HinglishEval: Evaluating the Effectiveness of Code-generation Models on Hinglish Prompts

Mrigank Pawagi[0009−0002−6169−4766], Anirudh Gupta[0009−0008−3188−4634], Siddharth Reddy Rolla[0009−0008−6538−8372], and Kintan Saha[0009−0005−0897−5237]

Indian Institute of Science, Bengaluru, Karnataka, India
{mrigankp,anirudhgupta,siddharthrr,kintansaha}@iisc.ac.in

Abstract. Code-generation Models are Large Language Models (LLMs) that are fine-tuned to generate code from natural-language prompts. Prior work shows that such models can democratize programming by allowing novice programmers to generate accurate code for simple coding tasks by providing clear English-language prompts. In this paper, we explore whether this democratization can extend to novice programmers who lack proficiency in English but are able to craft clear prompts in another language. Specifically, we consider prompts in *Hinglish*, a mixture of Hindi and English that many students in India are comfortable with. We make two contributions. First, we propose a semi-automated technique to translate English prompts into Hinglish, and we use this technique to create HinglishEval: a Hinglish translation of the widely-used code-generation benchmark HumanEval. Second, we compare the performance of several popular open- and closed-source code-generation models on Hinglish and English prompts. Our findings suggest that although code-generation models are generally more effective at generating accurate code for English prompts, their efficacy with Hinglish prompts is promising.

Keywords: Large Language Models · Native Language Speakers · Novice Programmers

1 Introduction

In the evolving landscape of Generative AI, software development has been significantly influenced by the usage of Large Language Models (LLMs) such as GPT [1], Gemini [4], Llama [31], MistralAI [15], and Codegen [26]. Software developers are rapidly adopting AI coding assistants and have been seen to improve programmer efficiency [30, 27]. The structure of the prompt provided to these models plays a crucial role in the quality of the generated code [28]. Most models exhibit a high proficiency in understanding English, as their training datasets predominantly consist of English text [23]. Thus there is a significant gap in the performances of LLMs between using English and other languages [8].

However, for non-native English-speaking (NNES) users, particularly in India, there is a demand to prompt these models in a more familiar language, such

as Hinglish. Hinglish is a blend of Hindi and English that is often used in technical contexts where important terminology may not have natural Hindi translations. It is marked by the frequent use of English words in sentences with usual Hindi grammar. This linguistic mix is not unique to Hindi. Similar blends exist for other languages as well, like Spanglish (Spanish and English) and Chinglish (Chinese and English). The prevalence of English in technical literature and discourse leads to a natural integration of English terms in conversations across various languages, resulting in these hybrid language forms.

Since technical documentation is primarily available in English, NNES novice programmers find it challenging to adequately explain their queries while prompting LLMs for programming tasks like code generation, debugging, or code explanation [3]. Moreover, these programmers are also prone to misinterpreting LLM responses in a non-native language such as English. Addressing this language barrier by allowing novice programmers to prompt in their native languages can enhance both their learning experience as well the quality of their work on programming tasks.

To evaluate the performance of LLMs for code generation tasks, Chen et al. [9] proposed the pass@k metric to estimate whether at least one of the k LLM-generated solutions for a given prompt is "correct". In this context, a "correct" solution is one that *passes all the test cases*. We will utilize this metric[1] to evaluate LLMs on code generation from prompts in Hinglish. We will focus on prompts written in the Latin script since this is the most natural way of typing Hinglish.

2 Background

Compatibility with native languages is important for increasing access to Computer Science education. Several recent studies have demonstrated that LLMs can help close this gap [17, 2, 11, 24]. These models have become ubiquitous in recent times for their natural language and technical capabilities including code generation [16, 21, 29, 33].

2.1 Native Languages in the Classroom

Previous studies have shown that the use of native languages in educational settings is crucial for effective learning [12, 13, 3]. For example, 122 Indian languages were recognized by the national census in 2011 [19]. Yet, technical communication often takes place in English or its blends with regional languages. Writing prompts in English can be challenging for NNES novice programmers who are more comfortable with Hinglish or other such blends with Enlish. By supporting diverse languages, LLMs can eliminate the need for extensive translation of educational material and technical literature, by acting as an interface between such content and NNES students.

[1] In this paper, we will consider $k = 1$.

