# Eliciting Better Multilingual Structured Reasoning from LLMs through Code

**Bryan Li**[1]*, **Tamer Alkhouli**[2]†, **Daniele Bonadiman**[2], **Nikolaos Pappas**[2], **Saab Mansour**[2]
[1]University of Pennsylvania, bryanli@seas.upenn.edu
[2]aws AI Labs, {alkhouli, dbonadim, nppappa, saabm}@amazon.com

## Abstract

The development of large language models (LLM) has shown progress on reasoning, though studies have largely considered either English or simple reasoning tasks. To address this, we introduce a multilingual structured reasoning and explanation dataset, termed xSTREET, that covers four tasks across six languages. xSTREET exposes a gap in base LLM performance between English and non-English reasoning tasks.[1]

We then propose two methods to remedy this gap, building on the insight that LLMs trained on code are better reasoners. First, at training time, we augment a code dataset with multilingual comments using machine translation while keeping program code as-is. Second, at inference time, we bridge the gap between training and inference by employing a prompt structure that incorporates step-by-step code primitives to derive new facts and find a solution. Our methods show improved multilingual performance on xSTREET, most notably on the scientific commonsense reasoning subtask. Furthermore, the models show no regression on non-reasoning tasks, thus demonstrating our techniques maintain general-purpose abilities.

## 1 Introduction

The ability to perform complex reasoning tasks is fundamental to human intelligence, where multiple steps of thought are required. Complex reasoning remains an open-problem for large language models (LLMs), despite some recent progress. Prior works consider complex reasoning tasks specified only in English. Such an English-centric perspective provides a limited assessment of the underlying reasoning capabilities of LLMs, given any specific language is largely a surface-form representation.

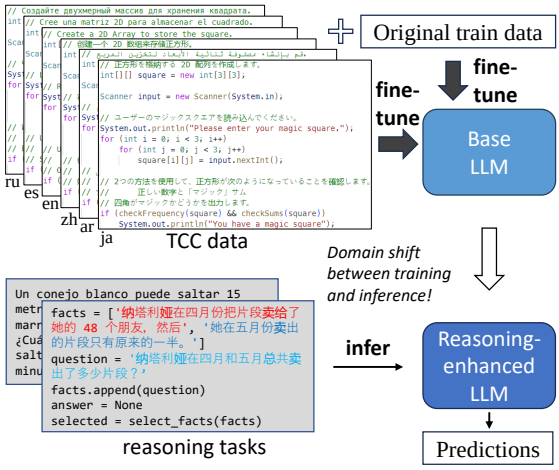


Figure 1: An overview of our methods to improve multilingual structured reasoning. First (top), we create the translated code comments (TCC) dataset, and use it in a fine-tuning setup. Second (bottom), we use the resulting LLM for inference on reasoning tasks. We find the most success with a code prompt format that bridges the representations between training and inference.

This motivates our first inquiry into the multilingual complex reasoning capabilities of LLMs.

We introduce the xSTREET reasoning and explanation dataset (as shown in Figure 2). xSTREET covers 4 tasks, and extends the English STREET benchmark (Ribeiro et al., 2022) to 5 additional diverse languages, inheriting the source's expert annotations and structured graphs for reasoning steps (7.8 average steps/answer). The tasks cover arithmetic, logic and science commonsense problems. We perform machine translation for the training and development data splits, and also perform human post-editing to the test sets, to ensure a high quality multilingual benchmark. We use xSTREET to evaluate several LLMs, identifying the multilingual setting as significantly challenging.

To remedy the non-English reasoning gap, we turn to the widely accepted hypothesis that LLMs trained on code are better at reasoning than those trained only on text. This *code and reasoning hy-*

*Work done during an internship at Amazon
†corresponding author
[1]https://github.com/amazon-science/xstreet released under CC-BY-4.0.



*Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5154–5169
August 11-16, 2024 

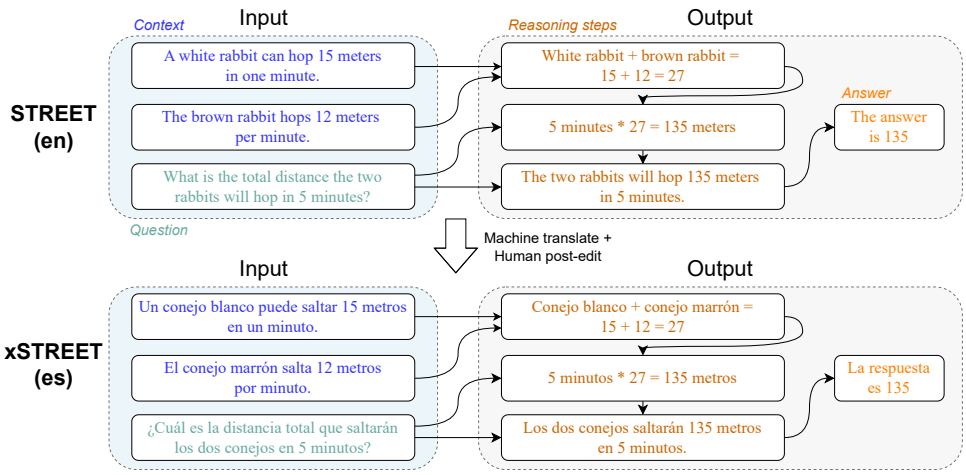


Figure 2: The translation process for an xSTREET entry. We start from an example from STREET (Ribeiro et al., 2022). The reasoning graphs are directly transferred, while each sentence text is translated. Note that this shows only one (of 4) task, GSM8K, and one (of 5) language, Spanish.

*pothesis* has been empirically corroborated by several papers (Suzgun et al., 2022; Liang et al., 2023; Hendy et al., 2023). Our work takes a further step in investigating the extent to which this hypothesis holds for non-English tasks. We proceed with the insight that code can be leveraged as a structured framework to represent the underlying reasoning steps, regardless of the surface-form language of the task. We thus propose two techniques to elicit better multilingual complex reasoning from LLMs (as shown in Figure 1): at training time through a lightweight fine-tuning recipe on code, and at inference time using a novel code prompt format.

In the LLM literature, many capabilities have been characterized as 'emergent' with model scale (Wei et al., 2022a; Patel et al., 2022). Recent work on complex reasoning has thus focused on huge (175B+) and closed-source models. In our work, we instead aim to boost performance on far smaller open-source LLMs (7B). To make our findings reproducible, we release our benchmark. Our contributions are:

1. We collect and release the first **dataset for multilingual structured reasoning, xSTREET**, covering 6 diverse languages and 4 tasks (5.5K entries total).
2. At train time: we enhance reasoning capabilities of off-the-shelf LLMs by further training on **program code data where code is interleaved with non-English comments**. To this end, we augment a source code corpus through translating code comments and apply low-rank parameter-efficient fine-tuning (LoRA (Hu et al., 2021)) to BLOOMZ (Muennighoff et al., 2022). Our method is effective yet lightweight, while preserving general-purpose LM capabilities.
3. At inference time: we design a code-like prompting format that mimics the structure of the reasoning tasks by **interweaving function calls and multilingual text**. We show this format outperforms several other prompt formats used.
4. We evaluate multiple LLMs (BLOOMZ, GPT-3, Falcon-40b-instruct) on our benchmark, and show improved performance — even for top-performing models — across structured reasoning tasks in different languages. As our inference and training-time techniques are orthogonal, we show that they can be used in tandem to achieve the best performance.
5. We perform qualitative and quantitative analysis to understand the roles of both the program code, and the code comments. Our findings taken together suggest that code elicits better multilingual structured reasoning by improving LLM's adherence to the reasoning format.

## 2 Related Work

We adopt the scope of complex reasoning from Ribeiro et al. (2022), and use complex reasoning and structured reasoning interchangeably. The goal is to study the reasoning process itself, and how an LLM can structure its output in steps to improve the final performance. The tasks are selected so that the knowledge to answer a question is

contained with the input question and context. For ease of evaluation, the answers are multiple-choice selections, or numbers for arithmetic reasoning. Further details are given in §2.4.

### 2.1 Code & Reasoning Hypothesis for LLMs

This hypothesis arose from empirical evidence by several concurrent works. Suzgun et al. (2022) state, "Codex, trained on both code and text data, shows better performance in following task instructions and exploiting algorithmic patterns based on the prompt exemplars." Liang et al. (2023) state, "for reasoning-intensive scenarios, we find that the code models, especially Codex davinci v2, consistently outperform the text models, even on synthetic reasoning scenarios posed in natural language." Hendy et al. (2023) state that "We hypothesize that the models acquire their reasoning capabilities through training on natural language multilingual data along with programming languages data". In summary, these works provide evidence that training LLMs on code serves as indirect supervision for complex reasoning tasks. One of our major goals is to explore the extent to which this hypothesis holds beyond English.

### 2.2 Code Prompts for Complex Reasoning

Reasoning tasks posed in natural language can be reformulated as **code prompts**. Using these code-like structures to interact with code-LLMs better aligns the representations seen at training time with those at inference time. Madaan et al. (2022) use few-shot prompting on the Codex LLM to convert tasks into Python graphs, deal with structured commonsense tasks. Zhang et al. (2023) proceed similarly, but for causal reasoning tasks. Chen et al. (2023) consider arithmetic reasoning tasks, and then execute the LLM-generated code on an external interpreter. The reformulation process from natural language specification to code prompts is an open-ended one, requiring manual annotation effort, creativity, and trial and error.

