ERNIE-Code: Beyond English-Centric Cross-lingual Pretraining for Programming Languages

Yekun Chai Shuohuan Wang Chao Pang Yu Sun Hao Tian Hua Wu
Baidu
{chaiyekun,wangshuohuan,sunyu02}@baidu.com

Abstract

Software engineers working with the same programming language (PL) may speak different natural languages (NLs) and vice versa, erecting huge barriers to communication and working efficiency. Recent studies have demonstrated the effectiveness of generative pre-training in computer programs, yet they are always English-centric. In this work, we step towards bridging the gap between multilingual NLs and multilingual PLs for large language models (LLMs). We release ERNIE-Code, a unified pre-trained language model for 116 NLs and 6 PLs. We employ two methods for universal cross-lingual pre-training: span-corruption language modeling that learns patterns from monolingual NL or PL; and pivot-based translation language modeling that relies on parallel data of many NLs and PLs. Extensive results show that ERNIE-Code outperforms previous multilingual LLMs for PL or NL across a wide range of end tasks of code intelligence, including multilingual code-to-text, text-to-code, code-to-code, and text-to-text generation. We further show its advantage of zero-shot prompting on multilingual code summarization and text-to-text translation. We release our code and pre-trained checkpoints[1].

1 Introduction

Recent trends in generative pre-training of programming languages (Feng et al., 2020; Chen et al., 2021; Li et al., 2022) have led to a proliferation of improvements in code intelligence scenarios, including program understanding and generation (Wang et al., 2021; Ahmad et al., 2021). In this context, a transformer-based large language model (LLM) is pre-trained on a large corpus of open source code (*e.g.*, from GitHub) and then finetuned or zero-shotly evaluated on downstream tasks, such as program synthesis (Austin et al., 2021; Fried et al., 2022; Nijkamp et al., 2022), code search (Husain et al., 2019; Li et al., 2021), clone detection (Lu et al., 2021b), and text-to-code generation (Clement et al., 2020).

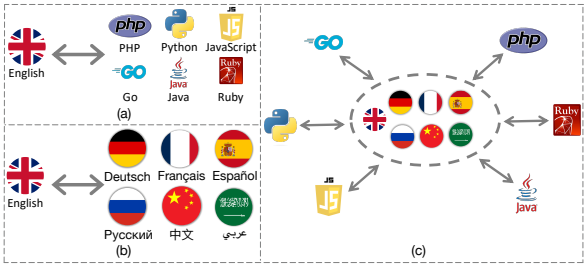


Figure 1: Comparison among (a) Multilingual code pre-training; (b) Multilingual text pre-training; (c) Universal multilingual text-code pre-training (ours).

Although there has been a surge of interest in learning general-purpose multilingual LLMs for source code (Feng et al., 2020; Ahmad et al., 2021; Wang et al., 2021; Fried et al., 2022; Xu et al., 2022), research in this area has been essentially connecting English texts (*e.g.*, comments or docstring) and multiple computer programs (*e.g.*, Python, C++, and Java), as shown in Figure 1(a), and primarily focused around English-centric corpora and benchmarks. This *English-centricity* issue dramatically limits their use and practice given that 95% of the world population does *not* have English as their native language (Guo, 2018).

As such, it is crucial to mitigate barriers and draw connections between non-English natural languages (NLs) and multiple programming languages (PLs). One engineering solution is to use English translation of non-English texts by engaging neural machine translation (NMT) systems before/after the code LLM as a pipeline. Unfortunately, most general-purpose NMT systems (Wu et al., 2016; Johnson et al., 2017) are not designed for code-specific texts and can be prone to accumulative errors due to cascaded prediction stages.

A more general way is to learn a multilingual

[1] https://github.com/PaddlePaddle/PaddleNLP/tree/develop/model_zoo/ernie-code

Findings of the Association for Computational Linguistics: ACL 2023, pages 10628–10650

LLM that encodes a mixture of multiple NLs and PLs into a shared cross-mode representation space. The success in learning universal representations of many languages (Conneau and Lample, 2019; Xue et al., 2021; Ahmad et al., 2021; Wang et al., 2021; Xu et al., 2022) that focuses on PLs or NLs suggests that it is possible to build a universal multilingual model that jointly represent multiple PLs and NLs.

In this work, we present ERNIE-Code, a unified cross-lingual pre-trained LLM for multiple NLs and PLs in hopes of mitigating the *English-centric* bias for program pre-training, as illustrated in Figure 1. Our model builds on the T5 (Raffel et al., 2020) encoder-decoder architecture that has been demonstrated to be effective in understanding and generation tasks for multilingual NL (Xue et al., 2021) and PL (Wang et al., 2021). For monolingual pre-training on mono-mode data (*i.e.*, unpaired multilingual code or text), we follow the same T5 recipe to employ the "span-corruption" denoising objective in the text-to-text format.

The good-quality parallel corpus between low-resource NLs and multilingual PLs is usually unavailable. Instead, most popular PLs, accompanying API documentation, code examples, and discussion forums are primarily written in English, which poses a bottleneck to drawing connections between low-resource NLs and PLs. Inspired by the pivot-based machine translation (Gispert and Mariño, 2006; Utiyama and Isahara, 2007) that uses a *pivot* language and decomposes the source↔target translation into source↔pivot and pivot↔target bilingual translation, we introduce the pivot-based translation language modeling (PTLM) with prompting that disassembles multi-NL↔multi-PL into multi-NL↔English and English↔multi-PL with pivoting through English.

Specifically, we leverage the PTLM training in dual direction for parallel corpus in different modes: (1) English↔multi-PL. For multi-PL↔English parallel data, *i.e.*, code snippets and their accompanying comments, the model learns to generate English comments from code fragments and vice versa. (2) English↔Multi-NL. It learns to translate between English and other NLs. The model thus encodes PL↔English and English↔NL at the same time, with English as a *pivot* language. We conduct extensive experiments on different downstream tasks: (1) Multilingual text-to-code generation; (2) Multilingual code summarization (code-to-text); (3) Documentation translation (text-to-text); (4) Code repair (code-to-code). Empirical results have shown that our model outperforms strong multilingual LLMs for PL or NL and have verified its universal multilingual capacity. We also provide examples to show its decent zero-shot capability on code summarization and text translation via zero-shot prompting.

To summarize, this paper makes the following contributions: (1) We first propose a unified cross-lingual pre-trained LLM for both multilingual NLs and multilingual PLs, enlarging the capacity of LLMs towards jointly learning the universal multilingualism. (2) We employ the pivot-based translation language modeling with prompting to build connections between multi-NLs and multi-PLs (with English pivots) and mitigate the problem when the parallel corpus of multilingual-NL↔multilingual-PL is unavailable. (3) We obtain superior performance compared with previous multilingual LLMs across a wide range of code intelligence tasks, including text-to-code, code-to-text, code repair, and code documentation translation. (4) To some extent, our model has shown zero-shot prompting ability on multilingual code-to-text, text-to-code, and text-to-text generation. Moreover, ERNIE-Code is well-performed at naming a function and completing corresponding arguments given multilingual NL instructions.

2 Related work

As text-based formal languages with strict syntax and semantics, PL differs from NL because NL is only used for human communication while PL requires the interaction between humans and computers. This work targets bridging the gap between human languages and computer programs in a cross-lingual manner for unified multilingual pre-training, which is closely related to LLMs in either multilingual PL or NL.

Multilingual PL pre-training The success of large-scale pre-training has led to impressive advances in computer programs. This line of research involves pre-training on multilingual PLs using bidirectional transformer encoders (Feng et al., 2020; Li et al., 2021), casual transformer decoders (Chen et al., 2021; Austin et al., 2021; Fried et al., 2022; Nijkamp et al., 2022; Xu et al., 2022), and transformer encoder-decoder architectures (Wang et al., 2021; Ahmad et al., 2021; Li et al., 2022). Those with bidirectional encoder focus on program understanding tasks, such as code search (Husain et al., 2019), while the encoder-

decoder ones target at building unified LLMs for both program understanding and generation. We observe that a large body of pre-trained models for PL tend to scale up their parameters under the framework of causal language modeling, mainly focusing on program synthesis (Chen et al., 2021; Austin et al., 2021; Fried et al., 2022; Nijkamp et al., 2022; Xu et al., 2022). Nevertheless, all of these works are almost *English-centric*, posing significant challenges to coping with PL end-tasks in non-English scenarios.

Multilingual NL pre-training This work is also related to the continual trend of multilingual LLMs. One line of this work focuses on encoding multiple NLs into a shared representation space (Conneau and Lample, 2019; Conneau et al., 2020), while some make efforts to extend the efficient monolingual pre-training method into multilingual settings (Xue et al., 2021; Liu et al., 2020).

Inheriting the recent success of LLMs in multilingualism, this work lies in the intersection between multilingual NL and PL pre-training. In contrast to the previous work that attends to either multilingual NL or multilingual PL, we seek to explicitly learn multiple NLs and PLs in a shared representation space in hopes of breaking the language barriers between these two modes.

3 Cross-lingual NL-PL pre-training

In this section, we introduce pre-training tasks (§3.1), model (§3.2), and pre-training data (§3.3) we use throughout this work.

3.1 Pre-training tasks

We pre-train on two pre-training tasks using both PL and NL data: one (§3.1.1) uses monolingual PL/NL data (unsupervised), while the other (§3.1.2) requires parallel NL-PL and NL-NL pairs (supervised). The former advances to learn intra-modal patterns from PL or NL only, while the latter endows the model with cross-lingual/modal alignment and zero-shot capabilities.

3.1.1 Task#1: Span-corruption language modeling (SCLM)

Denoising sequence-to-sequence pre-training has been highly effective across a broad set of tasks, including natural language processing (Liu et al., 2020; Raffel et al., 2020; Xue et al., 2021) and programming language processing (Wang et al., 2021; Ahmad et al., 2021). The denoising pre-training objective first corrupts input sequences by masking or adding noise; and then recovers the original inputs by forcing the model to predict corrupted spans, sentences, or documents. Raffel et al. (2020) finds that span-corruption denoising pre-training produces strong performance while being more computationally efficient on account of shorter target sequence lengths.

In similar vein, we extend the span-corruption denoising pre-training on both PL and NL. We refer to this task as span-corruption language modeling (SCLM), as illustrated in Figure 2. Specifically, it corrupts 15% of the original NL/PL input tokens with a mean span length of 3 by replacing contiguous, randomly-spaced spans of tokens as a single mask placeholder and then predicting the corrupted span on the target side.

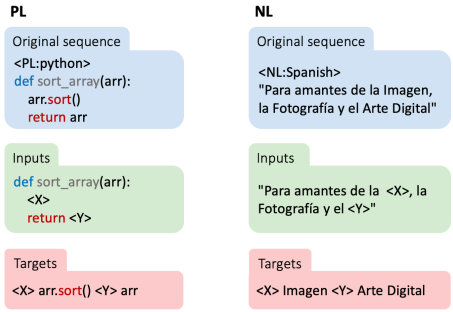


Figure 2: Schematic of the SCLM objective for PL (left) and NL (right) example.