2.2 Large Language Models on Native Language Prompts

Recent work suggests that LLMs have the ability to understand and respond to queries in different languages [35, 38, 37, 25, 18]. However, these capabilities are limited to only a few languages. More linguistically diverse datasets are therefore required to train LLMs in adequately comprehending queries in other languages. Our work focuses on the code-generation abilities of LLMs and indicates that the performance of LLMs when prompted in Hinglish is significantly lower than when prompted in English. This suggests that it is still challenging for LLMs to understand problems specified in regional languages.

2.3 Related Work

The rise of LLMs has revolutionized the field of natural language processing. Their code generation and code explanation abilities [16, 29] have impacted several aspects of CS education, especially introductory programming courses [2, 24]. The multilingual abilities of these models [38, 18] have shown promise in helping narrow the gap in access to technology for NNES users. In particular, they have been shown to help NNES programming students [12, 13].

Previous work on evaluating LLMs for CS education has shown the utility of LLMs for both students as well as instructors [2, 24, 7]. In general, several benchmarks have been proposed to evaluate the programming capabilities of LLMs in various aspects of software engineering [9, 6, 36, 5, 22, 14, 20]. Our work builds upon previous work by developing a benchmark for evaluating code-generation by LLMs from prompts in a native language like Hinglish.

3 Approach

3.1 Creating a Benchmark

Our study builds upon the HumanEval dataset [9] which contains basic programming exercises for evaluating the code generation capabilities of LLMs. Each exercise in HumanEval consists of a prompt, a set of test cases, and a ground truth solution. The prompt is an incomplete Python function definition consisting of a function header along with a *docstring*. The docstring is a comment that contains a purpose statement for the function, and optionally some test cases called *doctests*. We translate these purpose statements to Hinglish to create our benchmarking dataset. This translation is performed semi-automatically as described in the following sections.

Generating base translation. We extract the docstring from each prompt and provide one-shot prompts to GPT-4 to translate them to Hinglish, as illustrated in Figure 1. This provides us *base* translations which can then be manually verified and corrected. We adopt this approach based on previous work demonstrating that human-AI collaboration increases the efficiency and quality of translation [34].

System: You are a translator fluent in both Hindi and English. Today, you will convert docstrings of Python functions from English to Hinglish, which is a conversational form of Hindi in which we use English for technical words related to syntax or code, programming concepts, and mathematics. Note that all text must be in the Roman script, like in the example.

User:
```
Given a positive integer n, return the product of the odd digits.
Return 0 if all digits are even. For example:
digits(1) == 1
digits(4) == 0
digits(235) == 15
```

Assistant:
```
Diye gaye positive integer n ke odd digits ka product return karo.
Agar saare digits even ho to 0 return karo. Jaise ki:
digits(1) == 1
digits(4) == 0
digits(235) == 15
```

User: `<docstring>`

Fig. 1: Our prompt to GPT-4 for translating docstrings. We provide a persona through our system prompt, and then provide an example for the translation. We replace `<docstring>` by the docstring to be translated.

Manual verification and correction of translations. A group of seven undergraduate students who can natively speak Hindi manually verified the base translations. Each *base* translation was verified by one student. In case of an incorrect translation, the student opened a pull request with the proposed correction. Pull request were reviewed by one or two other students and merged only once a consensus was reached. We will refer to the final dataset of Hinglish prompts thus created, together with the test cases and ground truth solutions from HumanEval, as HinglishEval.
Besides grammatical correctness, these corrections usually focused on making the translations idiomatic to native speakers. In total, 74 base translations were manually corrected with an average edit distance of 24.76 characters. Listings 1, 2, and 3 respectively illustrate an original docstring in HumanEval, its base translation, and its manually corrected and verified translation.

3.2 Evaluation

Experimental setup. We prompted each model under evaluation with both the English prompts in HumanEval as well as the Hinglish prompts in Hingli-

```
Write a function that accepts a list of strings.
The list contains different words. Return the word with maximum
number of unique characters. If multiple strings have maximum number
of unique characters, return the one which comes first in
lexicographical order.
...
```

Listing 1: Docstring in problem 158 of HumanEval.

```
Ek function likho jo strings ki list ko accept karta hai.
List mein alag alag shabd hote hain. Unique characters ki maximum
number wala shabd return karo. Agar multiple strings mein maximum
number of unique characters ho, toh lexicographical order mein
sabse pehle aane wala shabd return karo.
...
```

Listing 2: Base translation for docstring in problem 158 of HumanEval. The highlighted text was removed during manual correction.

```
Ek function likho jo strings ki ek list accept karta hai.
List mein alag alag words hain. Sabse zyada unique characters wala
word return karo. Agar multiple strings mein maximum number of
unique characters ho, toh lexicographical order mein sabse pehle a
ane wala word return karo.
...
```

Listing 3: Manually corrected translation of docstring in problem 158 of HumanEval. The highlighted text was added during manual correction.

shEval. We evaluated the generated code for each prompt by testing its equivalence with the corresponding ground truth solution over the provided test cases. We configured the models to greedy decoding (0 temperature) and set the maximum number of output tokens to 512. We found that no model produced code beyond 512 tokens for any problem. At 0 temperature, models behave deterministically and so we only generate one output for every prompt.

