MCoNaLa: A Benchmark for Code Generation from Multiple Natural Languages

Zhiruo Wang[1*], **Grace Cuenca**[2*] **Shuyan Zhou**[1] **Frank F. Xu**[1] **Graham Neubig**[1]
[1]Carnegie Mellon University [2]Princeton University
{zhiruow,shuyanzh,fangzhex,gneubig}@cs.cmu.edu, gcuenca@princeton.edu

Abstract

While there has been a recent burgeoning of applications at the intersection of natural and programming languages, such as code generation and code summarization, these applications are usually English-centric. This creates a barrier for program developers who are not proficient in English. To mitigate this gap in technology development across languages, we propose a multilingual dataset, MCoNaLa, to benchmark code generation from natural language commands extending beyond English. Modeled off of the methodology from the English Code/Natural Language Challenge (CoNaLa) dataset, we annotated a total of 896 NL-code pairs in three languages: Spanish, Japanese, and Russian. We present a quantitative evaluation of performance on the MCoNaLa dataset by testing with state-of-the-art code generation systems. While the difficulties vary across these three languages, all systems lag significantly behind their English counterparts, revealing the challenges in adapting code generation to new languages. [1]

Spanish	¿Cómo sumar el campo `*precio*` de todos los elementos del modelo `*Precompra*` en Django? (How to sum the `*precio*` field of all the elements of the `*Precompra*` model in Django?)
	totaldos = Precompra.objects.aggregate(Sum(precio)).values()[0])
Japanese	2次元配列`*arr*`の要素となっている1次元配列から先頭の値のみを抜き出す (Extract only the first value from the 1D array that is the element of the 2D array `*arr*`)
	arr[:, 0]
Russian	Установить кодировку `*my_encode*` для переменных окружения пользователя '*username*' (Set `*my_encode*` encoding for "*username*" environment variables)
	os.environ('username').decode(my_encode)

Figure 1: Examples in the MCoNaLa dataset, that aim to generate general-purpose Python code snippets from source intent of multiple natural languages.

1 Introduction

Code intelligence is a nascent but rapidly developing application of language processing, brewing an increasing number of code-relevant applications such as code summarization (Allamanis et al., 2016; Hu et al., 2018; Ahmad et al., 2020) and natural language (NL) to code generation (Ling et al., 2016; Rabinovich et al., 2017; Yin et al., 2018a; Xu et al., 2020; Norouzi et al., 2021; Wang et al., 2021). Progress on these tasks is tested using a number of code-specific tasks and benchmarks (Oda et al., 2015; Zhong et al., 2017; Yin et al., 2018b; Lu et al., 2021). However, in the cases where these benchmarks include natural language, that language is almost invariably English. There are a few exceptions, but most of them either focus on languages of specific domains (Sherborne and Lapata, 2021; Sherborne et al., 2020; Moradshahi et al., 2020) or types (Oda et al., 2015; Liang et al., 2021), or contain natural language intents collected via automatic translation (Li et al., 2020). However, similarly to how Kwiatkowski et al. (2019) argue that "natural questions" are necessary to appropriately benchmark question answering systems, we argue that ensuring the naturalness and coverage of questions is essential for benchmarking code generation systems as well.

One English code generation dataset based on natural programming questions is the CoNaLa dataset (Yin et al., 2018a), which is based on natural developer questions harvested from the StackOverflow (SO) question answering forum[2], In fact, besides English, SO also supports four other languages (Spanish, Portuguese, Japanese, and Russian) that have strong developer communities and engage in non-English programming environments. In this work, we utilize this resource to construct the MCoNaLa dataset, consisting of 341, 210, and

[*]Equal contribution.
[1]The code and data are available at https://github.com/zorazrw/multilingual-conala

[2]https://stackoverflow.com

345 manually-curated parallel samples with natural intents in Spanish, Japanese, and Russian and code snippets implementing these intents. Similarly to CoNaLa, these snippets are collected from language-specific SO sites and annotated by native speakers who are also proficient in the Python programming language.

To provide insights on the state of code generation using this new resource, we conduct comprehensive experiments with three state-of-the-art seq2seq models. Further, to enable cross-lingual transfer, we experiment with standard cross-lingual transfer approaches based on translation (Ruder, 2021; Shi et al., 2021; Shima and Mitamura; Kuo; Hartrumpf et al., 2008), or utilizing a multilingual NL encoder (e.g., mBART (Liu et al., 2020)). Our results suggest that cross-lingual NL-to-code generation is challenging. Among all the cross-lingual transfer settings, the highest average BLEU score on the three testing languages is 7.28. This lags far behind their English counterpart which achieves a 33.41 BLEU score. In addition, we find that it is beneficial to have task-specific modules and task-aware pretraining. Models with them consistently outperform generic pretrained seq2seq models like mBART with the same amount of training data. Finally, varied language contexts and experiment settings consistently affect the model performance. All-in-all, our corpus and experiments demonstrate the varied difficulty of the NL-to-Code generation task under different language contexts, emphasizing the need to develop a language-comprehensive approach to code intelligence.