Suppose we have a total of M monolingual corpora of NL and PL corpora $\{C_m\}_{m=1\cdots M}$. We apply the SCLM pre-training objective on both NL and PL data in a multi-tasking fashion:

$$\mathcal{L}_{\text{SCLM}} = \sum_{m=1}^{M} \sum_{t=1}^{T} -\log P_\theta\left(x_{(i),t} | \mathbf{x}_{(m)}^{\backslash \text{mask}}, \mathbf{x}_{(m),<t}^{\text{mask}}\right) \tag{1}$$

where θ denotes trainable parameters, $\mathbf{x}_{(m)}^{\backslash \text{mask}}$ and $\mathbf{x}_{(m)}^{\text{mask}}$ are span-corrupted inputs and corresponding target spans from monolingual corpus C_m, respectively. $\mathbf{x}_{(m),<t}^{\text{mask}}$ indicates the generated tokens until the t-th time step out of the target (corrupted) sequence length T.

3.1.2 Task#2: Pivot-based translation language modeling (PTLM)

This work aims at narrowing the cross-modal cross-lingual gap between multiple NLs and PLs, yet good quality parallel corpora between non-English

NL and multilingual PL are unavailable. The lack of parallel corpus stems from the fact that most popular PLs, accompanying documentations, and discussion websites are primarily written in English. Early investigation of statistical machine translation proposed pivot-based approach (Gispert and Mariño, 2006; Utiyama and Isahara, 2007) to introducing a third language - named *pivot* language - for which there exist good-quality source-pivot and pivot-target bilingual corpora. Johnson et al. (2017) adopt a single NMT model to simultaneously learn many translation directions (including source↔pivot, pivot↔target), enabling the zero-shot translation between NLs implicitly.

In our context, the good-quality multi-PL to the multi-NL bilingual corpus is unavailable, yet there exists multi-NL to English and English to multi-PL parallel corpora, with pivoting through English. Motivated by the pivot-based NMT (Johnson et al., 2017) and translation language modeling (TLM; Conneau and Lample, 2019) approach, we apply a unified pivot-based training objective to the course of multilingual NL-PL pre-training, namely pivot translation language modeling (PTLM).

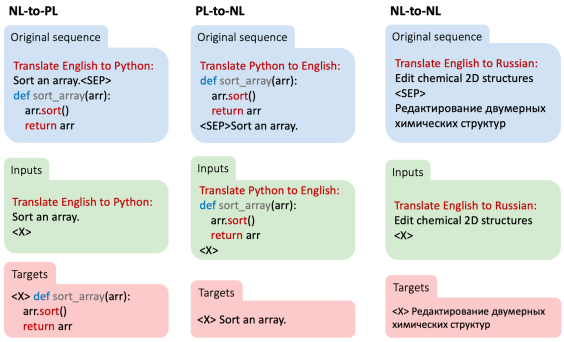


Figure 3: Schematic of the PTLM objective for NL-to-PL (left), PL-to-NL (middle), NL-to-NL (right) example. "<SEP>" indicates the delimiter token.

With bilingual PL-NL and NL-NL corpora, we jointly learn the parallelism with pivoting in dual directions: for instance, Python↔English and English↔Russian. This allows for implicit bridging between PL-NL pairs that are never seen explicitly in training data (Johnson et al., 2017). More precisely, we concatenate parallel source-target sentences and learn to predict the corrupted target language, as shown in Figure 3. Instead of masking random tokens (Conneau and Lample, 2019), we corrupt the *whole* sentence in either direction of bilingual data and predict on the target side. The model requires attending to complete representations of source sentences to recover the target sentence and learn the alignment between source-target pairs. Suppose we have N bilingual NL-NL and NL-PL parallel corpora $\{D_n\}_{n=1,\cdots,N}$. We can formulate the PTLM training as:

$$\mathcal{L}_{\text{PTLM}} = \sum_{n=1}^{N} \sum_{t=1}^{T} -\log P_\theta\left(x_{(n),t} | \mathbf{x}_{(n)}^{\text{source}}, \mathbf{x}_{(n),<t}^{\text{target}}\right) \tag{2}$$

where $\mathbf{x}_{(n)}^{\text{source}}$ and $\mathbf{x}_{(n)}^{\text{target}}$ denote source and target sentences from bilingual corpus D_n. $\mathbf{x}_{(n),<t}^{\text{target}}$ indicates the generated tokens until the t-th time step out of the target sequence length T. This training format is the same as an NMT task.

To enable a pivot-based approach and specify the target language, we reformat the PTLM by prompting with a task prefix (See Figure 3), in which we prepend a task instruction "translate `A` to `B`: `\n`" on the left of input sentences, where `A` and `B` denote the source and target language, respectively. This prompt instruction indicates the target language the model should translate to, resulting in descent zero-shot abilities (§5.3).

3.2 Model

Model architecture Our model follows the same architecture as T5-base (Raffel et al., 2020). Specifically, we build ERNIE-Code on "T5.1.1" version[2], which improves upon T5 using gated nonlinearities (Shazeer, 2020; Chai et al., 2020). We refer to §A.3.1 for pre-training settings.

Shared NL/PL encoding We base our tokenizer on SentencePiece tokenizer in Xue et al. (2021). However, the original SentencePiece tokenizer designed for encoding NLs does not effectively represent PL data. We thus add a set of tokens representing whitespace indentation of different lengths in PL. See tokenization details in §A.1.

3.3 Pre-training data

Code corpus For PL data, we use the same pre-training corpora - CodeSearchNet (Husain et al., 2019) - as previous models (Feng et al., 2020; Wang et al., 2021).[3] It covers six monolingual PLs (Go, Java, JavaScript, PHP, Python, and Ruby) and six NL-PL parallel data, *i.e.*, PL-NL query pairs. The

[2] https://github.com/google-research/text-to-text-transfer-transformer/blob/main/released_checkpoints.md#t511

[3] Note that for a fair comparison, we do not use additional data from public repositories hosted on GitHub.

Model	#Param	#PLs	#NLs	Data source
mBART (Liu et al., 2020)	680M	-	25	Common Crawl (CC25)
mT5 (Xue et al., 2021)	560M	-	101	Common Crawl (mC4)
PLBART (Ahmad et al., 2021)	390M	2	1	GitHub, StackOverflow
CodeT5 (Wang et al., 2021)	220M	8	1	CodeSearchNet, GitHub (C/C#)
ERNIE-Code (ours)	560M	6	116	CodeSearchNet, CC-100, OPUS

Table 1: Comparison of our model to existing massively multilingual pre-trained models for NLs and PLs.

majority of NL annotations in the parallel corpora is English. We defer data statistics and preprocessing details in §A.2.1.

Text corpus We pre-train on the following NL data corpus: (1) Monolingual data from CC-100 (Conneau et al., 2020) that built on a clean CommonCrawl corpus[4], containing 116 different NLs. [5] (2) Parallel data from OPUS website[6] covering 15 languages. The collected NL translation pairs include MultiUN (Ziemski et al., 2016), IIT Bombay (Kunchukuttan et al., 2018), OPUS (Tiedemann, 2012), WikiMatrix (Schwenk et al., 2021), *etc*. We refer to §A.2.2 for details.

To alleviate the bias towards high-resource languages, we follow Conneau and Lample (2019) to rebalance the data distribution on both corpora and up/down-sample sentences from each language (or language pair) i with a rescaled multinomial distribution q_i:

$$q_i = \frac{p_i^\alpha}{\sum_{j=1} p_j^\alpha} \tag{3}$$

where p_i is the data percentage of each monolingual or parallel corpus. Following Conneau and Lample (2019), we set $\alpha = 0.3$ for both monolingual and parallel corpus.

4 Experiments

In this section, we first introduce multilingual pre-trained models for comparison (§4.1), downstream tasks, and evaluation metrics (§4.2). Then we evaluate and show consistent performance gains on several multilingual NL/PL benchmarks, including code-to-text (§4.3), text-to-code (§4.4), text-to-text (§4.5), and code-to-code (§4.6) end tasks.

4.1 Comparison to related models

To contextualize our new model, we briefly compare it with existing multilingual LLMs for NLs/PLs. Considering that ERNIE-Code is the first LLM targeting multilingual NL and PL explicitly, for brevity, we focus on models that support either many NLs or many PLs. Table 1 reports the overall statistics of comparison models.

mBART (Liu et al., 2020) is a multilingual-NL variant of BART (Lewis et al., 2020) trained with a full-text denoising objective on a subset of 25 languages from CommonCrawl. It learns to reconstruct the full NL texts from corrupted ones with an arbitrary noising function. **mT5** (Xue et al., 2021) is a multilingual-NL encoder-decoder model adapted from T5. It is trained on 101 NLs using filtered CommonCrawl data (mC4) using the same SCLM objective as our model. **PLBART** (Ahmad et al., 2021) is a multilingual-PL version of BART with a denoising objective using three noising formats. It is trained on 210M Java functions, 470M Python functions from GitHub, and 47M English posts from StackOverflow. **CodeT5** (Wang et al., 2021) is a PL version of mT5 that is pre-trained on six-PL monolingual/parallel data from CodeSearchNet and extra C/C# data collected from GitHub. It additionally learns token-type information from identifiers and applies dual generation between English and PLs.

4.2 Evaluation datasets and metrics

Table 9 displays the statistics of evaluation dataset. We use the same public datasets and train-test splits for all downstream tasks. We refer to §A.3.3 for experimental settings of finetuning.

Multilingual code summarization is a code-to-text task that aims to generate multilingual texts given a code snippet. We use mCoNaLa (Wang et al., 2022) to evaluate the performance of generating multilingual NL from PL. It consists of 341/210/345 manually curated parallel samples with NL in Spanish/Japanese/Russian and PL in Python. As mCoNaLa does not provide the training and validation set, we use CoNaLa (Yin et al., 2018), an English-Python parallel data (consisting of #2,379 samples), as the train/dev set (with 10:1 data split) after translation. For "*translate-train*" settings, we use machine-translated CoNaLa as training and dev sets, while use mCoNaLa as the test set. Particularly, we translate CoNaLa's training set into three target languages using FLORES-101 (Goyal et al., 2022) and adopt them as train/dev set. We utilize ROUGE-L (R-L; Lin, 2004), BLEU-4 (B-4; Post, 2018), and chrF (Popović, 2015) for comprehensive comparison.

[4] https://data.statmt.org/cc-100

[5] Note that following Conneau et al. (2020), we count Romanized variants as separate languages.

[6] https://opus.nlpl.eu

Model	Spanish			Japanese			Russian			Avg.		
	B-4	R-L	chrF	B-4	R-L	chrF	B-4	R-L	chrF	B-4	R-L	chrF
Translate-train												
mBART	0.96	19.46	19.30	0.07	4.70	7.88	0.08	0.00	13.56	0.37	8.05	13.58
mT5	0.94	28.69	19.87	0.06	2.95	6.58	0.09	2.56	12.00	0.36	11.40	12.82
PLBART	0.16	14.33	11.72	0.06	4.11	7.87	0.24	2.98	14.06	0.15	7.14	11.22
CodeT5	1.00	22.93	20.09	0.04	5.42	7.13	0.13	1.48	12.97	0.39	9.94	13.40
Ours(L512)	1.90	32.51	23.22	0.30	**10.62**	**9.16**	**0.43**	5.01	**16.60**	0.88	**16.05**	**16.33**
Ours(L1024)	**2.51**	**33.87**	**24.00**	**0.58**	8.55	8.81	0.28	**5.69**	15.24	**1.12**	16.04	16.02
Zero-shot												
Ours(L512)	0.49	12.78	15.69	**1.46**	**32.07**	**11.02**	**1.98**	**30.46**	11.68	**1.31**	**25.10**	12.80

Table 2: Results of multilingual code summarization task. "L512/1024" indicates the maximum length of 512/1024.

