Bridging the Language Gap: Enhancing Multilingual Prompt-Based Code Generation in LLMs via Zero-Shot Cross-Lingual Transfer

Mingda Li, Abhijit Mishra, Utkarsh Mujumdar

School of Information, University of Texas at Austin

{mingdali, abhijitmishra, utkarsh.mujumdar@utexas.edu}@utexas.edu

Abstract

The use of Large Language Models (LLMs) for program code generation has gained substantial attention, but their biases and limitations with non-English prompts challenge global inclusivity. This paper investigates the complexities of multilingual prompt-based code generation. Our evaluations of LLMs, including CODELLAMA and CODEGEMMA, reveal significant disparities in code quality for non-English prompts; we also demonstrate the inadequacy of simple approaches like prompt translation, bootstrapped data augmentation, and fine-tuning. To address this, we propose a zeroshot cross-lingual approach using a neural projection technique, integrating a cross-lingual encoder like LASER (Artetxe and Schwenk, 2019) to map multilingual embeddings from it into the LLM's token space. This method requires training only on English data and scales effectively to other languages. Results on a translated and quality-checked MBPP dataset show substantial improvements in code quality. This research promotes a more inclusive code generation landscape by empowering LLMs with multilingual capabilities to support the diverse linguistic spectrum in programming.

1 Introduction

The use of Large Language Models (LLMs) for code generation, such as generating Python programs from problem specifications, has gained substantial interest due to their effectiveness in handling complex language tasks (Zhao et al., 2023; Gao et al., 2023; Austin et al., 2021). This capability has led to the development of innovative applications like GitHub Copilot (Yetistiren et al., 2022) and specialized LLMs such as CodeLLaMa (Roziere et al., 2023), underscoring the growing importance of this field. Although LLMs are globally prevalent and proficient in processing multilingual inputs, they often exhibit biases against non-English prompts (Talat et al., 2022; Choudhury and Deshpande, 2021), which is particularly

evident in code generation. Figure 1 under Baseline Output illustrates how the quality of generated code diminishes as prompts shift from English to other languages, a disparity linked to the availability of data used in LLM training and fine-tuning. Ensuring LLMs deliver consistent quality across languages is crucial for fostering fair and inclusive code generation, especially as the global programming community is increasingly composed of non-English speakers. Data from coding platforms like The Competitive Programming Hall of Fame¹ highlight the skew towards non-English-speaking regions. As the global population of coders grows, addressing these biases in LLMs is essential to promoting an equitable and accessible environment for developers worldwide.

This paper tries to bridge the language gap in multilingual prompt-based code generation by enhancing LLMs through self-supervised fine-tuning. We first evaluate the performance of LLMs like CODELLAMA and GPT-4 on English and five non-English languages, using a translated and qualitychecked version of the MBPP dataset (Austin et al., 2021). Significant disparities in code quality across languages were observed, even when using the same problem statements. Inspired by Shi et al. (2022a) and Qin et al. (2023), who improved LLM performance on multilingual tasks with Chain-of-Thought (CoT), and Awasthi et al. (2022), who used bootstrapping for multilingual data generation, we form strong baselines for code generation with CoT and fine-tuning on bootstrapped multilingual data. However, these approaches showed only marginal and inconsistent improvements.

To address data sparsity and limited multilingual exposure, we propose a novel approach: (a) using a pre-trained multilingual encoder like LASER (Artetxe and Schwenk, 2019) to encode multilingual inputs into a joint vector space; (b) projecting these embeddings into the LLM's input space

¹https://cphof.org/countries

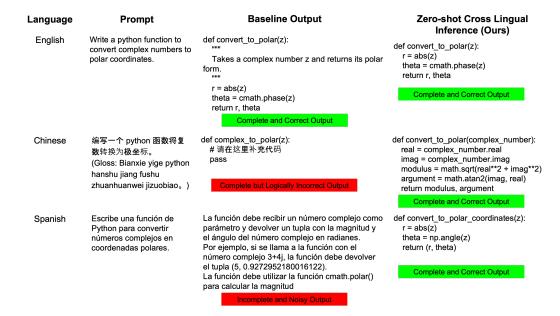


Figure 1: Disparity in output code generated by *CodeLLaMa-Instruct* model(Roziere et al., 2023) with 7B parameters for the same problem statement given in multiple languages

and aligning them through training solely on English data; and (c) employing this LASER→LLM pipeline at inference for zero-shot cross-lingual processing of non-English inputs. This method familiarizes models with multiple languages without needing additional external training data. Our evaluation shows improved code quality and reduced syntax and logical errors, as illustrated in Figure 1 under *Zero-shot Cross-Lingual Inference*. Our approach integrates seamlessly as a minor training step that does not require any expensive pretraining or fine tuning, thus offering a promising way to enhance LLMs' multilingual capabilities in code generation.

