

# The Cost of Asking in Turkish: Prompt Language Effects on Code LLMs

Mert İncidelen, Melike Kaya, Elif İşbilir, Yunus Santur

Department of Artificial Intelligence and Data Engineering, Firat University, Elazığ, Türkiye  
mincidelen@firat.edu.tr, km.kaya.melike@gmail.com  
isbilirelif.ie@gmail.com, ysantur@firat.edu.tr

## Abstract

Large language models (LLMs) have demonstrated strong performance on existing code generation benchmarks. However, these benchmarks are almost entirely based on English prompts. This study examines the cost of prompt language on code generation performance for Turkish, an agglutinative and morphologically rich language. For this purpose, EN-TR CodeBench, a benchmark consisting of 60 problems and 811 unit tests at easy, medium, and hard difficulty levels, was created. Five open-source code LLMs from three model families, one with three different parameter scales, were evaluated under English, Turkish, and Turkish-localized conditions using the pass@1 metric. The results show that the use of Turkish prompts leads to a performance decrease in all five models, and this decrease narrows as model scale increases within the Qwen2.5-Coder family. Analysis according to difficulty level reveals that, for most models, the language-related performance loss increases substantially with problem complexity. Error analysis shows that the performance difference between languages is largely driven by runtime and logic errors, while syntax errors remain rare.

## 1 Introduction

Large language models (LLMs) perform successfully in the task of converting natural language expressions into executable program code. Benchmarks such as HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021) have been among the pioneering efforts to measure the code generation performance of LLMs. However, evaluation studies of LLMs conducted with these and similar benchmarks have often concentrated primarily on the English language (Peng et al., 2024). This raises the question of the extent to which the code generation performance of LLMs is language-dependent. The ability of LLMs to correctly interpret prompts and

their ability to solve algorithmic problems cannot be disentangled in standard monolingual assessment frameworks.

Studies in which HumanEval was translated into different languages using machine translation methods have revealed that prompts in different languages for the same problem create significant performance differences (Peng et al., 2024; Raihan et al., 2025). Similarly, evaluations of MBPP translated into languages other than English and performed on code LLMs show significant performance differences (Li et al., 2025). In addition, token fertility differences between languages in LLMs have the potential to affect performance (Petrov et al., 2023; Ahia et al., 2023). Representing words with more tokens not only increases the processing cost and latency, but also limits the amount of information that the model can process within its context window. As Turkish is an agglutinative language, it is known that a Turkish word is represented as multiple sub-words when tokenized by LLMs (Toraman et al., 2023; Bayram et al., 2025). In addition, general-purpose Turkish LLM benchmarks such as TurkishMMLU (Yüksel et al., 2024), TR-MMLU (Bayram et al., 2024) and Cetvel (Er et al., 2026) evaluate the Turkish language capabilities of LLMs across various tasks. However, none of these benchmarks are based on the code generation performance of LLMs.

This study reveals the cost of asking questions in Turkish by creating EN-TR CodeBench, a parallel English-Turkish code generation benchmark consisting of 60 problems spread across three difficulty levels and validated with 811 unit tests. The benchmark also checks for the impact of the identifier language with a Turkish-localized condition where function and parameter names are localized along with the prompt. The performance of five open-source code LLMs, spanning three model families, one of which was further evaluated across three parameter scales, is compared. Furthermore, the code

generation performance of the models is analyzed through syntax, runtime, and logic errors.

## 2 The EN-TR CodeBench Dataset

EN-TR CodeBench<sup>1</sup> was created to evaluate the cost of prompt language in code generation tasks for LLMs. It consists of English and Turkish prompts for 60 problems of varying difficulty levels. The Turkish prompts were prepared under two separate conditions, one in which function and parameter names remained in English and one in which they were also localized into Turkish. Problem prompts were created and validated jointly by two people fluent in both English and Turkish. The aim was to ensure the problem prompts were equivalent in both languages. Figure 1 shows test inputs and outputs along with English, Turkish, and Turkish-localized prompts for an example problem in the benchmark.