2 The MCoNaLa Dataset

2.1 Task Definition

Concerning the task of converting natural language into machine-executable programs, our specific focus is to build a benchmark evaluation dataset that evaluates the ability of models to consume *intents* written in multiple natural languages (English, Spanish, Japanese, Russian) and generate Python code *snippets*. For each example in Fig. 1, the upper *intents* in different natural languages and asks for an operation, and the lower *snippets* respond with a programmatic execution.

2.2 Annotation Workflow

We collect MCoNaLa dataset in Spanish, Japanese, and Russian languages. For each language, we hired one native speaker who is proficient in both English and Python to annotate the data.[3]

intent	how to set global const variables in python
rewritten	assign float 9.8 to variable `GRAVITY`
snippet	`GRAVITY = 9.8`

Figure 2: Example to illustrate the annotation process.

We here outline the procedures and instructions used in the data annotation workflow here.

Source and Selection Besides the English version, StackOverflow also has available forums in four other languages: Spanish, Portuguese, Japanese, and Russian. We successfully collected intent-snippet parallel data from three language sources: Spanish, Japanese and Russian. We were unsuccessful in finding a Portuguese-speaking annotator to complete the annotation at time of the corpus collection.

Identifying How-to Questions We follow Yin et al. (2018a) and focus on how-to type questions. A how-to question is an imperative utterance that asks to achieve a particular goal achievable via code execution. As exemplified by Fig. 2, the intent asks how to “set global constant variables”. Posts are firstly checked by the annotators to identify if they contain any how-to questions, usually in the titles or question descriptions. Only a post containing how-to questions is selected for the subsequent annotation. We then use these manually categorized how-to questions to train a classifier. This classifier can then be used to classify future utterances without human intervention. We combine the manually and automatically filtered questions and their corresponding posts as the seed post collection.

Collecting Intent-Snippet Pairs We provide the seed post collection to the annotators and ask them to find at most three snippets of Python code that correctly answer the how-to question. These snippets are annotated as the answers to the identified question.

An intent without ambiguity should clearly convey an executable command and contain the necessary detail to precisely define the code snippet. However, the post title and question description

[3] Annotators were hired on https://www.upwork.com. Due to the relatively high cost and difficulty of hiring reliable annotators with such a specialized skill set, we only employ one annotator per language.

of a raw StackOverflow post often do not satisfy such criteria. For example, in Fig. 2, the original question indicates the user's intent to set a variable, but doesn't indicate which variable should be set to what value.

Therefore, for each post, we ask annotators to rewrite the title into an imperative command corresponding to the answer code snippet. If the title alone is not sufficiently clear, annotators can further refer to the descriptive texts below the title. The rewritten intents must contain the variable names that are mentioned in the code snippet. Furthermore, any variable names and data types in the rewritten intent are required to be surrounded by the ASCII grave accent marks (e.g., `` `data` ``). Distinctively, any string literals or file path names should be surrounded by singular typographic quotation marks (e.g., ‘`file1.txt`’, ‘`https://www.abc.com/`’).

After the above steps, we end up collected 341, 210, and 345 intent-snippet pairs in Spanish, Japanese, and Russian, respectively.

It is notable that our goal is to leverage the MCoNaLa dataset to benchmark the cross-lingual NL-to-Code generation task, instead of curating large scale dataset for each language independently. Therefore, the collected samples are for *testing* purposes only. Although the size of dataset in each language is relatively small, we believe that they are representative because they are naturally occurring questions on SO in respective language environments.

3 Experiment

In this section, we evaluate the three baseline models on the MCoNaLa dataset in their viable experiment settings. We first specify the dataset usage (§ 3.1) and explain three train-test settings (§ 3.2). We then introduce the systems we evaluate: two state-of-the-art NL-to-Code models and one pretrained multilingual encoder (§ 3.3). Lastly, we elaborate the experiment details (§ 3.4), present their results, and analyze with respect to each and across multiple languages (§ 3.5).

3.1 English CoNaLa as Training Data

As noted above, the size of multilingual parallel data is sufficient only for testing purposes, and thus we must rely on other datasets for training. Therefore, we resort to its larger English counterpart, CoNaLa (Yin et al., 2018a), to enable model training. The English CoNaLa dataset contains a relatively large 2879 manually annotated samples and 600k samples mined from the web or API documents, and hence can serve as a reasonable source for model training. In contrast to English, we refer to three test languages as *target* languages.