The contributions of the paper are as follows:

- We create a novel multilingual test dataset with quality-checked translations and new evaluation metrics.
- 2. We introduce a scalable projection technique by integrating the LASER multilingual encoder with popular open-source LLMs like CodeLLaMa (Roziere et al., 2023), CodeGemma (Team, 2024), and Mistral (Jiang et al., 2023) for zero-shot cross-lingual code generation.
- 3. Our evaluation of the approach against Chainof-Thought (CoT) and fine-tuning with multilingual bootstrapped data, highlights the strengths and limitations of each method.

We will make our code² and multilingual evaluation data³ publicly available for academic use.

2 Related Work

As AI technology advances, transformer-based LLMs like GPT (OpenAI, 2023), LLaMA (Touvron et al., 2023), Mistral (Jiang et al., 2023), and Gemma (Team et al., 2024) have become prominent in research and applications. While pre-trained models such as LLaMA2 and Gemma are finetuned for code generation, their English-centric training data limits multilingual proficiency (Lai et al., 2023; Akiki et al., 2022). Studies show these models face performance issues with non-English tasks (Shi et al., 2022b; Becker et al., 2023), and human supervision remains crucial for quality (Sarsa et al., 2022).

To address these gaps, Ahuja et al. (2023) developed a multilingual benchmark for evaluating LLMs, revealing performance drops across languages. Tan and Golovneva (2020) and Huang et al. (2023) suggest leveraging transfer learning and cross-lingual prompting to improve multilingual capabilities. Additional methods include language-specific pre-training (Pfeiffer et al., 2022) and consistency regularization for fine-tuning (Zheng et al., 2021).

Optimizing prompts enhances multilingual LLM

²https://github.com/lmd0420/Multilingual_Code_Gen

³https://huggingface.co/datasets/Mingda/MBPP-Translated

accuracy (Zhao and Schütze, 2021; Huang et al., 2022), and CoT techniques improve code generation (Ma et al., 2023). Fine-tuning with multilingual synthetic data, including translation and back-translation (Sennrich et al., 2015; Hoang et al., 2018), further refines LLMs, as demonstrated by Li et al. (2023) and Zhang et al. (2024). Contrary to these popular approaches, we take an orthogonal route by using specialized multimodal encoders and lightweight projectors to bridge language gaps in popular LLMs. Our work is inspired by multimodal AI literature, integrating projection techniques for different modalities such as language, vision and speech (Liu et al., 2024; Fathullah et al., 2024; Beyer et al., 2024).

3 Experimental Setup

This section details our experimental setup, including the creation of a multilingual benchmark dataset, the models evaluated, and the metrics used. Our focus is on Python code generation from multilingual prompts, though the methods and insights are applicable to other languages and contexts.

3.1 Evaluation Dataset

To the best of our knowledge, datasets with multilingual prompts for program generation code are elusive. To address this, we adapted the Mostly Basic Programming Problems (MBPP) dataset (Austin et al., 2021), specifically the sanitized version with its "test" split, containing 257 problems with solutions and three test cases each. We translated these prompts into five languages—Chinese-Simplified (*zh-cn*), Spanish (*es*), Japanese (*ja*), Russian (*ru*), and Hindi (*hi*)—using the Google Translate API⁴, chosen for their diverse linguistic representation.

Translation quality was assessed by (a) expert bilingual speakers via *Amazon Mechanical Turk*, who rated translations as acceptable or not (Note: guidelines were provided for binary rating and consent was obtained to report the statistics in the paper), with results showing superior quality (see Table 1), and (b) GPT-4, which rated translations on a scale of 1 to 5. Table 2 presents GPT-4's ratings, again indicating that translations are of high-quality with high mean scores and low standard deviations.

Translation	A1	A2	Agreement (%)
en_es	0.94	0.96	89.69
en-hi	0.93	0.96	89.11
en_ja	0.93	0.96	89.88
en_ru	0.93	0.96	90.43
en_zh	0.94	0.96	90.79

Table 1: Human Evaluation of Translated Prompts. Two distinct bilingual speakers from MTurk rated translations with 1 (acceptable) or 0 (not acceptable) for each translation. A1 and A2 represent their average scores.

Lang. Pair	Average Rating	St.Dev			
en-hi	4.88	0.40			
en-es	4.90	0.48			
en-ru	4.95	0.30			
en-zh_cn	4.93	0.39			
en-ja	4.87	0.55			

Table 2: Average Rating and Standard Deviation for Translation from English to Other Languages

3.2 Models Used for Evaluation

We consider three open source variants of instruction tuned models for evaluation, namely CODEL-LAMA7B⁵, CODEGEMMA7B⁶ and MISTRAL-7B-V0.3⁷. These models are specialized versions of their base models to programming-related tasks. They have demonstrated greater efficacy at code generation, infilling, and debugging capabilities compared to the standard versions. We access the models are accessed from HuggingFace (http://huggingface.co) hub. For benchmarking, we use GPT-4 as the reference system (a.k.a Skyline) due to its proven effectiveness in various tasks, including code generation.