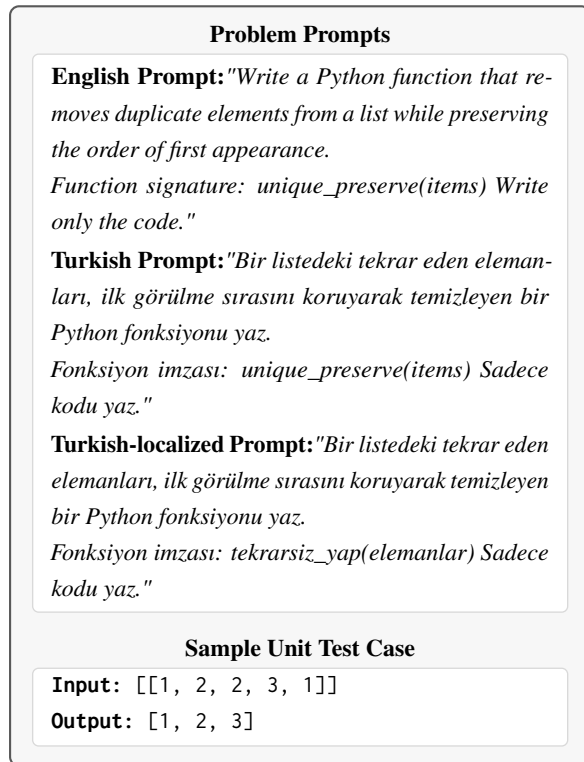


Figure 1: Anatomy of a benchmark task illustrating the three evaluation setups: English, Turkish, and Turkish-localized.

The benchmark includes 15 easy, 20 medium, and 25 hard problems. Difficulty levels broadly reflect problem and code complexity, ranging from simple operations at the easy level to multi-step tasks at the medium level and more advanced algorithms or interacting rules at the hard level. For

<sup>1</sup>[huggingface.co/datasets/incidelen/EN-TR-CodeBench](https://huggingface.co/datasets/incidelen/EN-TR-CodeBench)

each problem, unit tests were prepared to evaluate the logical correctness of the code. Each unit test consists of an input value and its corresponding expected output. To test the model’s ability to handle edge cases, the unit tests include boundary values in addition to values that test core functionality. Benchmark statistics are given in Table 1.

| Difficulty | Problems | Total Tests | Average Tests |
|------------|----------|-------------|---------------|
| Easy       | 15       | 200         | 13.3          |
| Medium     | 20       | 269         | 13.5          |
| Hard       | 25       | 342         | 13.7          |
| Total      | 60       | 811         | 13.5          |

Table 1: Distribution of problems and test cases across difficulty levels.

## 3 Methodology

### 3.1 Models

In this study, code LLMs were selected to allow both comparison of different model families with similar parameter scales and evaluation of the effect of parameter scale within the same model family. For this purpose, five open-source code LLMs from the CodeLlama (Roziere et al., 2023), DeepSeek-Coder (Guo et al., 2024), and Qwen2.5-Coder (Hui et al., 2024) model families were evaluated. For inter-family comparison, the CodeLlama-7B-Instruct, DeepSeek-Coder-6.7B, and Qwen2.5-Coder-7B models were selected. For intra-family comparison, in addition to the 7B model from the Qwen2.5-Coder family, variations with 14B and 32B parameters were evaluated.

### 3.2 Experimental Setup

The code generation experiments of the code LLMs were conducted for the Python programming language, based on the system prompt presented in Appendix A. The hyperparameters used in the experimental processes are given in Table 2.

| Parameter           | Value |
|---------------------|-------|
| Temperature         | 0.2   |
| Seed                | 1234  |
| Samples per problem | 5     |
| Maximum new tokens  | 1200  |

Table 2: Hyperparameters used in the experiments.

The evaluation process of the code outputs generated by code LLMs was carried out using a three-stage pipeline. In this context, the generated code was first checked for syntax errors, then for runtime errors. Finally, logic errors were evaluated,

and code that correctly executed all test scenarios was considered successful. Accordingly, the errors that occurred in each stage were evaluated separately and reported as syntax errors, runtime errors, and logic errors. All models were evaluated under the same hyperparameter settings.