System prompt. We provided a system prompt to each model under evaluation in order to provide a persona to the model along with relevant instructions. This system prompt is shown in Listing 4. The system prompt was not provided to models that do not offer such a facility. The main prompt to the models was prefixed by the following: “Can you complete the following Python function?

Retain the function header and docstring." These prompts were provided in the appropriate format depending on the model under evaluation.

```
You are an experienced Python programmer. Complete the Python
functions from the given docstrings. Do NOT write anything except
the function definition. Avoid print and input statements.
```

Listing 4: System prompt for code generation.

4 Results

4.1 Performance Metrics

Pass@1. For each model, M under evaluation, we calculate the pass@1 values on HumanEval (denoted by E) and HinglishEval (denoted by H), denoted respectively by $P_1(M, E)$ and $P_1(M, H)$. Note that $P_1(M, D)$ is the fraction of problems in dataset $D \in \{E, H\}$ for which M generated *correct* code. Under our evaluation, the generated code for a given problem is *correct* if it is equivalent to the corresponding ground truth solution over the provided test cases. Further, we define Δ_M to be the difference in the pass@1 values of M between HumanEval and HinglishEval, i.e., $\Delta_M = P_1(M, E) - P_1(M, H)$. Table 1 lists the values of $P_1(M, E)$, $P_1(M, H)$ and Δ_M for various models. Figure 2 shows the ratio of $P_1(M, E)$ and $P_1(M, H)$ values for each model.

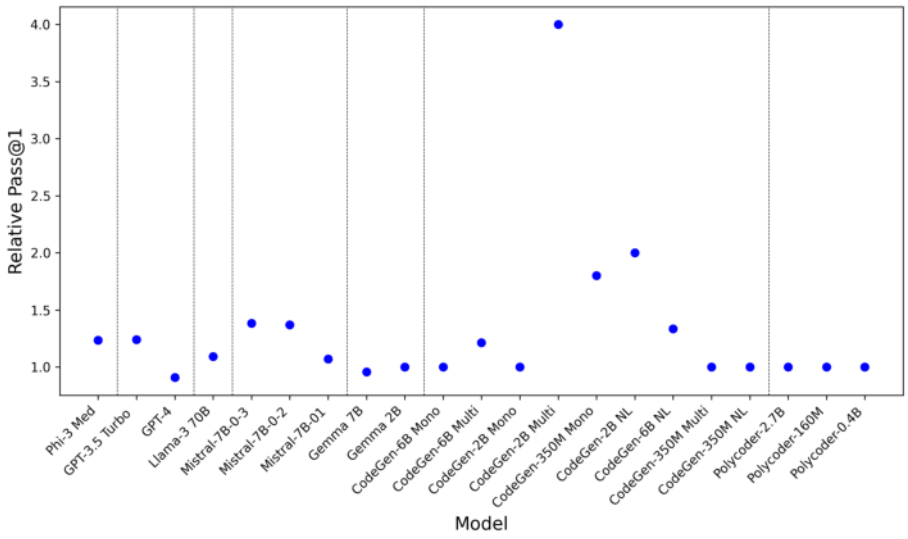

Fig. 2: Relative Pass@1 scores, i.e., $P_1(M, E)/P_1(M, H)$ values.