3.3 Inference Pipeline and Evaluation Metrics

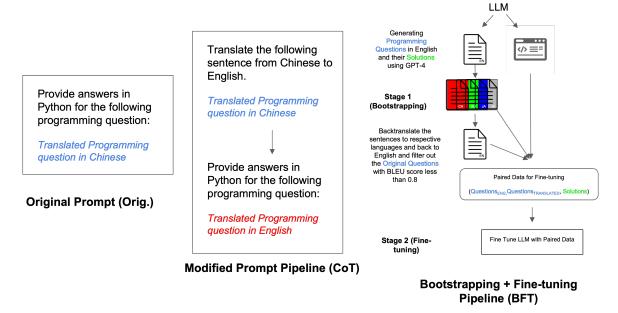
We developed a pipeline to process task descriptions in various languages. The pipeline feeds these prompts into the models and variants described in Sections 4 and 5 and stores the results. Python code from the outputs is extracted using regular expressions. We then identify function names in the code, replacing the function names in the MBPP test assertions with those from the model outputs. Finally, we generate bash scripts from the extracted code

⁴https://cloud.google.com/translate

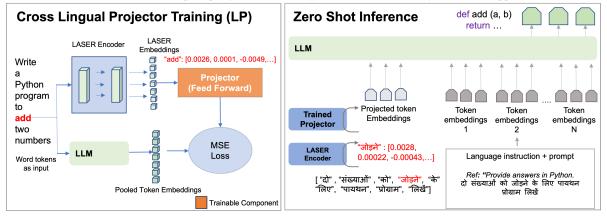
⁵codellama/CodeLlama-7b-Instruct-hf

⁶google/codegemma-7b-it

⁷mistralai/Mistral-7B-Instruct-v0.3



(a) Baselines with direct prompting, Chain of Thought (CoT) and fine-tuning with bootstrapped data



(b) Our proposed approach based on cross lingual encoder and projector training and zero shot inference

Figure 2: Explored approaches

and assertions and measure the following metrics:

- Logical Error Rate (LER): The ratio of code samples that execute successfully but produce incorrect results, to the total number of samples. Lower is better.
- Syntax Error Rate (SER): The ratio of code samples containing syntax errors, to the total number of samples. Lower is better.
- Total Error Rate (TotalER): The ratio of code samples that fail at least one test case, to the total number of samples. Lower is better.
- All Tests Passed Rate (ATPR): The ratio of code samples that pass all given test cases, to the total number of samples. Higher is better.

Additionally, we also observe the *Code Completion Rate* as a supplementary metric, which indicates the proportion of complete codes in model responses. A higher value represents a better result.

With this setup, we can now evaluate LLM code generation quality and propose mitigation strategies. Our approach and baselines are summarized in Figure 2, detailed in the following section.

4 Issues with Trivial Baselines

Given that language models exhibit emergent capabilities and scale effectively across tasks and languages, efficient prompting and prompt tuning are generally preferred over costly training or fine-tuning that demands extensive data curation. Based on our experimental setup, we highlight the challenges LLMs face with multilingual code genera-

tion in conventional settings, providing a detailed analysis of existing models' performance and their limitations. Throughout this section, we will reference Table 3 for a comprehensive discussion of the results.

4.1 Baseline 1. Original Prompt

Here, each query in the dataset is passed through the pipeline, where the model generates response code, filtered from extraneous information such as code explanations, and executed using an automatically constructed bash script. The results are presented in first column of each section of Table 3, with the following key observations:

GPT-4, recognized for its robustness and extensive engineering, reliably generates code across all language prompts, though with slightly varying error profiles-except for Hindi and Chinese. In contrast, open-source models like CodeLLaMa show more pronounced disparities between languages, with higher error rates and lower all-tests-passed rates compared to English. Notably, some models, such as CodeLLaMa-Instruct-7B, perform better in non-English languages like Spanish. This may seem unusual but aligns with findings from (Chen et al., 2024), which show that LLaMa 7B, when instruction-tuned for multilingual tasks, performs better in Spanish than English. Since CodeLLaMa is based on this instruction-tuned model, this could explain the atypical performance in Spanish. Overall, these results highlight a lack of consistency in code output quality as the language changes. We use the abbreviation Orig. to refer to this baseline henceforth.

4.2 Chain-of-Thought with Back-translation

Due to uneven language representation in LLM training datasets, achieving consistent results with direct prompting is challenging. A potential solution is to use back translation: translate non-English prompts into English and use the English version as the query. This Chain-of-Thought (CoT) approach involves translating the problem statement with the prompt: Translate the sentence \$PROBLEM from \$TARGET-LANG to English, generating code outputs from the translated prompt. Our experiments, detailed in the second column of Table 3, show that back translation did not significantly improve results. In some cases, it even reduced performance, as indicated by lower ATPR scores. Qualitative analysis

suggests that models struggle with non-canonical language representations and topic drift, despite the translations not being of poor quality. We use the abbreviation *CoT* to refer to this baseline henceforth.

