mHumanEval - A Multilingual Benchmark to Evaluate Large Language Models for Code Generation

Anonymous ACL submission

Abstract

Recent advancements in large language models (LLMs) have significantly enhanced code generation from natural language prompts. The HumanEval Benchmark, developed by OpenAI, remains the most widely used code generation benchmark. However, this and other Code LLM benchmarks face critical limitations, particularly in task diversity, test coverage, and linguistic scope. Current evaluations primarily focus on English-to-Python conversion tasks with limited test cases, potentially overestimating model performance. While recent works have addressed test coverage and programming language (PL) diversity, code generation from low-resource language prompts remains largely unexplored. To address this gap, we introduce mHumanEval[1], an extended benchmark supporting prompts in over 200 natural languages. We employ established machine translation methods to compile the benchmark, coupled with a quality assurance process. Furthermore, we provide expert human translations for 15 diverse natural languages (NLs). We conclude by analyzing the multilingual code generation capabilities of state-of-the-art (SOTA) Code LLMs, offering insights into the current landscape of cross-lingual code generation.

1 Introduction

LLMs have transformed software development with their ability to generate programming code from simple natural language instructions. LLMs are trained on extensive datasets that include diverse code samples, aiding programmers in code development and debugging. They also make programming more accessible to beginners. However, assessing the performance of these models across different coding tasks is still a major challenge. Comprehensive testing is essential to verify that these models are both effective and adaptable, rather than only performing well under specific conditions.

The most widely used benchmark for evaluating these models is OpenAI's HumanEval (Chen et al., 2021), which includes a collection of 164 tasks generated by human experts. Each task includes an English prompt, a canonical solution provided by the authors, and three test cases. Although this benchmark is commonly used, it has significant limitations, such as limited test coverage and minimal support for non-English and non-Python prompts. While recent variations (Peng et al., 2024; Cassano et al., 2023) of HumanEval address some of these issues, most do not include prompts in NLs other than English and, in particular, in low-resource NLs. Consequently, current benchmarks fail to provide key insights into the multilingual capabilities of LLMs in the context of code generation.

Figure 1 demonstrates one such example. While the widely used GPT3.5 (Brown et al., 2020) model performs perfectly for the original prompt "Write a Python code snippet that detects whether a year is a leap year or not.", it fails when the same prompt is given in a low-resource language (*Nyanja*, in this case).

```
# Chaka chomwe tikufuna kuyang'ana
# Yang'anani ngati chaka ndi chaka cha ziwalo
if (year % 4 == 0 and year % 100 != 0):
    # Ngati chaka chimagawika ndi 4 ndipo
    # sichimagawika ndi 100
    sindikiza(f"{year} ndi chaka cha ziwalo")
else:
    # Ngati sichigwirizana ndi zofunikira
    # za chaka cha ziwalo
    sindikiza(f"{year} si chaka cha ziwalo")
```

Figure 1: Code snippet generated by GPT3.5 when prompted to write a Python code to detect leap years in *Nyanja* language. Some Python keywords are transformed into *Nyanja* words, resulting in compilation issues.

Most LLMs, primarily pre-trained on large English corpora like Common Crawl, perform poorly on

[1] anonymous.4open.science/r/mHumanEval-0162/

multilingual tasks, further propagating inequalities in language technology access (Blasi et al., 2022). However, proprietary models like GPT-4 (Achiam et al., 2023) and Claude 3 (Anthropic, 2024), with undisclosed training data, show decent performance in multilingual scenarios. Peng et al. (2024) for instance show that GPT-4 excels in code generation even with mid-resource language prompts. The open-source community is also advancing with multilingual models like Aya (Üstün et al., 2024) and LLaMA 3. However, insights into their code generation performance in a massively multilingual setting are lacking due to the absence of comprehensive benchmarks.

In this work, we introduce `mHumanEval`, a novel multilingual code generation benchmark including coding prompts in 204 NLs and expert human translations for 15 NLs. `mHumanEval` further includes canonical solutions in 25 PLs, including 4 new PLs that are not covered by any prior benchmarks. The primary contributions of this paper are as follows:

1. The creation of `mHumanEval`, the first massively multilingual benchmark for code generation.
2. A translation quality evaluation for each prompt.
3. A thorough evaluation of existing SOTA Code LLMs using `mHumanEval`.

The paper addresses two research questions (RQs):

- **RQ1:** How do the code generation capabilities of LLMs vary when prompts are provided in English, or other high-, mid-, and low-resource NLs?
- **RQ2:** How does the performance of multilingual LLMs compare to specialized, fine-tuned Code LLMs in code generation tasks on the `mHumanEval` dataset?

Finally, we also report *secondary findings* related to the translation quality of machine translation (MT) methods on coding prompts.

2 Related Work

The most widely used benchmark dataset for evaluating Code LLMs is the aformentioned `HumanEval` (Chen et al., 2021). Another key benchmark is DeepMind's `MBPP` (Austin et al., 2021), which includes 974 tasks with 3 test cases each. Despite their popularity, these benchmarks have significant limitations, such as inadequate test case coverage, limited number of PLs, and small task sets that do not represent real-world scenarios. Other benchmarks, like `CONCODE` (Iyer et al., 2018) (Java), `AxiBench` (Hao et al., 2022) (Java), `CSEPrompts` (Raihan et al., 2024) (Python) and `CodeApex` (Fu et al., 2023) (C++) focus on a single PL.

To broaden PL coverage, Cassano et al. (2023) combine both `HumanEval` and `MBPP` and add 17 more popular PLs besides Python, such as C++, Java, Ruby, and PHP. However, all prompts remain in English, with only 3 test cases per task. Similarly, the authors of `BabelCode` (Orlanski et al., 2023) include 14 PLs and a more extensive test suite. To address test case coverage, Liu et al. (2024) introduce two datasets, `HumanEval+` and `MBPP+`, with significantly more test cases per task, ensuring both node and edge coverage. Notably, Code LLM performance decreases with the additional test cases, highlighting the initial benchmarks' limitations. Nevertheless, these benchmarks also use English prompts exclusively.

Few studies explore non-English coding prompts and evaluate Code LLMs on them. The recent benchmark, `HumanEval-XL` (Peng et al., 2024), extends coverage for both NLs and PLs. This benchmark includes coding prompts in 23 NLs and solutions in 12 PLs. The original prompts from HumanEval (Chen et al., 2021) are translated into 23 different NLs using GPT-4 (Achiam et al., 2023), with the quality of these translations assessed using a thresholded `BERTScore` (Zhang et al., 2019). While `HumanEval-XL` explores multilingual prompts for code generation (Table 1), its 23 predominantly high-resource NLs limit insights into mid and low-resource NLs. The *`BERTScore`* (Zhang et al., 2019) evaluation may be inadequate, with *`CometKiwi`* (Rei et al., 2023) and `X-Comet` (Guerreiro et al., 2023) offering more robust alternatives. Experimenting with SOTA Code LLMs like WizardCoder (Luo et al., 2023) or multilingual models like Aya (Üstün et al., 2024) could yield valuable insights. Also, they do not include any human translations.

We argue that NL coverage is more critical than PL coverage when compiling a code generation benchmark. While prompts and tests can be reused across PLs, different NLs require curating contextually and linguistically appropriate prompts. Thus, NL diversity introduces more complexity in benchmark creation than PL diversity. To bridge the

	Benchmarks					
	HumanEval	MBPP	Babel Code	MultiPL-E	HumanEval-XL	mHumanEval
NL-Covg (MT)	1 (eng)	1 (eng)	1 (eng)	1 (eng)	23	**204**
NL-Covg (Human)	✗	✗	✗	✗	✗	**15**
PL-Covg	1 (py)	1 (py)	14	18	12	**25**

Table 1: Comparing popular benchmarks in terms of NL and PL coverage.

gap, we present mHumanEval, offering comprehensive experiments with multilingual coding prompts across 204 NLs and 25 PLs—the most extensive coverage to date (see Appendix A for the full list) and the first one to include expert-human annotations (see Table 1). We describe mHumanEval in detail in this paper and we evaluate SOTA models on this dataset.

3 The mHumanEval Benchmark

The mHumanEval benchmark is curated based on prompts from the original HumanEval (Chen et al., 2021) dataset. It includes a total of 33,456 prompts, significantly expanding from the original 164. The curation process can be divided into several key steps, as illustrated in Figure 2 and elaborated upon in the following subsections.

3.1 Prompt Extraction

A typical prompt from the original dataset includes optional library imports, a function declaration, a docstring, and optional examples (as illustrated in Figure 3).

For translation, we only consider the docstrings (enclosed in triple quotes). These are manually extracted from all 164 prompts to ensure accuracy.

3.2 Prompt Translation

Upon extracting the prompts, we move on to translating them into different languages. We use three different machine translation strategies - leveraging OpenAI's GPT4-omni through API, MetaAI's NLLB (Costa-jussà et al., 2022), which is the SOTA model for multiple NLs, and Google Translate via API.

Our target languages are all 204 languages from the Flores 200 dataset-(Costa-jussà et al., 2022). While we employ GPT4-omni and NLLB for all the target languages, it is important to note that we use only Google Translator for the 108 languages it supports (available through the API). For each extracted prompt, we employ the three translation systems for each target language, generating 5 candidate translation prompts (3 for GPT4o, due to budget considerations). We then evaluate the quality of the translation and keep the best one (see Figure 2). The pseudocode is in Appendix B.

3.3 Evaluating Prompt Quality

We evaluate translation quality using BERTScore (Zhang et al., 2019), which focuses on similarity based on contextual embeddings, and CometKiwi (Rei et al., 2023), which is trained on human judgments of MT quality and incorporates linguistic features. While BERTScore uses BERT embeddings to measure candidate-reference translation similarity (Appendix C), CometKiwi evaluates translations reference-free, using human judgments and combining linguistic features with contextual embeddings (Appendix D). Using both ensures holistic evaluation, covering lexical similarity and human-assessed quality aspects.

As illustrated in Figure 2, we generate 13 candidate translations for each prompt. We also perform round-trip translations back to the original language (eng_Latn) to calculate the BERTScore. While CometKiwi is calculated as a reference-free metric. Both metrics generate scores in the [0,1] range. By computing the mean of the two metrics for each prompt, we select the candidate with the highest score. The mean scores for each language and system are provided in Appendices K, L and M. It is worth noting that the CometKiwi metrics are not available for all languages, as it relies on XLM-R models (Conneau et al., 2019; Goyal et al., 2021) supporting 100 languages (Rei et al., 2023). For the remaining 104 Flores 200 languages, we use round-trip translations to calculate BERTScore, similar to HumanEval-XL (Peng et al., 2024).

3.4 Categorization based on Language Classes

To better understand the performance of models on languages considered to be low- or high-resourced, we group the languages in mHumanEval following the methodology of Joshi et al. (2020), who identify six classes of languages based on digital re-

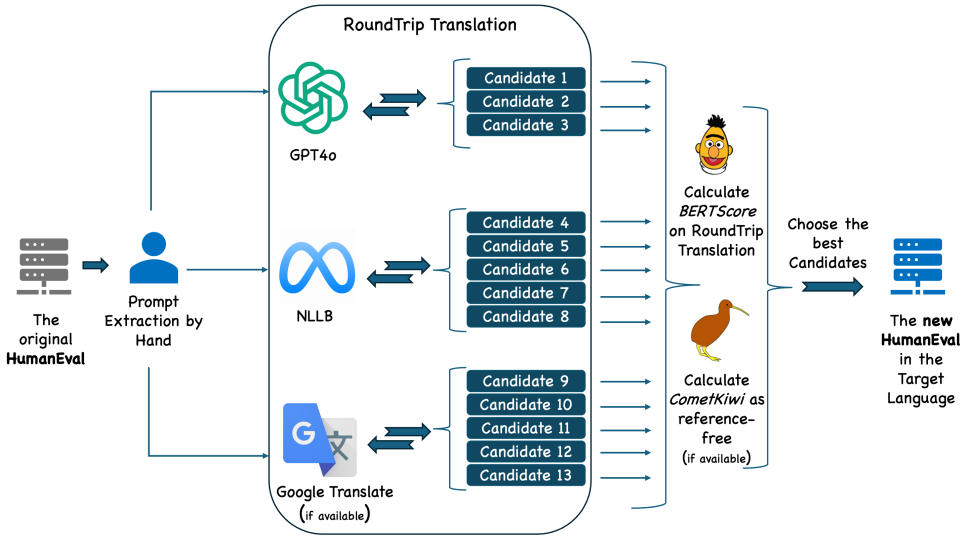

Figure 2: The workflow to generate prompts in a target language from the original `HumanEval`. Original prompts are first extracted manually. Then 3 Machine Translation models (GPT4o, NLLB, Google Translate) generate 13 candidates as well as roundtrip translations. Next, we evaluate each candidate's quality using *`BERTScore`* using RoundTrip translations and *`CometKiwi`* as a reference-free metric (if the language is supported). We then select the best candidate for each original prompt and compile the new benchmark for the target language.

```
from typing import List

def all_prefixes:
    """ Return list of all prefixes
    from shortest to longest of the
    input string. """

>>> all_prefixes('abc')
['a', 'ab', 'abc']
```

Figure 3: A sample prompt instance from the original `HumanEval` benchmark.

Class	Resource	Total	`mHumanEval`	`Expert`
5	High	7	7	6
4	Mid to High	18	18	4
3	Mid	28	27	1
2	Low to Mid	19	16	2
1	Low	222	98	1
0	Rare	2191	38	1
ALL	–	2485	204	15

Table 2: Class distribution of natural languages based on resource availability. **Expert** denotes human translations done by expert programmers.

source availability. These classes range from 0 to 5, with higher numbers indicating greater resource availability. Joshi et al. classify a total of 2,485 languages, of which `mHumanEval` includes 204, including 15 with expert translations, as detailed in Table 2.

We present the class-wise evaluation scores for the selected prompts in `mHumanEval` in Figure 4. The language-specific scores are provided in Appendices K, L, and M. Generally, the quality of the translation decreases as we address languages with fewer resources. However, by implementing Algorithm 1 and selecting from 13 candidate translations, the chosen candidates demonstrate improved quality compared to the model-specific results (see Appendices N and F). The final prompts in `mHumanEval` exhibit significantly better quality.

3.5 PL coverage

As noted in Section 2, most benchmarks in this subdomain are limited to Python, including `HumanEval` and `MBPP`. While recent benchmarks such as `MultiPL-E` and `HumanEval-XL` offer broader coverage, they still omit several widely used programming languages. With `mHumanEval`, we compile a comprehensive set of programming languages covered by existing multi-PL coding benchmarks and extend this set by incorporating four additional languages that have not been previously included: MATLAB, Visual Basic, Fortran, and COBOL (as shown in Table 7).

