minnesota. Association for Computational Linguistics.

Toufique Ahmed and Prem Devanbu. 2021. Multilingual training for software engineering. *2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*, pages 1443–1455.

Toufique Ahmed and Premkumar Devanbu. 2022. Multilingual training for software engineering. In *Proceedings of the 44th International Conference on Software Engineering*, ICSE '22, page 1443–1455, New York, NY, USA. Association for Computing Machinery.

Rami Al-Rfou', Bryan Perozzi, and Steven Skiena. 2013. Polyglot: Distributed word representations for multilingual NLP. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*, pages 183–192, Sofia, Bulgaria. Association for Computational Linguistics.

Waleed Ammar, George Mulcaire, Yulia Tsvetkov, Guillaume Lample, Chris Dyer, and Noah A. Smith. 2016. Massively multilingual word embeddings. *Preprint*, arXiv:1602.01925.

Naveen Arivazhagan, Ankur Bapna, Orhan Firat, Dmitry Lepikhin, Melvin Johnson, Maxim Krikun, Mia Xu Chen, Yuan Cao, George Foster, Colin Cherry, Wolfgang Macherey, Zhifeng Chen, and Yonghui Wu. 2019. Massively multilingual neural machine translation in the wild: Findings and challenges. *Preprint*, arXiv:1907.05019.

Mikel Artetxe and Holger Schwenk. 2019. Massively multilingual sentence embeddings for zero-shot cross-lingual transfer and beyond. *Transactions of the Association for Computational Linguistics*, 7:597–610.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. 2021. Program synthesis with large language models. *Preprint*, arXiv:2108.07732.

Yuri Balashov. 2025. Translation in the wild. *Information*, 16(12).

Razan Baltaji, Saurabh Pujar, Martin Hirzel, Louis Mandel, Luca Buratti, and Lav R. Varshney. 2025. Cross-lingual transfer in programming languages: An extensive empirical study. *Transactions on Machine Learning Research*.

Akhiad Bercovich, Itay Levy, Izik Golan, Mohammad Dabbah, Ran El-Yaniv, Omri Puny, Ido Galil, Zach

Moshe, Tomer Ronen, Najeeb Nabwani, Ido Shahaf, Oren Tropp, Ehud Karpas, Ran Zilberstein, Jiaqi Zeng, Soumye Singhal, Alexander Bukharin, Yian Zhang, Tugrul Konuk, and 117 others. 2025. Llama-nemotron: Efficient reasoning models. *Preprint*, arXiv:2505.00949.

Andrew Blinn, Xiang Li, June Hyung Kim, and Cyrus Omar. 2024. Statically contextualizing large language models with typed holes. *Proc. ACM Program. Lang.*, 8(OOPSLA2).

Eleftheria Briakou, Colin Cherry, and George Foster. 2023. Searching for needles in a haystack: On the role of incidental bilingualism in PaLM's translation capability. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9432–9452, Toronto, Canada. Association for Computational Linguistics.

Federico Cassano, John Gouwar, Francesca Lucchetti, Claire Schlesinger, Anders Freeman, Carolyn Jane Anderson, Molly Q Feldman, Michael Greenberg, Abhinav Jangda, and Arjun Guha. 2024. Knowledge transfer from high-resource to low-resource programming languages for code llms. *Proc. ACM Program. Lang.*, 8(OOPSLA2).

Federico Cassano, John Gouwar, Daniel Nguyen, Sydney Nguyen, Luna Phipps-Costin, Donald Pinckney, Ming-Ho Yee, Yangtian Zi, Carolyn Jane Anderson, Molly Q Feldman, Arjun Guha, Michael Greenberg, and Abhinav Jangda. 2023. Multipl-e: A scalable and polyglot approach to benchmarking neural code generation. *IEEE Trans. Softw. Eng.*, 49(7):3675–3691.

Linzheng Chai, Shukai Liu, Jian Yang, Yuwei Yin, JinKe, Jiaheng Liu, Tao Sun, Ge Zhang, Changyu Ren, Hongcheng Guo, Noah Wang, Boyang Wang, Xianjie Wu, Bing Wang, Tongliang Li, Liqun Yang, Sufeng Duan, Zhaoxiang Zhang, and Zhoujun Li. 2025. McEval: Massively multilingual code evaluation. In *The Thirteenth International Conference on Learning Representations*.