Model Family	Model (M)	$P_1(M, E)$	$P_1(M, H)$	Δ_M
Phi-3	Medium-4k Instruct	73.78	59.76	14.02
GPT	3.5-Turbo	72.56	58.54	14.02
	4	71.95	79.27	-7.32
Llama 3	70B	71.95	65.85	6.10
Mistral	7B-v0.3	39.63	28.66	10.98
	7B-v0.2	31.71	23.17	8.54
	7B-v0.1	27.44	25.61	1.83
Gemma	7B	13.41	14.02	-0.61
	2B	10.98	10.98	0.00
CodeGen	6B-Mono	15.24	15.24	0.00
	6B-Multi	10.37	8.54	1.83
	2B-Mono	9.76	9.76	0.00
	2B-Multi	7.32	1.83	5.49
	350M-Multi	5.49	3.05	2.44
	2B-NL	4.88	2.44	2.44
	6B-NL	4.88	3.66	1.22
	350M-Multi	3.05	3.05	0.00
	350M-NL	0.00	0.00	0.00
PolyCoder	2.7B	1.83	1.83	0.00
	160M	0.00	0.00	0.00
	0.4B	0.00	0.00	0.00

Table 1: Pass@1 scores of different models on the HumanEval and the HinglishEval datasets.

Item Response Theory. Item Response Theory (IRT) [10] is a psychometric model that utilizes statistical data to explain the relationship between intrinsic traits and observed outcomes. IRT has been employed to examine the usefulness of individual questions in standardized tests for human participants [32]. We utilize IRT to compare the relative performance of different models while accounting for each problem's difficulty and ability to discriminate models. This relative performance is called *latency*[2].

We assume that the difficulty of a problem specified in English (in HumanEval) is preserved during translation to Hinglish (in HinglishEval) and consider both versions to be the same problem. Therefore for every model M, we consider two *subjects* $E(M)$ and $H(M)$ which take the same problems in English and Hinglish respectively. We independently compare these two subjects from every model in

[2] Note that throughout our text, we will exclusively use this term in the context of IRT, and it should not be confused with the same term for the delay between prompting LLMs and receiving their output.

the same IRT evaluation. We will denote the latency of a subject, X by L_X. Note that a higher L_X indicates better performance of X relative to other subjects in the evaluation.

We implement IRT using the 2-Parameter Logistic (2PL) model, which computes the latencies of the subjects being evaluated using the difficulty and discrimination values of each problem. We did not use the three-parameter model which also accounts for a guessing parameter for each subject while computing latencies. This parameter captures the ability of the subjects to correctly guess answers to problems. Since we executed models with greedy decoding, the responses of the subjects were deterministic. Therefore, we did not include this parameter. Table 2 lists the latencies of all the subjects in our evaluation. Broadly, it can be observed that most models understand English prompts better than Hinglish prompts, even for the same problems.

Model Family	**Model (M)**	$L_{E(M)}$	$L_{H(M)}$
Phi-3	Medium-4k instruct	0.800	1.599
GPT	3.5-Turbo	0.800	0.716
	4	0.800	0.793
Llama 3	70B	0.801	0.800
Mistral	7B-v0.3	-0.781	-0.800
	7B-v0.2	-0.800	-0.816
	7B-v0.1	-0.800	-0.800
Gemma	7B	-1.66	-1.61
	2B	-2.36	-2.39
CodeGen	6B-Mono	-1.600	-1.612
	6B-Multi	-2.384	-2.426
	2B-Mono	-2.397	-2.391
	2B-Multi	-2.576	-3.955
	350M-Multi	-3.870	-3.827
	2B-NL	-3.414	-3.914
	6B-NL	-3.139	-3.584
	350M-Multi	-3.247	-3.827
	350M-NL	-3.985	-3.985
PolyCoder	2.7B	-3.888	-3.957
	160M	-3.985	-3.985
	0.4B	-3.985	-3.985

Table 2: IRT latencies of different subjects.

4.2 Analysis

The positive Δ_M values for all LLMs except GPT-4 and Gemma-7B indicate that these models perform poorly when prompted in Hinglish. A similar trend is seen by comparing the IRT latencies of the English and Hinglish subjects derived from each model. However, this gap is small and our evaluation demonstrates that both closed- and open-sourced models are promising in their utility for NNES novice programmers. We found that many models produced code with logical errors when prompted in Hinglish, even though they generated correct code from English prompts. This suggests a lack of comprehension of problem statements by these models when problems are specified in Hinglish. Listings 5, 7 and 6 illustrate a few such examples for problems with high difficulty values.

```
def words_string(s):
    """
    You will be given a string of words separated by commas or spaces.
    ↪  Your task is to split the string into words and return an array
    ↪  of the words...
    """
    words = s.split(', ')
    words = [word.strip() for word in words]
    return words
```