3.2 Train-Test Settings

While our training data is in English, the test data are in Spanish, Japanese and Russian respectively. To bridge the natural language gap between training and test, we follow existing works (Hu et al., 2020) and adopt two transfer paradigms: (1) applying a multilingual encoder that can encode both English and target languages. The encoder relies on the English training data to learn the mapping between NL and code, while generalizing its pretrained representations to learn to transfer between natural languages. (2) performing NL translation in advance to unify the training and the test languages (e.g., translating intents in English training data to Japanese). In this way, we could use any monolingual NL-to-Code model. We further explain the implementation details of these settings.

Zero-shot: Training on English Samples and Directly Testing on Target Language Samples The most straightforward usage of English CoNaLa is to train on its original English NL-to-Code samples and directly evaluate samples in target languages. Despite its simplicity in methodology, the difference in training and test languages adds complexity to the problem, requiring models to understand both languages and rephrase them into executable code. Multilingual models can be effective baselines since they can encode multiple natural languages (Devlin, 2018; Liu et al., 2020; Conneau et al., 2020; Xue et al., 2021) and translate between them. In this work, we examine whether similar methodology can be used to learn to translate natural languages to programming languages as well.

Fig. 3 exemplifies this *zero-shot* setting, where we train the baseline model using English CoNaLa samples, and the same trained model is used for evaluation in three target languages.

Translate-Train: English Training Samples Translated into Three Target Languages Instead of encoding multiple languages, one could also leverage translation to close the train-test language gap and enable a monolingual pipeline. In the *translate-train* setting, we translate English intent to each target language and pair it with the orig-

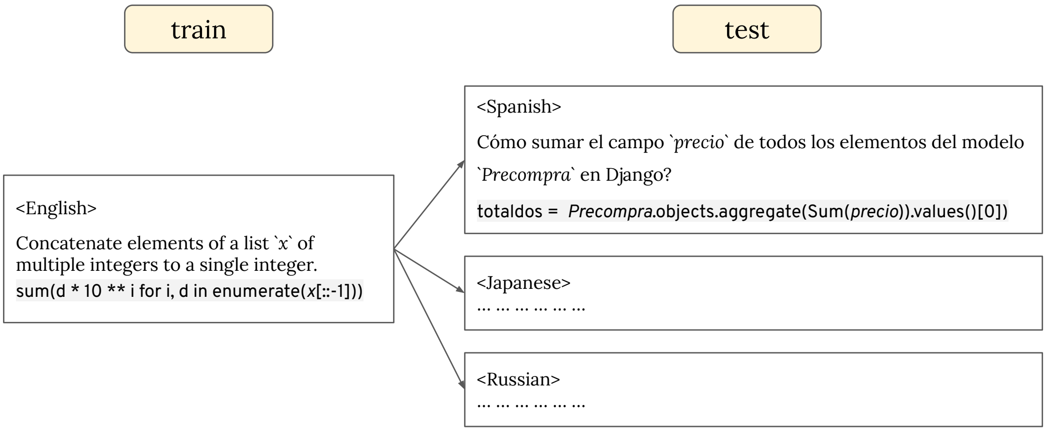

Figure 3: Example usage on the original English and Multilingual samples in the *zero-shot* setting.

inal snippet to create new samples. Because it is infeasible to manually translate millions of training examples, we use an existing multilingual neural machine translation (NMT) model to perform this translation. We benchmarked several open-source models, as elaborated in § 4.2, and eventually settled on the FLORES-101 (Goyal et al., 2021) model as our base model for experiments, although room for improvement still exists.

We train and test with each target language independently. Take the Spanish set in Fig. 4 for an example, we train the model with translated (from English to Spanish) intent-snippet pairs, then test it on our annotated MCoNaLa Spanish samples.

Translate-Test: MCoNaLa Test Samples Translated into English Instead of translating the training data, we could also translate the test samples in target languages to English. This is denoted as the *translate-test* setting. As shown in Fig. 5, models only need to be trained once on the English CoNaLa samples, and the resulting model can be readily used to synthesize from English intents translated from various target languages. Similar to *translate-train*, we also use FLORES-101 to translate by default. This setting adopts English as an intermediate natural language for code generation, and machine translation as a means to bridge multiple NLs. Although this is feasible, its performance is bounded by the English NL-to-Code ability and is affected by the machine translation quality.

3.3 Baseline Models

With the above mentioned settings, we select three models to form baseline evaluation.

For the first two models, we use two state-of-the-art code generation models on English CoNaLa dataset: TranX (Yin and Neubig, 2018) and TAE (Norouzi et al., 2021).

TranX is a BiLSTM-based encoder-decoder that can map NL intents into formal meaning representations such as Python code snippets. Because TranX operates in monolingual tasks, we evaluate it on *translate-train* and *translate-test* settings, where NL intents during training and testing reads in the same language.