While these works use code prompts for complex reasoning tasks with classification or numerical outputs, as we did, code prompts can also be applied to tasks with generative outputs, such as knowledge graph construction (Bi et al., 2023) and story understanding (Dong et al., 2023). To the best of our knowledge, our work is the first to use code prompts in multiple languages.

### 2.3 Multilingual Reasoning for LLMs

The MEGA benchmark (Ahuja et al., 2023) covers 70 languages and 16 tasks. MEGA considers only simple reasoning tasks, which, as discussed earlier, limits our understanding of how well LLMs can reason across languages.

MGSM (Shi et al., 2022) is an arithmetic reasoning dataset in 10 languages, translated from GSM8K (Cobbe et al., 2021).They find that the chain-of-thought technique (CoT) (Wei et al., 2022b), by adding to the prompt few-shot examples of step-by-step reasoning, is also effective in the multilingual setting. Interestingly, they find that for non-English questions, English CoT outperforms native language CoT. They further emphasize the reasoning ability increases with model scale. Our xSTREET benchmark is a more comprehensive view of multilingual complex reasoning. xSTREET covers not only arithmetic,[2] but adds logic and science tasks, has many more entries, and has ground-truth structured reasoning annotations.

### 2.4 STREET Complex Reasoning Benchmark

The STREET benchmark is a composite of several complex reasoning tasks (Ribeiro et al., 2022). The work adds expert human annotations for multi-premise, multistep explanations. Each task's explanation is structured in a reasoning graph. Reasoning graphs, as shown in Figure 2, consist of nodes which contain statements, and edges that connect nodes.

**Source Tasks** The tasks[3] and answer formats are:

- **ARC** science commonsense questions (multiple-choice)
- **GSM8k** arithmetic word problems (number)
- **AQUA_RAT** arithmetic word problems (multiple-choice)
- **AR_LSAT** logic problems from a standardized test (multiple-choice)

**Linearized prompt format** While a reasoning graph is abstract, to interface with an LLM, Ribeiro et al. (2022) use **linearized** prompts. This represents a graph as a sequence of tokens, as shown in Figure 3. Statements are given a number index;

[2]Instead of using MGSM, we perform our own translation of GSM8k given the intermediate reasoning annotations inherited from Ribeiro et al. (2022).

[3]STREET includes a fifth task, SCONE, which is omitted from xSTREET. SCONE is quite abstract, and involves state-tracking in a toy world. This requires careful consideration beyond translation, and is thus left to future work.

output statements (i.e., reasoning steps) include a trace of the nodes leading to the new statement.

Problems with the linearized format arise in that it is task-specific, and that it diverges from LLM's training data distribution. While in-context learning can help the model pattern-match the output format, the underlying reasoning abilities of the LLM may not be properly elicited. Following Madaan et al. (2022), we argue that interfacing with a code-LLM through code prompts is a more "intuitive" way for the LLM to reason through a task, leading to our novel code prompts format in §5.

### 2.5 Source Code Dataset

The Stack is a 3.1 TB dataset of permissively licensed source code in 30 programming languages (Kocetkov et al., 2022). In this work, we utilize the official small subset[4], and consider only 3 popular programming languages: Java, JavaScript, Python (10k files each, 30k total).

## 3 Multilingual Complex Reasoning Benchmark: xSTREET

We create the xSTREET dataset by translating STREET into 5 languages: Arabic (ar), Spanish (es), Russian (ru), Chinese (zh), and Japanese (ja). These languages have linguistic and script diversity; furthermore, they are the languages used in many online programming help websites.

To create the *xSTREET test split*, we hire expert human translators for all 5 languages through an internal team (detailed in §9). Translators are tasked with post-editing the machine translation of one sentence at a time; for context, they can refer to the entire STREET entry the sentence comes from. After receiving the translations, we re-use the reasoning graph edges, and replace English nodes with the translations to create xSTREET. This process is shown in Figure 2. We therefore extend the 914 English entries in STREET to 5484 examples in xSTREET (914 * 6 languages).

To create the *xSTREET train and development splits*, we use machine translation.[5] We then asked native speakers to evaluate the quality of 10 random sampled translations of each language. Annotators gave feedback that, despite some errors, the translations were of reasonable enough quality to use for training purposes.

[4] Available here

[5] We used an online translation API (anonymized here).

| Dataset | # entry /lang | # sents/ lang | avg # sents/entry |
|---|---|---|---|
| ARC | 340 | 4334 | 12.7 |
| AQUA RAT | 254 | 3436 | 13.5 |
| AR LSAT | 50 | 1158 | 23.2 |
| GSM8k | 270 | 2255 | 8.4 |
| Total | 914 | 11183 | 12.2 |
| x6 languages | 5484 | 67098 | |

Table 1: Statistics for the xSTREET test benchmark.

Dataset statistics for the xSTREET test benchmark are given in Table 1.

## 4 Code with Multilingual Comments as Indirect Supervision for Reasoning

Taking the idea of using code for reasoning, and comments for multilinguality a step further, we address the question: *can multilingual code serve as indirect supervision for multilingual reasoning?* In other words, we investigate whether the code & reasoning hypothesis holds multilingually. We therefore propose a lightweight fine-tuning recipe, which consists of creating a multilingually commented code dataset, then fine-tuning on it, which serves as *indirect supervision* for downstream reasoning tasks.

### 4.1 Translated Code Comments Dataset (TCC)

The first step of the recipe is creating a source code dataset with translated code comments, termed TCC. For each file from the source dataset, and for each target language, we perform the following. We parse the code to extract out comments, translate comments into the target language, then replace the original comments with translations. This is depicted in Appendix Figure 7.

We use two simple filters: for source code files that A) have >5 comments, and B) whose comments are over 50% in English.[6] This filters 30k source code files down to 20k. After translating into 5 additional languages, TCC consists of 20k*6=120k files total. See Appendix Table 5 for dataset statistics.

### 4.2 Train Time: fine-tuning on TCC

In the second step, we leverage low-rank adaptation (LoRA) (Hu et al., 2021) to finetune instruction-

[6] We performed other filtering experiments, described in Appendix E.2, which had similar performance.

tuned LLMs on TCC.[7] We use two methods to preserve the original model's capabilities despite the additional finetuning. First is by using LoRA itself, as it keeps the original base model's parameters frozen and introduces only a few learned parameters. Secondly, we replay 100k examples from the base model's training data, xP3 (Muennighoff et al., 2022), in a multitask setup with the TCC LM task.

The recipe for a reasoning-enhanced LLM is now complete, and this is depicted in Figure 1.

## 5 Multilingual Complex Reasoning as a Downstream Task

We hypothesize that structure, when applied to reasoning problems formulated in different languages, can abstract away some of the language-specific details, better surfacing the reasoning steps needed for a model. We thus propose the SIM (Select-and-infer multilingual comments) code prompts for complex reasoning tasks.

SIM code prompts utilize several functions. We do not provide the API definitions, instead, we expect the model to learn to use them from the in-context examples. The functions are:

- `select_facts(facts)`
- `infer_new_fact(selected)`
- `is_solved(fact, question)`
- `make_choice(fact, choices)`[8]
- `facts.append(fact)`

`select_facts` and `infer_new_fact` are loosely inspired by Selection-Inference (Creswell et al., 2023). A key difference, though, is that we use a single prompt, instead of iterative prompts. We therefore include `is_solved(fact, question)` as a signal for the LLM to stop generation.

Each function is annotated with its return value in an inline code comment. This is inspired by prior work (Zhang et al., 2023). `infer_new_fact` has a string return value, i.e., the text of the new fact. We experiment with two versions of the return value of `select_facts`. The first, termed SIM-indexed, uses variables `facts[i]` to reference the `facts` array (similar to the indices used in linearized format). The second, termed SIM-text, directly uses each fact's text, dereferenced from `facts[i]`. We find that SIM-text works best for smaller models, while SIM-indexed does for larger ones, and hence apply this going forward.

We write a rule-based Python script that converts existing structured graph annotations to SIM code prompts. SIM prompts express the exact same information as the linearized format. This property is unlike code prompts for prior work, wherein the conversion is done through in-context learning with an LLM, which can introduce errors as discussed in §2.2. The different prompting formats for LLMs are shown in Figure 3.

**Multilingual code prompts** We use multilingual input in SIM code prompts as follows. First, facts given in the question are listed in the language of the task in a list of strings. Second, new facts and selected facts are given as comment lines adjacent to the function calls. See Figure 3 for an example.

## 6 Experimental Setup

**Models Used** We primarily study open-source models, which allows for application of both train and inference-time techniques. We use **BLOOMZ** (Muennighoff et al., 2022) as our base LLM. This model is instruction-finetuned on prompts in 46 natural languages and 10 programming languages. For our experiments, we consider the 7.1B-parameter BLOOMZ, as well as BLOOMZ-TCC which is further finetuned on TCC.