We provide canonical solutions for the newly included four languages in the same format as `HumanEval` . These solutions are handwritten by human experts and successfully pass all test cases.

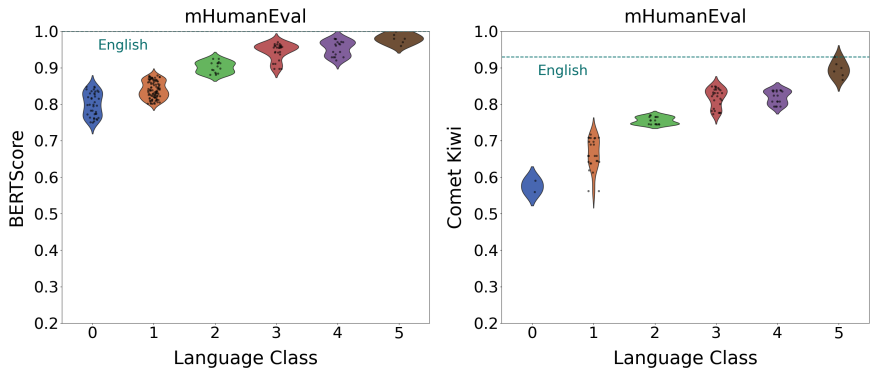

Figure 4: Evaluating the translated prompt qualities, chosen in `mHumanEval`. Our method results in better quality prompts compared to the model-specific translations (as depicted in Appendix F).

	Prompts	Note
`mHumanEval-{NL}`	164 each	Each NL
`mHumanEval-mini`	204	204 NLs
`mHumanEval-T500`	500	Top 500
`mHumanEval-R500`	500	Random 500
`mHumanEval-B500`	500	Bottom 500
`mHumanEval-Expert`	2460	Human Generated
`mHumanEval-{PL}`	4100 each	Each PL
`mHumanEval`	33456	Only Python
`mHumanEval-Max`	836400	All Prompts

Table 3: Subsets and Variants of `mHumanEval`. These enable practitioners to carry out both comprehensive and preliminary evaluations on the benchmark.

3.6 `mHumanEval` Subsets

We have a total of 33,456 prompts in `mHumanEval` spanning 204 NLs. Each prompt additionally supports 24 PLs, bringing the total number of prompts to 836,400. The entire dataset is publicly available on GitHub.

We also provide multiple subsets of the dataset for quick usability and interesting ablation studies (Table 3). Separate subsets are available for each NL and PL, in all possible combinations. Additionally, we create several variants for testing purposes- `mHumanEval-T500`: a subset consisting of the 500 highest-quality prompts based on `BERTScore` and `CometKiwi`, `mHumanEval-R500`: a randomly selected subset of 500 prompts, and `mHumanEval-B500`: a subset of the 500 lowest-quality prompts. Note that these prompts are drawn from the curated `mHumanEval`, which compiles the best prompts from 13 candidates each. Finally, we produce `mHumanEval-mini` which is a subset containing 204 prompts, with each prompt in a different language, where we select one prompt per language.

3.7 `mHumanEval` - Expert

The `mHumanEval`-Expert benchmark encompasses human translations across 15 languages, representing all six language classes (Table 2). Native speakers with computer science and engineering backgrounds perform these translations, ensuring precise interpretation of programming concepts and terminology. The curation process unfolds in three stages: (1) selection of 15 natural languages based on native speaker availability, ensuring representation from each language class; (2) translation by native speakers; and (3) quality assessment by expert programmers to verify the integrity of the coding prompts. Figure 5 illustrates the whole curation process.

A comparative analysis between human translations and `mHumanEval`'s machine-translated prompts yields comparable evaluation metrics, with BERTScore variations of ±0.02 and CometKiwi variations of ±0.03 across the selected languages. Interestingly, annotators report no significant terminology concerns when reviewing machine translations. Further examination of the original `HumanEval` prompts reveals that the docstrings—the primary translated content—predominantly comprise general task descriptions, minimizing the use of specialized coding terminology. This observation emphasizes the negligible discrepancies between human and machine translations in this context.

We conclude that human and machine translations of programming prompts across 15 languages show similar quality, with minimal differences in evaluation metrics. This similarity is attributed to the general nature of the content, which contains limited specialized coding terminology.

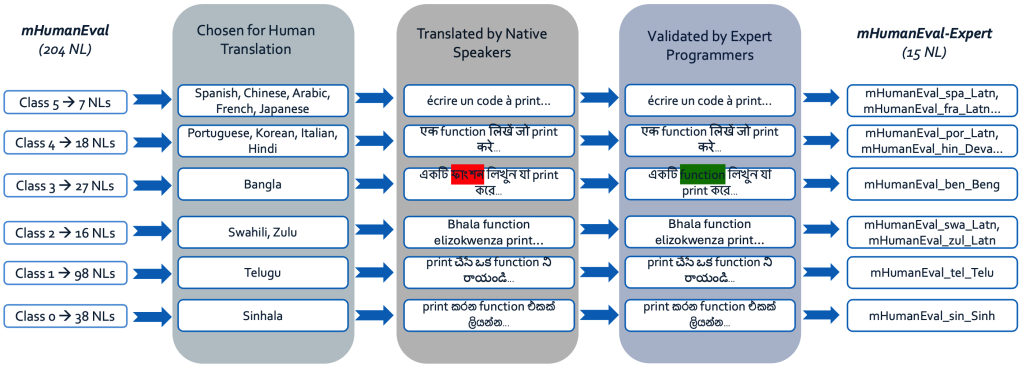

Figure 5: Curating `mHumanEval`-Expert via native human translation followed by expert programmer evaluation.

4 Experiments

Model Selection We experiment with `mHumanEval` using six models (Table 4), including both proprietary and open-source SOTA models for code generation. We use a mix of general-purpose and finetuned models to gather broader insights.

Model	Size	Type	Ref.
GPT4o	–	Base	(Achiam et al., 2023)
Claude-3.5-Opus	–	Base	(Anthropic, 2024)
GPT3.5	175B	Base	(Brown et al., 2020)
DeepSeek-Coder-V2	236B	Finetuned	(Dai et al., 2024)
WizardCoder	33B	Finetuned	(Luo et al., 2023)
Aya	33B	Finetuned	(Üstün et al., 2024)

Table 4: LLMs evaluated on `mHumanEval`.

Prompting We use the proprietary models through their APIs. Our experiments include all 33,456 prompts from `mHumanEval`, with 164 prompts for each language. We follow the standard prompt templates for each LLM. These templates are shown in Appendix G.

Code Execution Following code generation, we move to execution. The six models produce well-structured code blocks, requiring minimal cleaning. We use simple RegEx commands to extract these blocks, and evaluate them locally in batches using Python's `subprocess`[2] library, focusing exclusively on the **`Pass@1`** metric.

Results For each language, we present the **`Pass@1`** scores as percentages, categorizing them by the six language classes as discussed in Section 3.4. As illustrated in Figure 6, Claude3.5 and GPT4o exhibit the most consistent performance, maintaining strong results even with coding prompts in low-resource languages. In contrast, GPT3.5 and DeepSeek experience a significant decline in performance for low-resource classes. Although Aya shows the weakest results for higher resource classes, it maintains relative consistency, even in extremely low-resource languages. On the other hand, WizardCoder achieves excellent results in English and reasonable performance for Class 5, but its performance deteriorates significantly in other languages. The model and language-specific detailed results are presented in Appendix O.

Other PLs We extend our evaluation to four additional subsets of `mHumanEval`: `mHumanEval`-C++, `mHumanEval`-JAVA, `mHumanEval`-JavaScript, and `mHumanEval`-Ruby. The average **`Pass@1`** scores across all 204 NLs for the 5 PLs are shown in Table 5.

	Python	Java	C++	JavaScript	Ruby
GPT4o	0.738	0.650	0.652	0.477	0.480
GPT3.5	0.360	0.270	0.270	0.099	0.103
Claude3.5	0.739	0.651	0.649	0.483	0.477
DeepSeek-Coder	0.229	0.139	0.136	0.000	0.000
WizardCoder	0.098	0.009	0.007	0.000	0.000
Aya	0.445	0.355	0.356	0.186	0.183

Table 5: Mean performance of models across programming languages.

We observe that GPT-4o and DeepSeek-Coder achieve strong results in Classes 4 and 5, with scores consistently exceeding 0.85 in Python, Java, and C++. Python shows top performance, with scores reaching above 0.88 in Class 5. For lower classes (0-2), models like GPT-3.5, WizardCoder, and Aya underperform, often scoring below 0.70, particularly in JavaScript and Ruby, where scores frequently drop under 0.65. Even in higher classes, JavaScript and Ruby show challenges, with Class 4 scores for most models not exceeding 0.75. This

[2] `docs.python.org/3/library/subprocess.html`

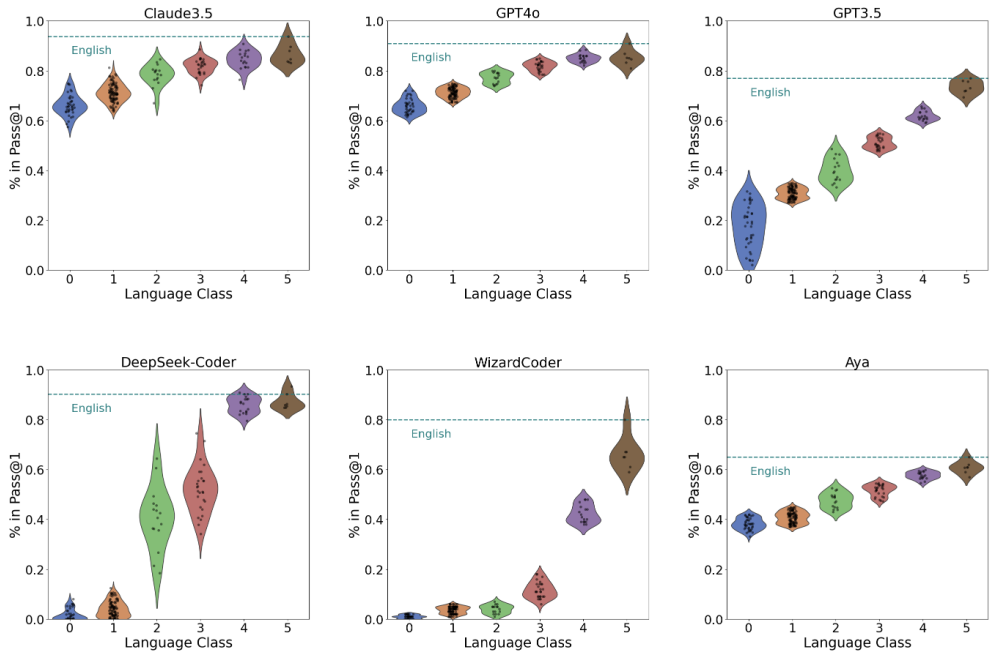

Figure 6: Comparing model performances (% in **Pass@1**) for the six models on mHumanEval-**Python**.

highlights the models' limitations in handling non-Python languages, particularly for lower classes and specific scripting languages. While every model's best scores are generated with English-Python pair, DeepSeek-Coder is the only exception with Chinese-Python.

A detailed analysis and discussion is provided in Appendix J.

5 Insights and Analysis

Upon curating the mHumanEval benchmark and completing the model evaluations, we now present some key analyses and gained insights based on the obtained results.

5.1 LLMs' Performance Analysis

We observe significant performance discrepancies among the models, as illustrated by Figure 6. While closed-source models perform better, their reliance on proprietary pretraining data complicates definitive conclusions. As suggested by the Chinchilla scaling hypothesis (Hoffmann et al., 2022), their superior performance may result from a larger parameter count and extensive training tokens, possibly including diverse and rare languages.

Aya, fine-tuned for multiple natural languages but not specifically for code generation, has the lowest Pass@1 score in English. However, low variability across language classes indicates that multilingual pretraining and fine-tuning enhances code generation across different NLs.

WizardCoder's poor performance in non-English languages is due to its fine-tuning on StarCoder (Li et al., 2023), which is primarily pretrained on code and documentation with minimal non-English content. In contrast, DeepSeek performs well for mid-resource languages but struggles with low-resource ones. These results suggest that effective multilingual code generation requires multilingual pretraining and/or finetuning datasets.

5.2 Performance based on Language Classes

While there are significant discrepancies among the models' performances, a key trend observed is a somewhat consistent performance decline as we move from high-resource to low-resource languages. This decline is not as pronounced for Claude and GPT-4o. However, it is quite substantial for others and exceptionally steep for WizardCoder and DeepSeek-Coder.

5.3 Error Analysis

In our analysis of errors, we observe several unique issues. Notably, the models rarely fail to generate any code. Specifically, GPT4o and GPT3.5 generate code with almost no compilation issues. However, a significant number of errors arise from misunderstandings of the problem, resulting in code that addresses incorrect tasks. This issue primarily occurs because translated keywords (e.g., string, list) do not always retain identical meanings in the

	GPT4o	GPT3.5	Aya	WizardCoder	Claude3.5	DeepSeek-Coder	LLaMA 3	CodeStral
mHumanEval-mini	.72	.44	.47	.12	.61	.57	.35	.15
mHumanEval-T500	.87	.76	.6	.63	.86	.73	.56	.36
mHumanEval-R500	.78	.53	.47	.16	.59	.63	.28	.17
mHumanEval-B500	.48	.21	.42	.00	.31	.22	.11	.10

Table 6: Comparison of different LLMs' based on % in **Pass@1** metric on multiple subsets of mHumanEval.

target language, as illustrated in Appendix H.1.

Furthermore, the Aya model often uses identifiers or keywords from different languages, leading to compilation errors (Appendix H.2). A recurring problem with DeepSeek-Coder and WizardCoder is the generation of nonsensical code, sometimes not even in Python, especially when prompted in a non-English language (Appendix H.3).

5.4 Ablation Study

We present results from a limited ablation study conducted on various subsets of mHumanEval as detailed in Table 3. This study incorporates two additional models including MetaAI's LLaMA 3 (70B), and MistralAI's code-finetuned CodeStral (22B) model.

As indicated by the results in Table 6, mHumanEval-mini serves as an effective preliminary test for evaluating a model's proficiency in code generation following multilingual prompts. Models fine-tuned on code but lacking multilingual exposure perform poorly, whereas base models with some multilingual exposure perform better. The three subsets of mHumanEval are curated by prompt quality: mHumanEval-T500 includes prompts from language class 5, mHumanEval-B500 from classes 0 or 1, and mHumanEval-R500 is randomly selected. These results align with our findings in Sections 5.1 and 5.2.