4.3 Bootstrapping Multilingual Data and Fine Tuning

Fine-tuning pre-trained models is effective for many NLP tasks but is often resource-intensive, requiring costly and time-consuming task-specific labeled data. Instead of manually creating such data for multiple languages—designing prompts, validating answers, and translating while preserving semantic meaning—we use a bootstrapping approach. In this method, we utilize a powerful LLM like ChatGPT to generate English programming problems and their answers. These problems are then translated into target languages and backtranslated into English. We assess the similarity of translations using the BLEU score (Papineni et al., 2002), retaining translations that meet a quality threshold (e.g., 0.8) to create new training data. This method preserves text quality in target languages and allows the model to validate its translations, as detailed in Algorithm 1 under Appendix A.

After bootstrapping data for all target languages, we shuffle and use it to fine-tune the LLMs with a single A100 GPU. Models are quantized to FP16 and fine-tuned using parameter-efficient techniques, including low-rank adaptation (Hu et al., 2021). We set the temperature to 0.8 for consistency and use two epochs. As shown in the third column of Table 3, while bootstrapping with ChatGPT reduces syntax errors. It also increases hallucinations, leading to lower test pass rates and higher total errors. This suggests that the model, although producing more complete code, struggles with accuracy and reliability. We use the abbreviation *BFT* to refer to this baseline henceforth.

5 Our Approach: Projection-Based Zero-Shot Transfer

Our approach focuses on avoiding the use of inlanguage training data, which can be costly and impractical. Instead, we utilize an intermediate, lightweight method that relies on abundant English data and the LASER multilingual encoder (Artetxe and Schwenk, 2019), which provides joint embeddings for over 200 languages. In this setup, the LASER encoder preprocesses and embeds input tokens before passing them to the LLM, which then operates on these embeddings rather than raw input IDs. This method enables efficient language scaling, as similar meanings are represented consistently across languages (e.g., the English token "add" and its Hindi counterpart "JoDaNe" are embedded similarly, as shown in Figure 2 part (b)).

Two key challenges arise with this approach: (A) differing tokenization between the multilingual encoder and the LLM, and (B) the LLM's unfamiliarity with the multilingual embeddings. To address (A), we use word tokens and extract mean-pooled embeddings from subwords using tokenizers such as NLTK 8 for space sparated lanaguge inputs, Jieba 9 for Chinese, and Janome 10 for Japanese. We then train a projector to align these embeddings. For a given word token, we compute the LLM's subword embeddings (\hat{H}_{llm}) through max pooling, and the multilingual embeddings (H_{laser}) from the LASER encoder. The projector, with learnable parameters \mathbf{W}_{llm} and \mathbf{b}_{llm} , is defined as:

$$\mathbf{H}_{llm} = \mathbf{W}_{llm} \cdot \mathbf{H}_{laser} + \mathbf{b}_{llm}$$

The model is trained by minimizing the Mean Squared Error (MSE) between \hat{H}_{llm} and \mathbf{H}_{llm} :

$$MSE = \frac{1}{N} \sum_{i=1}^{N} \left\| \hat{H}_{llm}^{i} - \mathbf{H}_{llm}^{i} \right\|^{2}$$

where N is the number of word tokens. Training utilizes English tokens from the MBPP dataset, which includes 127 examples. We do projector training on a single consumer grade NVIDIA 4060 GPU and training the projector happens in 200 epochs in less than one hour. During inference, tokens are first word-tokenized and embedded using LASER, then projected, and finally input to the LLM for multilingual processing without requiring in-language data. To enhance performance and align with baselines, we also concatenate system prompt embeddings with the original programming prompt embeddings. Notably, LASER embeddings are of size 1024, while LLM embeddings are typically 4096 or larger, necessitating a 4-fold upsampling. We achieve this using two linear projection layers as outlined in the above equations. We use the abbreviation LP to explain this system hence-

6 Results and Discussions

Table 3 presents the overall performance models and variants discussed in sections 4 and 5. Our observations indicate that across all metrics, our proposed model consistently reduces the performance gap between English and non-English languages, as reflected in the differences and deviations. This improvement is particularly evident when comparing the direct querying setup (Orig.) with our multilingual projector-based variant (LP), where deviations from English are generally smaller. We explore the details of each metric below.

6.1 Total Error Rate (TotalER)

The Total Error Rate (TotalER) is an important metric that quantifies the overall error rate of the generated code. Our proposed method, LP, consistently achieves the lowest TotalER across nearly all languages and models, demonstrating its effectiveness. For example, with the CodeLLaMa-7B model, LP significantly reduces the TotalER to 75.49 for English (en) and 82.1 for Chinese (zh), outperforming the original model (Orig) and other methods. This improvement is especially pronounced in languages with complex syntax and morphology, such as Hindi (hi) and Russian (ru), where LP reduces the TotalER by over 10% in some cases compared to the original model. Even in cases where LP is the second-best, its performance is very close to the top-performing method, highlighting its reliability. In contrast, finetuning on multilingual bootstrapped data (BFT), a strong trivial baseline, tends to increase the TotalER due to hallucinations, as observed in our data analysis, despite slightly improving the all test cases passed metric.

6.2 Logical Error Rate (LER)

The Logical Error Rate (LER) is a critical component of the total error, measuring the proportion of code samples that execute without errors but produce incorrect results. A lower LER indicates a model's ability to generate logically sound code, making it a key metric for evaluating performance. It's important to note that we classify a logical error not only when no valid code is generated but also when any of the test cases fail.

