RoCode: A Dataset for Measuring Code Intelligence from Problem Definitions in 🇷🇴 Romanian 🇷🇴

Adrian Cosma[1], **Bogdan Iordache**[2], **Paolo Rosso**[3,4]
[1]University POLITEHNICA of Bucharest
[2] University of Bucharest, HTL Research Center
[3] PRHLT Research Center, Universitat Politècnica de València
[4] ValgrAI - Valencian Graduate School and Research Network of Artificial Intelligence
[1,2]Bucharest, Romania, [3,4]València, Spain
ioan_adrian.cosma@upb.ro, iordache.bogdan1998@gmail.com, prosso@dsic.upv.es

Abstract

Recently, large language models (LLMs) have become increasingly powerful and have become capable of solving a plethora of tasks through proper instructions in natural language. However, the vast majority of testing suites assume that the instructions are written in English, the de facto prompting language. Code intelligence and problem solving still remain a difficult task, even for the most advanced LLMs. Currently, there are no datasets to measure the generalization power for code-generation models in a language other than English. In this work, we present RoCode, a competitive programming dataset, consisting of 2,642 problems written in Romanian, 11k solutions in C, C++ and Python and comprehensive testing suites for each problem. The purpose of RoCode is to provide a benchmark for evaluating the code intelligence of language models trained on Romanian / multilingual text as well as a fine-tuning set for pretrained Romanian models. Through our results and review of related works, we argue for the need to develop code models for languages other than English.

Keywords: code intelligence, language models, dataset, code-switching, Romanian

1. Introduction

Since the development of large language models (LLMs) (Brown et al., 2020; Hoffmann et al., 2022; Borgeaud et al., 2022; Chowdhery et al., 2022), few tasks have been left behind that cannot be reasonably tackled with proper prompting (Guo et al., 2023). Of particular interest in this area has been natural language to programming language (NL-PL) capabilities, in which models generate structured code with a precise intent (Replit, 2023; Luo et al., 2023; Rozière et al., 2023). LLMs have been particularly successful in this area, fueled by the massive amounts of available code on the internet (Kocetkov et al., 2022).

"Low-code" platforms, which enable users to develop software requiring less coding knowledge, require efficient interfacing between human operators and machine code. One of the most prominent tools in this direction is Github Copilot (Chen et al., 2021), a large language model trained to generate code based on natural language comments. In recent years, a wide array of methods that improve upon the state of the art of code execution have been proposed (Scholak et al., 2021; Christopoulou et al., 2022; Ni et al., 2023; Zhou et al., 2023), including prompt manipulation methods such as chain-of-thought (Wei et al., 2022) or similar approaches (Zhou et al., 2023) which appear to elicit reasoning capabilities. The most powerful commercial language model, GPT-4 (OpenAI, 2023), achieves great performance in a wide array of tasks, but its performance is still lacking in programming puzzles. While there is no information about the pretraining dataset composition, nor the composition of benchmarks, GPT-4's results on a benchmark dubbed "Leetcode" appears to be the worst performing, especially the hard subset, correctly solving only 3 out of the 45 problems. It is unclear how much the performance of GPT-4 on programming problems is due to a high degree of generalization, or due to data leakage from other websites such as LeetCode[1]. There are several existing datasets for semantic code search and competitive programming (Li et al., 2022; Chen et al., 2021; Husain et al., 2019; Iyer et al., 2018; Zavershynskyi et al., 2018; Kulal et al., 2019), but almost all of them have problem statements and comments written in English. Furthermore, out of the currently available open-sourced large language models (e.g., LLaMa (Touvron et al., 2023; Rozière et al., 2023)), the vastly predominant pretraining language is English.

By design, low-code systems promise the democratization of programming. In itself, coding is independent of the native language of the programmer. However, most NL-powered low-code platforms have a tacit requirement that the user is fluent in English.

Even through tremendous progress, multilingual

[1]https://leetcode.com/

LREC-COLING 2024, pages 14173–14185
20-25 May, 2024.

	CONCODE	NAPS	SPoC	APPS	**RoCode (ours)**
Programming Language	Java	UAST	C++	Python	**C / C++ / Python**
Test Cases	✗	✓	✓	✓	✓
Number of Programs	104,000	17,477	18,356	232,421	**11,250**
Lines per Program (Avg.)	26.3	21.7	14.7	18.0	**118.65**
Number of Exercices	104,000	2,231	677	10,000	**2,642**
Text Input	Docstrings	Pseudocode	Pseudocode	Problem Descriptions	**Problem Descriptions**

Table 1: Comparison with other existing NL-PL datasets. While RoCode has a comparable number of problems and solutions, its problem descriptions are formulated in native Romanian. Furthermore, solutions are written by Romanian students and can exhibit "code-code-switching".

systems (Wang et al., 2020; Scao et al., 2022) lag behind specialized models, especially in languages with fewer training resources available (Scao et al., 2022). For instance, RoBERT (Dumitrescu et al., 2020), the Romanian version of the popular BERT (Devlin et al., 2019) outperforms multilingual counterparts on Romanian tasks. Moreover, translation is imperfect due to the lexical gap between languages which makes some concepts to be difficult to translate directly and can induce a loss of meaning that might be crucial in certain high-stakes scenarios. The current state of the art for translating Romanian to English (Bhosale et al., 2020) has a reported 40.3 BLEU score, which is considered "understandable to good translation", leaving a lot to be desired in meaning-rich contexts such as code generation. Other methods such as NLLB (Team et al., 2022) provide higher quality translations, but are not open-source.

Moreover, code from non-English speaking countries often exhibits *code-code-switching*: programming syntax keywords are written in English, while comments and domain-specific attributes (i.e. variables, class names) are written in the native language. This phenomenon is, in fact, considered a best practice and falls in line with the idea of "ubiquitous language" (Evans, 2004): domain experts and developers need to share a single, common vocabulary such that the meaning is exact and not lost in translation.

