MCoNaLa: A Benchmark for Code Generation from Multiple Natural Languages

Zhiruo Wang∗♠ Grace Cuenca∗♦ Shuyan Zhou♠ Frank F. Xu♣ Graham Neubig∗♠
*Carnegie Mellon University ♦Princeton University ♠Inspired Cognition
{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 systematic evaluation on MCoNaLa by testing state-of-the-art code generation systems. Although the difficulties vary across three languages, all systems lag significantly behind their English counterparts, revealing the challenges in adapting code generation to new languages.1

1 Introduction

There are an increasing number of applications related to “code intelligence”, 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), accompanied by 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 of code (Oda et al., 2015; Liang et al., 2021), or contain NL intents collected via automatic translation (Li et al., 2021) (Appendix A). However, similarly to how Kwiatkowski et al. (2019) argue that “natural questions” are necessary to appropriately benchmark QA systems, we argue that ensuring the naturalness and coverage of questions is essential for benchmarking code generation systems as well.

A dataset for English code generation based on natural programming questions is the CoNaLa dataset (Yin et al., 2018a). It is based on natural developer questions harvested from the Stack Overflow (SO) question answering forum. In fact, in addition to 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 345 manually curated parallel samples with natural intents in Spanish, Japanese, and Russian, along with corresponding Python code snippets. Like CoNaLa, these snippets are collected from language-specific SO sites and annotated by na-

Figure 1: Examples in the MCoNaLa dataset, that aim to generate general-purpose Python code snippets from source intent of multiple natural languages.
tive speakers who are also proficient in the Python programming language.

To provide insights into the state of code generation on this new resource, we conduct comprehensive experiments with three state-of-the-art text generation models in the context of cross-lingual transfer, by unifying training and testing NL via translation (Ruder and Sil, 2021; Shi et al., 2021; Shima and Mitamura, 2010; Hartrumpf et al., 2008), or utilizing a multilingual NL encoder such as MBART (Liu et al., 2020). Our results suggest that cross-lingual NL-to-Code generation is challenging. Among all languages and experiment settings, the highest average BLEU score is 7.28, far behind that of English, which achieves 33.41, presumably because English resembles Python more than other NLs. In addition, we find models with task-specific modules and training outperform generic seq2seq models, yet the discrepancy between languages is consistent across all baseline models. In all, our corpus and experiments demonstrate the varied difficulty of the NL-to-Code generation task under different languages, emphasizing the need to develop a language-comprehensive approach to code intelligence.

2 The MCoNaLa Dataset

2.1 Task Definition

Concerning the task of answering natural language questions with machine-executable programs, our focus is to build a benchmark dataset to evaluate models for their ability to encode NL intents in multiple languages and generate code snippets. For each example in Figure 1, the intent above asks how to achieve a particular goal, and the snippet below responds with a piece of Python code.

2.2 Annotation Workflow

Our goal is to collect intent-snippet parallel data in multiple natural languages. In this section, we outline the main workflow for data annotation: (1) language source selection, (2) valid SO post identification, and (3) parallel sample annotation.

Language source and selection  Besides the English version, Stack Overflow also has forums available in four other languages: Spanish, Portuguese, Japanese, and Russian. Data annotation in each language requires a native speaker of that language, who should also be proficient in both English and Python. Due to the high cost and difficulty of hiring reliable annotators with such a specialized skill set, we only employ one Upwork annotator for each of Spanish, Japanese, and Russian. From the official SO data dump2 dated March 2021, we obtained all posts in these languages. However, we were unsuccessful in finding a Portuguese-speaking annotator at the time of corpus collection.

Identifying how-to questions  Following Yin et al. (2018a), annotators are first asked to identify valid posts that contain how-to type questions, which are imperative utterances seeking particular goals achievable by code. They are often in the post title or description, such as the example in Figure 2.

Posts are sent in 100-sample batches, and then categorized by annotators. To improve annotation efficiency, we bootstrapped a MBART how-to question classifier, with English examples, then iteratively multilingual samples. It achieves an accuracy of 72.50%. We then automatically filter the probable invalid posts using this classifier and designate the rest for manual annotation. We collect all valid posts and extract questions as raw intents, for subsequent parallel data annotation.