Our approach, LP, consistently outperforms other methods in terms of LER, with only a few exceptions where the difference is marginal and still better than other candidates. For instance, with the CodeGemma-7B model, LP achieved an LER

⁸https://www.nltk.org

⁹https://github.com/fxsjy/jieba

¹⁰ https://mocobeta.github.io/janome/en/

LLM	Lang	TotalER↓			LER↓			SER↓				ATPR ↑					
			CoT	BFT	LP	Orig.	CoT	BFT	LP	Orig.	CoT	BFT	LP	Orig.	CoT	BFT	LP
GPT-4 (Skyline)	en	58.37	-	-	-	10.9	-	-	-	47.47	-	-	-	41.63	-	-	
	es	62.65	-	-	-	12.85	-	-	-	49.8	-	-	-	37.35	-	-	-
	hi	67.7	-	-	-	17.9	-	-	-	49.8	-	-	-	32.3	-	-	-
	ja	64.2	-	-	-	13.62	-	-	-	50.58	-	-	-	35.8	-	-	-
	ru	65.37	-	-	-	17.12	-	-	-	48.25	-	-	-	34.63	-	-	-
	zh	67.7	-	-	-	16.73	-	-	-	50.97	-	-	-	32.3	-	-	-
	en	87.16	-	82.1	75.49	63.04	-	28.79	22.57	24.12*	_	53.31	52.92	12.84	-	17.9	24.51
	es	79.77	91.83	81.71	81.71	28.8	56.81	26.07	24.9	50.97	35.02*	55.64	56.81	20.23	8.17	18.29	18.29
Code	hi	96.5	97.66	96.5	95.72	65.37	61.08	61.87	25.29	31.13	36.58	34.63	70.43	3.5	2.34	3.5	4.28
LLaMa-7B	ja	89.49	84.82	84.82	84.44	50.58	52.91	34.24	22.96	38.91	31.91	50.58	61.48	10.51	15.18	15.18	15.56
	ru	82.1	86.38	85.21	82.88	39.69	61.87	31.51	23.35	42.41	24.51	53.7	59.53	17.9	13.62	14.79	17.12
	zh	93.77	96.5	88.72	82.1	77.43	73.15	35.41	26.46	16.34	23.35	53.31	55.64	6.23	3.5	11.28	17.9
Code	en	82.1	-	92.22	77.04	41.63	-	63.04	25.68	40.47	-	29.18	51.36	17.9	-	7.78	22.96
	es	86.38	89.1	91.05	77.82	47.86	42.02	57.59	24.51	38.52	47.08	33.46	53.31	13.62	10.9	8.95	22.18
	hi	89.49	91.05	94.16	81.71	49.41	50.58	74.71	29.18	40.08	40.47	19.45	52.53	10.51	8.95	5.84	18.29
Gemma-7B	ja	83.66	90.27	91.05	79.77	38.91	44.75	50.58	24.13	44.75	45.52	40.47	55.64	16.34	9.73	8.95	20.23
	ru	85.99	88.72	89.1	77.04	42.41	48.25	59.53	25.68	43.58	40.47	29.57	51.36	14.01	11.28	10.9	22.96
	zh	84.82	86.38	93.0	79.38	39.68	48.64	62.26	28.02	45.14	37.74	30.74	51.36	15.18	13.62	7.0	20.62
Mistral -7B-v0.3	en	85.21	-	92.61	83.27	35.41	-	28.41	27.24	49.8	-	64.2	56.03	14.79	-	7.39	16.73
	es	87.55	86.38	94.94	84.82	39.69	29.18	26.46	26.06	47.86	57.2	68.48	58.76	12.45	13.62	5.06	15.18
	hi	91.44	91.05	98.83	92.22	35.41	35.41	24.12	30.74	56.03	55.64	74.71	61.48	8.56	8.95	1.17	7.78
	ja	88.72	86.77	96.11	87.55	35.8	31.91	28.02	22.57	52.92	54.86	68.09	64.98	11.28	13.23	3.89	12.45
	ru	85.6	84.05	95.33	84.05	33.85	30.74	26.85	24.13	51.75	53.31	68.48	59.92	14.4	15.95	4.67	15.95
	zh	88.72	87.16	94.55	84.05	39.3	30.74	26.07	26.85	49.42	56.42	68.48	57.2	11.28	12.84	5.45	15.95

Table 3: Comprehensive comparison of different models across multiple languages and configurations. TotalER: Total Error Rate, LER: Logical Error Rate, SER: Syntax Error Rate, ATPR: All Test Passed Rate. *Orig:* Directly Querying LLMs, *CoT*: Chain of Thought with Translation, *BFT*: Fine tuning on Bootstrapped Multilingual Data, *LP* (Our approach): Fine tuning on Multilingual Projection with LASER Encoders

of 25.68 for English, significantly lower than the 41.63 in Orig and 63.04 in bootstrapped multilingual fine tuning (BFT). This trend is also evident in other languages, such as Spanish (es) and Japanese (ja), where LP substantially reduces LER, underscoring its effectiveness in ensuring logical correctness across multilingual scenarios.