Efforts to bring the power of natural language-powered systems to other languages apart from English are limited. For Romanian, only the RoGPT-2 (Niculescu et al., 2021), GPTNeo-Ro(Dumitrescu, 2022), and RoBERT (Dumitrescu et al., 2020) counterparts are available. These models achieved good performance on LiRo (Dumitrescu et al., 2021), the current Romanian benchmarking suite, compared to other similar multilingual models. However, there is no benchmark for evaluating code generation models for Romanian. Furthermore, small scale models (< 1B parameters) fare poorly on coding challenges (Hendrycks et al., 2021). Nevertheless, a feasible alternative is the construction of code understanding / code retrieval models adapted for Romanian, such as CodeBERT (Feng et al., 2020).

In this work, we propose RoCode, the first competitive programming dataset for measuring code intelligence for NL-PL models. RoCode consists of 2,642 problems written in Romanian under 3 difficulty levels, multiple associated solutions written in C / C++ and Python, alongside a set of test cases to evaluate the correctness and algorithmic complexity. RoCode attempts to bridge the gap between Romanian natural language and computer code. RoCode is the first dataset of competitive coding problems in a language different from English. Problems, solutions and test cases are made available through a collaboration with *infoarena.ro*, the most popular Romanian competitive programming platform. While problems and solutions are publicly available to be crawled, the test cases for each problem are not. We provide a filtered, curated and structured dataset, containing test cases for each problem, as well as an open-source environment to test generated solutions.

Compared to other existing datasets for competitive programming, such as APPS (Hendrycks et al., 2021), RoCode is similar in size and scope, while having its own particularities geared towards Romanian. RoCode has problem definitions written in Romanian, and solutions exhibit *code-code-switching*, creating a challenging set for fine-tuning monolingual models. In Table 1 we provide a comparison with other similar datasets (Iyer et al., 2018; Zavershynskyi et al., 2018; Kulal et al., 2019). Through RoCode, we aim to facilitate the development of NL-PL models in native Romanian, outperforming current multilingual models. RoCode aims to be a benchmark in neural code generation from Romanian prompts as well as a fine-tuning dataset for larger models.

This work makes the following contributions:

1. We propose RoCode, the first dataset for measuring code intelligence from problem definitions written in Romanian. We provide 2,642 problems under 3 difficulty levels, solutions in C / C++ and Python, and test cases for each problem. We release the dataset on *huggingface* for public use[2].

[2] huggingface.co/datasets/cosmadrian/rocode

2. We provide the first results on NL-PL code intelligence performance on small open-source models on a language different than English. We tested all the available Romanian language models (RoGPT-2 (Niculescu et al., 2021) and GPT-Neo-Ro (Dumitrescu, 2022)) and a set of open-source English models (Replit, 2023; Geng and Liu, 2023; Luo et al., 2023; Touvron et al., 2023). Unsurprisingly, none of the tested models are able to obtain a reasonable performance, proving that RoCode is a challenging dataset. We make our code publicly available on github[3].

3. At the same time, this work is also a position paper. Through our results and extensive review of related works, we argue for the development of multi-lingual and monolingual non-English code intelligence models and provide potential future research directions.

2. Related Work

2.1. Large Language Models for Code

Following the success of ChatGPT models to generate and understand code, multiple other very recent open-source alternatives have been proposed (Scao et al., 2022; Rozière et al., 2023; Fried et al., 2023; Luo et al., 2023; Li et al., 2023) that make use of publicly available data (e.g., The Stack dataset (Kocetkov et al., 2022)), as well as synthetically generated data (Wang et al., 2023). For instance, StarCoder (Li et al., 2023) was trained on publicly available data and obtained impressive results on code benchmarks, surpassing in some cases, proprietary models. WizardCoder (Luo et al., 2023) is a subsequent improvement through instruction fine-tuning of StarCoder with the Evol-Instruct method.

Notably, CodeLlama (Rozière et al., 2023), a LLaMa-derived (Touvron et al., 2023) model, has received increased attention due to its high quality, multiple model sizes and its permissive open-source language. CodeLLaMa is trained on 500B tokens of publicly available code. Several specialized variants are released, including a model tuned specifically for Python and an instruction-following model. The model is trained using infilling (Bavarian et al., 2022) (similar to InCoder (Fried et al., 2023)) and automatically generated instructions from a larger model using the self-instruct method (Wang et al., 2023).

However, almost all open-sourced LLMs are primarily geared towards English (only BLOOM-176B (Scao et al., 2022) is multilingual), and all of the code-focused LLMs are exclusively fine-tuned on code having comments and documentation written in English (Gao et al., 2021; Kocetkov et al., 2022).

2.2. Code Datasets

Large publicly available training datasets for code intelligence have made heavy use of codebases hosted on GitHub with permissive licenses (Gao et al., 2021; Kocetkov et al., 2022; Husain et al., 2019). More notably, the Pile (Gao et al., 2021) is an open 800GB dataset of text having a considerable fraction comprised of code in various languages. Similarly, the Stack (Kocetkov et al., 2022) is a 3TB dataset of code from 30 programming languages, used to train StarCoder (Li et al., 2023). Some works such as Codex (Chen et al., 2021) use GitHub to compile a dataset, but do not disclose the repository details or licensing.

For benchmarking problem solving capabilities of LLMs, one predominant dataset used across approaches is APPS (Hendrycks et al., 2021), a dataset of leetcode-style problems organised into difficulty ranges. The APPS dataset withstood the test of time and has proven to be a hard benchmark even for the most sophisticated models. Other benchmarking datasets have been proposed in the past (Iyer et al., 2018; Zavershynskyi et al., 2018; Kulal et al., 2019), but they are geared towards code generation from some text input (docstrings or pseudocode) rather than solving a specific programming problem.

3. RoCode: Romanian Competitive Programming

For the data collection, we collaborated with InfoArena[4], one of the most popular Romanian competitive programming websites. The platform hosts a total of 3,072 coding problems, with difficulty ranging from simple to high school International Olympiad level. The problems have a problem description written in Romanian, alongside descriptions of input and output requirements and several easy test cases for users to evaluate the solution. Users can submit solutions written in C / C++, which are automatically evaluated in a sandbox environment. Solutions are stored alongside the number of passed test cases. Test cases evaluate both the logical correctness of the solutions and their algorithmic complexity (i.e., sub-optimal solutions are given a lower score).