TAE is a generic transformer-based seq2seq model for code synthesis. It exploits mass monolingual programming data using a Target AutoEncoding (TAE) objective and achieves superior performance on the English CoNaLa benchmark. However, it is built with (English-)BERT and originally intended for English scenarios, so we only test in the *translate-train* setting.

For another, **mBART** (Liu et al., 2020) is a pre-trained seq2seq model that could encode 25 natural languages, including the three target languages in our MCoNaLa dataset. Therefore, in addition to *translate-train* and *translate-test*, mBART is also capable of handling the *zero-shot* settings where it only sees the English data during training. It can then be directly test on the data in various target languages. This setting has previously been investigated in the context of domain-specific semantic parsing by Procopio et al. (2021).

We encourage the readers to refer to the original papers for the details of each model.

3.4 Experimental Details

Throughout the experiments, our training samples are based on the English CoNaLa samples, essentially the 2k annotated training samples and 600k samples from automatic mining and API documen-

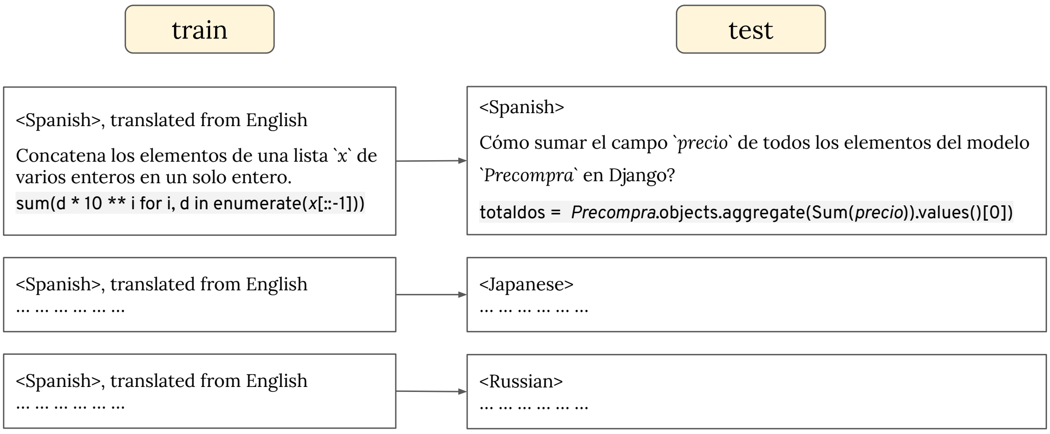

Figure 4: Example usage on the translated English and original Multilingual samples in the *translate-train* setting.

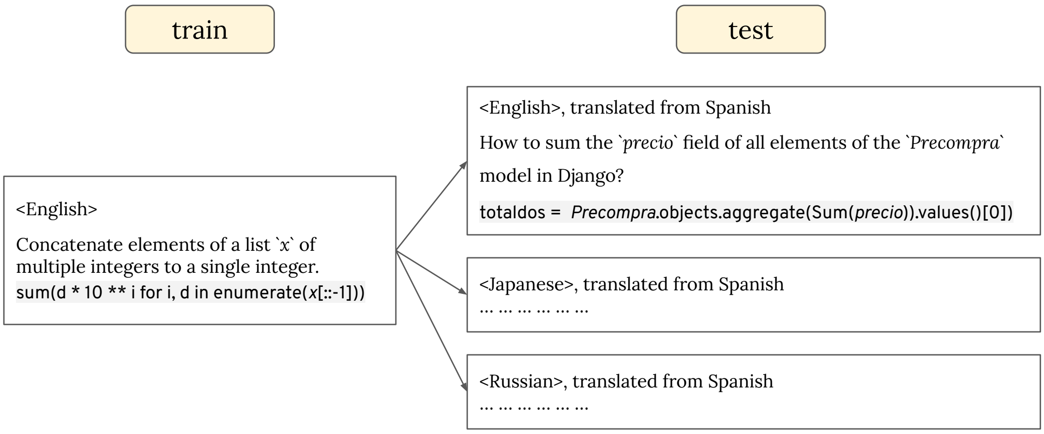

Figure 5: Example usage on the original English and translated multilingual samples in the *translate-test* setting.

tation. In different experimental settings, we train models that encode intents written in various natural languages (automatically translated when not in English) yet seek to generate the same code snippets written in Python. We split out a small subset of training samples and use it only for training-time validation, in which the number of samples is equal to the corresponding test set.

If not specified otherwise, we report the BLEU-4 score on the testing samples.

3.5 Experimental Result

We experiment with the three model baselines in their available settings. Tab. 1 shows the baseline performance under various train-test settings. While three baseline models perform roughly the same on Spanish samples, TAE performs significantly better on Japanese and Russian samples.