For inference-time only, we consider two larger LLMs. We use the instruction-finetuned version of **Falcon** (Almazrouei et al., 2023) (40B), another open-source LLM trained on text+code. Compared to BLOOMZ, Falcon is more performant on English tasks; however, it has limited multilingual abilities.[9] We also use **GPT-3** (175B)[10], a closed-source model that is popularly-used and powerful.

**Prompting setup** We use few-shot prompting, and random sample up to 5 exemplars from the train split (up to a model's context length).[11] For each inference example, the same exemplars are used for all models and prompt types. We use greedy decoding, and task the model with generating up to 682 tokens. (max context length of BLOOMZ 2048 // 3).[12]

[7] We used a g5.48xlarge instance from AWS, which has 8 NVIDIA A10G GPUs (24*8 GB=192GB vRAM).

[8] This function is not used for non-MC tasks, i.e. GSM8k.

[9] As stated in the model card for falcon-40b-instruct.

[10] `text-davinci-002` following Ribeiro et al. (2022)

[11] Due to brevity, Figure 1 uses a 0-shot prompt, and only depicts the SIM prompting format. The reported results use 5-shot prompts, and are given for all prompting formats.

[12] We acknowledge a limitation with the max of 682 tokens, as this will truncate output for questions which require longer

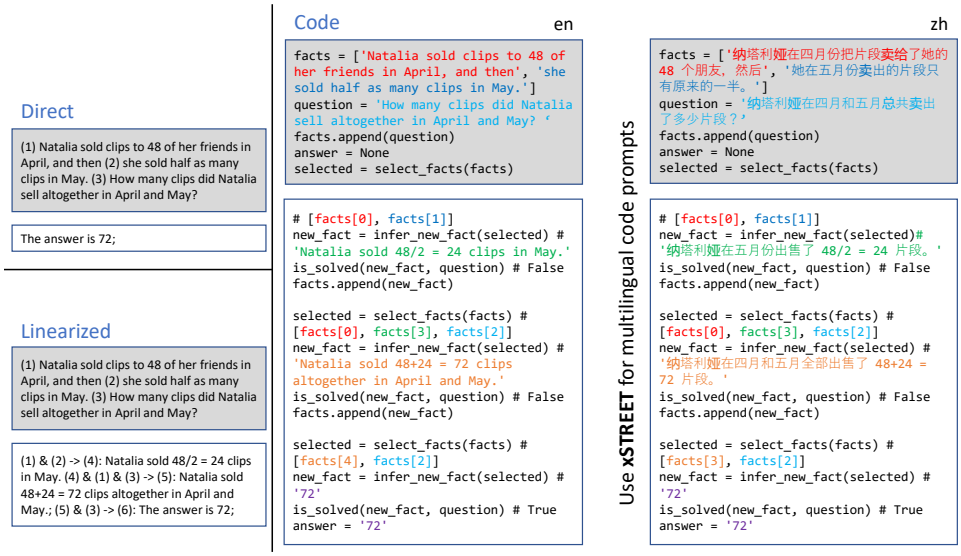


Figure 3: Depictions of 3 prompting formats for the xSTREET tasks. For each format, input is in a grey box, while expected output is in a white box. Top left: direct. Bottom left: linearized. Right: SIM code prompts (2 languages). In the code prompts, we color code facts which are aligned.

## 7 Results

We report results on the xSTREET benchmark. We use the answer accuracy metric, adapting evaluation from Ribeiro et al. (2022).[13]

Given the extensive nature of the xSTREET benchmark and our model experimentation, we highlight our findings iteratively. We first consider only BLOOMZ and BLOOMZ-TCC, with a particular focus on ARC, where our methods are the most impactful. We then consider GPT-3 and Falcon.

The full results are given in Appendix D. Here we provide numbers for all tasks, languages, models, and prompt formats (this also includes the direct prompting format).

### 7.1 Results for BLOOMZ Models

Results for the ARC task (science commonsense reasoning) are shown in Figure 4. Several takeaways arise. We see that code prompts greatly outperform linearized prompts across all languages. For results within a single model, reasoning performance drops greatly comparing English vs. average non-English (e.g. from 76.2% to 61.1% accuracy for BLOOMZ-TCC). This provides evidence that current multilingual LLMs are still optimized for English. This underscores the usefulness of xSTREET for developing LLMs with better underlying, language-agnostic abilities.

We next turn to comparing base BLOOMZ vs. our finetuned BLOOMZ-TCC. We see that BLOOMZ-TCC outperforms BLOOMZ for all languages and both formats. More interestingly, relative multilingual gain is much larger when using code prompts vs. linearized prompts (Avg non-en, 52.6 → 61.1 vs. 33.5 → 36.9). This is evidence that the code prompt format improves multilingual reasoning, likely by the explicit separation of the reasoning task (in code) vs the multilingual understanding (in comments). Finally, looking at per-language trends for BLOOMZ-TCC we see that code prompts are most effective for en, es, zh, and ar, while less so for ja and ru.[14]

**Results on GSM8K, AQUA_RAT, AR_LSAT** Our results show that BLOOMZ and BLOOMZ-TCC struggle for the other tasks, with performance being around random chance whether using the interventions or not. We hypothesize that these tasks are "too hard" for the BLOOMZ-7B used; to reiterate, GSM8K and AQUA_RAT are arithmetic reasoning, while AR_LSAT is logical reasoning. This concurs with the common view that complex reasoning capabilities of LLMs are emergent with model scale (Wei et al., 2022a). We further discuss these results, and expand our hypotheses, in

reasoning chains.

[13]While STREET also measure graph similarity between linearized output and reference graphs, we did not implement them for SIM prompts. This is because for the small LLMs (7B), even the linearized format had near 0 graph similarity.

[14]This is likely because the base model, BLOOM, was not trained on any ja or ru text.

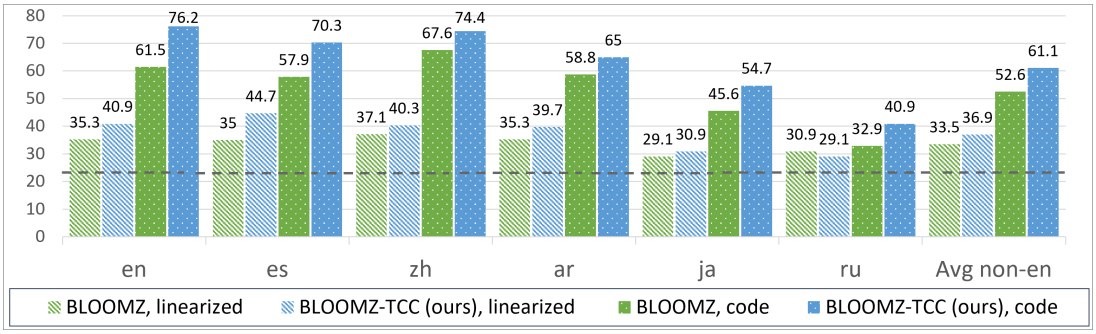


Figure 4: Results on ARC task of xSTREET, with BLOOMZ-based models. The random baseline is 25%. 'Avg' bars are across the 5 non-English languages. Linearized prompts use lines, while code prompts use dots.

| Model | XNLI | XStory-Cloze | XQUAD |
|---|---|---|---|
| BLOOMZ | 45.5 | 72.4 | 80.5 |
| BLOOMZ-TCC (ours) | 45.6 | 71.8 | 80.4 |

Table 2: Results for 3 non-complex multilingual reasoning tasks, averaged over all languages.

Appendix §A.1.

### 7.2 Results for Larger LLMs

As code prompts are at inference time, they can be used to interface with any LLM. We report results for GPT-3 in Figure 5. We see as before that the multilingual setting poses additional challenges for reasoning, as English results are always higher than corresponding non-English tasks.

First considering ARC, GPT-3 performs strongly in English for both formats, nearly solving the task. Comparing English to multilingual ARC, linearized suffers a sharp drop (93.2 → 73.2), while code prompts remain robust (99.1 → 94.2). This underscores the effectiveness of SIM prompts in disentangling the reasoning and multilingual components of the task.

For the other tasks, SIM always outperforms linearized format. Comparing relative gains, code prompts boost performance more in English than on multilingual settings. While still a very positive result, this differs from ARC as discussed above. To discuss why this is the case, we consider the dual effects of SIM code prompts, vs. linearized: the function calls capture the reasoning structure, while the multilingual comments capture the language understanding. Because the arithmetic and logical reasoning tasks are far more symbolic than the ARC commonsense reasoning task, multilingual language understanding is less effective.

### 7.3 Non-Complex Reasoning Task Results

Recall that our fine-tuning recipe aims to improve reasoning of an LLM, while maintaining its natural language understanding (NLU) abilities. We show this is the case by reporting results on 3 multilingual tasks:

- **XNLI**: natural language inference
- **XStoryCloze**: given 4 sentences from a short story, choose between 2 possible completions
- **XQUAD**: extractive question answering

To query LLMs, we follow the specific prompting guidelines for each task from Ahuja et al. (2023). Table 2 shows that for all 3 tasks, the differences between BLOOMZ and BLOOMZ-TCC are statistically insignificant. Therefore, the mitigation strategies we used, LoRA and training data replay, have proved effective.