3.1. Problem Statements

InfoArena is a "wiki", in which volunteers can submit problems and discuss about solutions. The website is primarily addressed to Romanian students, and

[3] github.com/cosmaadrian/rocode

[4] infoarena.ro

all problem statements are written in native Romanian. Problems usually follow a common format, containing an initial preamble, providing context, followed by the requirements and input/output data specification. Most problems also have concrete examples for the input and output data for a correct solution. However, not all hosted problems are valid. Out of the 3,072 problems, we filtered out problems that have no submitted solutions, contain no examples, or do not follow the suggested problem template. After filtering, we obtain a total of 2,642 problem statements. Originally, each problem statement was written using markdown. We cleaned any markdown formatting and left only the problem text in the same format for all problems. Figure 1 showcases the common problem format in RoCode. In all problems, input data is given in a file and programs must output the correct solution in an output file. This is different from, for instance, APPS (Hendrycks et al., 2021), in which input and output are served from standard input / output.

3.2. Solutions

Each problem contains an average of 260.92 solutions, with easier problems such as "greatest common divisor" containing the most submissions, while Olympiad-level problems contain just a few solutions. Originally, solutions were written in C / C++ or Pascal. We kept only C / C++ solutions, removed redundant comments and formatted every solution to a common code style using a public tool[5]. We provide all $653,094$ solutions, and also a curated set of $11k$ solutions, which correspond to the top-3 highest-scoring, shortest solutions for each problem. Moreover, for the same author, we filtered similar solutions that obtained the same score for a source problem using the standard Levenshtein distance.

Other datasets (Hendrycks et al., 2021; Chen et al., 2021) and code-generation models (Husain et al., 2019; Chen et al., 2021) are less focused on low-level C / C++, but more on high-level languages such as PHP, JavaScript and Python. Since Python has gained massive popularity in recent years, and is considered the de facto standard high-level programming language for machine learning, we also provide transpiled solutions in Python, using the OpenAI API. We used the "*gpt-3.5-turbo*" model variant, which is a language model based on InstructGPT (Ouyang et al., 2022). We used the following prompt to transpile solutions: "*Translate the following C / C++ code to Python. Output only code, without any explanation or comments. Omit type hinting in the Python code: {code}*". Transpiled solutions are automatically checked for correctness.

Code-Code-Switching in Solutions

```
int main(void)
{
    // a.k.a. "read"
    citire();
    nr = 0;
    // a.k.a "number of steps"
    max = nr_pasi(max);
    radx(max, w, n);

    // a.k.a. "solve"
    rezolvare();

    // a.k.a. "print"
    printare();
    return 0;
}
```

Table 2: Example of a snippet from a C / C++ solution that exhibits code-code-switching: function names / variables are written in Romanian, whereas language keywords are written in English. Comments added by the paper authors.

Table 3 showcases an example of a solution automatically transpiled into Python. The transpiled solution is more concise and preserves functionality.

Different from other English-oriented datasets, RoCode contains code that exhibits *code-code-switching*: some function and variable names are written in Romanian, while others (e.g. language-specific keywords) are written in English. Table 2 showcases a real snippet found in RoCode that has all function names written in Romanian, or abbreviated forms of Romanian words[6]. For instance, some problem solvers write "*rezolva()*" instead of "*solve()*", "*afisare()*" instead of "*print()*". This aspect provides additional complexity for adapting pretrained models to RoCode solutions. In Figure 2, we show the proportion of variable names and function names that contain Romanian words. We parsed the abstract syntax tree using ClangCheck LLVM (Lattner and Adve, 2004) and uniformized all declarations to snake_case format, since programmers use both camelCase, snake_case or PascalCase. If any of the strings separated by underscore is found to be Romanian, we count that name as a Romanian variable / function. We used the Romanian WordNet (RoWordNet) (Dumitrescu et al., 2018) to check if a word belongs to the Romanian language. We obtained that around 9% of function names and around 14% of variable names have explicit Romanian words. This approach counts only properly written words with only a subset of declensions, and omits abbreviations, which makes the actual counts higher than shown here. Examples of composite function names are: *"acopera_tot()" (cover_everything()), "descompunereNumar()" (de-*

[5] https://github.com/dawnbeen/c_formatter_42

[6] "*nr*" is the abbreviation for "*număr*", equivalent to "*no.*" for "*number*" in English

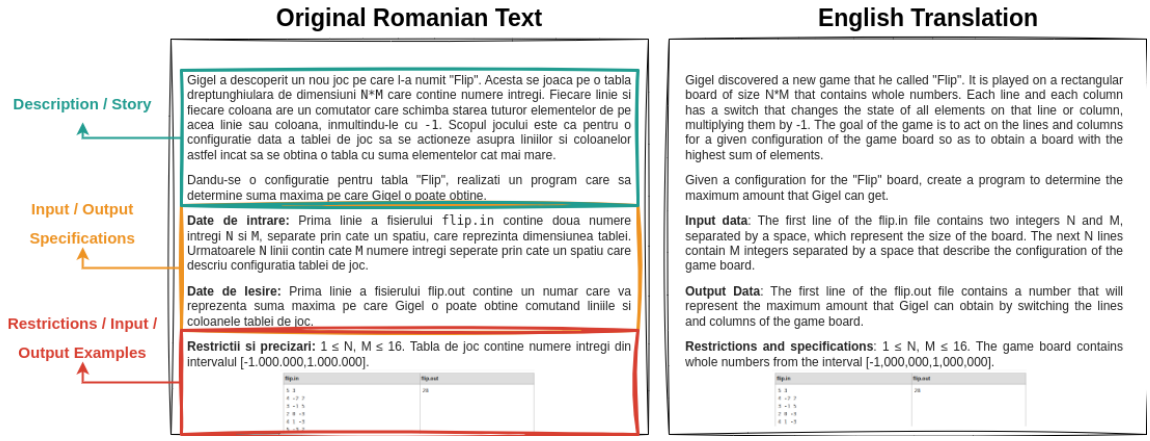

Figure 1: Example Romanian problem statement from RoCode. Problems specify a story-like description followed by input and output data specifications and restrictions. Finally, an example of an output from a correct solution is provided. We also show the English translation for non-Romanian readers.