Multilingual text-to-code generation refers to the code generation task that generates code fragments from multilingual NL instructions. We use the same train/dev/test set as the code summarization mentioned above. Specifically, under "*translate-train*" settings, we use translated CoNaLa data as training and dev set, mCoNaLa as the test set to generate Python code from NL instruction in three different NLs (*i.e.*, Spanish, Japanese, and Russian). We use ROUGE-L, BLEU-4, and CodeBLEU (C-B; Ren et al., 2020) for evaluating code predictions.

Documentation translation is a text-to-text task that translates code documentation from one NL to another. We use Microsoft Docs from CodeXGLUE dataset (Lu et al., 2021a) to verify the multilingual NL translation between English ↔ Danish, Latvian, Norwegian, and Chinese. We report BLEU-4 and exact match (EM) in our results.

Code repair is a code-to-code task that automatically fixes bugs given a piece of buggy code. We evaluate on Bugs2Fix (Tufano et al., 2019) dataset with two subsets: (i) "small" with tokens less than 50; (ii) "medium" with a length of between 50 and 100. We report BLEU-4[7] and EM for evaluation.

4.3 Multilingual code summarization

Table 2 shows the multilingual code-to-text results of generated NL summaries in Spanish, Japanese, and Russian. We use translated English CoNaLa as training sets in target three languages[8], denoted as "translate-train" evaluation. As shown in Table 2, our model outperforms all baseline LLMs for either NL (mBART, mT5) or PL (PLBART, CodeT5). In particular, ERNIE-Code, with a length of 1024, exceeds its counterpart of 512-length (1.12 vs. 0.88 on BLEU-4) in that it allows for learning more extended contexts from training NL/PL segments. PLBART performs worst among all baselines on average, while CodeT5, mT5, and mBART behave similarly. We conjecture that PLBART only learns data from Java/Python functions and English StackOverflow posts, whose training data lacks the diversity of multilingualism.

4.4 Multilingual text-to-code generation

Table 3 shows the "translate-train" results of multilingual text-to-code generation on mCoNaLa. ERNIE-Code outperforms all baselines on BLEU-4, ROUGE-L, and CodeBLEU scores, showing that our multilingual PL-NL pre-training can capture code syntax and semantics. Among all code generation tasks, multilingual models for NL behave worse than those counterparts of PL. PLBART beats all baselines on surface-form n-gram match (BLEU-4/ROUGE-L) and structured code-related match (CodeBLEU), even achieving on par with our model on CodeBLEU. In contrast, mT5 underperforms all the other models on either of three subtasks, suggesting that the mT5 tokenizer is ineffective in encoding PLs, as aforementioned in §3.2. By comparing mT5 and our models, the improvements suggest our approach's effectiveness in encoding whitespace characters for tokenization. Our model with more extended contexts (1024-length) overshadows that of 512-length on all three text-to-code subtasks.

4.5 Documentation translation (text-to-text)

We further investigate the multilingual text-to-text translation between English (en) and Danish (da)/Latvian (lv)/Norwegian(no)/Chinese(zh). Table 4 shows the documentation translation results of comparison models, including multilingual transformer (Johnson et al., 2017), XLM-R (Conneau

[7] https://github.com/microsoft/CodeXGLUE/blob/main/Code-Code/code-refinement/evaluator/evaluator.py

[8] https://conala-corpus.github.io/

Model	Spanish			Japanese			Russian			Avg.		
	B-4	R-L	C-B	B-4	R-L	C-B	B-4	R-L	C-B	B-4	R-L	C-B
Translate-train												
mBART	1.73	11.85	0.05	3.68	10.33	0.08	2.34	9.23	0.07	2.58	10.47	0.07
mT5	0.27	3.51	0.05	0.22	2.91	0.07	0.25	6.17	0.04	0.25	4.20	0.05
PLBART	2.19	14.47	**0.06**	6.56	18.26	0.09	3.27	19.92	**0.09**	4.01	17.55	**0.08**
CodeT5	1.97	14.47	0.05	7.46	18.58	0.09	4.26	17.96	0.07	4.56	17.00	0.07
Ours(L512)	2.25	**14.92**	**0.06**	**8.06**	**22.65**	**0.10**	6.12	**25.27**	0.08	5.48	**20.95**	**0.08**
Ours(L1024)	**2.51**	12.65	**0.06**	**8.08**	20.12	0.09	**6.55**	23.84	**0.09**	**5.71**	18.87	**0.08**
Zero-shot												
Ours(L512)	2.47	12.12	**0.10**	2.56	14.46	**0.15**	3.69	13.52	**0.14**	2.91	13.37	**0.13**

Table 3: Results of on multilingual text-to-code generation task.

Model	En-Da				En-Lv				En-No				En-Zh				Avg. B-4	Avg. EM
	→		←		→		←		→		←		→		←			
	B-4	EM	B-4	EM	B-4	EM	B-4	EM	B-4	EM	B-4	EM	B-4	EM	B-4	EM		
Transformer	53.31	-	58.73	-	37.85	-	50.37	-	53.84	-	57.73	-	59.90	-	50.00	-	52.67	-
XLM-R	67.09	-	67.02	-	51.92	-	68.30	-	68.00	-	71.84	-	70.60	-	64.47	-	66.16	-
mT5	67.39	10.6	68.72	24.1	57.69	8.5	64.95	22.2	68.40	12.3	68.02	23.3	72.26	20.0	68.64	24.7	67.01	18.21
Ours(L512)	**71.16**	13.2	**72.70**	27.2	60.98	**10.6**	69.28	24.3	**71.39**	**15.7**	72.28	26.3	**74.53**	24.3	72.43	**28.5**	70.59	21.26
Ours(L1024)	70.90	**13.6**	72.55	**27.3**	**61.30**	**10.6**	**69.85**	**25.1**	71.11	**15.7**	**72.49**	**26.7**	74.49	**24.7**	**72.49**	28.3	**70.65**	**21.50**

Table 4: Results of documentation translation. We report BLEU-4 (B-4) and exact match (EM) scores.

Model	Refine small		Refine medium	
	B-4	EM	B-4	EM
Naive copy	78.06	0	90.91	0
RoBERTa (code)	77.30	15.90	90.07	4.10
CodeBERT	77.42	16.40	91.07	5.20
PLBART	77.02	19.21	88.50	8.98
CodeT5	78.06	**22.59**	88.90	**14.18**
Ours (L512)	80.09	13.21	**91.20**	2.22
Ours (L1024)	**80.10**	12.43	91.17	2.00

Table 5: Results of program repair task.

et al., 2020), and mT5. Specifically, we finetune our model in a multilingual manner where all bilingual language pairs are learned simultaneously.

Our model surpasses mT5 and XLM-R in all eight translation directions, demonstrating that our model can perform code-related text-to-text translation. As the experiment design only aims to verify the NL translation ability of our model, we did not conduct comprehensive results to compare with state-of-art (SOTA) NMT methods.

4.6 Program repair (code-to-code)

We further validate that our model can perform code-to-code generation. Table 5 demonstrates the comparison model results on the Bugs2Fix benchmark. Baseline models include RoBERTa (code) - a PL variant of RoBERTa (Liu et al., 2019), CodeBERT (Feng et al., 2020), PLBART, and CodeT5.

On "small" and "medium" tasks, our model achieves 80.10 and 91.20 BLEU scores, outperforming or achieving competitive results compared with previous SOTA performance.[9] The results of 1024-length and 512-length models slightly differ, possibly because both "small" and "medium" Java data are of no more than 100-token length, far shorter than our model's length limit.

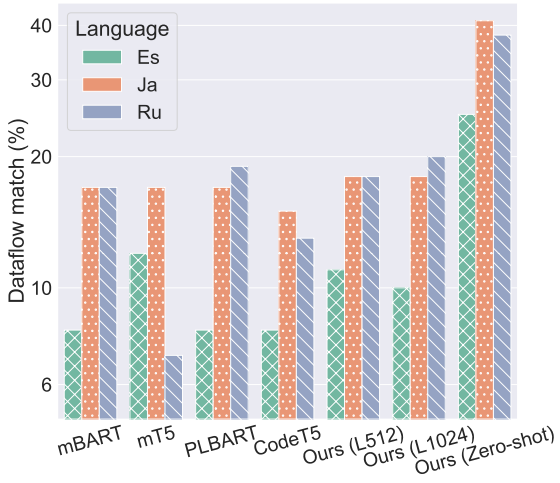
5 Analysis

5.1 Syntactic & semantic probing

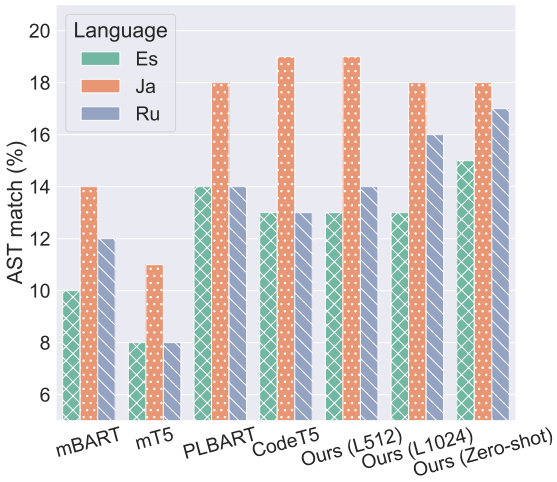
Code fragments with highly-overlapping surface forms but with different semantic and syntactic logic can be given high scores by NL evaluation metrics, such as BLEU and ROUGE. To evaluate the semantic and syntactic aspects of text-to-code generation, we follow Ren et al. (2020) to adopt dataflow and abstract syntax tree (AST) match to compute the accuracy of dataflow graph and AST subtrees between hypothesis and reference. We refer to Ren et al. (2020) for further details.

Figure 4 illustrates the dataflow and AST match results of comparison models. PL baselines tend to generate code with better AST structures than NL models. In particular, mT5 fails to produce code with proper AST syntax but can match or surpass others on dataflow evaluation except on Russian tasks. Our model (L512/1024) exceeds or matches

[9]Note that EM only serves as a reference indicator in that it is too strict and inaccurate for evaluation, especially for PL hypotheses with the same semantic logic but in various surface forms.



(a) Semantic dataflow match (w/ log-scaled y-axis).



(b) Syntactic AST match.

Figure 4: Semantic and syntactic comparison on multilingual text-to-code generation. All comparison models are evaluated under "translate-train" settings by default, unless otherwise specified (*i.e.*, "zero-shot").

baselines in terms of both the semantic dataflow and syntactic AST match.