Collecting intent-snippet pairs  For each post, we ask the annotators to find at most three snippets of Python code that correctly answer the extracted question. However, questions from post title or description are often ambiguous, especially in respective context of answer snippet, such as the example in Figure 2, that the question does not specify the names of “data” and “list” variables to allow precise code implementation. To disambiguate the intent and align it with a snippet, we ask annotators to rewrite the intent by: (1) specifying variable names appearing in the answer snippet, and (2) clarifying commands with reference question descriptions. Concretely, variable names and data types in the rewritten intent

2https://archive.org/details/stackexchange
Concatenate elements of a list `x` of multiple integers to a single integer.
```
sum(d * 10 ** i for i, d in enumerate(x[::-1]))
```

To provide insights about evaluating on MCoNaLa, we use existing multilingual machine translation (MMT) models to automate translation. We benchmarked several open-source options, as elaborated in § 4.2, and settled on the M2M-124 model used on the FLORES-101 dataset (Goyal et al., 2022). Because it is not feasible to manually translate 600k+ intents, we use existing multilingual machine translation (MMT) models to automate translation. We benchmarked several open-source options, as elaborated in § 4.2, and settled on the M2M-124 model used on the FLORES-101 dataset (Goyal et al., 2022).

Also, we can train models on English samples and directly evaluate on MCoNaLa samples in target languages zero-shot, requiring models to encode multiple NLs, further, transfer the code generation ability from English context to target ones.

3.2 Baseline Models

We introduce three baseline methods targeting the above train-test settings. We encourage readers to refer to the original papers for more details.

In a monolingual context, models should function in target languages for translate-train and En-
glish for translate-test. TRANX (Yin and Neubig, 2018) is a BiLSTM-based encoder-decoder model that uses a transition-based abstract syntax parser to map NLs into formal meaning representations (MR) such as Python programs. It is agnostic to input languages and hence can be evaluated on both translated settings. TAE (Norozi et al., 2021) is the state-of-the-art method on CoNaLa by training a generic transformer with an added target autocoder (TAE) objective. However, it is built with (English-)BERT and is intended for English scenarios, therefore only tested on translate-test.

As is required by zero-shot evaluation, we adopt a multilingual model, BART (Liu et al., 2020), which is a seq2seq model pre-trained on 25 natural languages including our target ones. Note that MBART can also function in monolingual contexts, for both translate-train and translate-test settings.

3.3 Experiment

We train baseline models in their available settings, then tokenize the generated and reference code snippets following Yin and Neubig (2018) to evaluate the BLEU-4 scores. We report the average scores of five rounds using different random seeds.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Language</th>
<th>mbart</th>
<th>tranX</th>
<th>tae</th>
<th>tie</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>translate-test</td>
<td>es</td>
<td>0.532</td>
<td>0.402</td>
<td>0.066</td>
<td>0.468</td>
<td></td>
</tr>
<tr>
<td>translate-train</td>
<td>en</td>
<td>0.522</td>
<td>0.396</td>
<td>0.102</td>
<td>0.478</td>
<td></td>
</tr>
<tr>
<td>translate-test</td>
<td>ja</td>
<td>0.000</td>
<td>1.000</td>
<td>0.000</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>translate-train</td>
<td>ja</td>
<td>0.000</td>
<td>1.000</td>
<td>0.000</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>translate-test</td>
<td>ru</td>
<td>0.000</td>
<td>1.000</td>
<td>0.000</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>translate-train</td>
<td>ru</td>
<td>0.000</td>
<td>1.000</td>
<td>0.000</td>
<td>0.000</td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Significance testing results between each pair of baseline models. '-' marks the model not in the pair.

In both translate-test and translate-train settings of Table 2, code-specific systems (TRANX and TAE) significantly outperform MBART on Japanese and Russian. However, no significant differences are shown in Spanish, as expected given its relative difficulty. With significance testing, one can obtain reliable results even on a small dataset. While small sizes are not entirely desirable for informative evaluation, we view them as practical reflections of data scarcity, supporting our call for more non-English resources.

4 Analysis

4.1 Variations between Languages

We first study the differences in size and snippet length between languages subsets in MCoNaLa. As listed in Table 3, snippet lengths vary across languages, and the average snippet length in Spanish is around 2.5/1.3 times of that in Japanese/Russian. A longer snippet is presumably more complex, suggesting that snippets in Spanish samples are harder to generate, and hence models perform poorer.