6 Conclusion

This study introduces mHumanEval, a comprehensive multilingual code generation benchmark for assessing LLMs across 204 languages. We curated high-quality prompts for each language and evaluated various models. Our analyses, including ablation studies, provided insights into LLMs' multilingual code-generation capabilities, addressing the RQs posed in Section 1:

RQ1: How do the code generation capabilities of LLMs vary when prompts are provided in English, or other high-, mid-, and low-resource NLs?

> LLMs generally demonstrate optimal performance when prompted in English. For prompts in other languages, performance varies based on the language's resource level. High-resource languages tend to yield superior results compared to mid- and low-resource languages. The extent of performance variation is contingent upon the specific language of the prompt and the model's prior exposure and training in that language. This variation is likely influenced by the model's training data and the relative abundance of resources available for each language.

RQ2: How does the performance of multilingual LLMs compare to specialized, fine-tuned Code LLMs in code generation tasks on the mHumanEval dataset?

> While code-finetuned language models excel at generating code from English prompts, multilingual models demonstrate strong proficiency across various NLs. Notably, even without specific code fine-tuning for different NLs, they achieve decent results in code generation. This phenomenon suggests that multilingual models can generalize coding capabilities across NLs, leveraging their understanding of multiple NLs to support diverse linguistic contexts in programming.

While we draw some insightful conclusions from curating and evaluating mHumanEval, to facilitate further research, we are making it publicly available. We plan to expand coverage to more NLs and PLs in future updates. Despite the high cost of human translation, we included human annotations for 15 NLs, including some low-resource and rare ones. Currently, our dataset includes 164 prompts per language, following the HumanEval benchmark, with plans to increase this number. We will also explore strategies to enhance low-resource language performance, such as transfer learning and diverse training datasets. Comparative studies between general-purpose multilingual LLMs and specialized code LLMs will help optimize multilingual code generation.

Limitations

We conducted primary evaluations on six LLMs, focusing on key performance metrics. Given the benchmark's extensive 33,456 prompts, the evaluation process is exceedingly costly. This cost is the primary reason why we adopted **`Pass@1`** as our evaluation metric, rather than more resource-intensive metrics like **`Pass@10`** or **`Pass@100`**. However, to ensure a thorough analysis, we incorporated additional models in our ablation study. In our next iteration, we plan to comprehensively evaluate all models across the entire benchmark. This future work aims to enhance the benchmark's robustness and provide deeper insights into the performance of various LLMs in multilingual code generation.

Ethical Considerations

The benchmark introduced in this paper, which focuses on analyzing code generation using large language models (LLMs), strictly adheres to the ACL Ethics Policy. Each prompt in `mHumanEval` was tested multiple times by different models, and none produced any malicious code. Although there can occasionally be garbage code snippets or similar issues, none have posed any threats to the system.

To ensure safety and reliability, we recommend executing code generated using prompts from `mHumanEval` in a contained virtual environment. This precaution helps prevent potential issues related to infinite execution loops and memory management. Running code in a safe environment can also stop problems like crashing the system or using too much memory. We believe and hope that researchers and practitioners can maintain a secure and controlled testing environment while utilizing `mHumanEval`. This approach ensures that users can confidently explore and innovate without risking system integrity.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, et al. 2023. Gpt-4 technical report.

Anthropic. 2024. Claude 3: A next-generation ai assistant. https://www.anthropic.com/news/claude-3-family.

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*.

Damian Blasi, Antonios Anastasopoulos, and Graham Neubig. 2022. Systematic inequalities in language technology performance across the world's languages. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5486–5505, Dublin, Ireland. Association for Computational Linguistics.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*.

Federico Cassano, John Gouwar, Daniel Nguyen, Sydney Nguyen, Luna Phipps-Costin, Donald Pinckney, et al. 2023. Multipl-e: a scalable and polyglot approach to benchmarking neural code generation. *IEEE Transactions on Software Engineering*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Unsupervised cross-lingual representation learning at scale. *arXiv preprint arXiv:1911.02116*.

Marta R Costa-jussà, James Cross, Onur Çelebi, et al. 2022. No language left behind: Scaling human-centered machine translation. *arXiv preprint arXiv:2207.04672*.

Damai Dai, Chengqi Deng, Chenggang Zhao, et al. 2024. Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models. *arXiv preprint arXiv:2401.06066*.

Lingyue Fu, Huacan Chai, Shuang Luo, Kounianhua Du, Weiming Zhang, et al. 2023. Codeapex: A bilingual programming evaluation benchmark for large language models. *arXiv preprint arXiv:2309.01940*.

Naman Goyal, Jingfei Du, Myle Ott, Giri Anantharaman, and Alexis Conneau. 2021. Larger-scale transformers for multilingual masked language modeling. In *Proceedings of the 6th Workshop on Representation Learning for NLP (RepL4NLP-2021)*.

Nuno M Guerreiro, Ricardo Rei, Daan van Stigt, Luisa Coheur, et al. 2023. xcomet: Transparent machine translation evaluation through fine-grained error detection. *arXiv preprint arXiv:2310.10482*.

Yiyang Hao, Ge Li, Yongqiang Liu, Xiaowei Miao, et al. 2022. Aixbench: A code generation benchmark dataset. *arXiv preprint arXiv:2206.13179*.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, et al. 2022. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*.

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. 2018. Mapping language to code in programmatic context. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*.

Pratik Joshi, Sebastin Santy, Amar Budhiraja, Kalika Bali, and Monojit Choudhury. 2020. The state and fate of linguistic diversity and inclusion in the nlp world. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*.

R Li, LB Allal, Y Zi, N Muennighoff, D Kocetkov, C Mou, M Marone, C Akiki, J Li, J Chim, et al. 2023. Starcoder: May the source be with you! *Transactions on machine learning research*.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2024. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. *Advances in Neural Information Processing Systems*.

Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, et al. 2023. Wizardcoder: Empowering code large language models with evol-instruct. In *The Twelfth International Conference on Learning Representations*.

Gabriel Orlanski, Kefan Xiao, Xavier Garcia, et al. 2023. Measuring the impact of programming language distribution. In *International Conference on Machine Learning*. PMLR.

Qiwei Peng, Yekun Chai, and Xuhong Li. 2024. Humaneval-xl: A multilingual code generation benchmark for cross-lingual natural language generalization. *arXiv preprint arXiv:2402.16694*.

Nishat Raihan, Dhiman Goswami, Sadiya Sayara Chowdhury Puspo, Christian Newman, Tharindu Ranasinghe, and Marcos Zampieri. 2024. Cseprompts: A benchmark of introductory computer science prompts. *arXiv preprint arXiv:2404.02540*.

Ricardo Rei, Nuno M Guerreiro, Daan van Stigt, Marcos Treviso, et al. 2023. Scaling up cometkiwi: Unbabel-ist 2023 submission for the quality estimation shared task. In *Proceedings of the Eighth Conference on Machine Translation*.

Ahmet Üstün, Viraat Aryabumi, Zheng-Xin Yong, Wei-Yin Ko, Daniel D'souza, Gbemileke Onilude, et al. 2024. Aya model: An instruction finetuned open-access multilingual language model. *arXiv preprint arXiv:2402.07827*.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q Weinberger, and Yoav Artzi. 2019. Bertscore: Evaluating text generation with bert. *arXiv preprint arXiv:1904.09675*.

A list of NLs and PLs in `mHumanEval`

`mHumanEval` supports 204 NLs and 25 PLs. The `Expert` subset contains human annotation for 15 NLs.

A.1 List of PLs

Comparing PL support provided by most widely used existing benchmarks -

	Benchmarks					
	HumanEval	MBPP	Babel Code	MultiPL-E	HumanEval-XL	mHumanEval
Python	✓	✓	✓	✓	✓	✓
Bash	✗	✗	✗	✓	✗	✓
C++	✗	✗	✓	✓	✗	✓
C#	✗	✗	✓	✓	✓	✓
D	✗	✗	✗	✓	✗	✓
Go	✗	✗	✓	✓	✓	✓
Haskell	✗	✗	✓	✗	✗	✓
Java	✗	✗	✓	✓	✓	✓
JavaScript	✗	✗	✓	✓	✓	✓
Julia	✗	✗	✗	✓	✗	✓
Kotlin	✗	✗	✓	✗	✓	✓
Lua	✗	✗	✗	✓	✗	✓
Perl	✗	✗	✗	✓	✓	✓
PHP	✗	✗	✓	✓	✓	✓
R	✗	✗	✗	✓	✗	✓
Racket	✗	✗	✗	✓	✗	✓
Ruby	✗	✗	✓	✓	✓	✓
Rust	✗	✗	✓	✓	✗	✓
Scala	✗	✗	✓	✓	✓	✓
Swift	✗	✗	✓	✓	✓	✓
TypeScript	✗	✗	✓	✓	✓	✓
MATLAB	✗	✗	✗	✗	✗	✓
Visual Basic	✗	✗	✗	✗	✗	✓
Fortran	✗	✗	✗	✗	✗	✓
COBOL	✗	✗	✗	✗	✗	✓

Table 7: Comparing popular benchmarks in terms of NL and PL coverage.

A.2 List of NLs: `mHumanEval`-Expert

Prompts in these languages are generated using translations done by native speakers, followed by evaluations done by expert programmers.

Language	Class
English	5
Spanish	5
French	5
Japanese	5
Arabic	5
Chinese	5
Portuguese	4
Italian	4
Korean	4
Hindi	4
Bangla	3
Swahili	2
Zulu	2
Telugu	1
Sinhala	0

Table 8: NLs along with their classes in `mHumanEval`-Expert.

A.3 List of NLs: `mHumanEval`

Language	Class	Language	Class	Language	Class	Language	Class
arb_Arab	5	zsm_Latn	3	gla_Latn	1	tat_Cyrl	1
deu_Latn	5	amh_Ethi	2	guj_Gujr	1	tel_Telu	1
eng_Latn	5	gle_Latn	2	hye_Armn	1	tgk_Cyrl	1
fra_Latn	5	hau_Latn	2	ibo_Latn	1	tpi_Latn	1
jpn_Jpan	5	isl_Latn	2	ilo_Latn	1	tso_Latn	1
spa_Latn	5	lao_Laoo	2	jav_Latn	1	tuk_Latn	1
zho_Hans	5	mar_Deva	2	kab_Latn	1	tum_Latn	1
cat_Latn	4	mlt_Latn	2	kan_Knda	1	twi_Latn	1
ces_Latn	4	pan_Guru	2	kas_Arab	1	uig_Arab	1
eus_Latn	4	san_Deva	2	kas_Deva	1	vec_Latn	1
fin_Latn	4	swh_Latn	2	khk_Cyrl	1	war_Latn	1
hin_Deva	4	tir_Ethi	2	khm_Khmr	1	ydd_Hebr	1
hrv_Latn	4	tsn_Latn	2	kik_Latn	1	zho_Hant	1
hun_Latn	4	wol_Latn	2	kin_Latn	1	awa_Deva	0
ita_Latn	4	xho_Latn	2	kir_Cyrl	1	bam_Latn	0
kor_Hang	4	yor_Latn	2	kmr_Latn	1	ban_Latn	0
nld_Latn	4	zul_Latn	2	lij_Latn	1	bem_Latn	0
pes_Arab	4	ace_Arab	1	lim_Latn	1	cjk_Latn	0
pol_Latn	4	ace_Latn	1	lin_Latn	1	dyu_Latn	0
por_Latn	4	acm_Arab	1	lmo_Latn	1	fon_Latn	0
rus_Cyrl	4	acq_Arab	1	ltg_Latn	1	fuv_Latn	0
srp_Cyrl	4	aeb_Arab	1	ltz_Latn	1	grn_Latn	0
swe_Latn	4	ajp_Arab	1	lug_Latn	1	hat_Latn	0
tur_Latn	4	aka_Latn	1	mai_Deva	1	hne_Deva	0
vie_Latn	4	als_Latn	1	mal_Mlym	1	kac_Latn	0
afr_Latn	3	apc_Arab	1	min_Arab	1	kam_Latn	0
arb_Latn	3	ars_Arab	1	min_Latn	1	kbp_Latn	0
arz_Arab	3	ary_Arab	1	mkd_Cyrl	1	kea_Latn	0
ben_Beng	3	asm_Beng	1	mri_Latn	1	kmb_Latn	0
bos_Latn	3	ast_Latn	1	mya_Mymr	1	knc_Arab	0
bul_Cyrl	3	ayr_Latn	1	nno_Latn	1	knc_Latn	0
ceb_Latn	3	azb_Arab	1	nob_Latn	1	kon_Latn	0
dan_Latn	3	azj_Latn	1	npi_Deva	1	lua_Latn	0
ell_Grek	3	bak_Cyrl	1	oci_Latn	1	luo_Latn	0
est_Latn	3	bel_Cyrl	1	ory_Orya	1	lus_Latn	0
glg_Latn	3	bho_Deva	1	pag_Latn	1	mag_Deva	0
heb_Hebr	3	bjn_Arab	1	pap_Latn	1	mni_Beng	0
ind_Latn	3	bjn_Latn	1	pbt_Arab	1	mos_Latn	0
kat_Geor	3	bod_Tibt	1	plt_Latn	1	nso_Latn	0
kaz_Cyrl	3	bug_Latn	1	quy_Latn	1	nus_Latn	0
lit_Latn	3	ckb_Arab	1	sag_Latn	1	nya_Latn	0
lvs_Latn	3	crh_Latn	1	sat_Olck	1	prs_Arab	0
ron_Latn	3	cym_Latn	1	scn_Latn	1	run_Latn	0
slk_Latn	3	dik_Latn	1	smo_Latn	1	shn_Mymr	0
slv_Latn	3	dzo_Tibt	1	sna_Latn	1	sin_Sinh	0
tam_Taml	3	epo_Latn	1	snd_Arab	1	sot_Latn	0
tgl_Latn	3	ewe_Latn	1	som_Latn	1	taq_Latn	0
tha_Thai	3	fao_Latn	1	srd_Latn	1	taq_Tfng	0
ukr_Cyrl	3	fij_Latn	1	ssw_Latn	1	tzm_Tfng	0
urd_Arab	3	fur_Latn	1	sun_Latn	1	umb_Latn	0
uzn_Latn	3	gaz_Latn	1	szl_Latn	1	yue_Hant	0

Table 9: All NLs and their classes included in `mHumanEval`.

B Prompt Translation and Evaluation Algorithm

The pseudocode version of the workflow, presented in Figure 2.