Fuxiang Chen, Fatemeh H. Fard, David Lo, and Timofey Bryksin. 2022. On the transferability of pre-trained language models for low-resource programming languages. In *Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension*, ICPC '22, page 401–412, New York, NY, USA. Association for Computing Machinery.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021. Evaluating large language models trained on code. *Preprint*, arXiv:2107.03374.

Nuo Chen, Qiushi Sun, Jianing Wang, Xiang Li, and Ming Gao. 2023. Pass-tuning: Towards structure-aware parameter-efficient tuning for code representation learning. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 577–591, Singapore. Association for Computational Linguistics.

Pinzhen Chen, Shaoxiong Ji, Nikolay Bogoychev, Andrey Kutuzov, Barry Haddow, and Kenneth Heafield. 2024. Monolingual or multilingual instruction tuning: Which makes a better alpaca. In *Findings of the Association for Computational Linguistics: EACL 2024*, pages 1347–1356, St. Julian's, Malta. Association for Computational Linguistics.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, and 48 others. 2022. Palm: Scaling language modeling with pathways. *Preprint*, arXiv:2204.02311.

Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. 2020a. Unsupervised cross-lingual representation learning at scale. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8440–8451, Online. Association for Computational Linguistics.

Alexis Conneau and Guillaume Lample. 2019. *Cross-lingual language model pretraining*. Curran Associates Inc., Red Hook, NY, USA.

Alexis Conneau, Ruty Rinott, Guillaume Lample, Adina Williams, Samuel Bowman, Holger Schwenk, and Veselin Stoyanov. 2018. XNLI: Evaluating cross-lingual sentence representations. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2475–2485, Brussels, Belgium. Association for Computational Linguistics.

Alexis Conneau, Shijie Wu, Haoran Li, Luke Zettlemoyer, and Veselin Stoyanov. 2020b. Emerging cross-lingual structure in pretrained language models. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6022–6034, Online. Association for Computational Linguistics.

Yangkai Du, Tengfei Ma, Lingfei Wu, Xuhong Zhang, and Shouling Ji. 2024. Adaccd: adaptive semantic contrasts discovery based cross lingual adaptation for code clone detection. In *Proceedings of the Thirty-Eighth AAAI Conference on Artificial Intelligence and Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence and Fourteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'24/IAAI'24/EAAI'24. AAAI Press.

Akiko Eriguchi, Melvin Johnson, Orhan Firat, Hideto Kazawa, and Wolfgang Macherey. 2018. Zero-shot

cross-lingual classification using multilingual neural machine translation. *Preprint*, arXiv:1809.04686.

Cristina España-Bonet, Ádám Csaba Varga, Alberto Barrón-Cedeño, and Josef van Genabith. 2017. An empirical analysis of nmt-derived interlingual embeddings and their use in parallel sentence identification. *IEEE Journal of Selected Topics in Signal Processing*, 11:1340–1350.

Kawin Ethayarajh. 2019. How contextual are contextualized word representations? Comparing the geometry of BERT, ELMo, and GPT-2 embeddings. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 55–65, Hong Kong, China. Association for Computational Linguistics.

Manaal Faruqui and Chris Dyer. 2014. Improving vector space word representations using multilingual correlation. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages 462–471, Gothenburg, Sweden. Association for Computational Linguistics.

Fangxiaoyu Feng, Yinfei Yang, Daniel Cer, Naveen Arivazhagan, and Wei Wang. 2022. Language-agnostic BERT sentence embedding. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 878–891, Dublin, Ireland. Association for Computational Linguistics.

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. CodeBERT: A pre-trained model for programming and natural languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1536–1547, Online. Association for Computational Linguistics.

Changjiang Gao, Hankun Lin, Xin Huang, Xue Han, Junlan Feng, Chao Deng, Jiajun Chen, and Shujian Huang. 2025. Understanding LLMs' cross-lingual context retrieval: How good it is and where it comes from. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 22808–22837, Suzhou, China. Association for Computational Linguistics.