5.2 Ablation study

Quantitative results We carry out ablation experiments by ablating either SCLM or PTLM tasks and report the average results in Figure 5. It is shown that removing either monolingual (\SCLM) or bilingual (\PTLM) pre-training task could deteriorate overall performance of all tasks. Specifically, ablating PTLM would vastly reduce the performance of PL-to-NL and NL-to-PL tasks compared to removing SCLM, showing that pivot-based bitext pre-training is crucial to implicit bridging between bilingual NL-to-PL or PL-to-NL pairs that never seen explicitly in training data. Meanwhile, PTLM contributes slightly more than SCLM in NL-to-NL translation. We suspect that although PLTM can provide explicit training on bilingual data, SCLM could implicitly learn NL patterns from amounts of monolingual training corpora. In contrast, SCLM makes a trivial contribution to PL-to-PL generation, indicating that PTLM allows the model to focus on full-sequence generation instead of partial span reconstruction. Considering that the training data size of the PL corpus is quite limited, we suspect that pre-training on more open-source repositories from GitHub would bring more significant performance gain. We refer to §A.4 for detailed results on each subtask.

Analyzing PL semantics & syntax We further analyze the semantic and syntactic structure of multilingual text-to-code generation for ablation comparison. Figure 7 shows dataflow and AST match

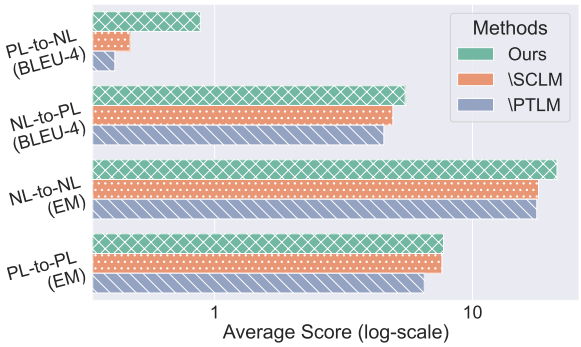


Figure 5: Ablation test performance (log-scale). The reported results are averaged among all subtasks.

performance on text-to-code generation given multilingual NL inputs. We find that removing SCLM does not overly impact the semantic dataflow and syntactic structures of generated PL. At the same time, ablating PTLM would generally cause more considerable fluctuation in the semantics and syntax of generated PL, suggesting that PTLM could allow the model to capture bilingual alignment and translation across multilingualism.

5.3 Zero-shot prompting

To verify the zero-shot ability of ERNIE-Code, we carry out code-to-text, text-to-code, and text-to-text experiments with zero-shot prompting. Precisely, we prepend a prompt prefix "translate `S` to `T`: `\n`" on the left of inputs, where `S` and `T` denote the source and target language respectively. Then we use beam search with five beams to obtain zero-shot predictions.

Quantitative analysis Table 2 (last row) shows the performance of *zero-shot* code-to-text genera-

```
def contar_vocales(s):
    return sum(map(s.lower().count, 'aeiouáéíóúü'))
```

Hypothesis	Compute la cifra de vocales de una palabra.
Translation	Compute the number of vowels in a word.
Reference (es)	¿Cómo contar las vocales en una cadena `s`?
Translation	How to count the vowels in a string `s`?

(a) Spanish code summarization

```
model = word2vec.Word2Vec.load('old_model')
model.train(sentences)
```

Hypothesis	新しいモデルを学習する
Translation	Learn a new model
Reference (ja)	既存のモデル'old_model'を読み込みデータ`sentences`を用いて学習させて新しいモデルを作る
Translation	Create a new model by reading an existing model 'old_model' and training it using the data `sentences`

(b) Japanese code summarization

```
def auth_vk_password():
    session = vk.AuthSession(app_id=APP_ID,
        user_login=LOGIN, user_password=PASS)
    file = open("auth_vk.ini", 'w')
    file.writelines(session.access_token)
    return session

try:
    file = open("auth_vk.ini", 'r')
except IOError as e:
    access_token = auth_vk_password().access_token
else:
    access_token = file.readline()

session = vk.Session(access_token=access_token)
if session:
    print('Авторизация прошла успешно!')
else:
    print('Авторизация не удалась!')
```

Hypothesis	Создать и загрузить authentication session.
Translation	Create and load an authentication session.
Reference (ru)	Авторизоваться в VK приложении с id `APP_ID` с логином `LOGIN` и паролем `PASS`
Translation	Authorize in VK application with id `APP_ID` with login `LOGIN` and password `PASS`

(c) Russian code summarization

Figure 6: Examples of zero-shot multilingual code summarization (code-to-text).

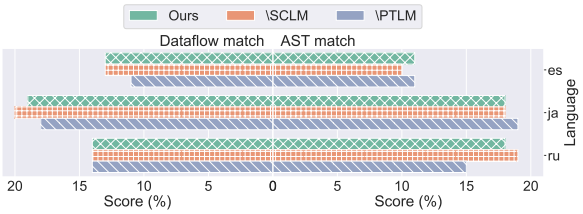


Figure 7: Ablation results on dataflow and AST match.

tion. Our model demonstrates excellent zero-shot capability on Japanese and Russian summary generation, even outperforming "translate-train" settings by 0.43 / 9.05 on BLEU / ROUGE-L in general. This is because the training data is automatically translated rather than human annotated (*i.e.*, "translate-train" settings), lowering the quality of training data. Table 3 shows that our model can zero-shotly produce code fragments with higher CodeBLEU scores than "translate-train" settings. This indicates that our cross-lingual NL-PL pre-training renders excellent transfer learning capability in bridging multilingual NLs and PLs.

Zero-shot PL-to-NL generation Figure 6 exhibits *zero-shot* multilingual code summarization examples in three target languages. Our model can attend to the whole picture of code semantics while ignoring blunt descriptions of detailed implementation, demonstrating the effectiveness of our approach on zero-shot prompting. To extend the evaluation to other NL, we further release a Python-Chinese test set by translating mCoNaLa into its Chinese variant via crowd-sourcing. Our model shows decent ability on zero-shot PL-to-Chinese generation. We give zero-shot demonstrations and provide data curation details in §A.5. We argue that our model captures many NL genres via cross-lingual pre-training. We encourage the community to release more multilingual code-to-text benchmarks for further evaluation.

Qualitative examples (zero-shot) We show a variety of qualitative examples with zero-shot prompting in §A.6: multilingual code summarization, NL-to-PL generation, zero-shot NL translation of technical jargon in eight randomly selected directions.

6 Conclusion

This work makes the first step towards explicitly connecting computer programs to human languages in a universal multilingual fashion. By virtue of cross-lingual pre-training on 116 NLs and 6 PLs, our model exhibits strong performance in various tasks across computer programs and natural languages, including PL-to-NL, NL-to-PL, NL-to-NL, and PL-to-PL. Our model shows descent zero-shot performance via prompting on PL summarization and NL translation. Finally, we provide discussions about limitations and future work for improvement.

Acknowledgements

We would like to thank Xuhong Li and Qiwei Peng for their helpful feedback on the initial manuscript of this work.

Limitations

Releasing multilingual NL-PL benchmark While our model has been shown to capture multilingual languages between humans and computer programs, we could not systemically evaluate its performance on a wide range of multilingual NLs due to the lack of corresponding benchmarks. Instead, we undertake NL-to-PL and PL-to-NL experiments on mCoNaLa that involves only three NLs and present demonstration examples via zero-shot prompting to reveal its cross-lingual capacity. We encourage researchers in the community to release more multilingual NL-PL benchmarks to accelerate the development of this intersecting area.

Scaling up the model size and data In this work, we only use the PL data from CodeSearchNet for a fair comparison to baselines, preventing the model from learning from more PL genres and billions of open-source repositories. Increasing the amount of data for bilingual NL-PL pairs is also a promising direction, such as using data augmentation. Moreover, the scaling law for large pre-training has been well studied and shown significant performance gains in the literature (Chen et al., 2021; Li et al., 2022). A targeted effort at expanding the pre-training data size and scaling up models could give rise to more considerable improvement toward universal multilingual NL-PL pre-training.

Curse of multilinguality We argue that the *curse of multilinguality* (Conneau et al., 2020) also exists in unified multilingual NL-PL pre-training, in which per-language capacity decreases as the number of languages increases given a fixed model size. It is an interesting direction to investigate the issue of *curse of multilinguality* upon this work.

References

Wasi Uddin Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. 2021. Unified pre-training for program understanding and generation. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*, pages 2655–2668. Association for Computational Linguistics.

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

Yekun Chai, Shuo Jin, and Xinwen Hou. 2020. Highway transformer: Self-gating enhanced self-attentive networks. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6887–6900, Online. Association for Computational Linguistics.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde, Jared Kaplan, Harrison Edwards, Yura Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, David W. Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William H. Guss, Alex Nichol, Igor Babuschkin, S. Arun Balaji, Shantanu Jain, Andrew Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew M. Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating large language models trained on code. *ArXiv*, abs/2107.03374.

Colin B. Clement, Dawn Drain, Jonathan Timcheck, Alexey Svyatkovskiy, and Neel Sundaresan. 2020. Pymt5: multi-mode translation of natural language and python code with transformers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, pages 9052–9065. Association for Computational Linguistics.

Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. 2020. Unsupervised cross-lingual representation learning at scale. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 8440–8451. Association for Computational Linguistics.

Alexis Conneau and Guillaume Lample. 2019. Cross-lingual language model pretraining. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 7057–7067.

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, Online Event, 16-20 November 2020*, volume EMNLP 2020 of *Findings of ACL*, pages 1536–1547. Association for Computational Linguistics.

Daniel Fried, Armen Aghajanyan, Jessy Lin, Sida I. Wang, Eric Wallace, Freda Shi, Ruiqi Zhong, Wen

tau Yih, Luke Zettlemoyer, and Mike Lewis. 2022. Incoder: A generative model for code infilling and synthesis. *ArXiv*, abs/2204.05999.

A. Gispert and José B. Mariño. 2006. Catalan-english statistical machine translation without parallel corpus : Bridging through spanish.

Naman Goyal, Cynthia Gao, Vishrav Chaudhary, Peng-Jen Chen, Guillaume Wenzek, Da Ju, Sanjana Krishnan, Marc'Aurelio Ranzato, Francisco Guzmán, and Angela Fan. 2022. The Flores-101 evaluation benchmark for low-resource and multilingual machine translation. *Transactions of the Association for Computational Linguistics*, 10:522–538.

Philip J. Guo. 2018. Non-native english speakers learning computer programming: Barriers, desires, and design opportunities. *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*.

Hamel Husain, Hongqi Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. 2019. Codesearchnet challenge: Evaluating the state of semantic code search. *ArXiv*, abs/1909.09436.

Melvin Johnson, Mike Schuster, Quoc V. Le, Maxim Krikun, Yonghui Wu, Z. Chen, Nikhil Thorat, Fernanda B. Viégas, Martin Wattenberg, Gregory S. Corrado, Macduff Hughes, and Jeffrey Dean. 2017. Google's multilingual neural machine translation system: Enabling zero-shot translation. *Transactions of the Association for Computational Linguistics*, 5:339–351.

Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. 2016. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*.

Taku Kudo. 2018. Subword regularization: Improving neural network translation models with multiple subword candidates. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 66–75, Melbourne, Australia. Association for Computational Linguistics.

Taku Kudo and John Richardson. 2018. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71, Brussels, Belgium. Association for Computational Linguistics.