Algorithm 1 Prompt Translation and Evaluation

```
for each extracted prompt from HumanEval do
    for each translation system do
        for each target language do
            if the language is supported then
                generate 5 translated candidate prompts
                do back translation
                calculate BERT_Score and Comet_Kiwi for each
                take the average of the two
                pick the best prompt
            else
                do back translation
                calculate only BERT_Score
                pick the best prompt
            end if
        end for
    end for
end for
```

It describes how the originally extracted prompts go through 13 candidate translations and evaluation via `BERTScore` and `CometKiwi` to build the new sets of benchmarks in the target natural languages.

C Evaluation Metric 1: `BERTScore`

`BERTScore` uses pre-trained BERT embeddings to assess similarity between candidate and reference translations. For a candidate sentence C and a reference sentence R, let E_C and E_R be the sets of BERT embeddings for tokens in C and R, respectively. The similarity $S(i, j)$ between tokens i and j is the cosine similarity of their embeddings:

$$S(i,j) = \frac{e_{C_i} \cdot e_{R_j}}{\|e_{C_i}\| \|e_{R_j}\|}$$

Precision P, recall R, and F1-score $F1$ are then:

$$P = \frac{1}{|E_C|} \sum_{e_{C_i} \in E_C} \max_{e_{R_j} \in E_R} S(i,j)$$

$$R = \frac{1}{|E_R|} \sum_{e_{R_j} \in E_R} \max_{e_{C_i} \in E_C} S(j,i)$$

$$F1 = 2 \cdot \frac{P \cdot R}{P + R}$$

Here, P and R denote precision and recall as average maximum similarities from candidate to reference and vice versa. The $F1$ score is their harmonic mean.

D Evaluation Metric 2: `CometKiwi`

`CometKiwi` (Knowledge Integration via Weighted Importance) evaluates translations without references, using human-judgment scores. Given source $\mathbf{x}$ and candidate $\mathbf{y}$, it maps these inputs to a quality score $Q(\mathbf{x}, \mathbf{y})$ using a neural network $\mathcal{N}$ trained on human scores $Q_{\text{human}}(\mathbf{x}, \mathbf{y})$:

$$Q(\mathbf{x}, \mathbf{y}) = f(\mathbf{E}_{\text{src}}(\mathbf{x}), \mathbf{E}_{\text{cand}}(\mathbf{y}), \mathbf{L}(\mathbf{x}, \mathbf{y}))$$

where $\mathbf{E}_{\text{src}}$ and $\mathbf{E}_{\text{cand}}$ are embeddings for $\mathbf{x}$ and $\mathbf{y}$, and $\mathbf{L}$ represents linguistic features. The function f is:

$$f = \mathcal{N}(\mathbf{E}_{\text{src}}, \mathbf{E}_{\text{cand}}, \mathbf{L})$$

The network $\mathcal{N}$ minimizes the loss:

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} (Q(\mathbf{x}_i, \mathbf{y}_i) - Q_{\text{human}}(\mathbf{x}_i, \mathbf{y}_i))^2$$

where N is the sample size.

E Annotator Details

As mentioned in Section 3.7, `mHumanEval`-Expert utilizes native-speaking volunteer translators for 15 NLs. Each translator was assigned 164 prompts, with no monetary compensation involved. The experts, also native speakers, possess backgrounds in Computer Science and/or Information Technology, complemented by substantial coding experience. Both translators and experts were carefully selected through a rigorous process, ensuring a diverse demographic representation. This methodological approach enhances the dataset's linguistic diversity and technical robustness across various cultural contexts.

F Comparison of the Prompt Qualities by the 3 models vs `mHumanEval`

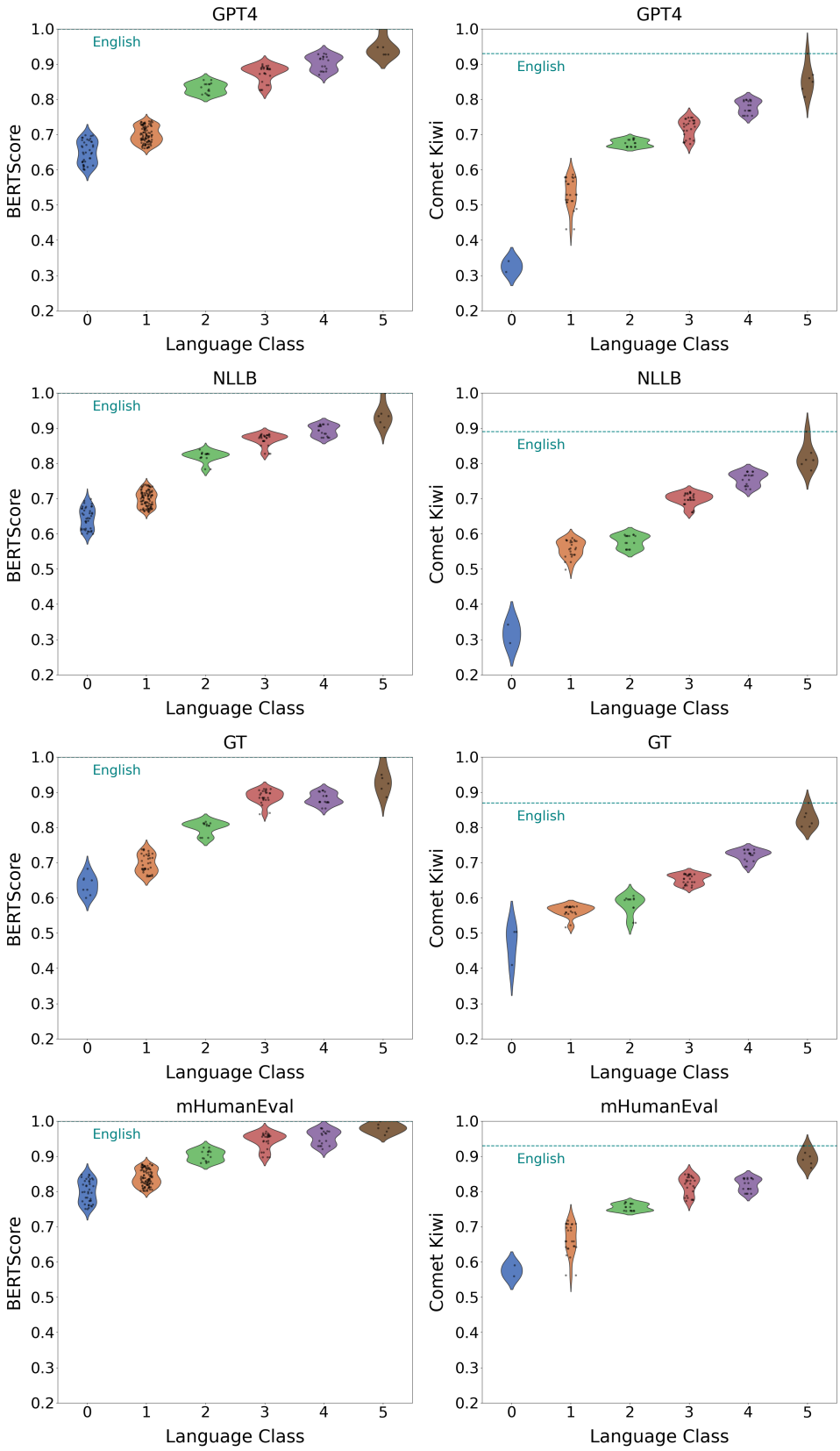

Figure 7: Comparing the Machine Translation Quality for GPT4o, NLLB and Google Translator. The metrics used are `BERTScore` and `CometKiwi`. As shown in the figure, the prompts chosen for `mHumanEval` are better in quality upon choosing from 13 different candidates.

G Prompt Templates

GPT4o and GPT3.5

```
prompt = "Write a Python function for
the following: " + mHumanEval[i] +
" Ensure your response includes a
Python code block."

messages=[
    {"role": "system", "content":
    "You are a helpful assistant
    trained to generate Python code.
    "},
    {"role": "user", "content":
    prompt}
]
```

Figure 8: Prompt template - GPT4o and GPT3.5.

WizardCoder

```
Below is an instruction that describes
a task. Write a response that
appropriately completes the request.

### Instruction:
"mHumanEval[i]"

### Response:
```

Figure 9: Prompt Template - WizardCoder

Aya

```
messages = [{"role": "user",
"content": mHumanEval[i]}]
```

Figure 10: Prompt Template - Aya.

Claude3.5

```
system="Write a Python Code snippet
for the following: ",
prompt = mHumanEval[i] + "Make sure
your response includes a code block."

messages=[
    {"role": "user",
    "content": prompt}
]
```

Figure 11: Prompt template - Claude3-Opus.

LLaMA 3

```
messages = [
    {"role": "system", "content":
    "You are a helpful AI assistant,
    who writes Python Code."},
    {"role": "user", "content":
    mHumanEval[i]},
]
```

Figure 12: Prompt template - LLaMA 3.

DeepSeek-Coder

```
messages =
    {"role": "user", "content":
    "write a quick sort algorithm
    in python."}
```

Figure 13: Prompt template - DeepSeek-Coder.

CodeStral

```
prompt = "Below is a programming
problem, paired with a language in
which the solution should be written.
Write a solution in the provided that
appropriately solves the programming
problem.

### Problem:

mHumanEval[i]

### Language: python

### Solution: "
```

Figure 14: Prompt template - MagiCoder.

H Error Analysis - Examples

H.1 Task Misunderstanding

GPT4o generates a code for `finding significant digits`, when it is prompted to generate a code for `detecting prime numbers` in `Zulu`. This is due to the reason of the translation of `'prime number'` in Zulu becomes `'Inombolo Eyinhloko'` which means `'primary numbers'`. The model interprets it as `'significant number'` instead.

```
def find_significant_digits(number):

    # Convert the number to a string to handle it
    num_str = str(number)

    # Remove any leading/trailing zeros and
    # the decimal point
    significant_digits = ''.join([char for char
    in num_str if char.isdigit()])

    return significant_digits

# Example usage
number = 123.456
significant_digits = find_significant_digits(number)
print(f"Significant digits in {number} are:
{significant_digits}")
```

Figure 15: GP4o generated Python Code to `find significant digits`, when prompted to write code to `detect prime numbers` in `Zulu`.

Issues like this persist, even sometimes when prompted multiple times.

H.2 Multilingual Keyword Issues

Some code snippets include `keywords` and `identifiers` in the target language as well, resulting in compilation errors. As shown in Figure 16, when prompted in `Rundi`, to `find the largest number in a list`, Aya generated a piece of code, where keywords like `for` and `return` turned into `kuri` and `subiza`.

```
def igitigiri_cane_kuruta(imibare):
    namba not imibare:
        subiza None
    cane = imibare[0]
    kuri imibare hagati imibare:
        namba imibare > cane:
            cane = imibare
    subiza cane
```

Figure 16: When prompted in `Rundi`, Aya generated Python code to `find the largest number in a list`, containing multiple `keywords` in `Rundi`, resulting in Compilation Error.

These kinds of errors are also persistent but the outputs differ with similar prompts. Still generate the same issue again and again.

H.3 Garbage Results

```
#include <stdio.h>

int main() {
    int x = 10; x = x / (x - x);
    printf("%d\n", x);
    x = x * "Hello World!";
}
```

Figure 17: When prompted in `Sinhala`, to `reverse a list`, WizardCoder generated a garbage code in C.

I Experimental Setup

I.1 Machine Translation

GPT4o is accessed via API key, eliminating the need for GPU hours. Hyperparameter tuning is not conducted; instead, recommended values are utilized. The `max_tokens` parameter is set to 1000, and the `temperature` is maintained at 0.7. Additionally, Google Translate is accessed through API key, and the NLLB model is employed using a single NVIDIA A100 GPU with 40 GB of memory.

I.2 Code Generation

GPT4o, GPT3.5, and Claude3-Opus are accessed through API keys, thereby negating the necessity for GPU hours. We adhere to the recommended hyperparameters without conducting hyperparameter searches. The `max_tokens` parameter is set to 1000, and the `temperature` is maintained at 0.7.

For WizardCoder and Aya, we utilize the full precision (FP32) models without employing any quantized versions. These models are run on four NVIDIA A100 GPUs, each with 40 GB of memory. Hyperparameter settings are maintained as per the authors' recommendations without additional tuning.

For MagiCoder, LLaMA 3, and Phi-3-mini, the full precision (FP32) models are employed on a single NVIDIA A100 GPU with 40 GB of memory. Hyperparameter configurations are again set to the recommended values as specified by the authors.

J Evaluation Results: `mHumanEval-PL`

We evaluate the six LLMs from Table 4 for all 204 NLs in four different PLs. More specifically, we evaluate them on four subsets of `mHumanEval` - `mHumanEval`-C++, `mHumanEval`-JAVA, `mHumanEval`-JavaScript, and `mHumanEval`-Ruby. The results with `mHumanEval`-Python are presented in Figure 6 and discussed in Section 4. The performance trend is similar to Python, as discussed in Section 4. However, the results are slightly worse than those of Python.

J.1 `mHumanEval-C++`

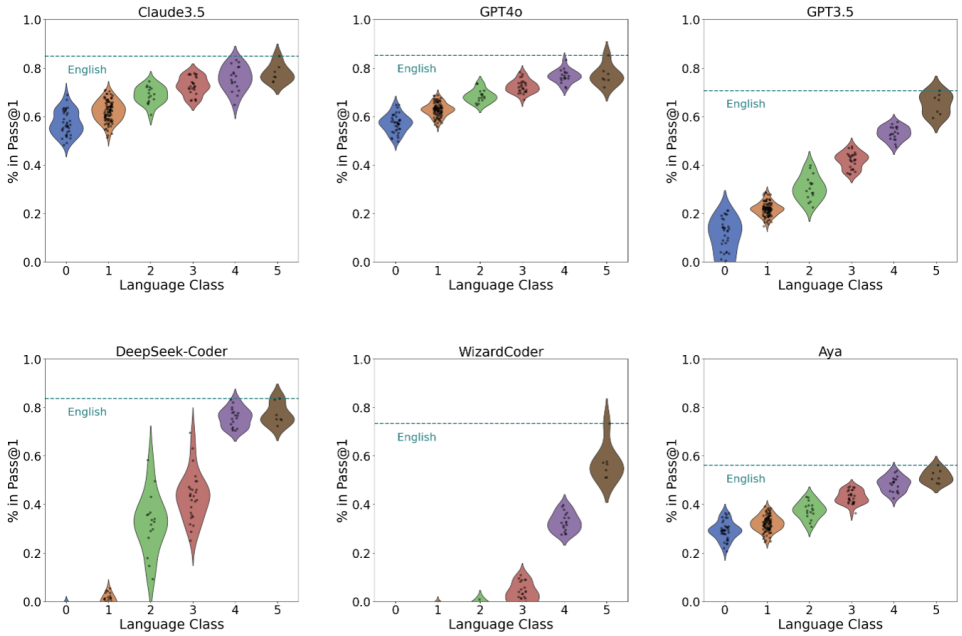

Figure 18: Comparing model performances (% in **`Pass@1`**) for the six models on mHumanEval-**C++**.