6.3 Syntax Error Rate (SER)

The Syntax Error Rate (SER) is a component of total error and indicates the proportion of code samples that contain syntax errors. A lower SER reflects the model's ability to generate syntactically correct code. Our overall observations with respect to this metric is that models like ours that often produce code than omitting it (which is indicated by the lower logical error) are more prone to syntax error due to the high recall. While syntax error solving is a crucial step in program debugging, we believe such a form of error is slightly easier to solve than logical errors. Thus, given that LP consistently achieves the lowest LER across all languages and models, we believe this demonstrates the LP's proficiency in helping with generating error-free code,

particularly in linguistically diverse contexts.

6.4 All Test Passed Rate (ATPR)

The All Tests Passed Rate (ATPR) measures the proportion of code samples that successfully pass all given test cases. A higher ATPR signifies greater reliability of the generated code, making it a crucial metric. Our observations show that LP consistently outperforms other methods in terms of ATPR across most cases. However, there are exceptions with the Mistral-7B-v0.3 model in a few languages. This model, being more recent, benefits from enhanced multilingual capabilities due to its diverse pretraining datasets and extended vocabulary. Overall, ATPR improvements are consistent across other languages, highlighting LP's superior performance in generating reliable and functional code

Our observations using Multilingual Projections with LASER Encoders reveal that LP not only reduces errors but also enhances the logical correctness and reliability of the generated code, establishing it as the leading approach for multilingual Python code generation. Additionally, we analyze

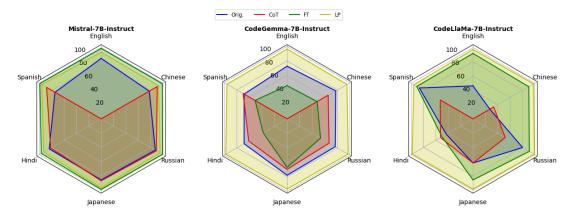


Figure 3: Code Completion Rate (CCR) for Models and Languages, with LP represented by perfect polygons, thus demonstrating between all languages and with highest surface area demonstrating higher CCR, often more than 90%

the Code Completion Rate (CCR) to assess the robustness of these models in generating meaningful code rather than nonsensical explanations across languages. LP consistently outperforms other variants in this regard, as shown in the spider graph in Figure 3. This graph illustrates LP's strong performance in producing complete code across all languages. Notably, the shapes representing LP in the graph are perfect polygons, reflecting its consistent behavior and reliability across different languages.

7 Conclusions and Future Work

In this paper, we demonstrated the significant potential of Large Language Models to bridge language gaps and promote inclusivity in multilingual prompt-based code generation. While LLMs exhibit promising capabilities across various languages, their performance can be inconsistent, particularly with non-English prompts. Our comprehensive analysis and evaluation using a benchmark dataset revealed both strengths and limitations in multilingual code generation, highlighting areas needing improvement.

We showcased the effectiveness of bootstrapping multilingual training data and fine-tuning LLMs to enhance code generation quality across multiple languages. Our zero-shot cross-lingual transfer approach, utilizing projected embeddings, proved effective, as evidenced by improved ATPR and reduced TotalER values. This method eliminates the need for extensive external multilingual data, maximizing the model's potential internally. Future work will expand this approach to include more languages, diverse prompt patterns, and programming languages beyond Python. Our findings underscore

the importance of advancing these techniques to enhance LLM adaptability and utility for a global audience, stressing the need for ongoing efforts to improve their effectiveness and versatility in diverse linguistic contexts.

8 Limitations

A major limitation of this work lies in the reliance on word tokenization and pooled token embeddings, which introduces external dependencies and may not scale effectively to extremely low-resource languages where tokenizers are not readily available. Furthermore, the sequence of projected embeddings from the target language can significantly differ from the canonical English order, potentially hindering the model's ability to fully leverage these embeddings. This misalignment could contribute to the generation of hallucinatory and erroneous outputs. To address this issue, some degree of finetuning of LLMs with denoising objectives may be necessary.

Moreover, our exploration is limited to only five non-English languages, which, while a promising start, is not comprehensive enough to establish the approach as a fully robust multilingual solution. Additionally, our study focuses solely on generating code from scratch and does not cover codefilling scenarios, which is another important aspect that warrants future exploration. Due to resource constraints, the scope of this study has been limited to Python, but expanding the approach to encompass other general-purpose and special-purpose programming languages is essential for broader applicability.

9 Ethical considerations

The models we utilized in our study are widely used ones from OpenAI, Google, MistralAI and Meta, and we employed Google Cloud Translator and the MBPP dataset on Hugging Face. All of these resources are publicly accessible; we did not introduce any additional real-world data, thus avoiding the creation of new ethical and privacy issues.