Anoop Kunchukuttan, Pratik Mehta, and Pushpak Bhattacharyya. 2018. The IIT bombay english-hindi parallel corpus. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation, LREC 2018, Miyazaki, Japan, May 7-12, 2018*. European Language Resources Association (ELRA).

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online. Association for Computational Linguistics.

Xiaonan Li, Yeyun Gong, Yelong Shen, Xipeng Qiu, Hang Zhang, Bolun Yao, Weizhen Qi, Daxin Jiang, Weizhu Chen, and Nan Duan. 2021. Coderetriever: Unimodal and bimodal contrastive learning for code search.

Yujia Li, David H. 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, James Molloy, Daniel Jaymin Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de, Freitas, Koray Kavukcuoglu, and Oriol Vinyals. 2022. Competition-level code generation with alphacode. *ArXiv*, abs/2203.07814.

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

Yinhan Liu, Jiatao Gu, Naman Goyal, Xian Li, Sergey Edunov, Marjan Ghazvininejad, Mike Lewis, and Luke Zettlemoyer. 2020. Multilingual denoising pre-training for neural machine translation. *Transactions of the Association for Computational Linguistics*, 8:726–742.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *ArXiv*, abs/1907.11692.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *International Conference on Learning Representations*.

Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin Clement, Dawn Drain, Daxin Jiang, Duyu Tang, Ge Li, Lidong Zhou, Linjun Shou, Long Zhou, Michele Tufano, MING GONG, Ming Zhou, Nan Duan, Neel Sundaresan, Shao Kun Deng, Shengyu Fu, and Shujie LIU. 2021a. Codexglue: A machine learning benchmark dataset for code understanding and generation. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1.

Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin B. Clement, Dawn Drain, Daxin Jiang, Duyu Tang, Ge Li, Lidong Zhou, Linjun Shou, Long Zhou, Michele Tufano, Ming Gong, Ming Zhou, Nan Duan, Neel Sundaresan, Shao Kun Deng, Shengyu Fu, and Shujie

Liu. 2021b. Codexglue: A machine learning benchmark dataset for code understanding and generation. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*.

Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Haiquan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. 2022. A conversational paradigm for program synthesis. *ArXiv*, abs/2203.13474.

Maja Popović. 2015. chrF: character n-gram F-score for automatic MT evaluation. In *Proceedings of the Tenth Workshop on Statistical Machine Translation*, pages 392–395, Lisbon, Portugal. Association for Computational Linguistics.

Matt Post. 2018. A call for clarity in reporting BLEU scores. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 186–191, Brussels, Belgium. Association for Computational Linguistics.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21:140:1–140:67.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. 2020. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–16. IEEE.

Shuo Ren, Daya Guo, Shuai Lu, Long Zhou, Shujie Liu, Duyu Tang, M. Zhou, Ambrosio Blanco, and Shuai Ma. 2020. Codebleu: a method for automatic evaluation of code synthesis. *ArXiv*, abs/2009.10297.

Holger Schwenk, Vishrav Chaudhary, Shuo Sun, Hongyu Gong, and Francisco Guzmán. 2021. Wikimatrix: Mining 135m parallel sentences in 1620 language pairs from wikipedia. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume, EACL 2021, Online, April 19 - 23, 2021*, pages 1351–1361. Association for Computational Linguistics.

Noam Shazeer and Mitchell Stern. 2018. Adafactor: Adaptive learning rates with sublinear memory cost. In *International Conference on Machine Learning*, pages 4596–4604. PMLR.

Noam M. Shazeer. 2020. Glu variants improve transformer. *ArXiv*, abs/2002.05202.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958.

Jörg Tiedemann. 2012. Parallel data, tools and interfaces in OPUS. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation, LREC 2012, Istanbul, Turkey, May 23-25, 2012*, pages 2214–2218. European Language Resources Association (ELRA).

Michele Tufano, Cody Watson, Gabriele Bavota, Massimiliano Di Penta, Martin White, and Denys Poshyvanyk. 2019. An empirical study on learning bug-fixing patches in the wild via neural machine translation. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 28:1 – 29.

Masao Utiyama and Hitoshi Isahara. 2007. A comparison of pivot methods for phrase-based statistical machine translation. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference*, pages 484–491, Rochester, New York. Association for Computational Linguistics.

Yue Wang, Weishi Wang, Shafiq R. Joty, and Steven C. H. Hoi. 2021. 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, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pages 8696–8708. Association for Computational Linguistics.

Zhiruo Wang, Grace Cuenca, Shuyan Zhou, Frank F. Xu, and Graham Neubig. 2022. Mconala: A benchmark for code generation from multiple natural languages. *ArXiv*, abs/2203.08388.

Guillaume Wenzek, Marie-Anne Lachaux, Alexis Conneau, Vishrav Chaudhary, Francisco Guzm'an, Armand Joulin, and Edouard Grave. 2019. Ccnet: Extracting high quality monolingual datasets from web crawl data. In *International Conference on Language Resources and Evaluation*.

Yonghui Wu, Mike Schuster, Z. Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason R. Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Gregory S. Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *ArXiv*, abs/1609.08144.

Frank F. Xu, Uri Alon, Graham Neubig, and Vincent J. Hellendoorn. 2022. A systematic evaluation of large language models of code. *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming*.

Linting Xue, Noah Constant, Adam Roberts, Mihir Kale, Rami Al-Rfou, Aditya Siddhant, Aditya Barua, and

Colin Raffel. 2021. mt5: A massively multilingual pre-trained text-to-text transformer. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*, pages 483–498. Association for Computational Linguistics.

Pengcheng Yin, Bowen Deng, Edgar Chen, Bogdan Vasilescu, and Graham Neubig. 2018. Learning to mine aligned code and natural language pairs from stack overflow. In *International Conference on Mining Software Repositories*, MSR, pages 476–486. ACM.

Michal Ziemski, Marcin Junczys-Dowmunt, and Bruno Pouliquen. 2016. The united nations parallel corpus v1.0. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation LREC 2016, Portorož, Slovenia, May 23-28, 2016*. European Language Resources Association (ELRA).

A Appendix

A.1 Input representation

We base our shared text/code lexer on the mT5 tokenizer - SentencePiece (Kudo and Richardson, 2018), specifically unigram language model (Kudo, 2018). Since the word distribution in PL essentially differs from that of NL, it is not feasible to directly apply the SentencePiece tokenization on PL. SentencePiece is ineffective in encoding whitespace characters - such as blank space, tab `\t`, and newline character `\n` - which are crucial in representing structures and indentations in source code. We thus add a set of additional tokens for encoding whitespace of different lengths in PL. Considering that developers with different programming habits may type indentations with various lengths and characters (tab or space), we add spaces of length-1/2/4 (denoted as `<space*1>`, `<space*2>`, `<space*4>`, respectively), and tab `\t` to represent various indentations. Moreover, we use the newline symbol `\n` to encode line breaks. Our tokenizer eventually consists of 250,105 SentencePiece vocabularies. Figure 8 exhibits a tokenization example of Python snippets. SentencePiece tends to normalize whitespaces and skip extra empty characters, while our modified tokenizer allows the model to cope with whitespace characters such as indentation in PL.

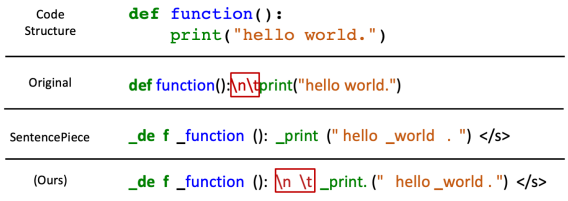


Figure 8: NL/PL-shared tokenization example (Python). "</s>" represents the end-of-sentence token.

A.2 Pre-training data

A.2.1 PL data

Table 6 shows the statistics of monolingual PL data and parallel NL-PL pairs, consisting of 6.5 million monolingual samples and 1.9 million NL-PL pairs in six different PLs. We do not use additional code repositories from GitHub for a fair comparison to baseline PL models.

The NL data may also exist in their paired PL data, serving as a comment or docstring. It could result in data leakage for PL-to-NL translation if NL has been given as a part of PL inputs, thereby hurting the code-to-text test performance. Accordingly, for all code-to-text generation and randomly 50% of text-to-code generation in PTLM training, we replace all NL sentences as an NL placeholder "<|removed|>" if it exists in the corresponding PL fragments.

PL	#Sample	#NL-PL pair
Go	726,768	317,832
Java	1,569,889	454,451
JavaScript	1,857,835	123,889
PHP	977,821	523,712
Python	1,156,085	412,178
Ruby	164,048	48,791
#Total	6,452,446	1,880,853

Table 6: Statistics of CodeSearchNet in six PLs, totaling 6.5 million monolingual PL instances and 1.5 million parallel NL-PL samples.

We additionally observe that parallel data in CodeSearchNet only contain few non-English NLs. Directly regarding all NLs in CodeSearchNet as English would confuse the model to distinguish various NLs. To better leverage this parallel supervision signal, we utilize FastText (Joulin et al., 2016) tools[10] to identify different NLs. Specifically, we only consider NL sentences with confidence higher than 80% predicted by FastText. In PTLM training, we use the predicted language genre with 50% probability at random; otherwise, we treat the sample as "text" other than "English". Therefore, the model could implicitly tell different language genres without being exposed to erroneous supervision.

A.2.2 NL data

Monolingual NL corpus CC-100[11] was constructed by processing CommonCrawl snapshots (Wenzek et al., 2019). The original CC-100 dataset comprises documents separated by double newlines. We maintain the document-level corpus by concatenating paragraphs within the same document page. Table 7 summarizes the statistics of our processed data, totaling 1.5 billion training document pages in 116 monolingual NLs. We rescale the data distribution according to page counts as aforementioned in Eq. (3) with $\alpha = 0.3$.