Listing 5: GPT-3.5-Turbo's response for the Hinglish version of problem 101 in HumanEval. The bug in this response can be fixed by changing the highlighted text to `s.replace(',', ' ').split()`. Note that we have replaced the Hinglish docstring with part of its original English version for the purpose of illustration.

```
def generate_integers(a, b):
    """
     Given two positive integers a and b, return the even digits between
     ↪  a and b, in ascending order...
    """
    return [i for i in range(min(a, b), max(a, b) + 1) if i % 2 == 0]
```

Listing 6: GPT-4's response for the Hinglish version of problem 163 in HumanEval. The bug in this response can be fixed by changing the highlighted text to `i % 2 == 0 and 0 <= i <= 9`. Note that we have replaced the Hinglish docstring with part of its original English version for the purpose of illustration.

```
def match_parens(lst):
    """
    You are given a list of two strings, both strings consist of open
    ↪  parentheses '(' or close parentheses ')' only. Your job is to
    ↪  check if it is possible to concatenate the two strings in some
    ↪  order, that the resulting string will be good. A string S is
    ↪  considered to be good if and only if all parentheses in S are
    ↪  balanced...
    """
    def is_balanced(s):
        ...
    return 'Yes' if is_balanced(lst[0] + lst[1]) else 'No'
```

Listing 7: Mistral-7B-v0.3's response for the Hinglish version of problem 119 in HumanEval. The bug in this response can be fixed by changing the highlighted text to `is_balanced(lst[0] + lst[1]) or is_balanced(lst[1] + lst[0])`. Note that we have replaced the Hinglish docstring with part of its original English version for the purpose of illustration.

5 Conclusion

This study evaluates the code-generation abilities of LLMs from prompts in regional languages. This provides a perspective on their usefulness for NNES users, particularly novice programmers who may use these models for assistance in programming tasks. We present a benchmark to track the performance of newer and upcoming language models on such tasks. One immediate direction for future work is to expand this benchmark to other regional languages and their blends with English. Further, our benchmark evaluates models on a relatively small number of Python code generation tasks. Future work can extend our approach to benchmarks involving other programming languages, other aspects of software engineering like code comprehension and debugging, or harder programming tasks. User studies involving students and instructors should also be conducted to better understand their interaction with LLMs in regional languages. Further studies can also explore the effect of different prompting strategies on the performance of LLMs with regional languages. Our benchmark, HinglishEval, all scripts and generated code-samples are publicly available at `github.com/mrigankpawagi/HinglishEval` to enable reproducibility and further work in this direction.

Acknowledgements

We thank the anonymous reviewers and Dr. Viraj Kumar for their valuable comments and suggestions for improving this paper, the Kotak-IISc AI-ML Centre for supporting this work, and Om Prakash Choudhary, Pratham Gupta and Adithya K Anil for their help in validating the translations.

References

[1] Josh Achiam et al. *GPT-4 Technical Report*. 2023. arXiv: 2303.08774.

[2] Vibhor Agarwal et al. *"Which LLM should I use?": Evaluating LLMs for tasks performed by Undergraduate Computer Science Students in India*. 2024. arXiv: 2402.01687.

[3] Suad Alaofi and Seán Russell. "Computer Terminology Test for Non-native English Speaking CS1 Students". In: *ACE*. 2022.

[4] Rohan Anil et al. *Gemini: A Family of Highly Capable Multimodal Models*. 2023. arXiv: 2312.11805.

[5] Ben Athiwaratkun et al. "Multi-lingual Evaluation of Code Generation Models". In: *International Conference on Learning Representations*. 2023.

[6] Jacob Austin et al. *Program Synthesis with Large Language Models*. 2021. arXiv: 2108.07732.

[7] Rishabh Balse et al. "Evaluating the Quality of LLM-Generated Explanations for Logical Errors in CS1 Student Programs". In: *COMPUTE*. 2023.

[8] Yejin Bang et al. "A Multitask, Multilingual, Multimodal Evaluation of ChatGPT on Reasoning, Hallucination, and Interactivity". In: *IJCNLP-AACL*. 2023.

[9] Mark Chen et al. *Evaluating Large Language Models Trained on Code*. 2021. arXiv: 2107.03374.

[10] Yunxiao Chen et al. *Item Response Theory–A Statistical Framework for Educational and Psychological Measurement*. 2021. arXiv: 2108.08604.

[11] Hagit Gabbay and Anat Cohen. "Combining LLM-Generated and Test-Based Feedback in a MOOC for Programming". In: *Learning @ Scale*. 2024.