First, compared to the BLEU score of 33.41 achieved by TAE on English NL-to-Code generation, the overall performance on multilingual examples is relatively low, revealing the difficulty of the multilingual code generation task.

Model	**Setting**	**Language**			
		Spanish	**Japanese**	**Russian**	**avg.**
mBART	translate-test	2.38	3.08	2.04	2.50
	translate-train	**2.62**	3.51	2.65	2.92
	zero-shot	2.49	1.81	2.30	2.20
TranX	translate-test	2.46	8.41	8.09	6.32
	translate-train	2.44	6.09	6.01	4.85
TAE	translate-test	2.39	**9.88**	**9.57**	**7.28**

Table 1: BLEU score on NL-to-Code baseline models in different train-test settings.

Second, two models designed for code generation (TAE and TranX) are in general better than mBART, and TAE is comparatively stronger. TAE and TranX are either implanted with inductive bias or trained on nl-code parallel data, therefore possessing better abilities to generate structured code. However, mBART results are lower and demonstrate few variations across settings or languages. Because mBART is intended more for natural rather than programming languages, it can

Language	Size	# Snippet Tokens		
		average	min	max
Spanish	341	42.6	343	4
Japanese	210	17.7	94	2
Russian	345	32.0	243	3

Table 2: Comparing the basic statistics between data samples that have intents written in different natural languages. Statistics including the size of dataset and minimum/maximum/average lengths of code snippets.

be inferior when used in a distant domain about programming languages.

Third, results across languages show consistent distributions, especially for the two code-specific models (TAE and TranX). In both *translate-train* and *translate-test* settings, scores are comparable on Japanese and Russian examples, but lower on Spanish ones. As we will discuss in § 4.1, this is possibly due to the distributional gap between languages signified by varied sample complexity and specification.

Lastly, comparing between the two settings, the *translate-test* strategy results better than *translate-train*. We conjecture that one reason is the degraded data quality introduced by this intermediate auto-translation. As we will discuss in § 4.2, the potentially inaccurate translation of NL intents gives rise to unmatched intents and snippets, rendering it even hard to generate correctly in the low-resource scenario.

4 Result Analysis

In this section, we analyze the results from a number of angles.

4.1 Variations between Languages

In this section, we examine differences in both the underlying dataset statistics and model performance across multiple languages.

Basic Statistics Tab. 2 shows the basic statistics of each language in our MCoNaLa dataset, including their sample sizes, and the snippet lengths that is measured by the number of tokens.

First, the length of code snippets in each language varies. The average length of the snippet in Spanish subset is around 2.5 times of that in Japanese subset Since a longer code snippet is presumably more complex, this length measurement can (partially) indicates the sample complexity in each language. We conjecture that the lower performance of Spanish data is mainly due to its greater complexity, compared to the other two languages.

Second, although the sizes of three language sets are in general small, the Japanese version has fewer samples than the Spanish and Russian sets. The reason behind this difference traces back to the data collection step, where samples are collected from different StackOverflow forums in respective language versions to maintain the naturalness of the questions. Compared to Spanish or Russian forums, the Japanese forum is less active, yielding a smaller number of valid annotations.

Mutual Coverage of Variable Names Besides sample complexity, we hypothesize that varied degrees of alignment of languages sets can also lead to different performance. To this end, we examine the degree of mutual specification of intent-snippet pairs, presumably on the coverage of variable names in both directions. According to the annotation guideline that variable names should be specifically quoted, we can automatically parse out all variable names mentioned in intents and check their coverage in corresponding snippets. In the reverse direction, however, it can be hard to accurately enumerate the variable names in the code snippets especially with little context. We therefore manually examine 20 random samples in each language subset and approximate the coverage rate.

Language	Coverage of Variable Names	
	intent-to-snippet	snippet-to-intent
Spanish	94.4	14/20
Japanese	97.9	18/20
Russian	93.8	18/20

Table 3: Coverage of variable name mentions in three language subsets. **intent-to-snippet** automatically measures the percentage of variables names stated in the intents are mentioned in the snippets, across all samples. **snippet-to-intent** approximates the percentage of variable names appeared in the snippets to be covered by the corresponding intents.

Tab. 3 lists the coverage rates in both directions. In all three target languages, most variables specified in the intent are covered by the snippet. However, compared to Japanese and Russian in which most samples have intents covering all variables mentioned in the snippet, Spanish code snippets contain rich contexts and has a relatively lower coverage rate. This lack of variable alignment may lead to certain difficulty during code generation.

4.2 Influence of Intent Auto-translation

Next, we conjecture that one key bottleneck on multilingual code generation is the quality of the translated NL intents. Both *translate-train* and *translate-test* setting require translations to fill in the gap between different training and testing languages. Due to the high cost of manual translation, we alternatively adopt the NMT models and perform automatic translations. However, because of a lack of machine translation data in the programming domain, we are only able to take the out-of-box NMT models without further domain adaptations. It is challenging for current MT models to perfectly translate these intents for code synthesis, such that certain degradation in intent-snippet alignment can be introduced by intent auto-translation.