4.2 Intent Auto-translation

In § 3.1 we use MMT models for intent translation. To optimize translation quality, we compare three best performing MMT models: OPUS-MT (Tiedemann and Thottingal, 2020), M2M-
Prepend string ‘hello’ to all items in list ‘a’

Preparación (prepare) de la cadena ‘hello’ a todos los elementos en la lista ‘a’

Add a colorbar to plot ‘plt’ using image ‘im’ on axes ‘ax’

 affidavit

Add a colorbar to plot ‘plt’ using image ‘im’ on axes ‘ax’

extend dictionary ‘a’ with key/value pairs of dictionary ‘b’

Russian

Table 3: Data size and snippet length (in number of tokens) of MCoNaLa samples between target languages.

Table 4: Comparing MMT models under translate-test.

4.3 Quality of Auto-translation

To better measure the quality of translated intents, we manually check the semantic alignment between the original and translated intents, with the assistance of the Google Translate API and dictionaries. Concretely, we take 20 English CoNaLa intents and check if their semantics preserve after being translated into three target languages (translate-train). We similarly examine 20 MCoNaLa intents in each target language and check their English translations (translate-test). We use the M2M-124 translations given its best results. As shown in Figure 4, MMT translations are still sub-optimal: often mis-translate, even omit, the key words. This is especially severe on verbs that indicate certain Python operations. Hence, the translation step may impair intent-snippet alignment, being one of the major factors to the poor results in translated settings.

5 Conclusion

In this work, we extend the task of NL-to-Code generation from English-centric to multilingual scenarios. We establish the MCoNaLa benchmark that contains NL intent and code snippet pairs available in Spanish, Japanese, and Russian. Our benchmark serves for the multilingual code generation task, requiring models of both multilingual understanding and code synthesis. We conduct systematic experiments on three baseline models and show varying difficulty across languages and settings. We hope to reveal the necessity to develop, and serve as a solid test bed for language-comprehensive approaches regarding code intelligence.

Acknowledgements

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

Although the MCoNaLa dataset makes a first step to include more natural languages aside from English, it is currently limited to the languages supported by the StackOverflow forum, since SO provides the source data for the MCoNaLa creation. This can be mitigated by extending to more languages using programming forums in other languages that have a similar purpose to SO. Besides, MCoNaLa dataset only supports literal evaluation methods such as BLEU. Given the executable nature of Python programs, it is beneficial to support more evaluation metrics such as functional correctness, robustness, and conciseness.

Ethics Statement

The MCoNaLa dataset is built to serve as a testbed for evaluating code generation systems from natural languages extending beyond English, given that an English-centric setting can harm universal accessibility to language technologies.

We hire annotators who are proficient in target languages and assist them with clearly documented instructions, flexible annotation interfaces (e.g., Google Sheets), and automated methods (e.g., using a neural classifier to filter out possibly invalid cases) to optimize the annotation efficiency. We carefully check in line with our instructions and standards, to ensure the quality of both the question posts given and the annotation results back from our annotators. We emphasize the differences between samples in different languages, because they are natural reflections of the questions that programmers asked in each specific language, similar to many works in fields such as multilingual question answering (Clark et al., 2020) and named entity recognition (Nothman et al., 2013). We reckon that it is of paramount importance to evaluate on data that was originally produced in the target language, and results may be less reliable otherwise.

Nevertheless, with the advances in models capable of generating code from natural language inputs, we should be aware of the potentially harmful usage such as concealing malicious code (Wallace et al., 2020), or generating code with security vulnerabilities (Verdi et al., 2020; Pearce et al., 2021).

References


Sergey Edunov, Myle Ott, Michael Auli, and David Grangier. 2018. Understanding back-translation at


Mengzhou Xia, Xiang Kong, Antonios Anastasopoulos, and Graham Neubig. 2019. Generalized data


A 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), CODE (Iyer et al., 2018), and CoNaLa (Yin et al., 2018a). Other examples include datasets for 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 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 have their NL 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 experiments. However, these techniques have rarely been utilized for NL-conditioned code generation. We present preliminary attempts in the experiments.