[10] https://fasttext.cc/docs/en/language-identification.html
[11] https://data.statmt.org/cc-100

ISO code	Language	#Pages (M)	Percent. (%)	ISO code	Language	#Pages (M)	Percent. (%)
af	Afrikaans	1.3	0.09	lt	Lithuanian	9.19	0.61
am	Amharic	0.24	0.02	lv	Latvian	5.83	0.39
ar	Arabic	15.04	1.0	mg	Malagasy	0.15	0.01
as	Assamese	0.05	0.0	mk	Macedonian	1.78	0.12
az	Azerbaijani	4.1	0.27	ml	Malayalam	1.9	0.13
be	Belarusian	1.45	0.1	mn	Mongolian	0.96	0.06
bg	Bulgarian	18.16	1.21	mr	Marathi	1.01	0.07
bn	Bengali	4.11	0.27	ms	Malay	11.92	0.79
bn_rom	Bengali Romanized	6.5	0.43	my	Burmese	0.22	0.01
br	Breton	0.14	0.01	my_zaw	Burmese (Zawgyi)	0.88	0.06
bs	Bosnian	0.4	0.03	ne	Nepali	1.13	0.08
ca	Catalan	7.01	0.47	nl	Dutch	31.16	2.08
cs	Czech	10.15	0.68	no	Norwegian	28.8	1.92
cy	Welsh	0.71	0.05	ns	Northern Sotho	0.03	0.0
da	Danish	30.19	2.01	om	Oromo	0.08	0.01
de	German	69.02	4.6	or	Oriya	0.19	0.01
el	Modern Greek	12.33	0.82	pa	Panjabi	0.33	0.02
en	English	247.59	16.49	pl	Polish	31.2	2.08
eo	Esperanto	0.58	0.04	ps	Pushto	0.26	0.02
es	Spanish	60.54	4.03	pt	Portuguese	39.0	2.6
et	Estonian	3.94	0.26	qu	Quechua	0.03	0.0
eu	Basque	1.86	0.12	rm	Romansh	0.03	0.0
fa	Persian	36.96	2.46	ro	Romanian	30.21	2.01
ff	Fulah	0.02	0.0	ru	Russian	123.18	8.2
fi	Finnish	28.12	1.87	sa	Sanskrit	0.12	0.01
fr	French	62.11	4.14	sc	Sardinian	0.0	0.0
fy	Western Frisian	0.2	0.01	sd	Sindhi	0.08	0.01
ga	Irish	0.52	0.03	si	Sinhala	0.67	0.04
gd	Scottish Gaelic	0.11	0.01	sk	Slovak	17.0	1.13
gl	Galician	1.85	0.12	sl	Slovenian	6.24	0.42
gn	Guarani	0.02	0.0	so	Somali	0.4	0.03
gu	Gujarati	0.75	0.05	sq	Albanian	2.72	0.18
ha	Hausa	0.46	0.03	sr	Serbian	2.7	0.18
he	Hebrew	12.77	0.85	ss	Swati	0.0	0.0
hi	Hindi	8.11	0.54	su	Sundanese	0.06	0.0
hi_rom	Hindi Romanized	1.97	0.13	sv	Swedish	46.77	3.12
hr	Croatian	16.54	1.1	sw	Swahili	1.13	0.08
ht	Haitian	0.09	0.01	ta	Tamil	4.12	0.27
hu	Hungarian	26.14	1.74	ta_rom	Tamil Romanized	1.6	0.11
hy	Armenian	2.14	0.14	te	Telugu	1.21	0.08
id	Indonesian	79.68	5.31	te_rom	Telugu Romanized	1.9	0.13
ig	Igbo	0.04	0.0	th	Thai	23.92	1.59
is	Icelandic	2.06	0.14	tl	Tagalog	2.64	0.18
it	Italian	24.67	1.64	tn	Tswana	0.24	0.02
ja	Japanese	65.61	4.37	tr	Turkish	18.42	1.23
jv	Javanese	0.31	0.02	ug	Uighur	0.11	0.01
ka	Georgian	2.68	0.18	uk	Ukrainian	24.98	1.66
kk	Kazakh	1.77	0.12	ur	Urdu	2.26	0.15
km	Central Khmer	0.61	0.04	ur_rom	Urdu Romanized	4.58	0.3
kn	Kannada	0.91	0.06	uz	Uzbek	0.46	0.03
ko	Korean	35.68	2.38	vi	Vietnamese	52.48	3.5
ku	Kurdish	0.24	0.02	wo	Wolof	0.13	0.01
ky	Kirghiz	0.41	0.03	xh	Xhosa	0.15	0.01
la	Latin	3.1	0.21	yi	Yiddish	0.15	0.01
lg	Ganda	0.09	0.01	yo	Yoruba	0.02	0.0
li	Limburgan	0.02	0.0	zh	Chinese (Simplified)	40.0	2.66
ln	Lingala	0.02	0.0	zh-Hant	Chinese (Traditional)	12.33	0.82
lo	Lao	0.2	0.01	zu	Zulu	0.07	0.0

Table 7: Statistics of CC-100 corpus, totaling 1.5 billion training document pages from 116 different NLs. Reported training pages and percentages are calculated according to the document distribution of original data. Note that our 116 NLs include 5 Romanized variants of existing languages denoted by "Romanized".

ISO code	Lang 1	Lang 2	#Pairs (M)	Percent. (%)	ISO code	Language 1	Language 2	#Pairs (M)	Percent. (%)
ar-bg	Arabic	Bulgarian	46.57	0.59	en-ru	English	Russian	312.91	3.99
ar-de	Arabic	German	44.58	0.57	en-sw	English	Swahili	9.41	0.12
ar-el	Arabic	Greek	45.66	0.58	en-th	English	Thai	26.11	0.33
ar-en	Arabic	English	199.26	2.54	en-tr	English	Turkish	196.96	2.51
ar-es	Arabic	Spanish	141.9	1.81	en-ur	English	Urdu	11.04	0.14
ar-fr	Arabic	French	118.52	1.51	en-vi	English	Vietnamese	79.56	1.02
ar-hi	Arabic	Hindi	7.24	0.09	en-zh	English	Chinese	156.31	1.99
ar-ru	Arabic	Russian	96.15	1.23	es-fr	Spanish	French	522.47	6.67
ar-sw	Arabic	Swahili	2.38	0.03	es-hi	Spanish	Hindi	15.93	0.2
ar-th	Arabic	Thai	9.42	0.12	es-ru	Spanish	Russian	166.12	2.12
ar-tr	Arabic	Turkish	58.32	0.74	es-sw	Spanish	Swahili	7.88	0.1
ar-ur	Arabic	Urdu	2.43	0.03	es-th	Spanish	Thai	10.15	0.13
ar-vi	Arabic	Vietnamese	17.36	0.22	es-tr	Spanish	Turkish	105.87	1.35
ar-zh	Arabic	Chinese	55.68	0.71	es-ur	Spanish	Urdu	0.8	0.01
bg-de	Bulgarian	German	57.71	0.74	es-vi	Spanish	Vietnamese	44.33	0.57
bg-el	Bulgarian	Greek	68.07	0.87	es-zh	Spanish	Chinese	74.93	0.96
bg-en	Bulgarian	English	151.04	1.93	fr-hi	French	Hindi	15.38	0.2
bg-es	Bulgarian	Spanish	86.31	1.1	fr-ru	French	Russian	154.58	1.97
bg-fr	Bulgarian	French	69.09	0.88	fr-sw	French	Swahili	8.91	0.11
bg-hi	Bulgarian	Hindi	3.35	0.04	fr-th	French	Thai	8.7	0.11
bg-ru	Bulgarian	Russian	66.25	0.85	fr-tr	French	Turkish	85.83	1.1
bg-sw	Bulgarian	Swahili	1.12	0.01	fr-ur	French	Urdu	0.74	0.01
bg-th	Bulgarian	Thai	6.98	0.09	fr-vi	French	Vietnamese	25.37	0.32
bg-tr	Bulgarian	Turkish	66.06	0.84	fr-zh	French	Chinese	70.14	0.9
bg-ur	Bulgarian	Urdu	0.59	0.01	hi-ru	Hindi	Russian	7.32	0.09
bg-vi	Bulgarian	Vietnamese	11.23	0.14	hi-sw	Hindi	Swahili	1.46	0.02
bg-zh	Bulgarian	Chinese	11.56	0.15	hi-th	Hindi	Thai	2.69	0.03
de-el	German	Greek	72.85	0.93	hi-tr	Hindi	Turkish	8.75	0.11
de-en	German	English	655.83	8.37	hi-ur	Hindi	Urdu	1.49	0.02
de-es	German	Spanish	242.73	3.1	hi-vi	Hindi	Vietnamese	6.11	0.08
de-fr	German	French	269.02	3.43	hi-zh	Hindi	Chinese	2.39	0.03
de-hi	German	Hindi	9.36	0.12	ru-sw	Russian	Swahili	2.17	0.03
de-ru	German	Russian	80.08	1.02	ru-th	Russian	Thai	8.12	0.1
de-sw	German	Swahili	3.22	0.04	ru-tr	Russian	Turkish	51.77	0.66
de-th	German	Thai	7.07	0.09	ru-ur	Russian	Urdu	2.56	0.03
de-tr	German	Turkish	57.14	0.73	ru-vi	Russian	Vietnamese	16.47	0.21
de-ur	German	Urdu	0.86	0.01	ru-zh	Russian	Chinese	61.53	0.79
de-vi	German	Vietnamese	20.77	0.27	sw-th	Swahili	Thai	0.49	0.01
de-zh	German	Chinese	22.8	0.29	sw-tr	Swahili	Turkish	4.16	0.05
el-en	Greek	English	190.87	2.44	sw-ur	Swahili	Urdu	0.39	0.0
el-es	Greek	Spanish	133.05	1.7	sw-vi	Swahili	Vietnamese	3.02	0.04
el-fr	Greek	French	117.73	1.5	sw-zh	Swahili	Chinese	1.08	0.01
el-hi	Greek	Hindi	4.55	0.06	th-tr	Thai	Turkish	9.26	0.12
el-ru	Greek	Russian	45.1	0.58	th-ur	Thai	Urdu	0.64	0.01
el-sw	Greek	Swahili	1.84	0.02	th-vi	Thai	Vietnamese	4.62	0.06
el-th	Greek	Thai	5.83	0.07	th-zh	Thai	Chinese	0.97	0.01
el-tr	Greek	Turkish	69.81	0.89	tr-ur	Turkish	Urdu	4.34	0.06
el-ur	Greek	Urdu	0.31	0.0	tr-vi	Turkish	Vietnamese	16.29	0.21
el-vi	Greek	Vietnamese	14.84	0.19	tr-zh	Turkish	Chinese	14.62	0.19
el-zh	Greek	Chinese	11.44	0.15	ur-vi	Urdu	Vietnamese	0.58	0.01
en-es	English	Spanish	1088.62	13.89	ur-zh	Urdu	Chinese	0.11	0.0
en-fr	English	French	884.16	11.28	vi-zh	Vietnamese	Chinese	9.31	0.12
en-hi	English	Hindi	27.42	0.35					

Table 8: Statistics of OPUS corpus, totaling 7.8 billion bilingual NL pairs from 105 different NL pairs. The reported count of bilingual pairs ("#Sent.") and percentage ("#Percent.") are calculated according to the original data.

Parallel NL corpus We use parallel NL data collected from OPUS website[12]. We summarize the statistics of collected OPUS data in Table 8. The data we use are in 15 different NLs, comprising of 105 various bilingual language pairs (ignoring the dual direction between two languages) and 7.8 billion sentence pairs in total. Similar to CC-100 preprocessing, we apply the same data resampling process by following Eq. (3), with $\alpha = 0.3$.

A.2.3 Data rebalance between NL and PL

Considering that the data amount of PL and NL data vastly differs, the data distribution across NL and PL will still be unbalanced even after rescaling as per Eq. (3), which could give rise in biases towards high-resource modality (*i.e.*, NL). To mitigate this issue, we set the data distribution of PL and NL as 1:1 by equating the training sample ratio of PL with that of NL during pre-training. In other words, we train the same sample counts for NL and PL corpora.

A.3 Experimental settings

A.3.1 Pre-training settings

We use the same T5 architecture with a 12-layer encoder, a 12-layer decoder, 768 hidden units (d_{model}), 12 heads, 2048 feedforward linear units (d_{ff}), GELU activations, a dropout (Srivastava et al., 2014) rate as 0.1, and no embedding tying. Chen et al. (2021) find no difference between training from pre-trained model weights and that from scratch, except that the former converges more quickly. To this end, we use mT5 checkpoint[13] for initialization, which already contains strong multilingual NL representations.