J.2 `mHumanEval-JAVA`

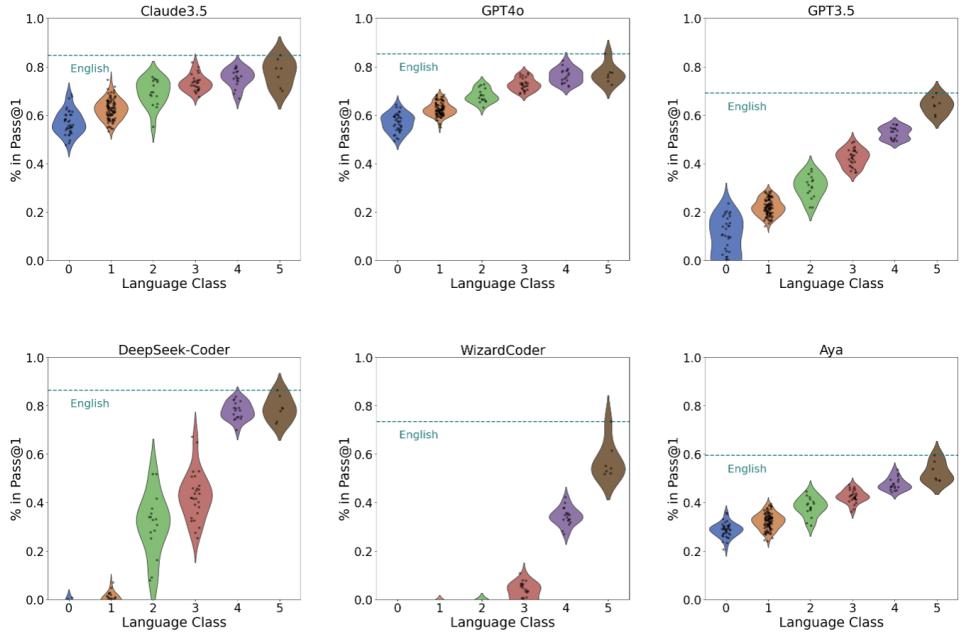

Figure 19: Comparing model performances (% in **`Pass@1`**) for the six models on mHumanEval-**JAVA**.

J.3 `mHumanEval`-JavaScript

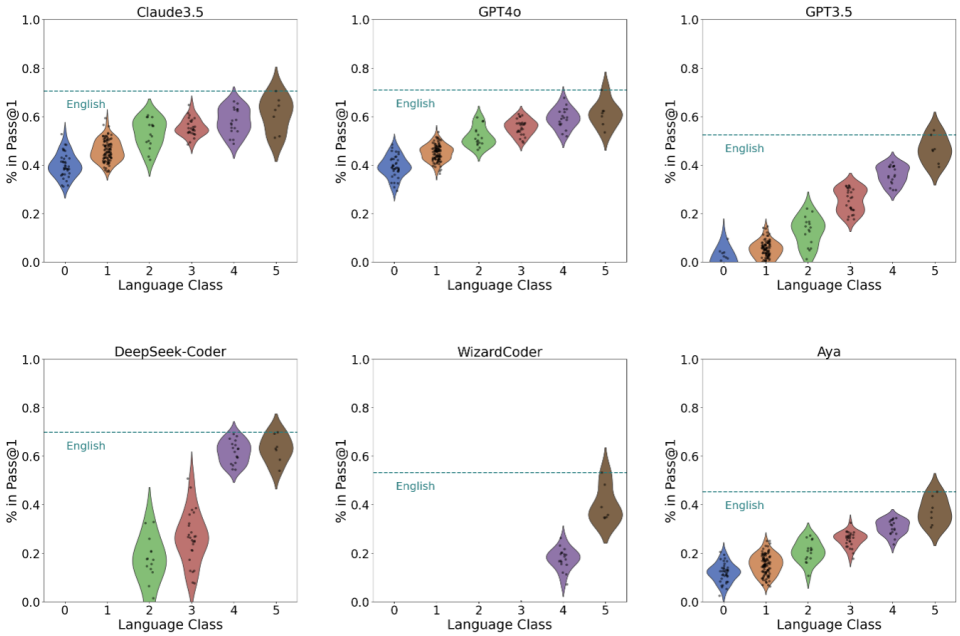

Figure 20: Comparing model performances (% in **`Pass@1`**) for the six models on mHumanEval-**JavaScript**.

J.4 `mHumanEval`-Ruby

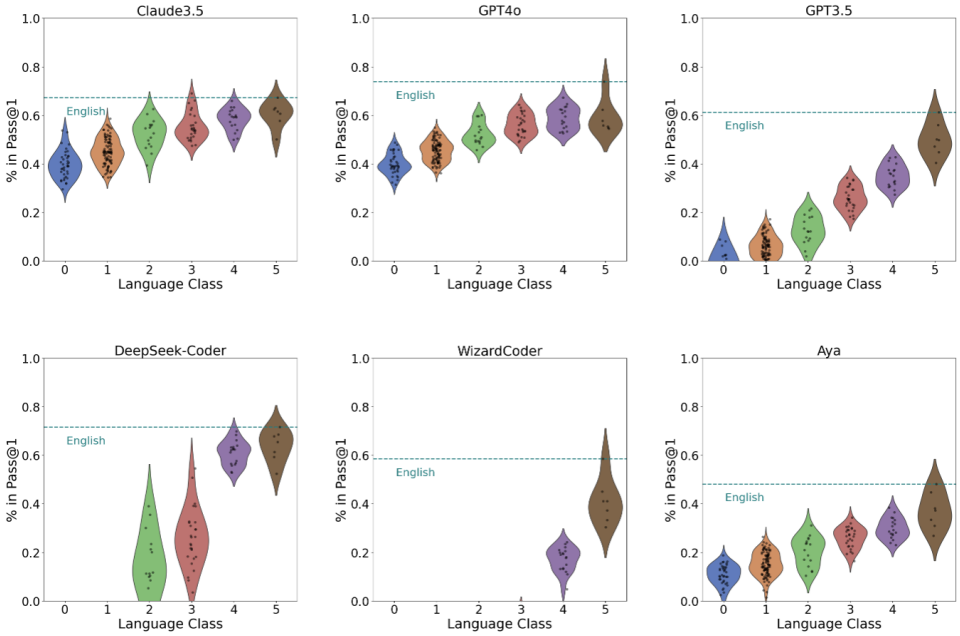

Figure 21: Comparing model performances (% in **`Pass@1`**) for the six models on mHumanEval-**Ruby**.

J.5 Analyzing PL-specific results

Performance Decline in Lower Classes (0-2) Models generally exhibit a noticeable performance decline in lower language classes, particularly Classes 0-2. Across all programming languages, scores in these classes fall well below the performance seen in Classes 4 and 5. This decline is especially pronounced in JavaScript and Ruby, where scores frequently drop to or near 0.000, suggesting these classes pose additional challenges.

Model	Python (C2)	Java (C2)	C++ (C2)	JavaScript (Cl1)	Ruby (C1)
GPT4o	0.600	0.590	0.591	0.000	0.000
GPT3.5	0.200	0.180	0.181	0.000	0.000
Claude3.5	0.620	0.600	0.601	0.478	0.473
DeepSeek-Coder	0.350	0.330	0.331	0.000	0.000

Table 10: Performance of models in lower classes (0-2) across programming languages, with pronounced drops, particularly in JavaScript and Ruby.

General Trends Across Language Classes In Classes 4 and 5, GPT-4 and Claude3.5 achieve high scores, often exceeding 0.85 in Python and Java. Python consistently demonstrates the highest scores, especially in Class 5, where models like GPT-4 and DeepSeek-Coder surpass 0.88. However, in Classes 0-3, performance drops across all models, particularly in JavaScript and Ruby, where scores frequently fall below 0.65.

Class	Python	Java	C++	JavaScript	Ruby
Class 5	0.880	0.850	0.852	0.650	0.653
Class 4	0.860	0.830	0.832	0.640	0.643
Class 3	0.750	0.720	0.721	0.530	0.533
Class 2	0.620	0.600	0.601	0.420	0.423

Table 11: General model performance across language classes, highlighting high scores in Classes 4 and 5 and lower scores in Classes 0-3, particularly in JavaScript and Ruby.

Underperformance of WizardCoder and Aya in JavaScript and Ruby Across Classes WizardCoder and Aya consistently struggle across all language classes in JavaScript and Ruby. In Classes 0-3, their scores frequently reach 0.000, underscoring limitations in handling these scripting languages regardless of language class.

Mixed Adaptability of DeepSeek-Coder Across Language Classes DeepSeek-Coder shows moderate scores in Python for higher classes (Classes 4 and 5) but drops to 0.000 in lower classes, particularly in JavaScript and Ruby, highlighting issues with adaptability across classes.

Model	Class 5	Class 4	Class 3	Class 2	Class 1
WizardCoder (JavaScript)	0.000	0.000	0.000	0.000	0.000
Aya (JavaScript)	0.186	0.165	0.143	0.120	0.100
WizardCoder (Ruby)	0.000	0.000	0.000	0.000	0.000
Aya (Ruby)	0.183	0.160	0.138	0.115	0.090

Table 12: Underperformance of WizardCoder and Aya in JavaScript and Ruby across language classes, with scores at 0.000 for WizardCoder across all classes.

Language Class	Python	Java	C++	JavaScript	Ruby
Class 5	0.880	0.850	0.852	0.000	0.000
Class 4	0.860	0.830	0.832	0.000	0.000
Class 3	0.500	0.480	0.482	0.000	0.000

Table 13: DeepSeek-Coder's performance across language classes, illustrating high scores in Python and Java in Classes 4 and 5, but collapsing to 0.000 in JavaScript and Ruby.

Claude3.5's Stable Performance Across Language Classes Claude3.5 consistently scores above 0.477 across all languages and classes, indicating versatility and robust adaptability across different language classes and programming languages.

Language Class	Python	Java	C++	JavaScript	Ruby
Class 5	0.880	0.850	0.852	0.483	0.477
Class 4	0.860	0.830	0.832	0.480	0.475
Class 3	0.750	0.720	0.721	0.480	0.475
Class 2	0.620	0.600	0.601	0.478	0.473

Table 14: Claude3.5's consistent performance across language classes and programming languages, with scores remaining stable above 0.477.

Implications for Future Model Development The significant underperformance in JavaScript and Ruby across language classes indicates a need for enhanced training in scripting languages. Models like GPT-4 and Claude3.5 excel in higher classes, particularly in Python and Java, but gaps in lower classes and scripting languages suggest a focus on diversifying training data to boost adaptability.

K Evaluating Prompt Translation by GPT4

Language	Class	BERTScore	CometKiwi	Language	Class	BERTScore	CometKiwi
arb_Arab	5	0.927	0.807	tha_Thai	3	0.874	0.749
deu_Latn	5	0.948	0.826	ukr_Cyrl	3	0.872	0.722
eng_Latn	5	1.000	0.930	urd_Arab	3	0.841	0.682
fra_Latn	5	0.927	0.807	uzn_Latn	3	0.885	0.740
jpn_Jpan	5	0.948	0.807	zsm_Latn	3	0.890	0.711
spa_Latn	5	0.927	0.839	amh_Ethi	2	0.825	0.690
zho_Hans	5	0.921	0.784	gle_Latn	2	0.824	0.666
cat_Latn	4	0.911	0.784	hau_Latn	2	0.810	0.666
ces_Latn	4	0.914	0.799	ibo_Latn	2	0.842	0.685
eus_Latn	4	0.920	0.754	kin_Latn	2	0.838	0.665
fin_Latn	4	0.870	0.798	lao_Laoo	2	0.844	0.690
hin_Deva	4	0.879	0.795	lug_Latn	2	0.824	0.685
hrv_Latn	4	0.921	0.768	lua_Latn	2	0.842	0.690
hun_Latn	4	0.879	0.784	luo_Latn	2	0.824	0.685
ita_Latn	4	0.916	0.768	mar_Deva	2	0.811	0.676
kor_Hang	4	0.930	0.768	npi_Deva	2	0.812	0.666
nld_Latn	4	0.887	0.799	orm_Latn	2	0.842	0.665
pes_Arab	4	0.929	0.768	prs_Arab	2	0.827	0.685
pol_Latn	4	0.894	0.754	quc_Latn	2	0.842	0.665
por_Latn	4	0.879	0.798	sag_Latn	2	0.811	0.676
rus_Cyrl	4	0.929	0.754	sna_Latn	2	0.812	0.666
srp_Cyrl	4	0.879	0.795	srd_Latn	2	0.842	0.665
swe_Latn	4	0.914	0.798	tso_Latn	2	0.842	0.665
tur_Latn	4	0.920	0.754	uzb_Latn	2	0.827	0.685
vie_Latn	4	0.894	0.795	zdj_Arab	2	0.811	0.676
arb_Latn	3	0.894	0.731	fuv_Latn	1	0.844	0.666
afr_Latn	3	0.890	0.711	gaz_Latn	1	0.839	0.665
arz_Arab	3	0.891	0.748	hin_Latn	1	0.841	0.682
ben_Beng	3	0.872	0.749	jav_Latn	1	0.776	0.508
bos_Latn	3	0.900	0.731	kan_Knda	1	0.755	0.489
bul_Cyrl	3	0.886	0.677	khm_Khmr	1	0.787	0.529
ceb_Latn	3	0.841	0.682	kir_Cyrl	1	0.765	0.578
dan_Latn	3	0.827	0.733	kmr_Latn	1	0.784	0.567
ell_Grek	3	0.887	0.727	mal_Mlym	1	0.785	0.529
est_Latn	3	0.885	0.715	mkd_Cyrl	1	0.737	0.578
glg_Latn	3	0.867	0.701	mya_Mymr	1	0.760	0.579
heb_Hebr	3	0.895	0.673	nob_Latn	1	0.750	0.515
ind_Latn	3	0.874	0.677	ory_Orya	1	0.776	0.579
kat_Geor	3	0.892	0.741	snd_Arab	1	0.788	0.432
kaz_Cyrl	3	0.850	0.745	som_Latn	1	0.750	0.512
lit_Latn	3	0.886	0.740	sun_Latn	1	0.778	0.567
lvs_Latn	3	0.827	0.688	tel_Telu	1	0.745	0.560
ron_Latn	3	0.895	0.722	uig_Arab	1	0.741	0.529
slk_Latn	3	0.886	0.731	ydd_Hebr	1	0.768	0.508
slv_Latn	3	0.890	0.745	zho_Hant	1	0.788	0.529
tam_Taml	3	0.887	0.677	sin_Sinh	0	0.690	0.410
tgl_Latn	3	0.827	0.731				

Table 15: Evaluating the quality of machine translation by GPT4 using `BERTScore` and `CometKiwi`. The languages are given as Flores-200 codes.