### 7.4 Effect of Code Comments on Downstream Reasoning

The code & reasoning hypothesis speaks to training on code improving LLM reasoning. However, an integral part of source code is comments, which have been underexplored by prior work. We study 2 ablation settings, with the same finetuning setup: **TCC-en**: original source code files (i.e. English-only comments). **TCC-del**: source code files without any comments (comments are deleted).

We evaluate the best prompt format (SIM) on the ARC subtask. Results are shown in Figure 6. We see that overall, finetuning on TCC is the best configuration, then TCC-en, and finally TCC-del. These trends generally hold over the 6 languages.[15]

This ablation study adds a new consideration to the code & reasoning hypothesis: that within code,

[15]Russian (ru) is an exception, where BLOOMZ-TCC-en outperforms BLOOMZ-TCC. We will investigate this further, but note this may be an artifact of the base LLM, BLOOM, not having tokenization for Cyrillic script.

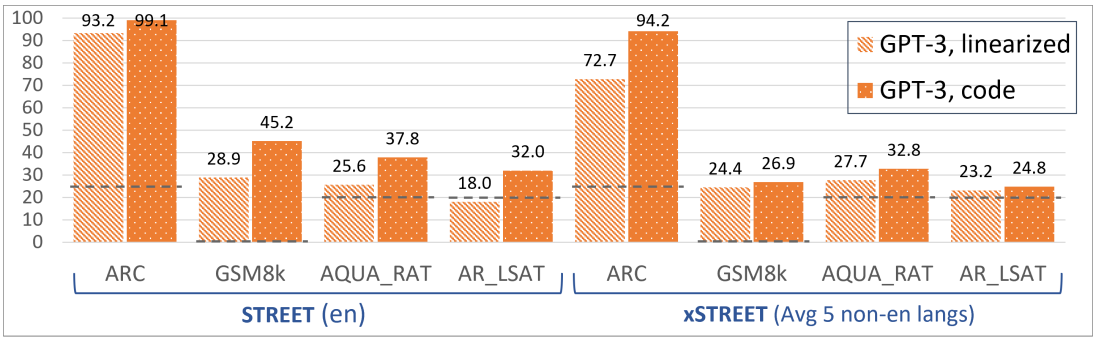


Figure 5: Results on GSM8k, AQUA_RAT, AR_LSAT tasks of STREET (left) and xSTREET (right), with GPT-3. For each task, the random baseline is shown with a dotted line. xSTREET results are averaged over 5 languages.

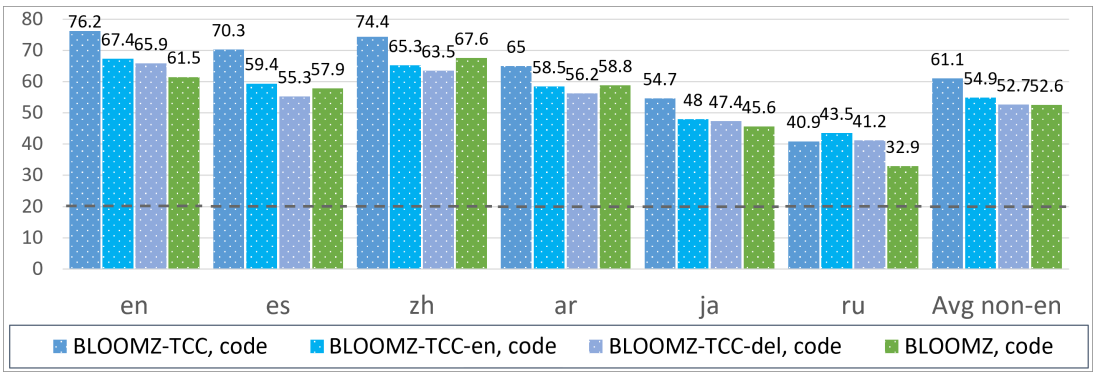


Figure 6: Results on ARC subtask of xSTREET for the code ablation experiments, where BLOOMZ is finetuned on different datasets. BLOOMZ-TCC uses our proposed multilingual code comment augmentation process, BLOOMZ-TCC-en uses the source code files with English comments, and BLOOMZ-TCC-del uses source code files with all comments deleted. As in Figure 4, we use SIM prompts with up to 5-shot examples, and show 'Avg' bars across the 5 non-English languages.

even the comments are influential in downstream reasoning performance. Furthermore, we see that the diversity of code comments introduced by our proposed data augmentation of TCC further boosts performance in all languages, including English.

## 8 Analysis

To further understand wherein our techniques help, or fail to help, model reasoning, we perform some manual analysis. For brevity, we focus on 2 languages, en and ar, and 2 tasks, ARC and GSM8K. We first perform error analysis on BLOOMZ, then perform a case study for each task.

We further perform 3 additional experiments, which are detailed in Appendix §E. To highlight one interesting finding, we show that training on diverse code comments, such as from the multilingual TCC, boosts xSTREET performance in all languages including English.

**Error Analysis for BLOOMZ English** For this task, and with base BLOOMZ, SIM achieves 61.5 ARC accuracy, while linearized achieves 35.3. Our manual analysis of outputs reveals that the performance discrepancy is largely due to *poor instruction-following when using linearized* vs. using SIM. Ribeiro et al. (2022) find that for linearized (and their model), 62% of generations fail to generate a parsable answer (i.e., reasoning graph is incomplete). Our findings concur, in that linearized has 66% (223/340) invalid generations. In contrast, SIM has only ˜19% invalid. BLOOMZ-TCC with SIM further reduces invalid rate to 9%, and increases accuracy to 76.2. We observe that in cases where all formats output successfully, the reasoning graph and answers are nearly identical. The difference is that SIM prompts allows the model to generate a complete reasoning graph far more often. We reiterate that this behavior is a novel finding given BLOOMZ-TCC was indirectly supervised on code, rather than directly on reasoning tasks. Further discussion is found in Appendix A.

We summarize this section with the following view: our techniques *elicit better instruction-*

*following* of the proscribed reasoning format from a base LLM, leading to improved benchmark performance. Within a reasoning step, the models are making similar decisions, but at the reasoning-graph level, our methods assist in harder cases.

### 8.1 Case Study on GSM8k English

We perform a case study of one GSM8k problem, comparing 3 models (BLOOMZ, BLOOMZ-TCC, GPT-3) and 2 formats (linearized, SIM) in Appendix Table 3. We observe that only GPT-3 with SIM achieves the correct answer, and reasoning steps also concur with the gold completion. GPT-3 with linearized representation makes an erroneous first step, which propagates the error downwards. Both BLOOMZ models with linearized formatting only follow the output format, and the text statements are copied from the input instead of being new statements. BLOOMZ with SIM has repetitive output and does not output an answer. While BLOOMZ-TCC still outputs a wrong answer, it does perform 2 rounds of reasoning through selecting and inferring facts. So, we see that both interventions elicit better underlying reasoning abilities of LLMs.

### 8.2 Case Study on ARC Arabic

We look at an Arabic example from ARC in Appendix Table 4. We observe that for the linearized format, the final answer is incorrect (A), given the model makes a wrong penultimate inference. The SIM format, meanwhile, allows GPT-3 to output the correct answer (D), given it makes a correct inference step (albeit 1 step less than the gold). In fact, directly prompting GPT-3 leads to a correct answer. This again highlights the importance of aligning the prompt format, which is code here, to the training format.

## 9 Conclusion

We introduced xSTREET, a multilingual structured reasoning benchmark which covers 5 diverse languages , spans science commonsense, arithmetic and logical reasoning tasks, and includes high-quality intermediate reasoning steps. We found that current multilingual LLMs underperform in the non-English setting, then proposed two methods to remedy this, based on the popular hypothesis that LLMs trained on code are better reasoners. At training, we propose translating the comments of a source code dataset, to use as indirect supervision data for parameter-efficient fine-tuning. During inference, we leverage code structure to represent reasoning graphs. We perform extensive experimentation, and both of our methods better elicit underlying reasoning abilities of LLMs.

Our work brings together two areas of challenge for LLMs — multilinguality, and complex reasoning. In particular, our fine-tuning recipe shows that the code & reasoning hypothesis can apply multilingually. We suspect that improvements can be amplified if multilingual comments are included at the pre-training, instead of the fine-tuning stage. We hope our findings underscore the key role that code should play in the development of LLMs with better reasoning capabilities across languages.

## Limitations

One limitation is that we were unable to apply our fine-tuning recipe to the stronger LLMs. "Stronger" refers to two characteristics. First and unavoidably, we can only apply the method to weaker open-source models, as closed-source models are proprietary; nevertheless, we explored them with our inference-time SIM prompts approach, and this worked well. Second, we only were able to fine-tune a 7B parameter model due to our resource constraints, so it is to-be-determined the effectiveness of the recipe on 70B+ models.

Between the submission and publication of this work (February to August 2024), LLM development has been brisk, and several recently released $\sim$7B LLMs have shown decent performance on arithmetic reasoning. In our work, we were limited to BLOOMZ-7B, which we saw was poor at math. For followup work, therefore, we are excited to try our finetuning approach on TCC while using these newer LLMs as base models.