For pre-training, we set the maximum length (L) of 512/1024, a micro-batch size of 8/4 with a gradient accumulation step of 15. We utilize the Adafactor (Shazeer and Stern, 2018) optimizer and a linear warmup of 1000 steps with a peak learning rate of 1e-4. All pre-training tasks are run on a cluster of 32 NVIDIA A100 GPUs with 40G memory for 100,000 training steps. To accelerate the pre-training, we utilize the ZeRO stage1 approach (Rajbhandari et al., 2020) for partitioning optimizer states and enable BFloat16 half-precision format for mixed-precision training. The total pre-training time lasts around four weeks.

[12] https://opus.nlpl.eu

[13] https://github.com/google-research/multilingual-t5#released-model-checkpoints

A.3.2 Evaluation datasets

Table 9 reports the detailed statistics of evaluation dataset across a suit of code benchmarks, including NL-to-PL, PL-to-NL, PL-to-PL, and NL-to-NL.

Task	Dataset	Language	Train	Valid	Test
NL-PL	mCoNaLa (Wang et al., 2022)	Spanish ↔ Python	-	-	341
		Japanese ↔ Python	-	-	210
		Russian ↔ Python	-	-	345
PL-PL	Bugs2Fix (Tufano et al., 2019)	Java-small	46,680	5,835	5,835
		Java-medium	52,364	6,545	6,545
NL-NL	Microsoft Docs (Lu et al., 2021a)	Danish↔English	42,701	1,000	1,000
		Latvian↔English	18,749	1,000	1,000
		Norwegian↔English	44,322	1,000	1,000
		Chinese↔English	50,154	1,000	1,000

Table 9: Statistics of downstream benchmark datasets.

A.3.3 Finetuning settings

When finetuning on end tasks, we use mini-batches of 8/4, and a maximum input length of 512. We set the maximum target length as 128, 64, 256, and 256 for code summarization, text-to-code, documentation translation, and code repair tasks, respectively. We use prompt-based finetuning by prepending a task prompt (as shown in Table 10) before each sample for training and evaluation. We finetune code-to-text, text-to-code, and documentation translation tasks for 100 epochs and train 10 epochs on the code repair dataset. For all finetuning experiments, we use the AdamW (Loshchilov and Hutter, 2019) optimizer with a learning rate of 5e-5. As to model inference, we apply beam search decoding with five beams. We conducted all finetuning experiments on 8 NVIDIA V100 GPUs with 32G memory.

A.4 Ablation results

Table 11 reports ablation results of code summarization (mCoNaLa), text-to-code generation (mCoNaLa), documentation translation (Microsoft Docs), and code repair (Bugs2Fix), showing that combining SCLM and PTLM can confer benefit for all of the end tasks.

A.5 Chinese code summarization

Data curation To expand the evaluation on Chinese code summarization, we release a translated variant of mCoNaLa dataset via crowd-sourcing. Specifically, we hire human translators who satisfy all following three criteria to undertake the crowd-sourcing:

Finetuning task	Prompt format
Code-to-text	"translate Spanish to Python: \n" "translate Japanese to Python: \n" "translate Russian to Python: \n"
Text-to-code	"translate Python to Spanish: \n" "translate Python to Japanese: \n" "translate Python to Russian: \n"
Documentation translation	"translate 'src_lang' to 'tgt_lang':\n"
Code repair	"fix bugs: \n"

Table 10: Task prompt we use for finetuning. For documentation translation, the "src_lang" and "tgt_lang" represent the source and target language (*e.g.*, English, Danish, Latvian, Norwegian, and Chinese), respectively.

1. Must be a native Chinese speaker;
2. Holding at least a master's degree in Spanish, Japanese, and Russian translation, literature, or related subjects;
3. Holding professional translation certificates in the corresponding language.

After human translation, we also employ professional engineers who are Chinese native speakers with at least five years of experience in Python to perform further translation refinement. We will release this dataset to speed up the research on multilingual code summarization.

Examples of Chinese code summarization (zero-shot prompting) We show the Chinese code summarization examples of our model under zero-shot prompting evaluation in Figure 9. We prepend the instruction prompt "translate Python to Chinese: \n" for training and evaluation. It demonstrates that our model equips the zero-shot ability on Chinese code summarization, affirming the positive effect of our cross-lingual pre-training. Moreover, as shown in Figure 9, our model focuses on the high-level meaning of the input code fragments, neglecting the implementation details. We guess this is because we use code search corpus as NL-PL bilingual training data, where NL instructions comprising high-level descriptions are usually extracted from code comments. It causes a discrepancy between the training and evaluation settings.

A.6 Qualitative examples (zero-shot prompting)

Zero-shot multilingual PL-to-NL generation Figure 9 and 10 illustrate the code summarization examples with zero-shot prompting. As mentioned earlier, As illustrated in Figure 9 and 10, we find that our model focuses on the global overview of code semantics rather than verbalizing the implementation process. Moreover, when explaining a short snippet of code, different people may interpret it with various meanings, which we refer to as "*program ambiguity*", making difficulties in annotating and evaluating the multilingual code summarization. This is because the test-set reference of mCoNaLa is human-rewritten, while the training NL is not. We also find that the model tends to copy small code snippets for code summarization. For instance, given inputs "# -*- utf-8 -*- ", our model tends to copy the original string rather than describe its usage using NL.

Zero-shot NL-to-PL generation Figure 11 and 12 demonstrate examples of zero-shot text-to-code generation. We also observe that ERNIE-Code is well-performed in generating function names, arguments, and docstrings. It tends to generate function-level snippets and call user-defined functions following the object-oriented logic while lacking the knowledge of builtin functions or user-defined contexts given multilingual NL inputs. The given Japanese instruction requires the model to memorize the API usage of `selenium`[14] library that our model may never see in the training data. We argue that training on data collected from GitHub and StackOverflow would confer benefits in memorizing and comprehending the API usage and instruction contexts. We suspect that the training on additional PL data from GitHub and StackOverflow rather than limited data of CodeSearchNet can lead to large improvements. Note that the generated "<|removed|>" docstring in Figure 11 is consistent with our preprocessing in §A.2.1.

Zero-shot multilingual NL-to-NL translation To further validate the zero-shot translation capability between multilingual NLs, we report several selected language pairs from different language families and translate technical terminologies with zero-shot prompting. Figure 13 exhibits examples of multilingual NL translation in eight randomly selected directions, such as Spanish to French and Russian to Arabic. This suggests that our cross-lingual pre-training can capture semantic alignment without seeing direct supervision from bilingual phrase or short-term pairs.

[14] https://selenium-python.readthedocs.io/

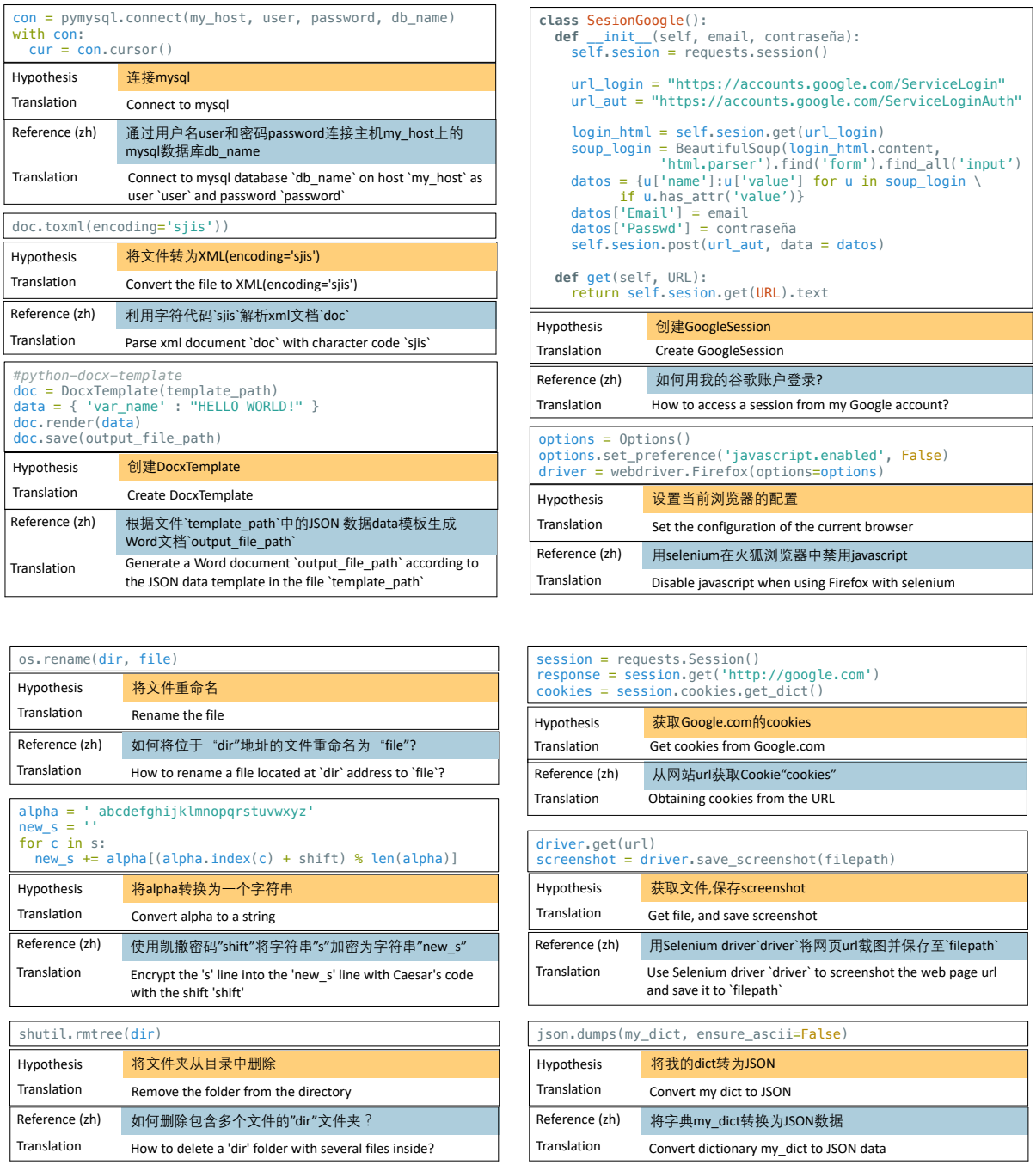


Figure 9: Examples of Chinese code summarization with zero-shot prompting.

Model	es			ja			ru			Avg.		
	B-4	R-L	chrF	B-4	R-L	chrF	B-4	R-L	chrF	B-4	R-L	chrF
Ours	1.90	32.51	23.22	0.30	10.62	9.16	0.43	5.01	16.60	0.88	16.05	16.33
\SCLM	1.04	23.96	19.56	0.17	7.62	8.88	0.21	2.69	15.53	0.47	11.42	15.10
\PTLM	0.96	22.47	24.00	0.06	5.71	8.22	0.20	4.92	14.66	0.41	11.03	14.15

(a) Ablation results on multilingual code summarization.