Language	Class	BERTScore	Language	Class	BERTScore
ace_Arab	1	0.666	quy_Latn	1	0.722
ace_Latn	1	0.719	sag_Latn	1	0.735
acm_Arab	1	0.680	sat_Olck	1	0.717
acq_Arab	1	0.714	scn_Latn	1	0.730
aeb_Arab	1	0.664	smo_Latn	1	0.738
ajp_Arab	1	0.675	sna_Latn	1	0.679
aka_Latn	1	0.683	srd_Latn	1	0.705
als_Latn	1	0.692	ssw_Latn	1	0.739
apc_Arab	1	0.697	szl_Latn	1	0.716
ars_Arab	1	0.727	tat_Cyrl	1	0.724
ary_Arab	1	0.700	tgk_Cyrl	1	0.673
ast_Latn	1	0.716	tpi_Latn	1	0.730
ayr_Latn	1	0.693	tso_Latn	1	0.696
azb_Arab	1	0.673	tuk_Latn	1	0.674
bak_Cyrl	1	0.702	tum_Latn	1	0.724
bho_Deva	1	0.708	twi_Latn	1	0.665
bjn_Arab	1	0.686	vec_Latn	1	0.698
bjn_Latn	1	0.701	war_Latn	1	0.735
bod_Tibt	1	0.735	awa_Deva	0	0.600
bug_Latn	1	0.678	bam_Latn	0	0.691
ckb_Arab	1	0.662	ban_Latn	0	0.677
crh_Latn	1	0.682	bem_Latn	0	0.656
dik_Latn	1	0.727	cjk_Latn	0	0.691
dzo_Tibt	1	0.727	dyu_Latn	0	0.648
ewe_Latn	1	0.706	fon_Latn	0	0.685
fao_Latn	1	0.662	fuv_Latn	0	0.613
fij_Latn	1	0.730	grn_Latn	0	0.680
fur_Latn	1	0.733	hat_Latn	0	0.649
gaz_Latn	1	0.669	hne_Deva	0	0.633
ibo_Latn	1	0.680	kac_Latn	0	0.665
ilo_Latn	1	0.723	kam_Latn	0	0.623
kab_Latn	1	0.690	kbp_Latn	0	0.646
kas_Arab	1	0.677	kea_Latn	0	0.698
kas_Deva	1	0.698	kmb_Latn	0	0.680
khk_Cyrl	1	0.686	knc_Arab	0	0.612
kik_Latn	1	0.694	knc_Latn	0	0.684
kin_Latn	1	0.704	kon_Latn	0	0.688
lij_Latn	1	0.683	lua_Latn	0	0.671
lim_Latn	1	0.691	luo_Latn	0	0.667
lin_Latn	1	0.699	lus_Latn	0	0.603
lmo_Latn	1	0.662	mag_Deva	0	0.665
ltg_Latn	1	0.720	mni_Beng	0	0.629
ltz_Latn	1	0.688	mos_Latn	0	0.610
lug_Latn	1	0.685	nso_Latn	0	0.601
mai_Deva	1	0.664	nus_Latn	0	0.614
min_Arab	1	0.669	nya_Latn	0	0.649
min_Latn	1	0.695	prs_Arab	0	0.698
mri_Latn	1	0.706	run_Latn	0	0.608
nno_Latn	1	0.703	shn_Mymr	0	0.670
npi_Deva	1	0.671	sot_Latn	0	0.624
oci_Latn	1	0.732	taq_Latn	0	0.658
pag_Latn	1	0.662	taq_Tfng	0	0.677
pap_Latn	1	0.728	tzm_Tfng	0	0.622
pbt_Arab	1	0.693	umb_Latn	0	0.643
plt_Latn	1	0.696	yue_Hant	0	0.696

Table 16: Evaluating the quality of machine translation by GPT4 using `BERTScore`. These languages are not supported by `CometKiwi`. The languages are given as Flores-200 codes.

L Evaluating Prompt Translation by NLLB

Language	Class	BERTScore	CometKiwi	Language	Class	BERTScore	CometKiwi
arb_Arab	5	0.941	0.798	tha_Thai	3	0.881	0.704
deu_Latn	5	0.902	0.809	ukr_Cyrl	3	0.883	0.685
eng_Latn	5	1.000	0.910	urd_Arab	3	0.872	0.697
fra_Latn	5	0.917	0.787	uzn_Latn	3	0.875	0.697
jpn_Jpan	5	0.935	0.807	zsm_Latn	3	0.864	0.697
spa_Latn	5	0.935	0.809	amh_Ethi	2	0.817	0.597
zho_Hans	5	0.911	0.831	gle_Latn	2	0.816	0.555
cat_Latn	4	0.909	0.777	hau_Latn	2	0.827	0.574
ces_Latn	4	0.899	0.743	ibo_Latn	2	0.816	0.579
eus_Latn	4	0.877	0.719	kin_Latn	2	0.817	0.573
fin_Latn	4	0.875	0.704	lao_Laoo	2	0.810	0.585
hin_Deva	4	0.875	0.697	lug_Latn	2	0.817	0.573
hrv_Latn	4	0.875	0.697	lua_Latn	2	0.816	0.579
hun_Latn	4	0.872	0.685	luo_Latn	2	0.818	0.585
ita_Latn	4	0.872	0.719	mar_Deva	2	0.817	0.573
kor_Hang	4	0.872	0.685	npi_Deva	2	0.816	0.579
nld_Latn	4	0.875	0.704	orm_Latn	2	0.817	0.573
pes_Arab	4	0.875	0.697	prs_Arab	2	0.816	0.579
pol_Latn	4	0.872	0.685	quc_Latn	2	0.818	0.585
por_Latn	4	0.875	0.704	sag_Latn	2	0.817	0.573
rus_Cyrl	4	0.875	0.697	sna_Latn	2	0.816	0.579
srp_Cyrl	4	0.872	0.685	srd_Latn	2	0.817	0.573
swe_Latn	4	0.875	0.704	tso_Latn	2	0.818	0.585
tur_Latn	4	0.872	0.685	uzb_Latn	2	0.817	0.573
vie_Latn	4	0.875	0.704	zdj_Arab	2	0.816	0.579
arb_Latn	3	0.875	0.697	fuv_Latn	1	0.818	0.585
afr_Latn	3	0.875	0.704	gaz_Latn	1	0.817	0.573
arz_Arab	3	0.872	0.685	hin_Latn	1	0.816	0.579
ben_Beng	3	0.872	0.697	jav_Latn	1	0.758	0.535
bul_Cyrl	3	0.875	0.697	kan_Knda	1	0.740	0.577
ceb_Latn	3	0.872	0.716	khm_Khmr	1	0.754	0.561
dan_Latn	3	0.851	0.666	kir_Cyrl	1	0.757	0.583
ell_Grek	3	0.883	0.709	kmr_Latn	1	0.770	0.579
est_Latn	3	0.877	0.661	mal_Mlym	1	0.736	0.550
glg_Latn	3	0.864	0.697	mkd_Cyrl	1	0.736	0.559
heb_Hebr	3	0.828	0.701	mya_Mymr	1	0.770	0.582
ind_Latn	3	0.864	0.697	nob_Latn	1	0.766	0.535
kat_Geor	3	0.880	0.709	ory_Orya	1	0.743	0.582
kaz_Cyrl	3	0.877	0.719	snd_Arab	1	0.743	0.582
lit_Latn	3	0.872	0.697	som_Latn	1	0.770	0.520
lvs_Latn	3	0.880	0.661	sun_Latn	1	0.743	0.540
ron_Latn	3	0.864	0.713	tel_Telu	1	0.754	0.540
slk_Latn	3	0.828	0.716	uig_Arab	1	0.751	0.579
slv_Latn	3	0.879	0.685	ydd_Hebr	1	0.757	0.556
tam_Taml	3	0.877	0.716	zho_Hant	1	0.736	0.583
tgl_Latn	3	0.851	0.713	sin_Sinh	0	0.645	0.490

Table 17: Evaluating the quality of machine translation by NLLB using `BERTScore` and `CometKiwi`. The languages are given as Flores-200 codes.

Language	Class	BERTScore	Language	Class	BERTScore
ace_Arab	1	0.672	quy_Latn	1	0.700
ace_Latn	1	0.716	sag_Latn	1	0.702
acm_Arab	1	0.664	sat_Olck	1	0.681
acq_Arab	1	0.669	scn_Latn	1	0.668
aeb_Arab	1	0.664	smo_Latn	1	0.739
ajp_Arab	1	0.690	sna_Latn	1	0.679
aka_Latn	1	0.720	srd_Latn	1	0.694
als_Latn	1	0.716	ssw_Latn	1	0.735
apc_Arab	1	0.683	szl_Latn	1	0.739
ars_Arab	1	0.669	tat_Cyrl	1	0.716
ary_Arab	1	0.666	tgk_Cyrl	1	0.672
ast_Latn	1	0.672	tpi_Latn	1	0.670
ayr_Latn	1	0.684	tso_Latn	1	0.706
azb_Arab	1	0.688	tuk_Latn	1	0.690
bak_Cyrl	1	0.699	tum_Latn	1	0.692
bho_Deva	1	0.719	twi_Latn	1	0.705
bjn_Arab	1	0.734	vec_Latn	1	0.709
bjn_Latn	1	0.668	war_Latn	1	0.684
bod_Tibt	1	0.692	awa_Deva	0	0.644
bug_Latn	1	0.670	bam_Latn	0	0.607
ckb_Arab	1	0.733	ban_Latn	0	0.645
crh_Latn	1	0.670	bem_Latn	0	0.613
dik_Latn	1	0.710	cjk_Latn	0	0.658
dzo_Tibt	1	0.726	dyu_Latn	0	0.664
ewe_Latn	1	0.737	fon_Latn	0	0.694
fao_Latn	1	0.710	fuv_Latn	0	0.615
fij_Latn	1	0.689	grn_Latn	0	0.677
fur_Latn	1	0.739	hat_Latn	0	0.666
gaz_Latn	1	0.708	hne_Deva	0	0.686
ibo_Latn	1	0.687	kac_Latn	0	0.651
ilo_Latn	1	0.722	kam_Latn	0	0.672
kab_Latn	1	0.680	kbp_Latn	0	0.600
kas_Arab	1	0.684	kea_Latn	0	0.672
kas_Deva	1	0.716	kmb_Latn	0	0.636
khk_Cyrl	1	0.725	knc_Arab	0	0.615
kik_Latn	1	0.668	knc_Latn	0	0.611
kin_Latn	1	0.705	kon_Latn	0	0.637
lij_Latn	1	0.719	lua_Latn	0	0.606
lim_Latn	1	0.706	luo_Latn	0	0.679
lin_Latn	1	0.723	lus_Latn	0	0.632
lmo_Latn	1	0.690	mag_Deva	0	0.600
ltg_Latn	1	0.681	mni_Beng	0	0.655
ltz_Latn	1	0.727	mos_Latn	0	0.688
lug_Latn	1	0.712	nso_Latn	0	0.635
mai_Deva	1	0.710	nus_Latn	0	0.674
min_Arab	1	0.666	nya_Latn	0	0.699
min_Latn	1	0.724	prs_Arab	0	0.609
mri_Latn	1	0.726	run_Latn	0	0.615
nno_Latn	1	0.703	shn_Mymr	0	0.657
npi_Deva	1	0.675	sot_Latn	0	0.601
oci_Latn	1	0.735	taq_Latn	0	0.658
pag_Latn	1	0.709	taq_Tfng	0	0.677
pap_Latn	1	0.664	tzm_Tfng	0	0.607
pbt_Arab	1	0.696	umb_Latn	0	0.643
plt_Latn	1	0.683	yue_Hant	0	0.677

Table 18: Evaluating the quality of machine translation by NLLB using `BERTScore`. These languages are not supported by `CometKiwi`. The languages are given as Flores-200 codes.

M Evaluating Prompt Translation by Google Translate

Language	Class	BERTScore	CometKiwi	Language	Class	BERTScore	CometKiwi
arb_Arab	5	0.886	0.802	lua_Latn	2	0.825	0.645
deu_Latn	5	0.910	0.811	ibo_Latn	2	0.816	0.637
eng_Latn	5	1.000	0.950	kin_Latn	2	0.829	0.617
fra_Latn	5	0.924	0.802	lao_Laoo	2	0.829	0.645
jpn_Jpan	5	0.910	0.812	lug_Latn	2	0.835	0.632
spa_Latn	5	0.900	0.802	luo_Latn	2	0.832	0.617
zho_Hans	5	0.933	0.807	mar_Deva	2	0.836	0.637
cat_Latn	4	0.903	0.689	npi_Deva	2	0.835	0.632
ces_Latn	4	0.894	0.722	orm_Latn	2	0.816	0.645
eus_Latn	4	0.898	0.725	prs_Arab	2	0.832	0.637
fin_Latn	4	0.842	0.655	quc_Latn	2	0.832	0.617
hin_Deva	4	0.853	0.630	sag_Latn	2	0.835	0.632
hrv_Latn	4	0.875	0.679	sna_Latn	2	0.829	0.645
hun_Latn	4	0.871	0.681	srd_Latn	2	0.836	0.637
ita_Latn	4	0.899	0.690	tso_Latn	2	0.823	0.604
kor_Hang	4	0.887	0.677	uzb_Latn	2	0.829	0.645
nld_Latn	4	0.886	0.676	zdj_Arab	2	0.835	0.632
pes_Arab	4	0.886	0.681	fuv_Latn	1	0.838	0.637
pol_Latn	4	0.881	0.666	gaz_Latn	1	0.841	0.637
por_Latn	4	0.880	0.689	hin_Latn	1	0.841	0.682
rus_Cyrl	4	0.887	0.667	jav_Latn	1	0.759	0.554
srp_Cyrl	4	0.880	0.669	kan_Knda	1	0.759	0.554
swe_Latn	4	0.878	0.676	khm_Khmr	1	0.767	0.517
tur_Latn	4	0.871	0.689	kir_Cyrl	1	0.766	0.522
vie_Latn	4	0.880	0.677	kmr_Latn	1	0.754	0.559
arb_Latn	3	0.898	0.726	mal_Mlym	1	0.766	0.573
afr_Latn	3	0.871	0.689	mkd_Cyrl	1	0.759	0.573
arz_Arab	3	0.880	0.667	mya_Mymr	1	0.767	0.576
ben_Beng	3	0.880	0.689	ory_Orya	1	0.766	0.558
bos_Latn	3	0.910	0.668	snd_Arab	1	0.754	0.576
bul_Cyrl	3	0.878	0.655	som_Latn	1	0.767	0.576
ceb_Latn	3	0.904	0.666	sun_Latn	1	0.766	0.554
dan_Latn	3	0.906	0.663	tel_Telu	1	0.766	0.574
ell_Grek	3	0.899	0.667	uig_Arab	1	0.766	0.558
est_Latn	3	0.893	0.645	ydd_Hebr	1	0.727	0.574
glg_Latn	3	0.837	0.635	zho_Hant	1	0.766	0.574
heb_Hebr	3	0.884	0.639	als_Latn	1	0.665	–
ind_Latn	3	0.880	0.669	azb_Arab	1	0.705	–
kat_Geor	3	0.871	0.661	ckb_Arab	1	0.720	–
kaz_Cyrl	3	0.841	0.654	khk_Cyrl	1	0.684	–
lat_Latn	3	0.910	0.667	mri_Latn	1	0.680	–
lit_Latn	3	0.897	0.630	npi_Deva	1	0.662	–
lvs_Latn	3	0.878	0.668	plt_Latn	1	0.673	–
ron_Latn	3	0.884	0.635	sna_Latn	1	0.713	–
slk_Latn	3	0.899	0.645	cos_Latn	1	0.714	–
slv_Latn	3	0.897	0.667	haw_Latn	1	0.719	–
tam_Taml	3	0.884	0.635	ibo_Latn	1	0.700	–
tgl_Latn	3	0.878	0.667	ltz_Latn	1	0.711	–
tha_Thai	3	0.884	0.635	nno_Latn	1	0.721	–
ukr_Cyrl	3	0.904	0.668	pbt_Arab	1	0.686	–
urd_Arab	3	0.884	0.655	smo_Latn	1	0.733	–
uzn_Latn	3	0.910	0.630	tgk_Cyrl	1	0.693	–
zsm_Latn	3	0.906	0.645	fry_Latn	0	0.683	0.522
amh_Ethi	2	0.835	0.623	sin_Sinh	0	0.685	0.410
gle_Latn	2	0.835	0.645	hat_Latn	0	0.608	–
hau_Latn	2	0.823	0.604	hmn_Latn	0	0.624	–
ibo_Latn	2	0.816	0.637	sot_Latn	0	0.623	–
kin_Latn	2	0.829	0.617	mni_Beng	0	0.650	–
lao_Laoo	2	0.829	0.645	nya_Latn	0	0.682	–
lug_Latn	2	0.835	0.632				