Given we are dealing with black-box Large Language Models as part of this study, there needs to be careful consideration of any potential biases that can be harmful in nature. Although we are focusing on a objective task with little to no opinion sourcing from the models, cultural and racial biases can occur given we are exposing the models to multi-lingual prompts. Since the applications we are focusing on are essentially user-centric in nature, a proper communication protocol should be established that can help clarify potential erratic behaviour of models, especially for low-resource languages. We would also like to share that we employed OpenAI's ChatGPT-4 system to enhance writing efficiency by generating LaTeX code, ensuring concise sentences, and aiding in error debugging.

References

- Kabir Ahuja, Harshita Diddee, Rishav Hada, Millicent Ochieng, Krithika Ramesh, Prachi Jain, Akshay Nambi, Tanuja Ganu, Sameer Segal, Maxamed Axmed, Kalika Bali, and Sunayana Sitaram. 2023. Mega: Multilingual evaluation of generative ai.
- Christopher Akiki, Giada Pistilli, Margot Mieskes, Matthias Gallé, Thomas Wolf, Suzana Ilić, and Yacine Jernite. 2022. Bigscience: A case study in the social construction of a multilingual large language model.
- Mikel Artetxe and Holger Schwenk. 2019. Massively multilingual sentence embeddings for zeroshot cross-lingual transfer and beyond. *Transactions of the association for computational linguistics*, 7:597–610.
- Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. 2021. Program synthesis with large language models. arXiv preprint arXiv:2108.07732.
- Abhijeet Awasthi, Nitish Gupta, Bidisha Samanta, Shachi Dave, Sunita Sarawagi, and Partha Talukdar. 2022. Bootstrapping multilingual semantic

- parsers using large language models. arXiv preprint arXiv:2210.07313.
- Brett A Becker, Paul Denny, James Finnie-Ansley, Andrew Luxton-Reilly, James Prather, and Eddie Antonio Santos. 2023. Programming is hard-or at least it used to be: Educational opportunities and challenges of ai code generation. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, pages 500–506.
- Lucas Beyer, Andreas Steiner, André Susano Pinto, Alexander Kolesnikov, Xiao Wang, Daniel Salz, Maxim Neumann, Ibrahim Alabdulmohsin, Michael Tschannen, Emanuele Bugliarello, et al. 2024. Paligemma: A versatile 3b vlm for transfer. *arXiv* preprint arXiv:2407.07726.
- Pinzhen Chen, Shaoxiong Ji, Nikolay Bogoychev, Andrey Kutuzov, Barry Haddow, and Kenneth Heafield. 2024. Monolingual or multilingual instruction tuning: Which makes a better alpaca.
- Monojit Choudhury and Amit Deshpande. 2021. How linguistically fair are multilingual pre-trained language models? In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 12710–12718.
- Yassir Fathullah, Chunyang Wu, Egor Lakomkin, Junteng Jia, Yuan Shangguan, Ke Li, Jinxi Guo, Wenhan Xiong, Jay Mahadeokar, Ozlem Kalinli, et al. 2024. Prompting large language models with speech recognition abilities. In *ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 13351–13355. IEEE.
- Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. Pal: Program-aided language models. In *International Conference on Machine Learning*, pages 10764–10799. PMLR.
- Cong Duy Vu Hoang, Philipp Koehn, Gholamreza Haffari, and Trevor Cohn. 2018. Iterative backtranslation for neural machine translation. In 2nd Workshop on Neural Machine Translation and Generation, pages 18–24. Association for Computational Linguistics.
- Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.
- Haoyang Huang, Tianyi Tang, Dongdong Zhang, Wayne Xin Zhao, Ting Song, Yan Xia, and Furu Wei. 2023. Not all languages are created equal in llms: Improving multilingual capability by cross-lingual-thought prompting.
- Lianzhe Huang, Shuming Ma, Dongdong Zhang, Furu Wei, and Houfeng Wang. 2022. Zero-shot cross-lingual transfer of prompt-based tuning with a unified multilingual prompt. *arXiv* preprint *arXiv*:2202.11451.

- Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023. Mistral 7b. arXiv preprint arXiv:2310.06825.
- Viet Dac Lai, Nghia Trung Ngo, Amir Pouran Ben Veyseh, Hieu Man, Franck Dernoncourt, Trung Bui, and Thien Huu Nguyen. 2023. Chatgpt beyond english: Towards a comprehensive evaluation of large language models in multilingual learning.
- Xian Li, Ping Yu, Chunting Zhou, Timo Schick, Luke Zettlemoyer, Omer Levy, Jason Weston, and Mike Lewis. 2023. Self-alignment with instruction backtranslation. *arXiv* preprint arXiv:2308.06259.
- Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. 2024. Visual instruction tuning. *Advances in neural information processing systems*, 36.
- Yingwei Ma, Yue Yu, Shanshan Li, Yu Jiang, Yong Guo, Yuanliang Zhang, Yutao Xie, and Xiangke Liao. 2023. Bridging code semantic and Ilms: Semantic chain-of-thought prompting for code generation. *arXiv* preprint arXiv:2310.10698.
- OpenAI. 2023. Gpt-4 technical report.
- Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the* 40th annual meeting of the Association for Computational Linguistics, pages 311–318.
- Jonas Pfeiffer, Naman Goyal, Xi Lin, Xian Li, James Cross, Sebastian Riedel, and Mikel Artetxe. 2022. Lifting the curse of multilinguality by pre-training modular transformers. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3479–3495, Seattle, United States. Association for Computational Linguistics.
- Libo Qin, Qiguang Chen, Fuxuan Wei, Shijue Huang, and Wanxiang Che. 2023. Cross-lingual prompting: Improving zero-shot chain-of-thought reasoning across languages.
- Baptiste Roziere, 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.
- Sami Sarsa, Paul Denny, Arto Hellas, and Juho Leinonen. 2022. Automatic generation of programming exercises and code explanations using large language models. In *Proceedings of the 2022 ACM Conference on International Computing Education Research-Volume 1*, pages 27–43.
- Rico Sennrich, Barry Haddow, and Alexandra Birch. 2015. Improving neural machine translation models with monolingual data. *arXiv preprint arXiv:1511.06709*.