Another limitation is for the xSTREET benchmark, we performed human translation on only the test set of the source STREET dataset. As we used machine translation for the train set, but also drew few-shot exemplars from these, the lower exemplar quality worsens performance compared to a gold standard exemplars. We also fine-tuned on machine-translated TCC.

While we tried to be inclusive with the languages chosen, studying 6 languages from different families and using different scripts, we acknowledge that more community effort will need to go into expanding the study of multilingual complex reasoning to lower-resource languages. We further

acknowledge the limits of the translation of English reasoning tasks and intermediate steps alone, in that reasoning processes may differ for speakers of different languages. So too may a multilingual LLM respond inconsistently to queries posted in different languages (Li et al., 2024), which warrants future studies into how this holds for the reasoning tasks studied in this work.

Finally, in this work, we considered only the final answer accuracy for the tasks. The original STREET tasks from Ribeiro et al. (2022) included various graph similarity metrics used to consider the intermediate reasoning steps as well – a definite strength of their structured reasoning approach vs. unstructured approaches such as CoT. We did not do this consideration due to the difficulty of reimplementing the graph similarity metric calculation for the different languages, and leave this to follow up work. Furthermore, we note that the 7B LLM we used had overall poor graph similarity (near 0 for all metrics) using the original STREET evaluation scripts and dataset.

**Data Statement** We provide a data statement in adherence with the ACL code of conduct and recommendations laid out in Bender and Friedman (2018). Linguists working on the Machine Translation Post Editing project for the multilingual dataset into Arabic, Chinese, Japanese, Russian, and Spanish are in-country, native speakers. They all are certified translators with more than 5 years of full-time translation experience, according to the 17100 Translation ISO Standard. These linguists were hired through vendors and were remunerated above industry standard rates. Instructions were to post-edit machine translated output and included guidelines on what to localize (artist names, city names, metric conversions), format (capitalization, punctuation) and structure (sentence level breaks). The vendor project managers made sure the instructions were adhered to. The QA process consisted of content review based on the Multidimensional Quality Metric (MQM) model that allocates different weights to 5 error severities (0-none to 5-critical) in several error topics. Total sample reviewed was 5 (5k words) of the total (100k words) source word count.

## 10 Acknowledgements

We would like to thank Danilo Neves Ribeiro for his guidance on working with the STREET benchmark, and insightful conversations on how tackle our multilingual extension of complex reasoning. We thank several colleagues for providing annotations: Etsuko Ishii, Igor Shalyminov, Yuwei Zhang. We thank these people for discussion and feedback: Salvatore Romeo, Yi Zhang, Sam Davidson, and Sailik Sengupta.

## References

Kabir Ahuja, Rishav Hada, Millicent Ochieng, Prachi Jain, Harshita Diddee, Samuel Maina, Tanuja Ganu, Sameer Segal, Maxamed Axmed, Kalika Bali, et al. 2023. Mega: Multilingual evaluation of generative ai. *arXiv preprint arXiv:2303.12528*.

Ebtesam Almazrouei, Hamza Alobeidli, Abdulaziz Alshamsi, Alessandro Cappelli, Ruxandra Cojocaru, Mérouane Debbah, Étienne Goffinet, Daniel Hesslow, Julien Launay, Quentin Malartic, Daniele Mazzotta, Badreddine Noune, Baptiste Pannier, and Guilherme Penedo. 2023. The falcon series of open language models.

Zhen Bi, Jing Chen, Yinuo Jiang, Feiyu Xiong, Wei Guo, Huajun Chen, and Ningyu Zhang. 2023. Codekgc: Code language model for generative knowledge graph construction. *arXiv preprint arXiv:2304.09048*.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W Cohen. 2023. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Transactions on Machine Learning Research*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Antonia Creswell, Murray Shanahan, and Irina Higgins. 2023. Selection-Inference: Exploiting Large Language Models for Interpretable Logical Reasoning. In *The Eleventh International Conference on Learning Representations*.

Yijiang Dong, Lara Martin, and Chris Callison-Burch. 2023. Corrpus: Code-based structured prompting for neurosymbolic story understanding. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 13152–13168.

Amr Hendy, Mohamed Gomaa Abdelrehim, Amr Sharaf, Vikas Raunak, Mohamed Gabr, Hitokazu Matsushita, Young Jin Kim, Mohamed Afify, and Hany Hassan Awadalla. 2023. How good are gpt models at machine translation? a comprehensive evaluation. *ArXiv*, abs/2302.09210.

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*.

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*.

Bryan Li, Samar Haider, and Chris Callison-Burch. 2024. This land is Your, My land: Evaluating geopolitical biases in language models.

Percy Liang, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, Michihiro Yasunaga, Yian Zhang, Deepak Narayanan, Yuhuai Wu, Ananya Kumar, Benjamin Newman, Binhang Yuan, Bobby Yan, Ce Zhang, Christian Cosgrove, Christopher D. Manning, Christopher R'e, Diana Acosta-Navas, Drew A. Hudson, E. Zelikman, Esin Durmus, Faisal Ladhak, Frieda Rong, Hongyu Ren, Huaxiu Yao, Jue Wang, Keshav Santhanam, Laurel J. Orr, Lucia Zheng, Mert Yuksekgonul, Mirac Suzgun, Nathan S. Kim, Neel Guha, Niladri S. Chatterji, Omar Khattab, Peter Henderson, Qian Huang, Ryan Chi, Sang Michael Xie, Shibani Santurkar, Surya Ganguli, Tatsunori Hashimoto, Thomas F. Icard, Tianyi Zhang, Vishrav Chaudhary, William Wang, Xuechen Li, Yifan Mai, Yuhui Zhang, and Yuta Koreeda. 2023. Holistic evaluation of language models. *Annals of the New York Academy of Sciences*, 1525:140 – 146.

Aman Madaan, Shuyan Zhou, Uri Alon, Yiming Yang, and Graham Neubig. 2022. Language models of code are few-shot commonsense learners. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 1384–1403, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Niklas Muennighoff, Thomas Wang, Lintang Sutawika, Adam Roberts, Stella Biderman, Teven Le Scao, M Saiful Bari, Sheng Shen, Zheng-Xin Yong, Hailey Schoelkopf, et al. 2022. Crosslingual generalization through multitask finetuning. *arXiv preprint arXiv:2211.01786*.

Ansong Ni, Jeevana Priya Inala, Chenglong Wang, Oleksandr Polozov, Christopher Meek, Dragomir R. Radev, and Jianfeng Gao. 2022. Learning math reasoning from self-sampled correct and partially-correct solutions. In *International Conference on Learning Representations*.

Ajay Patel, Bryan Li, Mohammad Sadegh Rasooli, Noah Constant, Colin Raffel, and Chris Callison-Burch. 2022. Bidirectional language models are also few-shot learners. In *The Eleventh International Conference on Learning Representations*.

Danilo Neves Ribeiro, Shen Wang, Xiaofei Ma, Henghui Zhu, Rui Dong, Deguang Kong, Juliette Burger, Anjelica Ramos, William Yang Wang, George Karypis, et al. 2022. Street: A multi-task structured reasoning and explanation benchmark. In *The Eleventh International Conference on Learning Representations*.

Freda Shi, Mirac Suzgun, Markus Freitag, Xuezhi Wang, Suraj Srivats, Soroush Vosoughi, Hyung Won Chung, Yi Tay, Sebastian Ruder, Denny Zhou, et al. 2022. Language models are multilingual chain-of-thought reasoners. In *The Eleventh International Conference on Learning Representations*.

Mirac Suzgun, Nathan Scales, Nathanael Scharli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V. Le, Ed Huai hsin Chi, Denny Zhou, and Jason Wei. 2022. Challenging big-bench tasks and whether chain-of-thought can solve them. In *Annual Meeting of the Association for Computational Linguistics*.

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. 2022a. Emergent abilities of large language models. *Transactions on Machine Learning Research*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022b. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Zhewei Yao, Reza Yazdani Aminabadi, Olatunji Ruwase, Samyam Rajbhandari, Xiaoxia Wu, Ammar Ahmad Awan, Jeff Rasley, Minjia Zhang, Conglong Li, Connor Holmes, Zhongzhu Zhou, Michael Wyatt, Molly Smith, Lev Kurilenko, Heyang Qin, Masahiro Tanaka, Shuai Che, Shuaiwen Leon Song, and Yuxiong He. 2023. DeepSpeed-Chat: Easy, Fast and Affordable RLHF Training of ChatGPT-like Models at All Scales. *arXiv preprint arXiv:2308.01320*.