Comparison of Machine Translation Models In addition, we compare three publicly available machine translation models as representatives of the state-of-the-art in NL intent translation: FLORES-101 (Goyal et al., 2021), OPUS-MT (Tiedemann and Thottingal, 2020), and M2M (Fan et al., 2021). Comparing translation quality in the *translate-train* setting requires intensive re-training with translations produced by different models, especially when there are compounding factors such as the training strategy that are challenging to isolated. We hence ablate in the more direct *translate-test* setting, where a single model is trained in English CoNaLa samples, and test intents in different target langauges are translated into English. For comparison, we translate the test intents with different MT models for evaluation.

Tab. 4 shows the BLEU scores tested on translations of different MT models given different baseline models. While the three MT models perform roughly comparable, FLORES-101 tends to be more stable across both languages and baseline models.

As we will show in the subsequent manual examination, despite its relative superiority among three MT model candidates, quality of the resulting translations may still lag behind that of human annotations.

Quality of Auto-Translation To have an intuitive grasp of the quality of translated NL intents, we perform a manual check in the degree of alignment between the translated and original intents (in

Baseline	MT	Language		
		Spanish	Japanese	Russian
mBART	flores-101	2.38	3.08	2.04
	OPUS-MT	2.28	3.21	2.46
	m2m	1.83	2.79	2.00
TranX	flores-101	2.46	8.41	8.09
	OPUS-MT	2.46	5.09	5.00
	m2m	2.04	7.38	8.48
TAE	flores-101	2.39	9.88	9.57
	OPUS-MT	3.15	3.89	5.30
	m2m	2.21	8.20	9.32

Table 4: Ablating different machine translation methods to the performance of baseline models and target natural languages.

the context of the paired code snippet), with the assistance of the Google Translate API and dictionaries. Concretely, in the *translate-train* setting, we randomly select 20 samples in the English CoNaLa dataset and check if the intent-snippet alignment is preserved after translating the intents into the three target languages. Similarly for the *translate-test* setting, we sample 20 NL-code pairs from each of the three languages and compare them with the translated English intents.

As shown in Fig. 6, the NMT model often translates the important words incorrectly, sometimes even omitting these words. This loss is especially severe on verbs that strongly indicate certain operations. As a result, the translation process may detach the alignment between intents and snippets, being one of the major factors to the overall poor baseline performance.

5 Related Work

Natural Language to Code Generation Datasets There have been several benchmark datasets for NL-to-Code generation, such as Hearthstone (Ling et al., 2016), Django (Oda et al., 2015), CONCODE (Iyer et al., 2018), and CoNaLa (Yin et al., 2018a). Other examples include datasets more towards problem solving, such as HumanEval (Chen et al., 2021), MBPP (Austin et al., 2021), and APPS (Hendrycks et al., 2021). A number of methods have been proposed to mine intent-snippet pairs for the purpose of code search, summarization, or generation. While our work falls in the line of mining from SO (Wong et al., 2013; Iyer et al., 2016; Yao et al., 2018; Yin et al., 2018b), other work also attempts to exploit other data sources

original intent (English)	**Prepend** string '*hello*' to all items in list '*a*'
translated intent (Spanish)	**Preparación** (prepare) de la cadena '*hello*' a todos los elementos en la lista `a`
snippet	['*hello*{0}'.format(i) for i in *a*]
original intent (English)	add a colorbar **to plot `plt`** using image `im` on axes `ax`
translated intent (Japanese)	画像`im`を使って`ax`の軸にカラーバーを追加
snippet	*plt*.colorbar(*im*, ax=*ax*)
original intent (English)	extend dictionary `a` with key/**value** pairs of dictionary `b`
translated intent (Russian)	расширить словарь `a` с ключевыми/**значительными** (significant) парами словаря `b`
snippet	*a*.update(*b*)

Figure 6: Examples showing that the translation errors or omits critical words in the original intent.

such as API documentation (Chatterjee et al., 2009; Movshovitz-Attias and Cohen, 2013; Xu et al., 2020), code comments (Wong et al., 2015), specialized sites (Quirk et al., 2015), and developer communications (Panichella et al., 2012). One prior methodology to automatically collect large-scale parallel data is using heuristics to extract intent-snippet pairs (Chatterjee et al., 2009; Wong et al., 2013; Zagalsky et al., 2012), but this often results in compromised data quality (Xu et al., 2020). Our work resorts to a manual-annotation strategy that often yields accurately aligned intent-snippet pairs.