Original C / C++ Solution

```cpp
#include <bits/stdc++.h>
#define all(cont) cont.begin(), cont.end()
#define pb push_back
using namespace std;
ifstream f("benzina.in");
ofstream g("benzina.out");
int main(void)
{
        int n; long long ans = 0;
        f >> n; n *= 2;
        vector<int> a(n);
        priority_queue<int> heap;
        for (int i = 0; i < n; ++i)
                f >> a[i];
        for (int i = 0; i < n; ++i)
        { int cost; f >> cost; ans += cost; a[i] -= cost; }
        for (int i = 0; i < n; ++i)
        {
                heap.push(a[i]);
                if (i % 2 == 0)
                { ans += heap.top(); heap.pop(); }
        }
        g << ans << '\n';
        f.close(); g.close();
}
```

Transpiled into Python

```python
import heapq
with open('benzina.in') as f, open('benzina.out', 'w') as g:
    n = int(f.readline().strip())
    n *= 2
    a = [int(x) for x in f.readline().split()]
    ans = 0
    for i in range(n):
        cost = int(f.readline())
        ans += cost
        a[i] -= cost
    heap = []
    for x in a:
        heapq.heappush(heap, x)
        if len(heap) % 2 == 1:
            continue
        ans += heapq.heappop(heap)
    g.write(str(ans) + '\n')
```

Table 3: Example of transpiled solution into Python, using "*gpt-3.5-turbo*". The Python code preserves functionality, while being comparatively smaller. For the C / C++ version, we formatted the code to be more concise for display purposes.

composeNumber()), "RezolvareDistantaMinimaCoborare()" (SolveMinimalDistanceDown()). Examples of composite variable names are: *"AdaugaValoare" (AddValue), "viziteazaTraseu" (visitRoute), deja_castigat (already_won)*.

3.3. Test Cases

Each solution is accompanied by a series of tests which are used to measure the correctness and computational complexity of the provided solution. The tests consist of an input file containing input data and an output file containing the desired output. There are a total of $35,758$ tests, and each problem has an average of 13 tests, while some problems have upwards of 100 tests. Since some tests are upwards of 100MB, we provide the smallest 5 tests for each problem, as well as an environment that automatically scores the provided generated solution. Similar to other works, problems are graded using accuracy, strict accuracy and "pass@k" metric (Chen et al., 2021).

3.4. Estimating Problem Difficulty

Following similar code datasets (Hendrycks et al., 2021; Chen et al., 2021) which provide different difficulty splits, we compute a difficulty score for each problem. For each problem, we computed the average score for the submitted user solutions, divided by the number of unique users. Since the publication date for problems ranges from the year 2006 up to 2022, we divided this score by the recency. We subsequently split RoCode into "easy", "medium" and "hard" problems, by splitting the difficulty distribution into tertiles. Consequently, we obtain a total of 790, 922 and 934 easy, medium and hard problems, respectively. Figure 3 showcases the distribution of problem difficulties across RoCode. Furthermore, Figure 4 showcases the

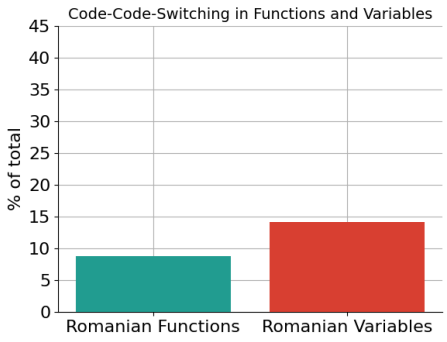

Figure 2: Percentage of Romanian variable names and function names that contain explicit Romanian words in submitted **human** C / C++ solutions. Abbreviations are omitted, making the actual proportion larger than shown here.

correlation between problem lengths and average solution lengths for each difficulty label - we found no significant correlation between lengths of problems and solutions, but consistently harder problems require longer solutions. Regarding *code-code-switching*, perhaps surprisingly, we found the same distribution of Romanian variable names and function names across difficulty labels.

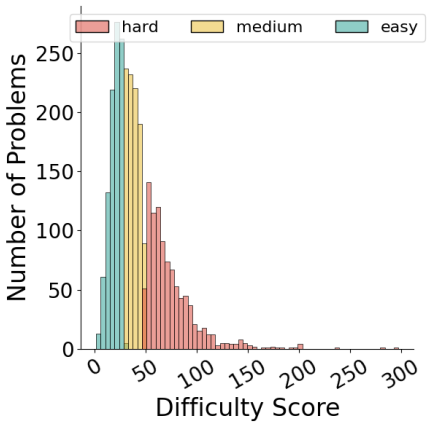

Figure 3: Distribution of problem difficulties in RoCode. Problem difficulty is estimated automatically based on the number of correct solutions, unique users, and the date of the problem.

3.5. Dataset Splits

We split the dataset into training, validation and test subsets, to enable researchers to fine-tune code generation models and to also provide a common testing split to compare approaches. The training, validation and test splits contain 2112, 264 and 266 problems, respectively. We sampled problems uniformly across problem difficulty for each data split.

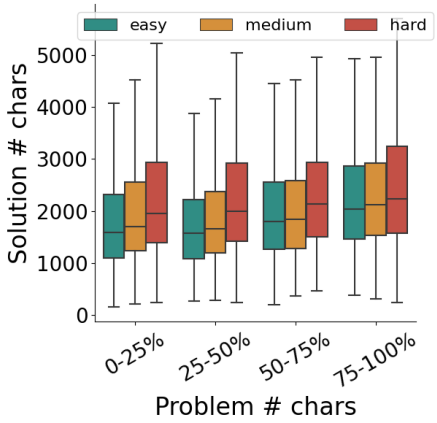

Figure 4: Correlation between problem statement length and solution lengths across problem difficulties. Harder problems require longer solutions, and problem statement length is not correlated with solution length.