Li Zhang, Hainiu Xu, Yue Yang, Shuyan Zhou, Weiqiu You, Manni Arora, and Chris Callison-Burch. 2023. Causal reasoning of entities and events in procedural texts. In *Findings of the Association for Computational Linguistics: EACL 2023*, pages 415–431, Dubrovnik, Croatia. Association for Computational Linguistics.

| | |
|---|---|
| SIM code input | facts = ['Raymond and Samantha are cousins.', 'Raymond was born 6 years before Samantha.', 'Raymond had a son at the age of 23.', 'If Samantha is now 31,']<br>question = "how many years ago was Raymond's son born?"<br>answer = None<br># call select_facts and infer_new_fact until answer is set<br>selected = select_facts(facts) # |
| Linearized input | QUESTION = sent1: Raymond and Samantha are cousins. sent2: Raymond was born 6 years before Samantha. sent3: Raymond had a son at the age of 23. sent4: If Samantha is now 31, sent5: how many years ago was Raymond's son born?<br>PROOF = |

| | Gold | GPT-3 | BLOOMZ-tcc | BLOOMZ |
|---|---|---|---|---|
| SIM code output | [facts[1], facts[2]]<br>new_fact = infer_new_fact(selected) # "When Raymond's son was born Samantha was 23 - 6 = 17 years old."<br>is_solved(new_fact, question) # False<br>facts.append(new_fact)<br><br>selected = select_facts(facts) # [facts[3], facts[3], facts[4]]<br>new_fact = infer_new_fact(selected) # "Thus it has been 31 - 17 = 14 years since Raymond's son was born."<br>is_solved(new_fact, question) # False<br>facts.append(new_fact)<br><br>selected = select_facts(facts) # [facts[3], facts[5]]<br>new_fact = infer_new_fact(selected) # 'The answer is 14'<br>is_solved(new_fact, question) # True<br>answer = extract_answer(new_fact) # '14' | [facts[1], facts[2]]<br>new_fact = infer_new_fact(selected) # 'Raymond's son was born 23 - 6 = 17 years ago.'<br>is_solved(new_fact, question) # False<br>facts.append(new_fact)<br><br>selected = select_facts(facts) # [facts[3], facts[4]]<br>new_fact = infer_new_fact(selected) # 'Samantha is 31, so Raymond's son was born 31 - 17 = 14 years ago.'<br>is_solved(new_fact, question) # False<br>facts.append(new_fact)<br><br>selected = select_facts(facts) # [facts[3], facts[5]]<br>new_fact = infer_new_fact(selected) # 'The answer is 14'<br>is_solved(new_fact, question) # True<br>answer = extract_answer(new_fact) # '14' | [facts[0]]<br>new_fact = infer_new_fact(selected) # facts[2]<br>is_solved(new_fact, question) # False<br>facts.append(new_fact)<br><br>selected = select_facts(facts) # [facts[0], facts[2]]<br>new_fact = infer_new_fact(selected) # facts[2]<br>is_solved(new_fact, question) # True<br>facts.append(new_fact)<br><br>selected = select_facts(facts) # [question, facts[2]]<br>new_fact = infer_new_fact(selected) # facts[2]<br>is_solved(new_fact, question) # True<br>answer = extract_answer(new_fact) # '23' | new_fact = infer_new_fact(selected) # facts[2]<br>is_solved(new_fact, question) # False<br>facts.append(new_fact)<br><br>###<br>selected = select_facts(facts) # [facts[0], facts[1], facts[2], facts[3]]<br>new_fact = infer_new_fact(selected) # facts[2]<br>is_solved(new_fact, question) # True<br>facts.append(new_fact)<br>*### repeated 4 times, without a parseable answer* |
| linearized output | sent2 & sent3 ->int1: When Raymond's son was born Samantha was 23 - 6 = 17 years old.; int1 & sent4 & sent5 ->int2: Thus it has been 31 - 17 = 14 years since Raymond's son was born.; int2 & sent5 ->int3: The answer is 14; | sent2 & sent4 ->int1: Raymond was born 6 years before Samantha, so Samantha is now 31 - 6 = 25 years old.; int1 & sent3 ->int2: Raymond had a son at the age of 23, so Raymond's son was born 23 - 6 = 17 years ago.; int2 & sent5 ->int3: The answer is 17; | sent1 & sent2 ->int1: Raymond was born 6 years before Samantha, so Raymond was born 6 years before Samantha.; int1 & sent2 & sent3 ->int2: Raymond had a son at the age of 23, so Raymond was 23 years old when he had a son.; int1 & int2 & sent3 ->int3: Samantha is now 31, so Samantha was 31 years old when she had a son.; int1 & int3 & sent4 ->int5: Thus, Raymond had a son 6 years ago.; int1 & int4 & sent5 ->int6: The answer is 6 years.; | sent1 & sent2 & sent3 ->int1: Raymond was born 6 years before Samantha, so Raymond was born 6 years before Samantha's son was born.; int1 & int2 & sent3 & sent4 ->int2: Samantha is now 31, so Samantha's son was born 6 years ago.; int1 & int2 & int3 & sent4 ->int5: The answer is 6 years.; |

Table 3: Case study for a GSM8K entry, in English. The correct answer is '14' (in green). Only GPT-3 with SIM prompts gets it correct.

# A Further Discussion

We find that regardless of answer correctness, BLOOMZ-based models often fail to generate new text, instead of copying text from the input. This again is likely due to the weaknesses of BLOOMZ, as this is not observed for GPT-3 with any format. Our use of both interventions, greatly reduces the incidence of this problem, which as we have discussed leads to BLOOMZ-TCCbetter eliciting the model's underlying reasoning abilities.

## A.1 BLOOM Results for GSM8K, AQUA_RAT, AR_LSAT

These results are shown in Appendix Figure 8.

For all tasks, performance is around random chance. For GSM8K, random chance is 0, and the models fails to solve nearly any math problem. While all numbers are close and likely statistically insignificant, we see that BLOOMZ-TCC slightly underperforms base BLOOMZ, and linearized and code prompts perform similarly.

Our hypothesis on why this happens, as discussed before, builds on the view that truly complex reasoning capabilities are emergent with LLM's model scale. The 7B BLOOMZ model used has no baseline ability for these 3 tasks (while it did for ARC), and therefore our interventions, which are indirect supervision on code, cannot help elicit better reasoning.

We discuss the two interventions separately. First, we study the effectiveness of code prompts on larger LLMs in §7.2. Second, for the finetuning recipe, we draw some initial points in Appendix A, given our resource constraints on small LLMs.[16]

This suggests limitations to the code+reasoning hypothesis, which have not been adequately discussed in prior work. Indirectly supervising LLMs for reasoning by training on code is effective for specific types of reasoning, such as ARC's commonsense reasoning, and less so for math problems like GSM8K, though, intuitively, code probably does not help, given code rarely includes arithmetic

[16]With 192 GB vRAM, we could finetune at most 7B multilingual models (which have much larger vocabulary sizes and thus larger embeddings). We leave future work to use our recipe with larger models, such as by using 4-bit quantization.

| | |
|---|---|
| SIM code input | facts = [ 'القمر هو الجسم السماوي الأقرب إلى الأرض من الشمس',<br>'جاذبية القمر على محيطات الأرض تتسبب في المد والجزر',<br>'جاذبية الشمس على محيطات الأرض تتسبب في المد والجزر',<br>'«كلما انخفضت المسافة من جسم ما، تزداد قوة الجاذبية على هذا الجسم<br>question = 'يتأثر المد والجزر على الأرض بالقمر أكثر من الشمس لأن القمر<br>choices = {'A': 'أكثر إشراقًا من الشمس.', 'B': 'أكبر من الأرض.', 'C': 'أبرد من الأرض.', 'D': 'أقرب من الشمس.'}<br>answer = None<br># call select_facts and infer_new_fact until answer is set<br>selected = select_facts(facts) # " |
| Linearized input | QUESTION = sent1: القمر هو الجسم السماوي الأقرب إلى الأرض من الشمس sent2: جاذبية القمر على محيطات الأرض تتسبب في المد والجزر sent3: جاذبية الشمس على محيطات الأرض تتسبب في المد والجزر sent4: «كلما انخفضت المسافة من جسم ما، تزداد قوة الجاذبية على هذا الجسم sent5: يتأثر المد والجزر على الأرض بالقمر أكثر من الشمس لأن القمر sent6: A) أكثر إشراقًا من الشمس. sent7: B) أكبر من الأرض. sent8: C) أبرد من الأرض. sent9: D) أقرب من الشمس.<br>PROOF = |

| | Gold | GPT-3 |
|---|---|---|
| SIM code output | [facts[0], facts[3]]<br>new_fact = infer_new_fact(selected) # 'قوة جاذبية القمر على الأرض أكبر من جاذبية الشمس على الأرض'<br>is_solved(new_fact, choices) # False<br>facts.append(new_fact)<br>selected = select_facts(facts) # [facts[1], facts[2], facts[4]]<br>new_fact = infer_new_fact(selected) # 'جاذبية القمر على الأرض لها تأثير على ظاهرة المد والجزر بشكل أكبر من جاذبية الشمس'<br>is_solved(new_fact, choices) # False<br>facts.append(new_fact)<br>selected = select_facts(facts) # [facts[5]]<br>new_fact = infer_new_fact(selected) # 'The answer is D'<br>is_solved(new_fact, choices) # True<br>answer = make_choice(new_fact, choices) # 'D'", | [facts[0], facts[1], facts[2], facts[3]]<br>new_fact = infer_new_fact(selected) # 'جاذبية القمر على الأرض تتسبب في المد والجزر بسبب قربه من الأرض'<br>is_solved(new_fact, choices) # False<br>facts.append(new_fact)<br>selected = select_facts(facts) # [facts[4]]<br>new_fact = infer_new_fact(selected) # 'The answer is D'<br>is_solved(new_fact, choices) # True<br>answer = make_choice(new_fact, choices) # 'D' |
| linearized output | sent1 & sent4 -> int1: قوة جاذبية القمر على الأرض أكبر من جاذبية الشمس على الأرض; sent2 & sent3 & int1 -> int2: جاذبية القمر على الأرض لها تأثير على ظاهرة المد والجزر بشكل أكبر من جاذبية الشمس; int2 -> int3: The answer is D); | sent1 & sent2 -> int1: القمر هو الجسم السماوي الأقرب إلى الأرض من الشمس; sent3 & sent4 -> int2: جاذبية الشمس تسبب المد والجزر ويسبب المد والجزر; sent5 & sent6 -> int3: القمر يسبب المد والجزر أكثر من الشمس لأنه أكثر إشراقًا; int3 -> int4: The answer is A |

Table 4: Case study for a ARC entry, in Arabic. The correct answer is 'D' (in green). Only GPT-3 with SIM prompts gets it correct.

equations.[17] As for the logical reasoning problems of AR_LSAT, we defer study to future work applying our finetuning recipe to larger LLMs.