Multilingual Learning While the bulk of code-related tasks has their natural language components in English, program developers native in other languages cannot enjoy the advances in code intelligence techniques, leading to the current lacunae in multilingual learning.Our work intends to mitigate this gap by facilitating NL-to-Code generation in multiple languages beyond English. To enable language understanding across multiple languages, a number of works propose to train language models with corpus in multiple languages (Devlin, 2018; Liu et al., 2020; Conneau et al., 2020; Xue et al., 2021). In addition to multilingual training, other data augmentation techniques commonly used in machine translation (MT), such as back-translation (Edunov et al., 2018), monolingual (Sennrich et al., 2016; Siddhant et al., 2020) or generalized data augmentation (Xia et al., 2019), also inspired our experiment settings. However, these techniques have rarely been utilized for NL-conditioned code generation, and we present preliminary attempts in the experiments.

6 Conclusion

In this work, we extend the task of NL-to-Code generation from an English-centric setting to multilingual scenarios. We present the MCoNaLa data benchmark, that involves NL intent and code snippet pairs available in Spanish, Japanese, and Russian. Our benchmark serves for the task of multilingual code generation, requiring the baseline models of multilingual understanding and code synthesis. We conduct systematic experiments on three state-of-the-art baseline models and demonstrate varying difficulty across languages and settings. Coupled with the quantitative result analysis and qualitative dataset analysis, we hope to reveal the necessity to develop, and serve as a solid testbed for language-comprehensive approaches regarding code intelligence.

Acknowledgements

We thank all the annotators for the contributions. This work was supported by the National Science Foundation under grant number 1815287.

References

Wasi Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. 2020. A transformer-based approach for source code summarization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4998–5007.

Miltiadis Allamanis, Hao Peng, and Charles Sutton. 2016. A convolutional attention network for extreme summarization of source code. In *International conference on machine learning*, pages 2091–2100. PMLR.

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

Shaunak Chatterjee, Sudeep Juvekar, and Koushik Sen. 2009. Sniff: A search engine for java using free-form queries. In *International Conference on Fundamental Approaches to Software Engineering*, pages 385–400. Springer.

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

Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Édouard 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*, pages 8440–8451.

Jacob Devlin. 2018. Multilingual bert readme.

Sergey Edunov, Myle Ott, Michael Auli, and David Grangier. 2018. Understanding back-translation at scale. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 489–500.

Angela Fan, Shruti Bhosale, Holger Schwenk, Zhiyi Ma, Ahmed El-Kishky, Siddharth Goyal, Mandeep Baines, Onur Celebi, Guillaume Wenzek, Vishrav Chaudhary, et al. 2021. Beyond english-centric multilingual machine translation. *Journal of Machine Learning Research*, 22(107):1–48.

Naman Goyal, Cynthia Gao, Vishrav Chaudhary, Peng-Jen Chen, Guillaume Wenzek, Da Ju, Sanjana Krishnan, Marc'Aurelio Ranzato, Francisco Guzmán, and Angela Fan. 2021. The flores-101 evaluation benchmark for low-resource and multilingual machine translation.

Sven Hartrumpf, Ingo Glöckner, and Johannes Leveling. 2008. Efficient question answering with question decomposition and multiple answer streams. In *Workshop of the Cross-Language Evaluation Forum for European Languages*, pages 421–428. Springer.

Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, et al. 2021. Measuring coding challenge competence with apps. *arXiv preprint arXiv:2105.09938*.

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.

Xing Hu, Ge Li, Xin Xia, David Lo, Shuai Lu, and Zhi Jin. 2018. Summarizing source code with transferred api knowledge.

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. 2016. Summarizing source code using a neural attention model. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2073–2083.

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. 2018. Mapping language to code in programmatic context. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1643–1652.

Chuan-Jie Lin Yu-Min Kuo. Description of the ntou complex qa system.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, et al. 2019. Natural questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:452–466.

Haoran Li, Abhinav Arora, Shuohui Chen, Anchit Gupta, Sonal Gupta, and Yashar Mehdad. 2020. Mtop: A comprehensive multilingual task-oriented semantic parsing benchmark. *arXiv preprint arXiv:2008.09335*.

Qingyuan Liang, Zeyu Sun, Qihao Zhu, Wenjie Zhang, Lian Yu, Yingfei Xiong, and Lu Zhang. 2021. Lyra: A benchmark for turducken-style code generation. *arXiv preprint arXiv:2108.12144*.

Wang Ling, Phil Blunsom, Edward Grefenstette, Karl Moritz Hermann, Tomáš Kočiský, Fumin Wang, and Andrew Senior. 2016. Latent predictor networks for code generation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 599–609.

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.

Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin Clement, Dawn Drain, Daxin Jiang, Duyu Tang, et al. 2021. Codexglue: A machine learning benchmark dataset for code understanding and generation. *arXiv preprint arXiv:2102.04664*.