Table 19: Evaluating the quality of machine translation by Google Translator using `BERTScore` and `CometKiwi`. The languages are given as Flores-200 codes.

N Evaluating Prompt Quality in mHumanEval

Language	Class	BERTScore	CometKiwi	Language	Class	BERTScore	CometKiwi
eng_Latn	5	1.000	0.961	urd_Arab	3	0.911	0.782
spa_Latn	5	0.98	0.919	bul_Cyrl	3	0.956	0.777
deu_Latn	5	0.99	0.927	ind_Latn	3	0.944	0.777
fra_Latn	5	0.98	0.896	tam_Taml	3	0.957	0.777
zho_Hans	5	0.96	0.89	heb_Hebr	3	0.965	0.773
jpn_Jpan	5	0.97	0.88	amh_Ethi	2	0.895	0.77
arb_Arab	5	0.96	0.867	mlt_Latn	2	0.904	0.77
ces_Latn	4	0.964	0.839	isl_Latn	2	0.898	0.765
nld_Latn	4	0.937	0.839	tir_Ethi	2	0.913	0.765
fin_Latn	4	0.92	0.838	yor_Latn	2	0.913	0.765
por_Latn	4	0.929	0.838	zul_Latn	2	0.913	0.765
swe_Latn	4	0.964	0.838	lao_Laoo	2	0.887	0.756
hin_Deva	4	0.929	0.835	mar_Deva	2	0.881	0.756
srp_Cyrl	4	0.929	0.835	xho_Latn	2	0.881	0.756
vie_Latn	4	0.944	0.835	gle_Latn	2	0.894	0.746
cat_Latn	4	0.961	0.824	pan_Guru	2	0.916	0.746
hun_Latn	4	0.929	0.824	san_Deva	2	0.925	0.746
hrv_Latn	4	0.971	0.808	wol_Latn	2	0.884	0.746
ita_Latn	4	0.966	0.808	hau_Latn	2	0.884	0.745
kor_Hang	4	0.98	0.808	swh_Latn	2	0.913	0.745
pes_Arab	4	0.979	0.808	tsn_Latn	2	0.925	0.745
eus_Latn	4	0.97	0.794	guj_Gujr	1	0.828	0.717
pol_Latn	4	0.944	0.794	epo_Latn	1	0.813	0.709
rus_Cyrl	4	0.979	0.794	mya_Mymr	1	0.828	0.709
tur_Latn	4	0.97	0.794	ory_Orya	1	0.84	0.709
ben_Beng	3	0.942	0.849	kir_Cyrl	1	0.819	0.708
tha_Thai	3	0.944	0.849	mkd_Cyrl	1	0.873	0.708
arz_Arab	3	0.961	0.848	cym_Latn	1	0.865	0.707
kaz_Cyrl	3	0.92	0.845	kmr_Latn	1	0.86	0.697
slv_Latn	3	0.96	0.845	sun_Latn	1	0.854	0.697
kat_Geor	3	0.962	0.841	gla_Latn	1	0.846	0.69
lit_Latn	3	0.956	0.84	tel_Telu	1	0.823	0.69
uzn_Latn	3	0.955	0.84	khm_Khmr	1	0.867	0.659
bos_Latn	3	0.97	0.839	mal_Mlym	1	0.871	0.659
dan_Latn	3	0.897	0.833	uig_Arab	1	0.809	0.659
arb_Latn	3	0.964	0.831	zho_Hant	1	0.876	0.659
slk_Latn	3	0.956	0.831	asm_Beng	1	0.871	0.645
tgl_Latn	3	0.897	0.831	nob_Latn	1	0.823	0.645
ell_Grek	3	0.957	0.827	hye_Armn	1	0.852	0.642
ron_Latn	3	0.965	0.822	som_Latn	1	0.812	0.642
ukr_Cyrl	3	0.942	0.822	jav_Latn	1	0.828	0.638
est_Latn	3	0.955	0.815	ydd_Hebr	1	0.875	0.638
afr_Latn	3	0.96	0.811	kan_Knda	1	0.825	0.619
zsm_Latn	3	0.96	0.811	bel_Cyrl	1	0.809	0.613
glg_Latn	3	0.937	0.801	azj_Latn	1	0.829	0.562
lvs_Latn	3	0.897	0.788	snd_Arab	1	0.851	0.562
ceb_Latn	3	0.911	0.782	sin_Sinh	0	0.859	0.56

Table 20: Observing improved prompt quality in mHumanEval upon choosing the best ones from 13 candidates each, evaluated using BERTScore and CometKiwi. The languages are given as Flores-200 codes.

Language	Class	BERTScore	Language	Class	BERTScore
ace_Arab	1	0.806	quy_Latn	1	0.862
ace_Latn	1	0.859	sag_Latn	1	0.875
acm_Arab	1	0.82	sat_Olck	1	0.857
acq_Arab	1	0.854	scn_Latn	1	0.87
aeb_Arab	1	0.804	smo_Latn	1	0.878
ajp_Arab	1	0.815	sna_Latn	1	0.819
aka_Latn	1	0.823	srd_Latn	1	0.845
als_Latn	1	0.832	ssw_Latn	1	0.879
apc_Arab	1	0.837	szl_Latn	1	0.856
ars_Arab	1	0.867	tat_Cyrl	1	0.864
ary_Arab	1	0.84	tgk_Cyrl	1	0.813
ast_Latn	1	0.856	tpi_Latn	1	0.87
ayr_Latn	1	0.833	tso_Latn	1	0.836
azb_Arab	1	0.813	tuk_Latn	1	0.814
bak_Cyrl	1	0.842	tum_Latn	1	0.864
bho_Deva	1	0.848	twi_Latn	1	0.805
bjn_Arab	1	0.826	vec_Latn	1	0.838
bjn_Latn	1	0.841	war_Latn	1	0.875
bod_Tibt	1	0.875	awa_Deva	0	0.75
bug_Latn	1	0.818	bam_Latn	0	0.841
ckb_Arab	1	0.802	ban_Latn	0	0.827
crh_Latn	1	0.822	bem_Latn	0	0.806
dik_Latn	1	0.867	cjk_Latn	0	0.841
dzo_Tibt	1	0.867	dyu_Latn	0	0.798
ewe_Latn	1	0.846	fon_Latn	0	0.835
fao_Latn	1	0.802	fuv_Latn	0	0.763
fij_Latn	1	0.87	grn_Latn	0	0.83
fur_Latn	1	0.873	hat_Latn	0	0.799
gaz_Latn	1	0.809	hne_Deva	0	0.783
ibo_Latn	1	0.82	kac_Latn	0	0.815
ilo_Latn	1	0.863	kam_Latn	0	0.773
kab_Latn	1	0.83	kbp_Latn	0	0.796
kas_Arab	1	0.817	kea_Latn	0	0.848
kas_Deva	1	0.838	kmb_Latn	0	0.83
khk_Cyrl	1	0.826	knc_Arab	0	0.762
kik_Latn	1	0.834	knc_Latn	0	0.834
kin_Latn	1	0.844	kon_Latn	0	0.838
lij_Latn	1	0.823	lua_Latn	0	0.821
lim_Latn	1	0.831	luo_Latn	0	0.817
lin_Latn	1	0.839	lus_Latn	0	0.753
lmo_Latn	1	0.802	mag_Deva	0	0.815
ltg_Latn	1	0.86	mni_Beng	0	0.779
ltz_Latn	1	0.828	mos_Latn	0	0.76
lug_Latn	1	0.825	nso_Latn	0	0.751
mai_Deva	1	0.804	nus_Latn	0	0.764
min_Arab	1	0.809	nya_Latn	0	0.799
min_Latn	1	0.835	prs_Arab	0	0.848
mri_Latn	1	0.846	run_Latn	0	0.758
nno_Latn	1	0.843	shn_Mymr	0	0.82
npi_Deva	1	0.811	sot_Latn	0	0.774
oci_Latn	1	0.872	taq_Latn	0	0.836
pag_Latn	1	0.802	taq_Tfng	0	0.774
pap_Latn	1	0.868	tzm_Tfng	0	0.772
pbt_Arab	1	0.833	umb_Latn	0	0.795
plt_Latn	1	0.836	yue_Hant	0	0.846

Table 21: Observing improved prompt quality in `mHumanEval` upon choosing the best ones from 13 candidates each, evaluated using `BERTScore`. These languages are not supported by `CometKiwi`. The languages are given as Flores-200 codes.

O Evaluation Results on mHumanEval

Language	Class	Claude3.5	GPT4o	GPT3.5	DeepSeek-Coder	WizardCoder	Aya
arb_Arab	5	0.831	0.846	0.719	0.859	0.650	0.590
deu_Latn	5	0.846	0.833	0.730	0.863	0.670	0.620
eng_Latn	5	0.938	0.910	0.770	0.902	0.800	0.650
fra_Latn	5	0.835	0.850	0.693	0.849	0.650	0.608
jpn_Jpan	5	0.896	0.868	0.757	0.849	0.670	0.609
spa_Latn	5	0.880	0.852	0.759	0.854	0.610	0.609
zho_Hans	5	0.838	0.810	0.720	0.933	0.590	0.570

Table 22: Comparing LLMs' performance (% in **Pass@1**) on mHumanEval - Class 5 languages. The languages are given as Flores-200 codes.

Language	Class	Claude3.5	GPT4o	GPT3.5	DeepSeek-Coder	WizardCoder	Aya
cat_Latn	4	0.764	0.832	0.613	0.827	0.420	0.584
ces_Latn	4	0.908	0.837	0.649	0.883	0.390	0.591
eus_Latn	4	0.880	0.884	0.617	0.902	0.480	0.599
fin_Latn	4	0.857	0.882	0.611	0.882	0.390	0.565
hin_Deva	4	0.854	0.859	0.600	0.872	0.480	0.572
hrv_Latn	4	0.831	0.833	0.608	0.865	0.450	0.595
hun_Latn	4	0.838	0.860	0.594	0.824	0.410	0.568
ita_Latn	4	0.870	0.860	0.607	0.796	0.430	0.563
kor_Hang	4	0.814	0.850	0.605	0.909	0.390	0.577
nld_Latn	4	0.809	0.849	0.649	0.843	0.440	0.546
pes_Arab	4	0.885	0.859	0.607	0.902	0.380	0.586
pol_Latn	4	0.840	0.850	0.634	0.821	0.390	0.569
por_Latn	4	0.861	0.862	0.657	0.835	0.440	0.576
rus_Cyrl	4	0.814	0.822	0.615	0.831	0.470	0.565
srp_Cyrl	4	0.815	0.842	0.591	0.892	0.400	0.595
swe_Latn	4	0.832	0.840	0.634	0.867	0.380	0.551
tur_Latn	4	0.867	0.860	0.618	0.882	0.480	0.585
vie_Latn	4	0.883	0.833	0.637	0.833	0.400	0.591

Table 23: Comparing LLMs' performance (% in **Pass@1**) on mHumanEval - Class 4 languages. The languages are given as Flores-200 codes.

Language	Class	Claude3.5	GPT4o	GPT3.5	DeepSeek-Coder	WizardCoder	Aya
afr_Latn	3	0.886	0.846	0.542	0.554	0.180	0.505
arb_Latn	3	0.792	0.839	0.548	0.592	0.110	0.541
arz_Arab	3	0.807	0.832	0.495	0.399	0.130	0.528
ben_Beng	3	0.797	0.792	0.541	0.565	0.090	0.523
bos_Latn	3	0.826	0.812	0.502	0.746	0.140	0.546
bul_Cyrl	3	0.848	0.796	0.491	0.379	0.120	0.536
ceb_Latn	3	0.850	0.827	0.499	0.473	0.150	0.479
dan_Latn	3	0.825	0.825	0.504	0.533	0.090	0.527
ell_Grek	3	0.742	0.784	0.484	0.479	0.180	0.539
est_Latn	3	0.821	0.786	0.529	0.554	0.090	0.516
glg_Latn	3	0.820	0.805	0.531	0.407	0.110	0.492
heb_Hebr	3	0.837	0.847	0.494	0.449	0.090	0.518
ind_Latn	3	0.849	0.809	0.478	0.511	0.080	0.482
kat_Geor	3	0.836	0.849	0.548	0.507	0.110	0.532
kaz_Cyrl	3	0.814	0.824	0.522	0.715	0.110	0.543
lit_Latn	3	0.788	0.812	0.491	0.413	0.140	0.476
lvs_Latn	3	0.791	0.798	0.522	0.555	0.140	0.520
ron_Latn	3	0.830	0.829	0.507	0.491	0.090	0.488
slk_Latn	3	0.772	0.822	0.501	0.440	0.120	0.528
slv_Latn	3	0.784	0.784	0.495	0.619	0.090	0.545
tam_Taml	3	0.837	0.818	0.532	0.529	0.160	0.526
tgl_Latn	3	0.794	0.836	0.485	0.342	0.140	0.473
tha_Thai	3	0.829	0.823	0.538	0.642	0.080	0.488
ukr_Cyrl	3	0.846	0.837	0.546	0.507	0.060	0.505
urd_Arab	3	0.794	0.823	0.477	0.513	0.110	0.537
uzn_Latn	3	0.847	0.838	0.516	0.591	0.170	0.540
zsm_Latn	3	0.826	0.804	0.483	0.543	0.080	0.514

Table 24: Comparing LLMs' performance (% in **`Pass@1`**) on `mHumanEval` - Class 3 languages. The languages are given as Flores-200 codes.