## B Hyperparameters

For the TCC finetuning recipe, we set maximum sequence length to 1024, set learning rate to 1e-5 (with 0.1 weight decay), do not use warm-up, and use cosine learning rate schedule. We trained for 2 epochs using a batch size of 3 and gradient accumulation over 20 steps. We set the LoRA layers dimension to 128. The implementation is done with the DeepSpeed-Chat framework (Yao et al., 2023) and the `transformers` library (Wolf et al., 2020).

## C Dataset Statistics

Statistics for TCC are shown in Appendix Table 5.

[17] Arithmetic reasoning can be improved by *directly* finetuning on math – Ni et al. (2022) achieve 19.5% on GSM8k on a 2.7B LLM with this approach.

| | |
|---|---|
| # files from source | 30000 |
| TCC # files/lang | 20289 |
| x6 langs | 121734 |
| # tokens per language | 55-60m |

Table 5: Statistics for the TCC dataset. The source dataset is the small subset of The Stack for {js, py, java}. The # of tokens is calculated with the BLOOMZ tokenizer, which has a 250k vocabulary size including tokens from multiple natural and programming languages.

## D Full Results

We now report results of all experiments and settings studied in this work. Table 6 gives results for all models and prompt formats, on STREET and xSTREET (averaged across 5 languages). Table 7 gives per-language xSTREET results. We first discuss BLOOMZ and GPT-3, which were analyzed in the paper, then separately discuss Falcon.

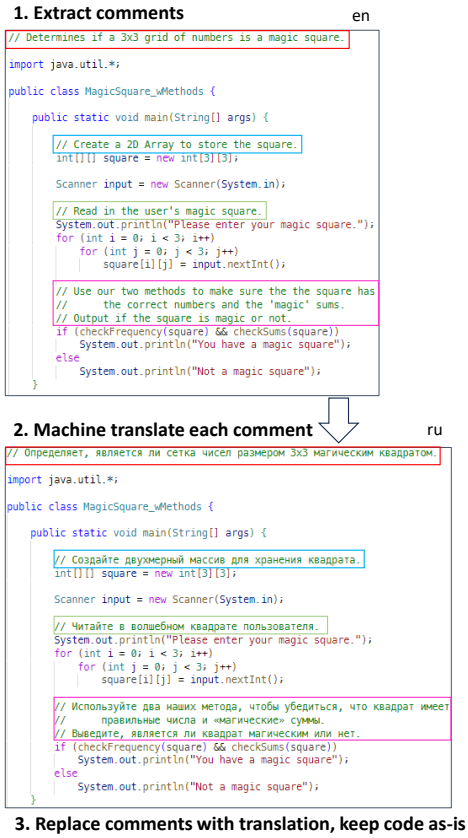


Figure 7: The data augmentation approach used to create the TCC (translated code comments) dataset.

### D.1 Direct Prompting

These tables include the direct prompting strategy, in which the model is given the same input as linearized, but needs to generate only the answer without intermediate reasoning.

For ARC, we find that, surprisingly, direct outperforms linearized (all LLMs). This is likely because the ARC questions are relatively easy already, and directly solving them given the context as well is possible. As discussed earlier, linearized format is artificial and hard to follow, which causes many reasoning graphs without valid answers. In contrast, SIM prompts are captured in code, which the models have seen, and therefore SIM results outperform direct and linearized.

For GSM8k and AQUA_RAT (using GPT-3), direct prompting fails. Using intermediate reasoning, as found by many prior works, is essential for arithmetic problem-solving. Linearized boosts performance significantly, and SIM code even further.

Regarding xSTREET, overall trends are relatively consistent with those discussed above.

### D.2 Results for Falcon

Falcon is an open-source LLM that is more performant than BLOOMZ, albeit English-centric. We chose the 40B instruction-finetuned variant, intermediate between BLOOMZ 7B and GPT-3 175B.[18]

First, we consider STREET results. For ARC, Falcon fails with direct prompts (34.7), but does much better with linearized (76.5) and SIM (81.5). For GSM8k, now that the base model has some math ability with direct prompts (4.4), it can improve with linearized (28.9) and SIM (19.6). SIM underperforming linearized here is because of a context size issue.

Falcon has a max context length of 2048 tokens, while GPT-3 and BLOOMZ can accept up to 4096. SIM code prompts use a lot of tokens for the code structure, and therefore, Falcon will run out of tokens quickly and therefore fail to generate a full reasoning graph in more cases than when using linearized. This is more so a limitation of Falcon than our work (recall that most prior work considers complex reasoning tasks with huge closed-source models).

For AQUA_RAT, Falcon performance is near random (20) for all 3 prompt formats; i.e., it is "too hard". For AR_LSAT, Falcon is near random (20) for direct and linearized, but achieves 34.0 with SIM prompts.

**Multilingual performance** Even though Falcon is an English-centric LLM, we evaluate its performance on xSTREET. We see that linearized performs the best across all tasks, with code prompts behind, and direct even further behind. Again, we attribute this to Falcon's shorter context length of 2048 – which is especially non-optimal for 4 of 5 languages studied which do not use the Latin script. The Falcon tokenizer did not see these scripts, resulting in byte-level tokenization, which further uses up the budget.

## E Additional Experiments

We perform 2 additional ablation experiments below.

### E.1 Finetuning on SIM Code Prompts

We experiment with directly finetuning on SIM code prompts (all 6 languages), so as to have a

[18] BLOOMZ has 7B and 176B variants, but nothing in between.

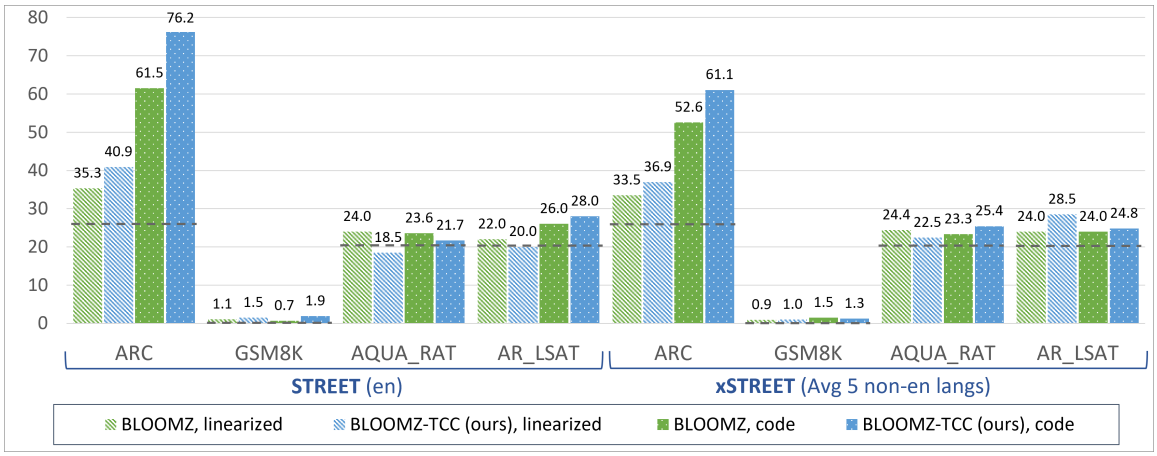


Figure 8: Results on GSM8K, AQUA_RAT, AR_LSAT subtasks for BLOOMZ-based models, with (up-to) 5-shot prompts. Random baselines are indicated with dashed grey lines. For each task, we report results 1) for, and 2) averaged over the 5 languages.