4. Results

4.1. Experimental Setup

For benchmarking existing language models on RoCode, we follow the evaluation procedure proposed by Hendrycks et al. (2021). We used the following prompt[7] for all models:

```
Se dă urmatoarea problemă de
programare:
<PROBLEM STATEMENT>
<INPUT / OUTPUT SPEFICIATIONS>
<EXAMPLES>
Codul în Python3 care rezolvă
problema, fără comentarii sau
explicații, este:
```

In our experiments, we generated 10 solutions per problem. We computed accuracy, strict accuracy and pass@k (Chen et al., 2021) metrics. For accuracy, we counted the average maximum number of tests passed per problem. For strict accuracy, we counted the average number of times a problem has passed all tests. Additionally, we used the "pass@k" metric for evaluation. We generate 10 samples per problem and count the number of correct samples c. In particular, due to computational constraints, we chose $k \in \{1, 10\}$. The metric is defined as:

$$\texttt{pass@k} := \mathbb{E}\left[1 - \frac{\binom{n-c}{k}}{\binom{n}{k}}\right] \quad (1)$$

All models in this work were executed with a temperature of 0.2 and top-p sampling of 0.90. We

[7]English Translation: "The following programming problem is given. The code in Python3 that solves the problem, without comments or explanations, is:"

used a timeout of 4 seconds for solution run time. Models are run on a machine with 2x NVIDIA RTX 3060 with 12GB of VRAM each. Bigger models that did not fit a single GPU were parallelized across the two GPUs. All models' performance is measured in zero-shot settings (no fine-tuning, no additional examples or solution peaking (Hendrycks et al., 2021)). We used publicly available models from HuggingFace in all instances. In case the generated solution does not write to a file (as required by the problem statement), and instead expects the input and output to be managed through standard input and output, we obliged and provided the test data accordingly.

4.2. Benchmarking Romanian Language Models

We evaluated existing Romanian language models: Ro-GPT2 (Niculescu et al., 2021) in three model sizes (124M, 354M and 774M parameters) and GPT-Neo-Ro (Dumitrescu, 2022). Unsurprisingly, **None** of the currently available Romanian language models are able to understand the problem definition or to produce code, and all generated code could not be compiled. Performance in terms of accuracy, strict accuracy, pass@1 and pass@10 is exactly 0 for all metrics. This performance can be partially explained by model size, as similar English-based models (e.g. GPT-Neo (Gao et al., 2021)) also have below 3% pass rate on easy English problems (Hendrycks et al., 2021), but have a comparatively larger pretraining dataset. We point out that RoGPT-2's corpus (for example) contains around 3,400M tokens, but contains little to no code tokens, which explains the poor performance. We provide a more detailed discussion below.

4.3. Benchmarking English Models

In Table 4, we showcased results for 4 open-source models with a relatively small number of parameters (maximum $7B$). We chose a selection of small and efficient models due to computational limitations. We leave more extensive evaluations, as well as different fine-tuning schemes (Hu et al., 2021), as future work. Similar to the Romanian models, all models' performance is measured in zero-shot settings. We evaluated replit-code-v1-3b (Replit, 2023), a small but capable model trained on the Stack 3T (Kocetkov et al., 2022), that outperforms bigger models on code intelligence tasks. Furthermore, we also evaluated LLaMA-7b (Touvron et al., 2023) and OpenLLaMA-7b (Geng and Liu, 2023), which are trained on 1T and 300B tokens, respectively, of both English and code. Finally, we evaluated WizardCoder-Python-7b (Luo et al., 2023), an instruction fine-tuned variant of LLaMA-7b, through automatically generated instructions, which showcases very good performance on code intelligence tasks, surpassing in some cases commercial models. English models we included in our study are pretrained on some Romanian data – for example, for LLaMA-2 (Touvron et al., 2023), the pretraining dataset contains 0.03% Romanian tokens out of a total of 2T tokens ($\sim$ 600M Romanian tokens) – a very small proportion of the pretraining data, smaller than, for instance RoGPT-2s's corpus, but not a small number of Romanian words in absolute terms. It is assumed that performance is improved by the model's ability to exploit cross-lingual commonalities. Evidently, the English-oriented models have poor performance, only solving a handful of easy problems. It is clear that more recent, larger models with code in the training set output plausible code solutions, which most of the time compile properly: on average, around 18% of solutions result in compilation errors across models. In Table 5, we showcased selected model outputs for 2 Romanian models and 2 English-oriented models. We further provide some insights to the unsatisfactory performance of existing models.

Model	# Params	Acc. %	Strict Acc. %	pass@1	pass@10
llama-2-7b	7B	0.55	0.0	0.02	0.98
open-llama-2-7b	7B	1.50	0.0	0.05	2.37
replit-code-v1-3b	3B	1.81	0.39	0.1	7.74
wizard-coder-7b	7B	3.86	0.79	0.27	3.83

Table 4: Performance on RoCode for open-sourced English-oriented code models. While performance is different then zero, current models struggle to solve the problems expressed in Romanian.

4.4. Discussion

There are several reasons for the bad performance of Romanian models, which also provide future research directions for specialized monolingual and general-purpose models on languages other than English:

No code data present in the pre-training dataset. Romanian models have been trained on datasets derived from OSCAR corpus (Ortiz Suárez et al., 2019), Wikipedia and books, and do not have dedicated code splits annotated with Romanian text. Even if code is present in the dataset (from Common Crawl), it is described in English and comments and documentation are in English. The same argument can be made for mathematics and other scientific disciplines. A large, highly curated and dedicated Romanian dataset containing scientific data, mathematics and coding splits is needed.