Linyuan Gong, Mostafa Elhoushi, and Alvin Cheung. 2024. AST-t5: Structure-aware pretraining for code generation and understanding.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, and 542 others. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Alex Gu, Naman Jain, Wen-Ding Li, Manish Shetty, Kevin Ellis, Koushik Sen, and Armando Solar-Lezama. 2025. Tasks, challenges, and paths towards AI for software engineering. In *ICLR 2025 Workshop: VerifAI: AI Verification in the Wild*.

Daya Guo, Shuai Lu, Nan Duan, Yanlin Wang, Ming Zhou, and Jian Yin. 2022. UniXcoder: Unified cross-modal pre-training for code representation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7212–7225, Dublin, Ireland. Association for Computational Linguistics.

Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt. 2021. Measuring coding challenge competence with APPS. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

Junjie Hu, Sebastian Ruder, Aditya Siddhant, Graham Neubig, Orhan Firat, and Melvin Johnson. 2020. Xtreme: A massively multilingual multi-task benchmark for evaluating cross-lingual generalisation. In *International conference on machine learning*, pages 4411–4421. PMLR.

Haoyang Huang, Yaobo Liang, Nan Duan, Ming Gong, Linjun Shou, Daxin Jiang, and Ming Zhou. 2019. Unicoder: A universal language encoder by pre-training with multiple cross-lingual tasks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2485–2494, Hong Kong, China. Association for Computational Linguistics.

Minyoung Huh, Brian Cheung, Tongzhou Wang, and Phillip Isola. 2024. The platonic representation hypothesis. In *Proceedings of Machine Learning Research (ICML)*.

Mohammad Abdullah Matin Khan, M Saiful Bari, Xuan Long Do, Weishi Wang, Md Rizwan Parvez, and Shafiq Joty. 2024. XCodeEval: An execution-based large scale multilingual multitask benchmark for code understanding, generation, translation and retrieval. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6766–6805, Bangkok, Thailand. Association for Computational Linguistics.

Tomáš Kočiský, Karl Moritz Hermann, and Phil Blunsom. 2014. Learning bilingual word representations by marginalizing alignments. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 224–229, Baltimore, Maryland. Association for Computational Linguistics.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman,

Lester James V. Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, Yuling Gu, Saumya Malik, Victoria Graf, Jena D. Hwang, Jiangjiang Yang, Ronan Le Bras, Oyvind Tafjord, Chris Wilhelm, Luca Soldaini, and 4 others. 2025. Tulu 3: Pushing frontiers in open language model post-training. *Preprint*, arXiv:2411.15124.

Guillaume Lample, Alexis Conneau, Marc'Aurelio Ranzato, Ludovic Denoyer, and Hervé Jégou. 2018a. Word translation without parallel data. In *International Conference on Learning Representations*.

Guillaume Lample, Myle Ott, Alexis Conneau, Ludovic Denoyer, and Marc'Aurelio Ranzato. 2018b. Phrase-based & neural unsupervised machine translation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 5039–5049, Brussels, Belgium. Association for Computational Linguistics.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, and 7 others. 2022a. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, and 7 others. 2022b. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097.

Yingwei Ma, Yue Yu, Shanshan Li, Zhouyang Jia, Jun Ma, Rulin Xu, Wei Dong, and Xiangke Liao. 2023. Mulcs: Towards a unified deep representation for multilingual code search. In *2023 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 120–131.

Meta AI. 2025. Introducing llama 4: Advancing multimodal intelligence.

Tomas Mikolov, Quoc V Le, and Ilya Sutskever. 2013. Exploiting similarities among languages for machine translation. *arXiv preprint*.

Indraneil Paul, Goran Glavaš, and Iryna Gurevych. 2024. IRCoder: Intermediate representations make language models robust multilingual code generators. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15023–15041, Bangkok, Thailand. Association for Computational Linguistics.

Guilherme Penedo, Anton Lozhkov, Hynek Kydlíček, Loubna Ben Allal, Edward Beeching, Agustín Piqueres Lajarín, Quentin Gallouédec, Nathan Habib, Lewis Tunstall, and Leandro von Werra. 2025. Codeforces. https://huggingface.co/datasets/open-r1/codeforces.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana. Association for Computational Linguistics.