Mehrad Moradshahi, Giovanni Campagna, Sina J Semnani, Silei Xu, and Monica S Lam. 2020. Localizing open-ontology qa semantic parsers in a day using machine translation. *arXiv preprint arXiv:2010.05106*.

Dana Movshovitz-Attias and William Cohen. 2013. Natural language models for predicting programming comments. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 35–40.

Sajad Norouzi, Keyi Tang, and Yanshuai Cao. 2021. Code generation from natural language with less prior knowledge and more monolingual data. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 776–785.

Yusuke Oda, Hiroyuki Fudaba, Graham Neubig, Hideaki Hata, Sakriani Sakti, Tomoki Toda, and Satoshi Nakamura. 2015. Learning to generate pseudo-code from source code using statistical machine translation. In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 574–584. IEEE.

Sebastiano Panichella, Jairo Aponte, Massimiliano Di Penta, Andrian Marcus, and Gerardo Canfora. 2012. Mining source code descriptions from developer communications. In *2012 20th IEEE International Conference on Program Comprehension (ICPC)*, pages 63–72. IEEE.

Luigi Procopio, Rocco Tripodi, and Roberto Navigli. 2021. SGL: Speaking the graph languages of semantic parsing via multilingual translation. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 325–337, Online. Association for Computational Linguistics.

Chris Quirk, Raymond Mooney, and Michel Galley. 2015. Language to code: Learning semantic parsers for if-this-then-that recipes. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 878–888.

Maxim Rabinovich, Mitchell Stern, and Dan Klein. 2017. Abstract syntax networks for code generation and semantic parsing. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1139–1149.

Sebastian Ruder. 2021. Multi-domain multilingual question answering.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Improving neural machine translation models with monolingual data. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 86–96.

Tom Sherborne and Mirella Lapata. 2021. Zero-shot cross-lingual semantic parsing. *arXiv preprint arXiv:2104.07554*.

Tom Sherborne, Yumo Xu, and Mirella Lapata. 2020. Bootstrapping a crosslingual semantic parser. *arXiv preprint arXiv:2004.02585*.

Peng Shi, Rui Zhang, He Bai, and Jimmy Lin. 2021. Cross-lingual training with dense retrieval for document retrieval. *arXiv preprint arXiv:2109.01628*.

Hideki Shima and Teruko Mitamura. Bootstrap pattern learning for open-domain clqa.

Aditya Siddhant, Ankur Bapna, Yuan Cao, Orhan Firat, Mia Xu Chen, Sneha Kudugunta, Naveen Arivazhagan, and Yonghui Wu. 2020. Leveraging monolingual data with self-supervision for multilingual neural machine translation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2827–2835.

Jörg Tiedemann and Santhosh Thottingal. 2020. OPUS-MT — Building open translation services for the World. In *Proceedings of the 22nd Annual Conferenec of the European Association for Machine Translation (EAMT)*, Lisbon, Portugal.

Yue Wang, Weishi Wang, Shafiq Joty, and Steven CH 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*, pages 8696–8708.

Edmund Wong, Taiyue Liu, and Lin Tan. 2015. Clocom: Mining existing source code for automatic comment generation. In *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pages 380–389. IEEE.

Edmund Wong, Jinqiu Yang, and Lin Tan. 2013. Autocomment: Mining question and answer sites for automatic comment generation. In *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 562–567. IEEE.

Mengzhou Xia, Xiang Kong, Antonios Anastasopoulos, and Graham Neubig. 2019. Generalized data augmentation for low-resource translation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5786–5796.

Frank F Xu, Zhengbao Jiang, Pengcheng Yin, Bogdan Vasilescu, and Graham Neubig. 2020. Incorporating external knowledge through pre-training for natural language to code generation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6045–6052.

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*, pages 483–498.

Ziyu Yao, Daniel S Weld, Wei-Peng Chen, and Huan Sun. 2018. Staqc: A systematically mined question-code dataset from stack overflow. In *Proceedings of the 2018 World Wide Web Conference*, pages 1693–1703.

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

Pengcheng Yin, Bowen Deng, Edgar Chen, Bogdan Vasilescu, and Graham Neubig. 2018b. Learning to mine aligned code and natural language pairs from stack overflow. In *2018 IEEE/ACM 15th international conference on mining software repositories (MSR)*, pages 476–486. IEEE.

Pengcheng Yin and Graham Neubig. 2018. Tranx: A transition-based neural abstract syntax parser for semantic parsing and code generation. *arXiv preprint arXiv:1810.02720*.

Alexey Zagalsky, Ohad Barzilay, and Amiram Yehudai. 2012. Example overflow: Using social media for code recommendation. In *2012 Third International Workshop on Recommendation Systems for Software Engineering (RSSE)*, pages 38–42. IEEE.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*.