Language	Class	Claude3.5	GPT4o	GPT3.5	DeepSeek-Coder	WizardCoder	Aya
amh_Ethi	2	0.765	0.742	0.373	0.214	0.020	0.454
gle_Latn	2	0.753	0.748	0.466	0.425	0.010	0.449
hau_Latn	2	0.670	0.739	0.431	0.382	0.070	0.447
isl_Latn	2	0.795	0.770	0.419	0.606	0.030	0.439
lao_Laoo	2	0.783	0.745	0.449	0.440	0.050	0.516
mar_Deva	2	0.764	0.773	0.464	0.493	0.050	0.519
mlt_Latn	2	0.826	0.790	0.348	0.184	0.020	0.439
pan_Guru	2	0.730	0.747	0.363	0.356	0.060	0.496
san_Deva	2	0.799	0.799	0.391	0.407	0.050	0.496
swh_Latn	2	0.801	0.794	0.363	0.363	0.030	0.488
tir_Ethi	2	0.802	0.792	0.343	0.457	0.020	0.473
tsn_Latn	2	0.786	0.781	0.396	0.464	0.040	0.468
wol_Latn	2	0.835	0.799	0.333	0.435	0.030	0.430
xho_Latn	2	0.805	0.756	0.486	0.644	0.050	0.490
yor_Latn	2	0.771	0.773	0.414	0.364	0.060	0.490
zul_Latn	2	0.847	0.791	0.364	0.267	0.050	0.526

Table 25: Comparing LLMs' performance (% in **`Pass@1`**) on `mHumanEval` - Class 2 languages. The languages are given as Flores-200 codes.

Language	Class	Claude3.5	GPT4o	GPT3.5	DeepSeek-Coder	WizardCoder	Aya
ace_Arab	1	0.812	0.736	0.281	0.070	0.050	0.423
ace_Latn	1	0.712	0.675	0.338	0.005	0.040	0.433
acm_Arab	1	0.673	0.671	0.276	0.019	0.010	0.437
acq_Arab	1	0.786	0.750	0.284	0.019	0.030	0.387
aeb_Arab	1	0.716	0.739	0.324	0.036	0.010	0.413
ajp_Arab	1	0.640	0.686	0.282	0.046	0.030	0.376
aka_Latn	1	0.687	0.708	0.345	0.124	0.020	0.371
als_Latn	1	0.739	0.720	0.272	0.081	0.050	0.414
apc_Arab	1	0.686	0.704	0.309	0.055	0.040	0.372
ars_Arab	1	0.713	0.699	0.315	0.028	0.040	0.374
ary_Arab	1	0.695	0.707	0.330	0.016	0.020	0.436
asm_Beng	1	0.652	0.671	0.338	0.000	0.040	0.416
ast_Latn	1	0.690	0.724	0.298	0.096	0.040	0.405
ayr_Latn	1	0.728	0.733	0.284	0.015	0.060	0.447
azb_Arab	1	0.684	0.688	0.290	0.046	0.050	0.419
azj_Latn	1	0.727	0.726	0.296	0.046	0.020	0.435
bak_Cyrl	1	0.743	0.722	0.326	0.049	0.030	0.435
bel_Cyrl	1	0.705	0.705	0.297	0.067	0.020	0.378
bho_Deva	1	0.705	0.747	0.338	0.065	0.050	0.378
bjn_Arab	1	0.709	0.697	0.272	0.092	0.030	0.383
bjn_Latn	1	0.733	0.716	0.276	0.062	0.050	0.407
bod_Tibt	1	0.769	0.730	0.291	0.012	0.010	0.427
bug_Latn	1	0.661	0.686	0.350	0.016	0.050	0.447
ckb_Arab	1	0.650	0.685	0.288	0.043	0.030	0.387
crh_Latn	1	0.703	0.731	0.285	0.024	0.020	0.418
cym_Latn	1	0.779	0.747	0.318	0.000	0.040	0.424
dik_Latn	1	0.713	0.711	0.335	0.027	0.030	0.382
dzo_Tibt	1	0.682	0.701	0.300	0.014	0.010	0.419
epo_Latn	1	0.714	0.718	0.313	0.103	0.040	0.409
ewe_Latn	1	0.653	0.674	0.309	0.051	0.020	0.371
fao_Latn	1	0.677	0.729	0.318	0.073	0.040	0.400
fij_Latn	1	0.657	0.713	0.326	0.064	0.050	0.386
fur_Latn	1	0.713	0.690	0.326	0.042	0.030	0.397
gaz_Latn	1	0.692	0.741	0.285	0.000	0.020	0.421
gla_Latn	1	0.722	0.688	0.319	0.051	0.030	0.382
guj_Gujr	1	0.762	0.730	0.273	0.000	0.020	0.392
hye_Armn	1	0.761	0.735	0.290	0.069	0.020	0.400
ibo_Latn	1	0.732	0.707	0.285	0.000	0.010	0.393
ilo_Latn	1	0.752	0.706	0.336	0.096	0.060	0.381
jav_Latn	1	0.771	0.747	0.286	0.004	0.040	0.386
kab_Latn	1	0.777	0.738	0.337	0.077	0.050	0.434
kan_Knda	1	0.747	0.745	0.296	0.007	0.040	0.436
kas_Arab	1	0.707	0.705	0.294	0.004	0.010	0.412
kas_Deva	1	0.733	0.698	0.317	0.000	0.050	0.398
khk_Cyrl	1	0.745	0.730	0.289	0.043	0.020	0.413
khm_Khmr	1	0.682	0.739	0.310	0.029	0.030	0.444
kik_Latn	1	0.656	0.719	0.314	0.080	0.040	0.413
kin_Latn	1	0.676	0.695	0.328	0.025	0.020	0.422
kir_Cyrl	1	0.689	0.693	0.276	0.056	0.050	0.412
kmr_Latn	1	0.735	0.723	0.294	0.105	0.040	0.378
lij_Latn	1	0.725	0.732	0.294	0.034	0.050	0.423
lim_Latn	1	0.750	0.727	0.349	0.032	0.050	0.384
lin_Latn	1	0.722	0.721	0.295	0.003	0.050	0.409
lmo_Latn	1	0.781	0.716	0.331	0.014	0.020	0.443
ltg_Latn	1	0.690	0.698	0.325	0.083	0.050	0.418
ltz_Latn	1	0.688	0.676	0.312	0.104	0.060	0.383
lug_Latn	1	0.669	0.673	0.317	0.000	0.050	0.449
mai_Deva	1	0.721	0.679	0.292	0.000	0.050	0.388
mal_Mlym	1	0.748	0.728	0.293	0.033	0.050	0.370
min_Arab	1	0.673	0.698	0.333	0.000	0.010	0.424
min_Latn	1	0.757	0.737	0.291	0.000	0.030	0.375
mkd_Cyrl	1	0.739	0.696	0.322	0.099	0.050	0.450
mri_Latn	1	0.703	0.708	0.310	0.050	0.020	0.439
mya_Mymr	1	0.744	0.710	0.329	0.009	0.020	0.441
nno_Latn	1	0.642	0.704	0.340	0.047	0.030	0.380
nob_Latn	1	0.689	0.733	0.311	0.010	0.040	0.425
npi_Deva	1	0.740	0.715	0.272	0.015	0.040	0.385
nno_Latn	1	0.642	0.704	0.340	0.047	0.030	0.380
nob_Latn	1	0.689	0.733	0.311	0.010	0.040	0.425
npi_Deva	1	0.740	0.715	0.272	0.015	0.040	0.385
oci_Latn	1	0.714	0.701	0.286	0.020	0.020	0.417
ory_Orya	1	0.700	0.714	0.307	0.064	0.050	0.438
pag_Latn	1	0.690	0.723	0.294	0.051	0.050	0.393
pap_Latn	1	0.764	0.729	0.347	0.095	0.020	0.393
pbt_Arab	1	0.706	0.722	0.281	0.076	0.030	0.446
plt_Latn	1	0.706	0.717	0.286	0.000	0.040	0.371
quy_Latn	1	0.685	0.689	0.334	0.072	0.050	0.374
sag_Latn	1	0.710	0.740	0.271	0.103	0.060	0.438
sat_Olck	1	0.702	0.708	0.320	0.020	0.040	0.408
scn_Latn	1	0.687	0.703	0.295	0.039	0.040	0.422
smo_Latn	1	0.706	0.699	0.321	0.049	0.040	0.377
sna_Latn	1	0.676	0.697	0.320	0.048	0.050	0.444
snd_Arab	1	0.714	0.717	0.344	0.021	0.040	0.425
som_Latn	1	0.732	0.718	0.339	0.000	0.030	0.392
srd_Latn	1	0.710	0.745	0.343	0.000	0.050	0.441
ssw_Latn	1	0.708	0.687	0.310	0.014	0.030	0.407
sun_Latn	1	0.728	0.718	0.321	0.060	0.030	0.427
szl_Latn	1	0.752	0.735	0.311	0.069	0.060	0.436
tat_Cyrl	1	0.719	0.709	0.315	0.056	0.050	0.420
tel_Telu	1	0.708	0.676	0.347	0.107	0.060	0.397
tgk_Cyrl	1	0.669	0.690	0.328	0.026	0.050	0.404
tpi_Latn	1	0.699	0.738	0.327	0.081	0.060	0.372
tso_Latn	1	0.777	0.728	0.287	0.042	0.040	0.394
tuk_Latn	1	0.707	0.711	0.284	0.042	0.050	0.417
tum_Latn	1	0.669	0.702	0.286	0.017	0.020	0.411
twi_Latn	1	0.749	0.737	0.302	0.000	0.020	0.414
uig_Arab	1	0.645	0.694	0.325	0.021	0.020	0.429
vec_Latn	1	0.744	0.743	0.336	0.033	0.020	0.380
war_Latn	1	0.681	0.717	0.270	0.041	0.020	0.402
ydd_Hebr	1	0.719	0.722	0.338	0.007	0.040	0.390
zho_Hant	1	0.636	0.680	0.300	0.023	0.020	0.385

Table 26: Comparing LLMs' performance (% in **Pass@1**) on mHumanEval - Class 1 languages. The languages are given as Flores-200 codes.

Language	Class	Claude3.5	GPT4o	GPT3.5	DeepSeek-Coder	WizardCoder	Aya
awa_Deva	0	0.653	0.628	0.191	0.033	0.020	0.353
bam_Latn	0	0.645	0.634	0.268	0.081	0.010	0.410
ban_Latn	0	0.639	0.641	0.285	0.060	0.010	0.398
bem_Latn	0	0.675	0.654	0.308	0.000	0.000	0.415
cjk_Latn	0	0.750	0.720	0.316	0.000	0.010	0.366
dyu_Latn	0	0.620	0.636	0.039	0.000	0.010	0.367
fon_Latn	0	0.719	0.658	0.072	0.016	0.000	0.396
fuv_Latn	0	0.657	0.665	0.212	0.000	0.010	0.357
grn_Latn	0	0.698	0.689	0.021	0.000	0.010	0.356
hat_Latn	0	0.597	0.621	0.142	0.012	0.010	0.363
hne_Deva	0	0.670	0.626	0.215	0.008	0.000	0.403
kac_Latn	0	0.679	0.670	0.047	0.051	0.000	0.332
kam_Latn	0	0.637	0.673	0.140	0.057	0.020	0.383
kbp_Latn	0	0.694	0.683	0.107	0.000	0.000	0.376
kea_Latn	0	0.677	0.720	0.065	0.000	0.010	0.346
kmb_Latn	0	0.667	0.661	0.175	0.000	0.000	0.381
knc_Arab	0	0.664	0.647	0.218	0.000	0.020	0.398
knc_Latn	0	0.586	0.621	0.291	0.061	0.010	0.348
kon_Latn	0	0.745	0.691	0.093	0.000	0.010	0.361
lua_Latn	0	0.689	0.660	0.283	0.000	0.010	0.411
luo_Latn	0	0.692	0.615	0.228	0.004	0.020	0.380
lus_Latn	0	0.616	0.640	0.132	0.018	0.000	0.383
mag_Deva	0	0.657	0.700	0.128	0.000	0.010	0.418
mni_Beng	0	0.574	0.628	0.275	0.033	0.010	0.368
mos_Latn	0	0.659	0.657	0.232	0.021	0.010	0.414
nso_Latn	0	0.635	0.647	0.038	0.000	0.000	0.408
nus_Latn	0	0.636	0.707	0.227	0.018	0.000	0.418
nya_Latn	0	0.746	0.667	0.124	0.000	0.000	0.387
prs_Arab	0	0.633	0.644	0.283	0.000	0.010	0.364
run_Latn	0	0.715	0.707	0.252	0.005	0.000	0.382
shn_Mymr	0	0.664	0.637	0.214	0.044	0.020	0.377
sin_Sinh	0	0.645	0.633	0.187	0.000	0.020	0.391
sot_Latn	0	0.723	0.703	0.194	0.053	0.010	0.417
taq_Latn	0	0.655	0.671	0.042	0.052	0.020	0.383
taq_Tfng	0	0.643	0.639	0.128	0.000	0.020	0.351
tzm_Tfng	0	0.654	0.670	0.114	0.014	0.020	0.376
umb_Latn	0	0.647	0.622	0.176	0.000	0.000	0.372
yue_Hant	0	0.613	0.666	0.282	0.021	0.020	0.419

Table 27: Comparing LLMs' performance (% in **`Pass@1`**) on `mHumanEval` - Class 0 languages. The languages are given as Flores-200 codes.