Matthew E. Peters, Sebastian Ruder, and Noah A. Smith. 2019. To tune or not to tune? adapting pretrained representations to diverse tasks. In *Proceedings of the 4th Workshop on Representation Learning for NLP (RepL4NLP-2019)*, pages 7–14, Florence, Italy. Association for Computational Linguistics.

Telmo Pires, Eva Schlinger, and Dan Garrette. 2019. How multilingual is multilingual BERT? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4996–5001, Florence, Italy. Association for Computational Linguistics.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, and 25 others. 2025. Qwen2.5 technical report. *Preprint*, arXiv:2412.15115.

Machel Reid and Mikel Artetxe. 2023. On the role of parallel data in cross-lingual transfer learning. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 5999–6006, Toronto, Canada. Association for Computational Linguistics.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, and 7 others. 2024. Code llama: Open foundation models for code. *Preprint*, arXiv:2308.12950.

Holger Schwenk and Matthijs Douze. 2017. Learning joint multilingual sentence representations with neural machine translation. In *Proceedings of the 2nd Workshop on Representation Learning for NLP*, pages 157–167, Vancouver, Canada. Association for Computational Linguistics.

Uri Shaham, Jonathan Herzig, Roee Aharoni, Idan Szpektor, Reut Tsarfaty, and Matan Eyal. 2024. Multilingual instruction tuning with just a pinch of multilinguality. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 2304–2317, Bangkok, Thailand. Association for Computational Linguistics.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *Preprint*, arXiv:2402.03300.

Yingli Shen, Wen Lai, Shuo Wang, Ge Gao, Kangyang Luo, Alexander Fraser, and Maosong Sun. 2025. From unaligned to aligned: Scaling multilingual LLMs with multi-way parallel corpora. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 7368–7390, Suzhou, China. Association for Computational Linguistics.

Luisa Shimabucoro, Ahmet Ustun, Marzieh Fadaee, and Sebastian Ruder. 2025. A post-trainer's guide to multilingual training data: Uncovering cross-lingual transfer dynamics. *arXiv preprint arXiv:2504.16677*.

Jacob Mitchell Springer, Suhas Kotha, Daniel Fried, Graham Neubig, and Aditi Raghunathan. 2025. Repetition improves language model embeddings. In *The Thirteenth International Conference on Learning Representations*.

Marc Szafraniec, Baptiste Roziere, Hugh James Leather, Patrick Labatut, Francois Charton, and Gabriel Synnaeve. 2023. Code translation with compiler representations. In *The Eleventh International Conference on Learning Representations*.

Saiteja Utpala, Alex Gu, and Pin-Yu Chen. 2024. Language agnostic code embeddings. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 678–691, Mexico City, Mexico. Association for Computational Linguistics.

Haoxiang Wang, Han Zhao, and Bo Li. 2021a. Bridging multi-task learning and meta-learning: Towards efficient training and effective adaptation. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 10991–11002. PMLR.

Yue Wang, Weishi Wang, Shafiq Joty, and Steven C.H. Hoi. 2021b. CodeT5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 8696–8708, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Chris Wendler, Veniamin Veselovsky, Giovanni Monea, and Robert West. 2024. Do llamas work in English? on the latent language of multilingual transformers. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15366–15394, Bangkok, Thailand. Association for Computational Linguistics.

Mingqi Wu, Zhihao Zhang, Qiaole Dong, Zhiheng Xi, Jun Zhao, Senjie Jin, Xiaoran Fan, Yuhao Zhou, Huijie Lv, Ming Zhang, Yanwei Fu, Qin Liu, Songyang Zhang, and Qi Zhang. 2025a. Reasoning or memorization? unreliable results of reinforcement learning due to data contamination. *Preprint*, arXiv:2507.10532.

Shijie Wu and Mark Dredze. 2019. Beto, bentz, becas: The surprising cross-lingual effectiveness of BERT. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 833–844, Hong Kong, China. Association for Computational Linguistics.

Zhaofeng Wu, Ananth Balashankar, Yoon Kim, Jacob Eisenstein, and Ahmad Beirami. 2024. Reuse your rewards: Reward model transfer for zero-shot cross-lingual alignment. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 1332–1353, Miami, Florida, USA. Association for Computational Linguistics.