[12] Philip J Guo. "Non-Native English Speakers Learning Computer Programming: Barriers, Desires, and Design Opportunities". In: *Conference on Human Factors in Computing Systems*. 2018.

[13] Carmen Nayeli Guzman, Anne Xu, and Adalbert Gerald Soosai Raj. "Experiences of Non-Native English Speakers Learning Computer Science in a US University". In: *SIGCSE Technical Symposium*. 2021.

[14] Dan Hendrycks et al. "Measuring Coding Challenge Competence With APPS". In: *Neural Information Processing Systems*. 2021.

[15] Albert Q Jiang et al. *Mistral 7B*. 2023. arXiv: 2310.06825.

[16] Juyong Jiang et al. *A Survey on Large Language Models for Code Generation*. 2024. arXiv: 2406.00515.

[17] Majeed Kazemitabaar et al. "How Novices Use LLM-based Code Generators to Solve CS1 Coding Tasks in a Self-Paced Learning Environment". In: *International Conference on Computing Education Research*. 2023.

[18] Aditi Khandelwal et al. "Do Moral Judgment and Reasoning Capability of LLMs Change with Language? A Study using the Multilingual Defining Issues Test". In: *Conference of the European Chapter of the Association for Computational Linguistics*. 2024.

[19] *Language Census Data*. URL: https://language.census.gov.in/showLanguageCensusData.

[20] Yujia Li et al. “Competition-Level Code Generation with AlphaCode”. In: *Science* (2022).
[21] Fang Liu et al. *Exploring and Evaluating Hallucinations in LLM-Powered Code Generation*. 2024. arXiv: 2404.00971.
[22] Jiawei Liu et al. “Is Your Code Generated by ChatGPT Really Correct? Rigorous Evaluation of Large Language Models for Code Generation”. In: *Neural Information Processing Systems*. 2024.
[23] Yang Liu et al. *Datasets for Large Language Models: A Comprehensive Survey*. 2024. arXiv: 2402.18041.
[24] Wenhan Lyu et al. “Evaluating the Effectiveness of LLMs in Introductory Computer Science Education: A Semester-Long Field Study”. In: *Learning @ Scale*. 2024.
[25] Junho Myung et al. *BLEnD: A Benchmark for LLMs on Everyday Knowledge in Diverse Cultures and Languages*. 2024. arXiv: 2406.09948.
[26] Erik Nijkamp et al. *A Conversational Paradigm for Program Synthesis*. 2022. arXiv: 2203.13474.
[27] Eirini Kalliamvakou. *Research: quantifying GitHub Copilots impact on developer productivity and happiness*. 2022. URL: https://github.blog/news-insights/research/research-quantifying-github-copilots-impact-on-developer-productivity-and-happiness/.
[28] Laria Reynolds and Kyle McDonell. “Prompt Programming for Large Language Models: Beyond the Few-Shot Paradigm”. In: *Conference on Human Factors in Computing Systems*. 2021.
[29] Claudio Spiess et al. *Calibration and Correctness of Language Models for Code*. 2024. arXiv: 2402.02047.
[30] *The world’s most widely adopted AI developer tool*. 2024. URL: https://github.com/features/copilot.
[31] Hugo Touvron et al. *Llama 2: Open Foundation and Fine-Tuned Chat Models*. 2023. arXiv: 2307.09288.
[32] *Understanding Item Analyses*. 2024. URL: https://www.washington.edu/assessment/scanning-scoring/scoring/reports/item-analysis.
[33] Yuvraj Virk, Premkumar Devanbu, and Toufique Ahmed. *Enhancing Trust in LLM-Generated Code Summaries with Calibrated Confidence Scores*. 2024. arXiv: 2404.19318.
[34] Wang, Lan. “The Impacts and Challenges of Artificial Intelligence Translation Tool on Translation Professionals”. In: *SHS Web Conf.* (2023).
[35] Zheng Xin Yong et al. “Prompting Multilingual Large Language Models to Generate Code-Mixed Texts: The Case of South East Asian Languages”. In: *Computational Approaches to Linguistic Code-Switching*. 2023.
[36] Hao Yu et al. “CoderEval: A Benchmark of Pragmatic Code Generation with Generative Pre-trained Models”. In: *ICSE*. 2024.
[37] Jun Zhao et al. *LLaMA Beyond English: An Empirical Study on Language Capability Transfer*. 2024. arXiv: 2401.01055.
[38] Yiran Zhao et al. *How do Large Language Models Handle Multilingualism?* 2024. arXiv: 2402.18815.