- Freda Shi, Mirac Suzgun, Markus Freitag, Xuezhi Wang, Suraj Srivats, Soroush Vosoughi, Hyung Won Chung, Yi Tay, Sebastian Ruder, Denny Zhou, Dipanjan Das, and Jason Wei. 2022a. Language models are multilingual chain-of-thought reasoners.
- Freda Shi, Mirac Suzgun, Markus Freitag, Xuezhi Wang, Suraj Srivats, Soroush Vosoughi, Hyung Won Chung, Yi Tay, Sebastian Ruder, Denny Zhou, et al. 2022b. Language models are multilingual chain-of-thought reasoners. *arXiv preprint arXiv:2210.03057*.
- Zeerak Talat, Aurélie Névéol, Stella Biderman, Miruna Clinciu, Manan Dey, Shayne Longpre, Sasha Luccioni, Maraim Masoud, Margaret Mitchell, Dragomir Radev, Shanya Sharma, Arjun Subramonian, Jaesung Tae, Samson Tan, Deepak Tunuguntla, and Oskar Van Der Wal. 2022. You reap what you sow: On the challenges of bias evaluation under multilingual settings. In Proceedings of BigScience Episode #5 Workshop on Challenges & Perspectives in Creating Large Language Models, pages 26–41, virtual+Dublin. Association for Computational Linguistics.
- Lizhen Tan and Olga Golovneva. 2020. Evaluating cross-lingual transfer learning approaches in multi-lingual conversational agent models. *arXiv* preprint *arXiv*:2012.03864.
- CodeGemma Team. 2024. Codegemma: Open code models based on gemma. *arXiv preprint arXiv:2406.11409*.
- Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, et al. 2024. Gemma: Open models based on gemini research and technology. *arXiv* preprint arXiv:2403.08295.
- Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.
- Burak Yetistiren, Isik Ozsoy, and Eray Tuzun. 2022. Assessing the quality of github copilot's code generation. In *Proceedings of the 18th International Conference on Predictive Models and Data Analytics in Software Engineering*, pages 62–71.
- Shimao Zhang, Changjiang Gao, Wenhao Zhu, Jiajun Chen, Xin Huang, Xue Han, Junlan Feng, Chao Deng, and Shujian Huang. 2024. Getting more from less: Large language models are good spontaneous multilingual learners.
- Mengjie Zhao and Hinrich Schütze. 2021. Discrete and soft prompting for multilingual models. *arXiv* preprint arXiv:2109.03630.
- Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. 2023. A

survey of large language models. arXiv preprint arXiv:2303.18223.

Bo Zheng, Li Dong, Shaohan Huang, Wenhui Wang, Zewen Chi, Saksham Singhal, Wanxiang Che, Ting Liu, Xia Song, and Furu Wei. 2021. Consistency regularization for cross-lingual fine-tuning. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 3403–3417, Online. Association for Computational Linguistics.

A Appendix

Algorithm 1 Bootstrap Training Data

```
1: function BOOTSTRAPDATA(LLM,Lang)
 2:
         n \leftarrow \text{number of attempts}
 3:
         threshold \leftarrow 0.9
 4:
         Initialize a query set Q \leftarrow \{\}
 5:
         Initialize training data TD \leftarrow \{\}
 6:
         squery \leftarrow "Generate 100 python problems"
         trquery \leftarrow "Translate from English into Lanq"
 7:
 8:
         btrquery \leftarrow "Translate from Lang into English"
 9:
         for i \leftarrow 1 to n do
10:
              q \leftarrow \text{LLM}(squery)
             Push q into query set Q
11:
12:
         end for
13:
         for q in Q do
14:
              a \leftarrow \text{LLM}(q)
15:
             Push a into answer set A
16:
             Push < q, a > into TD
17:
         end for
         for q, a in TD do
18:
19:
             t \leftarrow \mathsf{LLM}(trquery, q)
20:
             bt \leftarrow \text{LLM}(btrquery, t)
21:
              score \leftarrow \text{BLEU}(t,bt)
22:
             if score > threshold then
23:
                  Push < t, a > into TD
24:
             end if
25:
         end for
26:
         return TD
27: end function
```