Model	es			ja			ru			Avg.		
	B-4	R-L	C-B	B-4	R-L	C-B	B-4	R-L	C-B	B-4	R-L	C-B
Ours	2.25	14.92	0.06	8.06	22.65	0.10	6.12	25.27	0.08	5.48	20.95	0.08
\SCLM	2.42	14.27	0.06	6.89	21.31	0.10	5.41	23.09	0.08	4.91	19.56	0.08
\PTLM	2.08	13.94	0.06	6.40	17.77	0.10	5.11	23.17	0.08	4.53	18.29	0.08

(b) Ablation results on multilingual text-to-code generation.

Model	En-Da				En-Lv				En-No				En-Zh				Avg. B-4	Avg. EM
	→		←		→		←		→		←		→		←			
	B-4	EM	B-4	EM	B-4	EM	B-4	EM	B-4	EM	B-4	EM	B-4	EM	B-4	EM		
Ours	71.16	13.2	72.70	27.2	60.98	10.6	69.28	24.3	71.39	15.7	72.28	26.3	74.53	24.3	72.43	28.5	70.59	21.26
\SCLM	67.70	11.3	68.50	23.4	55.98	7.5	64.39	21.6	68.05	11.4	68.03	24.1	72.52	20.1	68.56	24.8	66.72	18.03
\PTLM	66.91	10.4	67.66	23.9	55.84	7.5	63.87	21.6	67.71	11.3	66.86	23.5	71.91	19.6	67.98	24.2	66.09	17.75

(c) Ablation results on documentation translation.

Model	Refine small		Refine medium		Avg.	
	B-4	EM	B-4	EM	B-4	EM
Ours	80.09	13.21	91.20	2.22	85.65	7.72
\SCLM	79.65	13.04	91.19	2.17	85.42	7.61
\PTLM	79.73	11.31	91.13	1.68	85.43	6.50

(d) Ablation results on code repair.

Table 11: Ablation results of downstream tasks, including multilingual code summarization (a), text-to-code generation (b), documentation translation (c), and code repair (d).

Qualitative findings We also observe that our model allows for naming functions and completing corresponding arguments according to multilingual textual instructions, as shown in Figure 14, confirming that our model learns to bridge the semantics and syntax between multilingual NL instructions and PL functions.

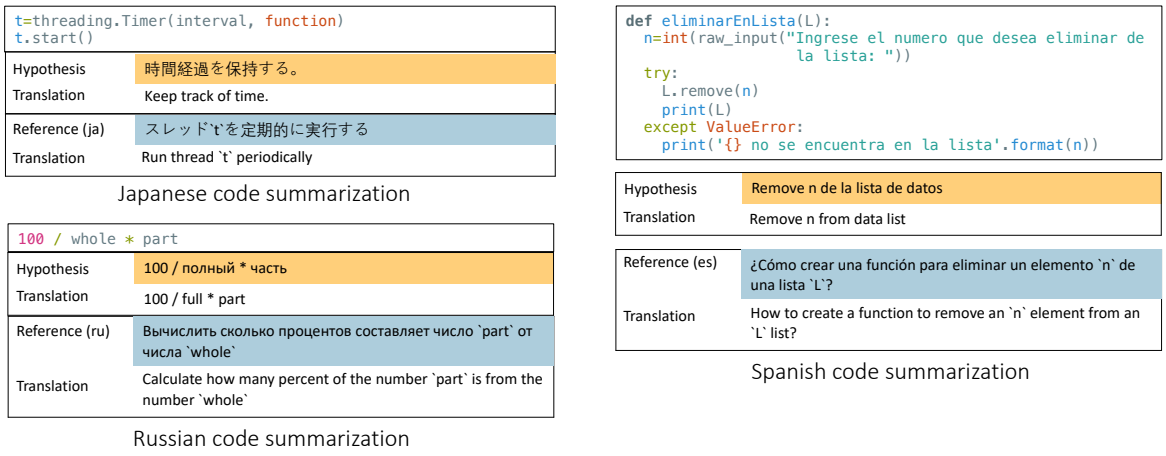


Figure 10: Multilingual code summarization (code-to-text) examples with zero-shot prompting.

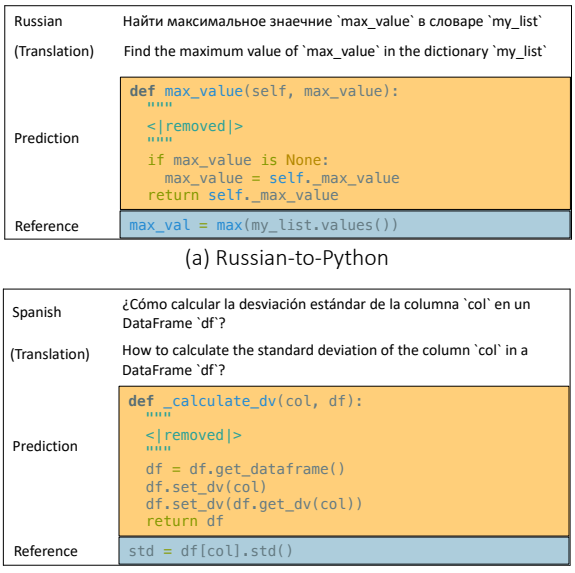


Figure 11: Multilingual code summarization (text-to-code) examples with zero-shot prompting.

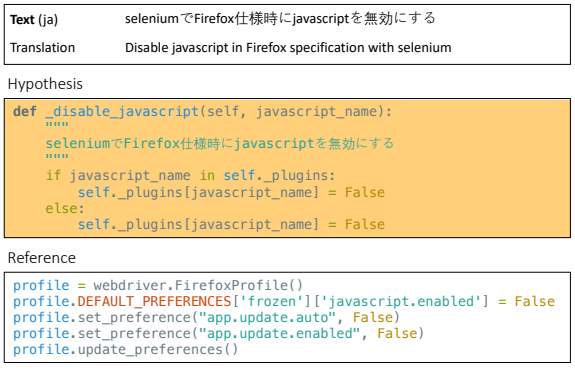


Figure 12: Examples of zero-shot multilingual text-to-code generation (Japanese). The region highlighted in orange is a hypothesis generated by our model.

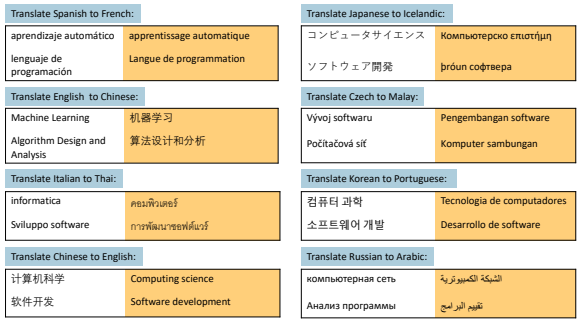


Figure 13: Examples of zero-shot text-to-text translation on technical jargon. The region highlighted in orange is the target language, whereas that in blue is the prefixed prompt we use for zero-shot translation.

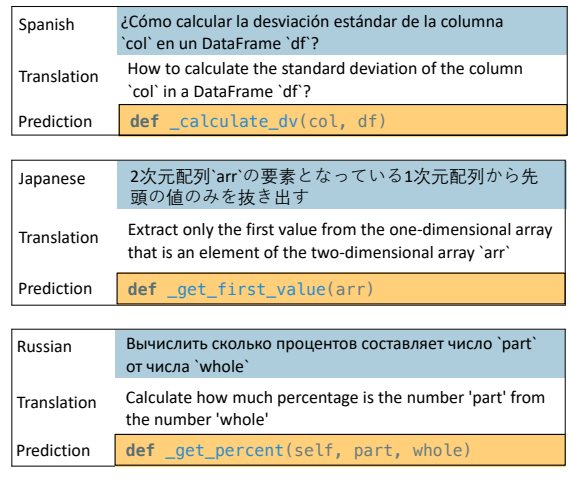


Figure 14: Examples of function naming and argument filling in text-to-code generation (zero-shot).

ACL 2023 Responsible NLP Checklist

A For every submission:

☑ A1. Did you describe the limitations of your work?
Section 6

☒ A2. Did you discuss any potential risks of your work?
Similar to large language models (LLMs) for natural language, LLMs for code trained on a large amount of human data can be prone to the societal stereotypes and cultural bias about names and genders. Besides, code generation can potentially have security risks. For examples, code LLMs could generate code for vulnerabilities or malware, leading to security risks.

☑ A3. Do the abstract and introduction summarize the paper's main claims?
Section 1

☒ A4. Have you used AI writing assistants when working on this paper?
Left blank.

B ☒ Did you use or create scientific artifacts?

Left blank.

☐ B1. Did you cite the creators of artifacts you used?
No response.

☐ B2. Did you discuss the license or terms for use and / or distribution of any artifacts?
No response.

☐ B3. Did you discuss if your use of existing artifact(s) was consistent with their intended use, provided that it was specified? For the artifacts you create, do you specify intended use and whether that is compatible with the original access conditions (in particular, derivatives of data accessed for research purposes should not be used outside of research contexts)?
No response.

☐ B4. Did you discuss the steps taken to check whether the data that was collected / used contains any information that names or uniquely identifies individual people or offensive content, and the steps taken to protect / anonymize it?
No response.

☐ B5. Did you provide documentation of the artifacts, e.g., coverage of domains, languages, and linguistic phenomena, demographic groups represented, etc.?
No response.

☐ B6. Did you report relevant statistics like the number of examples, details of train / test / dev splits, etc. for the data that you used / created? Even for commonly-used benchmark datasets, include the number of examples in train / validation / test splits, as these provide necessary context for a reader to understand experimental results. For example, small differences in accuracy on large test sets may be significant, while on small test sets they may not be.
No response.

The Responsible NLP Checklist used at ACL 2023 is adopted from NAACL 2022, with the addition of a question on AI writing assistance.

C ☑ Did you run computational experiments?

Appendix A.5

☑ C1. Did you report the number of parameters in the models used, the total computational budget (e.g., GPU hours), and computing infrastructure used?
Parameter count: Table 5 in appendix; Computational budgets and computing infrastructure: Appendix A.5.

☑ C2. Did you discuss the experimental setup, including hyperparameter search and best-found hyperparameter values?
Appendix A.5.

☐ C3. Did you report descriptive statistics about your results (e.g., error bars around results, summary statistics from sets of experiments), and is it transparent whether you are reporting the max, mean, etc. or just a single run?
No response.

☑ C4. If you used existing packages (e.g., for preprocessing, for normalization, or for evaluation), did you report the implementation, model, and parameter settings used (e.g., NLTK, Spacy, ROUGE, etc.)?
Section 3.1

D ☑ Did you use human annotators (e.g., crowdworkers) or research with human participants?

Appendix A.7

☐ D1. Did you report the full text of instructions given to participants, including e.g., screenshots, disclaimers of any risks to participants or annotators, etc.?
No response.

☑ D2. Did you report information about how you recruited (e.g., crowdsourcing platform, students) and paid participants, and discuss if such payment is adequate given the participants' demographic (e.g., country of residence)?
Left blank.

☐ D3. Did you discuss whether and how consent was obtained from people whose data you're using/curating? For example, if you collected data via crowdsourcing, did your instructions to crowdworkers explain how the data would be used?
No response.

☐ D4. Was the data collection protocol approved (or determined exempt) by an ethics review board?
No response.

☐ D5. Did you report the basic demographic and geographic characteristics of the annotator population that is the source of the data?
No response.