Zhaofeng Wu, Xinyan Velocity Yu, Dani Yogatama, Jiasen Lu, and Yoon Kim. 2025b. The semantic hub hypothesis: Language models share semantic representations across languages and modalities. In *The Thirteenth International Conference on Learning Representations*.

Ruiyang Xu, Jialun Cao, Yaojie Lu, Ming Wen, Hongyu Lin, Xianpei Han, Ben He, Shing-Chi Cheung, and Le Sun. 2025. CRUXEVAL-X: A benchmark for multilingual code reasoning, understanding and execution. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 23762–23779, Vienna, Austria. Association for Computational Linguistics.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025a. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

Jian Yang, Shawn Guo, Lin Jing, Wei Zhang, Aishan Liu, Chuan Hao, Zhoujun Li, Wayne Xin Zhao, Xianglong Liu, Weifeng Lv, and Bryan Dai. 2025b. Scaling laws for code: Every programming language matters. *Preprint*, arXiv:2512.13472.

A Cross-Programming-Language Evaluation of Custom-Trained SFT Models

In §3, we observed limited transferability using Llama-3.1-8B-Instruct. It is an SFT + DPO model, and we append a code RL training stage. While this is consistent with state-of-the-art practices that include multiple stages of policy optimization (Yang et al., 2025a; Bercovich et al., 2025; *i.a.*), the existing DPO stage may confound transfer effects in the RL stage. Therefore, we conduct a controlled experiment where we train custom SFT models, initializing from the Llama-3.1-8B *base* model. We replicate the 1 Language (source) setup from §5.2 and Figure 1. For each of our 11 possible source PLs, we generate an SFT mixture that contains both the general-purpose Tulu 3 SFT data and our coding SFT data exclusively in the *source* language. The subsequent RL training stage uses the same source PL. This simulates the setting where we only have access to the source PL data for both SFT and RL. As shown in Figure 5, the results are qualitatively similar to those in §3: same-language RL consistently leads to performance improvements, but RL-training using a different PL has limited benefits. This confirms that the transfer failure is fundamental, rather than an artifact of the specific instruct model used.

B RL Training Details

Code generation. We use the CodeForces dataset with its existing train/test split. We exclude instances that have custom checker functions as tests and those that do not use stdin-stdout tests. The resulting dataset has 6,617 training instances and 377 test instances. We train with batch size 16.

Code validation. We use the CodeForces submissions dataset. We use C++ and Python as our source languages because they have the largest amount of data in this dataset. For these two PLs, we manually perform a train-test split. We use batch size 32 and notice that the performance plateaus early and only report performance for 200 steps. We train with all distinct samples for these 6,400 instances. For evaluation, we use the remaining instances for each PL. To prevent class imbalance from skewing the accuracy, we manually balance the correct and incorrect solutions via truncation, on a per-PL basis. The statistics of the final evaluation instances are: Python: 19996 samples, Go: 1502 samples, PHP: 992 samples, Ruby: 1150 samples, Java: 75194 samples, C++: 19998 samples, Rust: 534 samples, Javascript: 2440 samples, C#: 8074 samples.

For both settings, we use a learning rate of 5×10^{-7}.

C Synthetic Translation Prompt

We use the below prompt for synthetic translation (§5.1). `{code}` is the original Python code snippet in the coding dataset; `{instruction}` is the natural language description of the code, from the same source; `{lg_long}` is the human-readable name of the target programming language (e.g., `C++`); and `{lg_short}` is the short name used for Markdown formatting (e.g., cpp).

````
Translate the following code from Python
to {lg_long}. The functionality should
be exactly the same across all possible
inputs.

```python
{code}
```

For reference, the original Python code
was generated according to the following
request:
'''
{instruction}
'''

Now translate the code into {lg_long}
and wrap it in
```{lg_short}
```
Closely follow the stylistic aspects of
the original Python code, for example:
(1) Respecting the amount of comment
(e.g., function-level/line-level/etc.)
and not be over-verbose or under-verbose;
(2) Not writing sample inputs **if** the
original Python code doesn't have them;
(3) The original Python code can be
directly run with top-level statements.
If this is allowed in {lg_long}, follow
this behavior; if this is impossible in
{lg_long}, put the top-level statements
in a canonical entry point, such as a
main function.
````