Current Romanian models are small and pretraining dataset is too noisy. Small language models are prone to hallucinations and cannot follow instructions reliably without a high-quality cu-

RoGPT2-large (Niculescu et al., 2021)	GPT-Neo-Ro (Dumitrescu, 2022)	replit-code-v1-3b (Replit, 2023)	Open-LLaMa-7b (Geng and Liu, 2023)

"Votul este acordat(x) din X/C(n).Se presupune ca ambii(X,n) este numărul de X% pentru care Tony are dreptul%(y,j,k,o) voturi de votat.JPGPunctul (y) xq voturi din 1/3.pdfqq.Este numarulq voturile Xq pe votul fanilor ingustequale. Poses(q) Q(sq(t) voturile de votati inqual pentru votul lui.qualitatiunilor pentru al X(z(Z(k)z-ului).pdfPseud.p(m). ....."	*"Python 3 Rezolvă această problemă, nu sunt necesare nici un fel de comentarii. Doar faceți click pe această pagină, și verificați codul. Există și o variantă pentru limba română: Pyyp4. În ea se scrie următorul cod: -y Un exemplu clasic este prezentat în continuare. O mașină de măsurat cu trei axe dispune de un sistem de operare care rulează într-o bază de date, numit "TANNO", în care pot exista fișiere cu datele măsurate de cele trei modele ale mașinii ....."*	`n = int(input())`
		`candidates = list(map(`
		`  str, input().split())`
		`)`
		`candidates.sort()`
		`votes = list(map(`
		`  int,`
		`  input().split())`
		`)`
		`votes.sort()`
		`Q = int(input())`
		`for q in range(Q):`
		`  L, R = map(int, input().split())`
		`  print(votes.count(L) +\`
		`  votes.count(R))`
			`#include <stdio.h>`
			`#include <stdlib.h>`
			`#define MAX_N 1000000`
			`#define MAX_S 1000000`
			`int main()`
			`{`
			`  int n, q;`
			`  scanf("%d %d", &n, &q);`
			`  char *votes[MAX_N];`
			`  for (int i = 0; i < n; i++){`
			`  votes[i] = (char *)malloc(MAX_S);`
			`  scanf("%s", votes[i]);}`
			`  int ans = 0;`
			`  for (int i = 0; i < q; i++)`
			`  {`
			`  int l, r;`
			`  scanf("%d %d", &l, &r);`
			`  int cnt = 0;`
			`  for (int j = l; j <= r; j++)`
			`  if (votes[j][0] == 'C')`
			`  cnt++;`
			`  if (cnt < r - l + 1){`
			`  for (int j = l; j <= r; j++)`
			`  if (votes[j][0] == 'C')`
			`  votes[j][0] = 'T';`
			`  ans++;}`
			`  }`
			`  printf("%d\n", ans);`
			`  return 0;`
			`}`

Table 5: Selected outputs of open-source language models for an easy problem from RoCode. Romanian models are not able to output code, and instead hallucinate text vaguely similar to the problem description. We truncated the output for display purposes, as the quality degrades severely. The *replit-code-v1-3b* model outputs plausible Python3 code, but it does not correctly address the problem. *Open-LLaMa-7b* outputs C / C++ code instead of the required Python.

rated dataset. Recently, models such as Mistral-7b (Jiang et al., 2023) have shown that a relatively small model ($< 7B$ parameters) can obtain comparable performance to much larger models by training for longer on a highly curated dataset. In the work by Eldan and Li (2023), the authors show that a small language model can still produce coherent text while only being trained on a small, easy to understand and curated dataset, raising questions whether model scale is the principal factor in model performance. Furthermore, the use of augmentation through retrieval (similar to RETRO (Borgeaud et al., 2022)) has been shown to increase model performance without increasing its size - such techniques have not been explored in monolingual models, for instance, retrieving tokens from translated English text during training.

There is no post-training refinement for code. Post-training techniques such as instruction-tuning (Mishra et al., 2022) is a proven method for better performance and controllability of LLM output by following natural language instructions. A Romanian dataset of instructions has not yet been compiled outside of automatically translated versions (Dac Lai et al., 2023).

These negative results have not been addressed so far in the literature, and our hope is that it inspires future directions in training general-purpose Romanian or otherwise low-resourced language models. Further, we discuss the performance of English-oriented models.

The pre-training set for English-oriented models might have data leakage from RoCode. It is surprising that English-oriented models can somewhat follow the Romanian text description, essentially performing translation from Romanian text to Python code with symbols in English. However, it is unclear if correct model outputs can be attributed to proper text understanding or if it is a form of data leakage from larger corpora. By manually investigating the problems with high pass rate, the problems with the most tests passed are easy problems describing, for example, the edit distance, computing graph diameter (maximal distance between leafs), and detecting repeating subsequences. These are classic programming problems, and it is very likely that they appeared in other contexts in the pre-training sets, for instance in some parts of Common Crawl (Kúdela et al., 2017), since the problem definitions are publicly available and are likely discussed on other websites. Moreover, as shown in Table 5 the output from *open-llama-7b* is in C / C++ even though the prompt explicitly mentioned Python3. This is a further indication of data leakage.

English models exhibit even more *code-code-switching* compared to human solutions. Furthermore, code generated from the language models exhibits even more *code-code-switching* when generating Python code (see Figure 5). The models we tested tend to use many more Romanian variable names and slightly more Romanian func-

tions compared to the human-submitted solutions (see Figure 2): we found around 35% of variable names and 12% of function names contain explicit Romanian words in the generated Python solutions. The larger amount of code-code-switching is presumably an artifact of the next token prediction objective, and the models usually follow the terminology present in the problem text and in the output examples. This leads the model to adopt a similar style of Romanian variable naming in further code blocks.

Translating the problem definitions in English worsened results. Evaluating a *replit-v1-3b* (Replit, 2023) on translated problem definitions resulted in 1.26% accuracy, down from 1.81% using original Romanian descriptions. This is due both to imprecise translation and obfuscating exact formulations that might be present in the pretraining set. Translation is not a long-term solution to multilingual code intelligence models and it does not substitute proper language and code understanding of specialized monolingual models.