| | | STREET (English) | | | | xSTREET (average 5 languages) | | | |
|---|---|---|---|---|---|---|---|---|---|
| | Model | ARC | GSM8k | AQUA_RAT | AR_LSAT | ARC | GSM8k | AQUA_RAT | AR_LSAT |
| direct | BLOOMZ | 54.7 | 2.6 | 24.8 | 20.0 | 41.7 | 1.7 | 21.1 | 20.8 |
| | BLOOMZ-TCC (ours) | 70.3 | 3.3 | 20.9 | 24.0 | 49.0 | 1.8 | 21.9 | 23.2 |
| | falcon-40b-instruct | 34.7 | 8.5 | 24.8 | 26.0 | 32.0 | 6.1 | 24.6 | 24.0 |
| | GPT-3 | 97.6 | 4.4 | 21.7 | 22.0 | 90.7 | 1.6 | 25.6 | 24.4 |
| linearized | BLOOMZ | 35.3 | 1.1 | 24.0 | 22.0 | 33.5 | 0.9 | 24.4 | 24.0 |
| | BLOOMZ -TCC (ours) | 40.9 | 1.5 | 18.5 | 20.0 | 36.9 | 1.0 | 22.5 | 28.5 |
| | falcon-40b-instruct | 76.5 | 28.1 | 18.9 | 24.0 | 54.6 | 10.6 | 23.4 | 24.4 |
| | GPT-3 | 93.2 | 28.9 | 25.6 | 18.0 | 72.7 | 24.4 | 27.7 | 23.2 |
| | GPT-3 (Ribeiro et al., 2022)* | 72.9 | 34.8 | 40.2 | 19.0 | - | - | - | - |
| SIM code | BLOOMZ | 61.5 | 0.7 | 23.6 | 26.0 | 52.6 | 1.5 | 23.3 | 24.0 |
| | BLOOMZ -TCC (ours) | 76.2 | 1.9 | 21.7 | 28.0 | 61.1 | 1.3 | 25.4 | 24.8 |
| | falcon-40b-instruct | 81.5 | 19.6 | 24.4 | 34.0 | 43.9 | 4.6 | 23.5 | 26.8 |
| | GPT-3 | 99.1 | 45.2 | 37.8 | 32.0 | 94.2 | 26.9 | 32.8 | 24.8 |

Table 6: Full results (STREET and xSTREET average) for all models, tasks, prompt formats, and languages. Rows are grouped by prompt type and model, while columns are grouped by the subtask and the language.
*We include the reported results of Ribeiro et al. (2022) for STREET. This is for reference, as we cannot reproduce their exact prompts and examples chosen (and so cannot run on xSTREET).

| | | ARC | | | | | GSM8K | | | | | AQUA_RAT | | | | | AR_LSAT | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Model | es | zh | ar | ja | ru | es | zh | ar | ja | ru | es | zh | ar | ja | ru | es | zh | ar | ja | ru |
| direct | BLOOMZ | 46.5 | 41.2 | 57.4 | 33.5 | 30.0 | 2.2 | 1.9 | 1.1 | 1.1 | 2.2 | 19.7 | 24.8 | 21.3 | 24.0 | 15.7 | 22.0 | 24.0 | 16.0 | 24.0 | 18.0 |
| | BLOOMZ -TCC | 63.2 | 44.7 | 68.5 | 31.2 | 37.6 | 1.9 | 1.5 | 2.2 | 1.5 | 1.9 | 22.0 | 25.6 | 19.7 | 22.0 | 20.1 | 18.0 | 28.0 | 26.0 | 24.0 | 20.0 |
| | falcon-40b-instruct | 34.7 | 32.4 | 30.3 | 31.8 | 30.9 | 10.0 | 6.7 | 3.3 | 4.8 | 5.6 | 24.8 | 24.0 | 25.2 | 24.8 | 24.0 | 26.0 | 22.0 | 24.0 | 26.0 | 22.0 |
| | GPT-3 | 97.4 | 95.0 | 81.8 | 90.6 | 88.5 | 3.0 | 1.5 | 0.0 | 1.5 | 1.9 | 25.2 | 24.8 | 25.6 | 26.0 | 26.4 | 24.0 | 24.0 | 24.0 | 24.0 | 26.0 |
| linearized | BLOOMZ | 35.0 | 37.1 | 35.3 | 29.1 | 30.9 | 2.2 | 1.1 | 0.4 | 0.4 | 0.4 | 22.0 | 26.8 | 23.6 | 24.0 | 25.6 | 24.0 | 26.0 | 24.0 | 20.0 | 26.0 |
| | BLOOMZ-TCC | 44.7 | 40.3 | 39.7 | 30.9 | 29.1 | 1.5 | 0.4 | 1.5 | 0.4 | 1.1 | 17.3 | 21.7 | 21.3 | 22.8 | 24.0 | 14.0 | 30.0 | 28.0 | 28.0 | 28.0 |
| | falcon-40b-instruct | 77.4 | 65.6 | 34.7 | 49.4 | 45.9 | 23.3 | 16.3 | 2.2 | 7.0 | 4.4 | 25.2 | 25.2 | 21.7 | 22.0 | 22.8 | 18.0 | 26.0 | 24.0 | 18.0 | 36.0 |
| | GPT-3 | 83.5 | 75.6 | 57.4 | 69.7 | 77.4 | 26.3 | 29.6 | 19.3 | 17.0 | 30.0 | 27.6 | 27.6 | 29.5 | 26.4 | 27.6 | 26.0 | 28.0 | 20.0 | 26.0 | 16.0 |
| SIM code | BLOOMZ | 57.9 | 67.6 | 58.8 | 45.6 | 32.9 | 0.7 | 1.5 | 2.2 | 2.2 | 0.7 | 21.3 | 25.6 | 23.6 | 22.4 | 23.6 | 26.0 | 30.0 | 30.0 | 12.0 | 22.0 |
| | BLOOMZ-TCC | 70.3 | 74.4 | 65.0 | 54.7 | 40.9 | 0.4 | 1.5 | 1.5 | 2.2 | 0.7 | 27.2 | 24.0 | 28.3 | 22.8 | 24.8 | 26.0 | 34.0 | 16.0 | 30.0 | 18.0 |
| | falcon-40b-instruct | 57.1 | 55.3 | 30.3 | 44.1 | 32.6 | 8.1 | 8.1 | 1.9 | 2.6 | 2.2 | 26.0 | 24.4 | 23.2 | 20.5 | 23.2 | 32.0 | 26.0 | 26.0 | 24.0 | 26.0 |
| | GPT-3 | 96.5 | 96.5 | 90.3 | 94.1 | 93.5 | 37.4 | 28.1 | 18.5 | 21.9 | 28.5 | 36.6 | 32.7 | 31.1 | 32.7 | 31.1 | 26.0 | 28.0 | 26.0 | 18.0 | 26.0 |

Table 7: Per-language xSTREET results for all models, tasks, prompt formats, and languages. Rows are grouped by prompt type and model, while columns are grouped by the subtask and the language.

model that can perform the SIM-formatted reasoning without in-context examples. We use the same hyperparameter and configuration from §4.2, again using LoRA to fine-tune a subset of the 7B model's parameters, but omitting the data replay to maximize performance. We train one model for all tasks, and all 6 languages. This differentiates our SIM finetuned model from the linearized finetuned model of Ribeiro et al. (2022), which finetunes a separate T5 (0.8B) model for each task with full finetuning.

As multilingual trends remain similar, we will just discuss English results (STREET).Using BLOOMZ-TCC or BLOOMZ as the base model does not make a difference. The SIM finetuned models achieves 85.9 (vs. 76.2) on ARC; the other 3 tasks are still near random chance. We suspect that performing full finetuning instead of LoRA should overcome this, and omit this experiment due to resource constraints.

### E.2 Improving the Code Comment Quality of TCC

Our initial and used version of TCC, as described in the main text, simply took 30k source code files from The Stack, then filtered down to 20k files, using criteria of >5 comments, of which >50% of comments in English.

To validate the quality of the translated code files, we recruited human annotators who were proficient programmers, and native speakers of each language. While overall, translations were judged to be reasonable, the main feedback points were:

- Some files had non-useful comments, such as long copyright statements in the header, or linting messages.
- Some files had comments which were actually commented-out code (i.e. unused functions).
- Terms related to programming, or referencing function or variable names in the code, were often mistranslated, if they should have been translated at all.

We leave the last point to future work, as we used an online MT API, and programming-specific MT is out of scope. For the other two, we tried to develop a version of TCC to specifically select files which have plenty of meaningful comments. We describe this filtering experiment below.

We now consider the entire Stack dataset,[19], instead of just 30k from the official small subset. As the Stack totals 6 TB, we considered only the first 3 million examples (1m each for Java, Python, JavaScript). Our scripts performed the following steps in order:

1. Delete copyrights, headers, linting comments from files.
2. Keep only those files with >1 standard deviation of number of comments:number of lines ratio. Note that 1 comment can span multiple lines.
3. Keep only those files with >5 comments.

This resulted in about 250K examples. We performed the code comment extraction and translation process, and termed the resulting dataset TCC-v2. We then applied the finetuning recipe as we did with the original TCC; furthermore, we keep the data size consistent as the original, using a 67k subset of TCC-v2. The resulting BLOOMZ-TCC-v2 models had similar downstream reasoning performance as BLOOMZ-TCC, and therefore we did not use it in the main text.

We hypothesize this experiment did not improve performance because, the program code plays a much bigger role in LLM's reasoning abilities than the comments.

[19] https://huggingface.co/datasets/bigcode/the-stack-dedup