D In-Language Performance of Parallel-SFT

We empirically test the extent to which parallel program training degrades in-language performance (i.e., training and testing on the same PL). We performed in-language RL training and evaluation (using Python), starting from both the 8-language non-parallel SFT model and the 8-language Parallel-SFT model. After 500 RL steps,

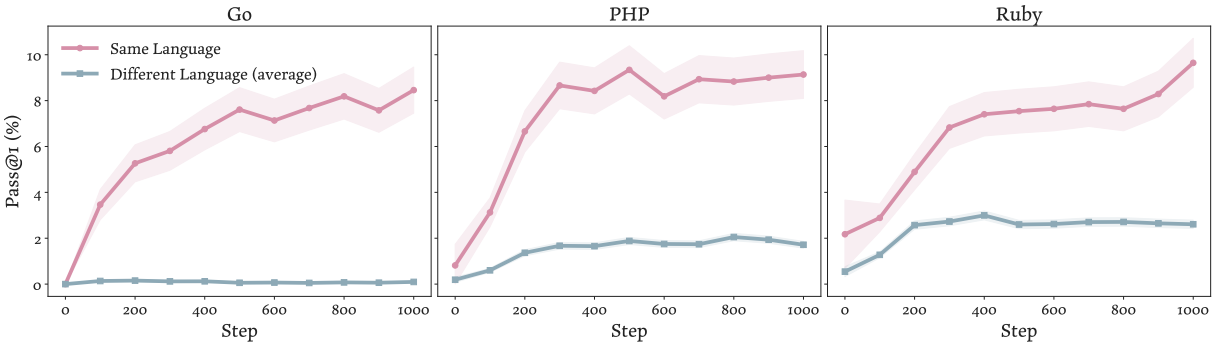


Figure 5: **Validation of Transfer Failure with Custom SFT.** We replicate the experiment from §3 and Figure 2 using custom SFT models trained from the Llama-3.1-8B base model, removing the DPO priors. We visualize the pass@1 performance on a target language (e.g., Go) during RL training on the same vs. different source languages (averaged over 10 source PLs). Shaded regions indicate 95% confidence intervals. Consistent with the instruct model results, cross-PL transfer from a different source language leads to limited improvements compared to same-language training.

non-parallel training leads to 12.47% pass@1 (95% CI: [11.32, 13.71]) and parallel training leads to 12.36% pass@1 (95% CI: [11.22, 13.60]). The difference is not statistically significant, confirming that parallel training does not sacrifice in-language performance.

E Additional Results

Figure 6 shows the code generation results using the pass@8 metric, complementing the pass@1 results in Figure 3a. The trends are consistent, corroborating the effectiveness of Parallel-SFT.

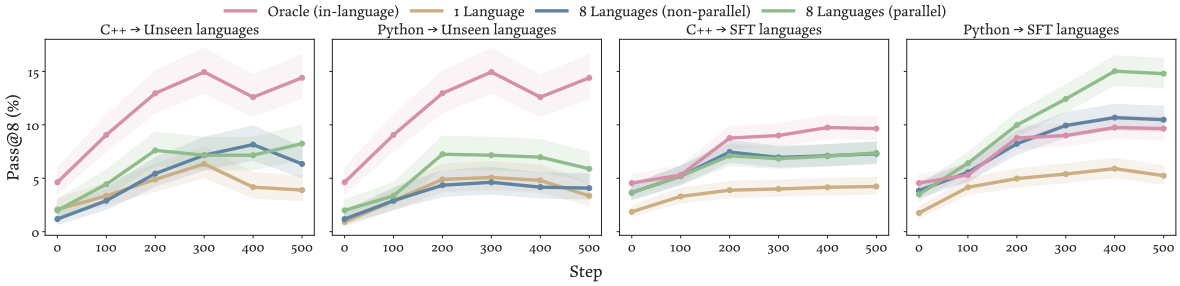


Figure 6: **Parallel-SFT improves cross-PL RL transfer (pass@8).** We report pass@8 performance across two source languages. We report the (average) transfer effect to 3 unseen languages (left) and the SFT languages (right; 7 PLs excluding the source). Shaded regions denote 95% confidence intervals. Consistent with the pass@1 results in Figure 3a, parallel-SFT yields the best transferability, outperforming the baselines and, in some cases, surpassing the target-language oracle.