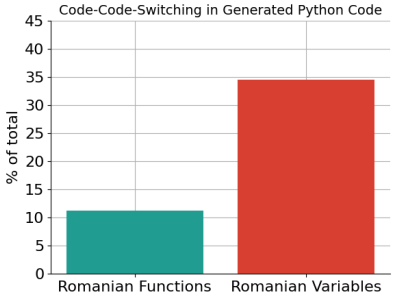

Figure 5: Percentage of Romanian variable and function names that contain explicit Romanian words in **model-generated** Python solutions. Generated solutions tend to have more explicit Romanian names.

5. Conclusions

In this work, we presented RoCode, a benchmarking dataset for code intelligence systems that measures the understanding of problem definitions in Romanian to provide algorithms that correctly solve the problems. Competitive programming benchmarks (Hendrycks et al., 2021) are still a challenging task, even for current state-of-the-art commercial models. However, all training sets containing code and benchmarks are implicitly geared towards English, with documentation, comments and problem definitions written solely in English. RoCode fills the gap in the benchmarking suite for Romanian NLP systems such as LiRo (Dumitrescu et al., 2021), which do not have any tasks for code generation for Romanian. Our dataset is challenging: several Romanian and English-oriented language models that we tested have poor performance, managing to correctly solve only a handful of problems from the test set. This work paves the way for further research of large language models for code intelligence in non-English languages and is a preliminary step in the democratization of programming for non-English speakers.

Acknowledgements

We thank the *infoarena.ro* team for providing the raw data for RoCode. The work of Adrian Cosma was performed as part of Short-Term Research Mission (STSM) at Universitat Politècnica de València, part of COST Action CA18231, Multi3Generation: Multi-task, Multilingual, Multi-modal Language Generation. The work of Paolo Rosso was in the framework of the FairTransNLP research project (PID2021-124361OB-C31), funded by MCIN/AEI/10.13039/501100011033 and by ERDF, EU A way of making Europe.

6. References

Openai api reference. https://platform.openai.com/docs/api-reference. Accessed: 2023-04-29.

Mohammad Bavarian, Heewoo Jun, Nikolas Tezak, John Schulman, Christine McLeavey, Jerry Tworek, and Mark Chen. 2022. Efficient training of language models to fill in the middle.

Shruti Bhosale, Kyra Yee, Sergey Edunov, and Michael Auli. 2020. Language models not just for pre-training: Fast online neural noisy channel modeling. In *Proceedings of the Fifth Conference on Machine Translation*, pages 584–593, Online. Association for Computational Linguistics.

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George Bm Van Den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, Diego De Las Casas, Aurelia Guy, Jacob Menick, Roman Ring, Tom Hennigan, Saffron Huang, Loren Maggiore, Chris Jones, Albin Cassirer, Andy Brock, Michela Paganini, Geoffrey Irving, Oriol Vinyals, Simon Osindero, Karen Simonyan, Jack Rae, Erich Elsen, and Laurent Sifre. 2022. Improving language models by retrieving from trillions of tokens. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 2206–2240. PMLR.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

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*.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. 2022. Palm: Scaling language modeling with pathways.

Fenia Christopoulou, Gerasimos Lampouras, Milan Gritta, Guchun Zhang, Yinpeng Guo, Zhongqi Li, Qi Zhang, Meng Xiao, Bo Shen, Lin Li, et al. 2022. Pangu-coder: Program synthesis with function-level language modeling. *arXiv preprint arXiv:2207.11280*.

Viet Dac Lai, Chien Van Nguyen, Nghia Trung Ngo, Thuat Nguyen, Franck Dernoncourt, Ryan A Rossi, and Thien Huu Nguyen. 2023. Okapi: Instruction-tuned large language models in multiple languages with reinforcement learning from human feedback. *arXiv e-prints*, pages arXiv–2307.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Stefan Dumitrescu. 2022. Github: Romanian transformers. https://github.com/dumitrescustefan/Romanian-Transformers. Accessed: 2023-04-29.

Stefan Dumitrescu, Andrei-Marius Avram, and Sampo Pyysalo. 2020. The birth of Romanian BERT. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4324–4328, Online. Association for Computational Linguistics.

Stefan Daniel Dumitrescu, Andrei Marius Avram, Luciana Morogan, and Stefan-Adrian Toma. 2018. Rowordnet–a python api for the romanian wordnet. In *2018 10th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)*, pages 1–6. IEEE.

Stefan Daniel Dumitrescu, Petru Rebeja, Beata Lorincz, Mihaela Gaman, Andrei Avram, Mihai Ilie, Andrei Pruteanu, Adriana Stan, Lorena Rosia, Cristina Iacobescu, Luciana Morogan, George Dima, Gabriel Marchidan, Traian Rebedea, Madalina Chitez, Dani Yogatama, Sebastian Ruder, Radu Tudor Ionescu, Razvan Pascanu, and Viorica Patraucean. 2021. Liro: Benchmark and leaderboard for romanian language tasks. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*.

Ronen Eldan and Yuanzhi Li. 2023. Tinystories: How small can language models be and still speak coherent english? *arXiv preprint arXiv:2305.07759*.

Eric Evans. 2004. *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley.

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. CodeBERT: A pre-trained model for programming and natural languages. In *Findings of*

the Association for Computational Linguistics: EMNLP 2020, pages 1536–1547, Online. Association for Computational Linguistics.

Daniel Fried, Armen Aghajanyan, Jessy Lin, Sida Wang, Eric Wallace, Freda Shi, Ruiqi Zhong, Scott Yih, Luke Zettlemoyer, and Mike Lewis. 2023. Incoder: A generative model for code infilling and synthesis. In *The Eleventh International Conference on Learning Representations*.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. 2021. The pile: An 800gb dataset of diverse text for language modeling. *CoRR*, abs/2101.00027.

Xinyang Geng and Hao Liu. 2023. Openllama: An open reproduction of llama.

Zishan Guo, Renren Jin, Chuang Liu, Yufei Huang, Dan Shi, Linhao Yu, Yan Liu, Jiaxuan Li, Bojian Xiong, Deyi Xiong, et al. 2023. Evaluating large language models: A comprehensive survey. *arXiv preprint arXiv:2310.19736*.

Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt. 2021. Measuring coding challenge competence with apps. *NeurIPS*.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katherine Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Oriol Vinyals, Jack William Rae, and Laurent Sifre. 2022. An empirical analysis of compute-optimal large language model training. In *Advances in Neural Information Processing Systems*.

Edward J Hu, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. 2021. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. 2019. CodeSearchNet challenge: Evaluating the state of semantic code search. *arXiv preprint arXiv:1909.09436*.

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, Brussels, Belgium. Association for Computational Linguistics.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b.

Denis Kocetkov, Raymond Li, Loubna Ben Allal, Jia Li, Chenghao Mou, Carlos Muñoz Ferrandis, Yacine Jernite, Margaret Mitchell, Sean Hughes, Thomas Wolf, et al. 2022. The stack: 3 tb of permissively licensed source code. *arXiv preprint arXiv:2211.15533*.

Jakub Kúdela, Irena Holubová, and Ondřej Bojar. 2017. Extracting parallel paragraphs from common crawl. *The Prague Bulletin of Mathematical Linguistics*, 107(1):39–56.

Sumith Kulal, Panupong Pasupat, Kartik Chandra, Mina Lee, Oded Padon, Alex Aiken, and Percy S Liang. 2019. Spoc: Search-based pseudocode to code. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Chris Lattner and Vikram Adve. 2004. LLVM: A compilation framework for lifelong program analysis and transformation. In *CGO*, pages 75–88, San Jose, CA, USA.

Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, et al. 2023. Starcoder: may the source be with you! *Submitted to Transactions on Machine Learning Research*. Under review.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals. 2022. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097.

Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao,

Jing Ma, Qingwei Lin, and Daxin Jiang. 2023. Wizardcoder: Empowering code large language models with evol-instruct. *arXiv preprint arXiv:2306.08568*.

Swaroop Mishra, Daniel Khashabi, Chitta Baral, and Hannaneh Hajishirzi. 2022. Cross-task generalization via natural language crowdsourcing instructions. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3470–3487.

Ansong Ni, Srini Iyer, Dragomir Radev, Ves Stoyanov, Wen-tau Yih, Sida I Wang, and Xi Victoria Lin. 2023. Lever: Learning to verify language-to-code generation with execution. In *Proceedings of the 40th International Conference on Machine Learning (ICML'23)*.

Mihai Alexandru Niculescu, Stefan Ruseti, and Mihai Dascalu. 2021. Rogpt2: Romanian gpt2 for text generation. In *2021 IEEE 33rd International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 1154–1161.

OpenAI. 2023. Gpt-4 technical report.

Pedro Javier Ortiz Suárez, Benoît Sagot, and Laurent Romary. 2019. Asynchronous Pipeline for Processing Huge Corpora on Medium to Low Resource Infrastructures. In *7th Workshop on the Challenges in the Management of Large Corpora (CMLC-7)*, Cardiff, United Kingdom. Leibniz-Institut für Deutsche Sprache.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744.

Inc Replit. 2023. replit/replit-code-v1-3b. https://huggingface.co/replit/replit-code-v1-3b. Accessed: 2023-10-17.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.

Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilic, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, Matthias Gallé, Jonathan Tow, Alexander M. Rush, Stella Biderman, Albert Webson, Pawan Sasanka Ammanamanchi, Thomas Wang, Benoît Sagot, Niklas Muennighoff, Albert Villanova del Moral, Olatunji Ruwase, Rachel Bawden, Stas Bekman, Angelina McMillan-Major, Iz Beltagy, Huu Nguyen, Lucile Saulnier, Samson Tan, Pedro Ortiz Suarez, Victor Sanh, Hugo Laurençon, Yacine Jernite, Julien Launay, Margaret Mitchell, Colin Raffel, Aaron Gokaslan, Adi Simhi, Aitor Soroa, Alham Fikri Aji, Amit Alfassy, Anna Rogers, Ariel Kreisberg Nitzav, Canwen Xu, Chenghao Mou, Chris Emezue, Christopher Klamm, Colin Leong, Daniel van Strien, David Ifeoluwa Adelani, and et al. 2022. BLOOM: A 176b-parameter open-access multilingual language model. *CoRR*, abs/2211.05100.

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. PICARD: Parsing incrementally for constrained auto-regressive decoding from language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 9895–9901, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

MosaicML NLP Team. 2023. Introducing mpt-7b: A new standard for open-source, commercially usable llms. Accessed: 2023-05-05.

NLLB Team, Marta R. Costa-jussà, James Cross, Onur Çelebi, Maha Elbayad, Kenneth Heafield, Kevin Heffernan, Elahe Kalbassi, Janice Lam, Daniel Licht, Jean Maillard, Anna Sun, Skyler Wang, Guillaume Wenzek, Al Youngblood, Bapi Akula, Loic Barrault, Gabriel Mejia Gonzalez, Prangthip Hansanti, John Hoffman, Semarley Jarrett, Kaushik Ram Sadagopan, Dirk Rowe, Shannon Spruit, Chau Tran, Pierre Andrews, Necip Fazil Ayan, Shruti Bhosale, Sergey Edunov, Angela Fan, Cynthia Gao, Vedanuj Goswami, Francisco Guzmán, Philipp Koehn, Alexandre Mourachko, Christophe Ropers, Safiyyah Saleem, Holger Schwenk, and Jeff Wang. 2022. No language left behind: Scaling human-centered machine translation.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor

Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open foundation and fine-tuned chat models.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. 2020. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. *Advances in Neural Information Processing Systems*, 33:5776–5788.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2023. Self-instruct: Aligning language models with self-generated instructions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13484–13508, Toronto, Canada. Association for Computational Linguistics.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. 2022. Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*.

Maksym Zavershynskyi, Alex Skidanov, and Illia Polosukhin. 2018. Naps: Natural program synthesis dataset. *NAMPI: Neural Abstract Machines & Program Induction Workshop*.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. 2023. Large language models are human-level prompt engineers. In *The Eleventh International Conference on Learning Representations*.