### 3.3 Evaluation Metrics and Statistical Testing

To evaluate the code generation performance of code LLMs, the  $\text{pass}@k$  metric, preferred in similar studies, was used (Chen et al., 2021). This metric calculates the probability of finding at least one correct solution among  $k$  samples taken from the model for a problem, estimated from  $n \geq k$  generated samples. In this study,  $n = 5$  samples were generated by the models for each problem, and  $\text{pass}@1$  ( $k = 1$ ), reflecting the success of the models in a single trial, was taken as the main metric. The statistical significance of the performance differences between prompt language conditions was tested for each model with 10,000 iterations using paired bootstrap resampling, and 95% confidence intervals were calculated. Holm-Bonferroni correction was applied to the p-values to control for the risk of Type I errors arising from multiple comparisons.

## 4 Results and Discussion

As can be seen in the  $\text{pass}@1$  findings presented in Table 3, a performance decrease was observed between English and Turkish prompts for all evaluated code LLMs. Among the evaluated models, the highest performance decrease due to the prompt language was observed in the CodeLlama-7B-Instruct model with a decrease of 15.3 percentage points. In addition, within the Qwen2.5-Coder family, it was observed that the performance difference between English and Turkish languages narrowed with increasing model scale. Among the variations of the Qwen2.5-Coder family with three different parameter scales, it was observed that the difference between English and Turkish languages

decreased from 14.0 to 2.3 percentage points as the number of parameters increased. The results obtained for Qwen2.5-Coder-32B show that the difference is not statistically significant. These results suggest that the language-related performance loss is not a fixed loss, but an effect that may narrow with increasing model scale. Under the Turkish-localized condition, where function and parameter names were localized along with the prompt, results varied from model to model. In smaller-scale models, an additional decrease in performance was observed due to this condition. However, in the Qwen2.5-Coder-14B model, identifier localization had the opposite effect compared to other models, and no performance difference was observed for the Qwen2.5-Coder-32B model.

| Model                 | Language          | Easy  | Medium | Hard |
|-----------------------|-------------------|-------|--------|------|
| CodeLlama-7B-Instruct | English           | 100.0 | 40.0   | 10.4 |
|                       | Turkish           | 69.3  | 30.0   | 0.0  |
|                       | Turkish-localized | 58.7  | 23.0   | 0.0  |
| DeepSeek-Coder-6.7B   | English           | 86.7  | 68.0   | 30.4 |
|                       | Turkish           | 86.7  | 55.0   | 17.6 |
|                       | Turkish-localized | 86.7  | 54.0   | 15.2 |
| Qwen2.5-Coder-7B      | English           | 93.3  | 80.0   | 36.0 |
|                       | Turkish           | 89.3  | 71.0   | 12.0 |
|                       | Turkish-localized | 86.7  | 69.0   | 14.4 |
| Qwen2.5-Coder-14B     | English           | 93.3  | 82.0   | 52.8 |
|                       | Turkish           | 93.3  | 75.0   | 29.6 |
|                       | Turkish-localized | 93.3  | 74.0   | 32.8 |
| Qwen2.5-Coder-32B     | English           | 100.0 | 88.0   | 47.2 |
|                       | Turkish           | 98.7  | 86.0   | 44.0 |
|                       | Turkish-localized | 97.3  | 90.0   | 41.6 |

Table 4: Pass@1 accuracy (%) broken down by problem difficulty across prompt-language conditions.

Analysis based on the difficulty levels given in Table 4 shows that, for most models, language-related performance loss increases substantially with problem complexity. While the effect of prompt language remains limited in most models at the easy level, this effect becomes markedly pronounced at the hard level. CodeLlama-7B-Instruct was able to solve 10.4% of hard problems in En-

| Model                 | EN   | TR   | TR-Loc. | $\Delta\text{TR-EN}$ | 95% CI <sub>TR-EN</sub> | $\Delta\text{Loc.-TR}$ | 95% CI <sub>Loc.-TR</sub> |
|-----------------------|------|------|---------|----------------------|-------------------------|------------------------|---------------------------|
| CodeLlama-7B-Instruct | 42.7 | 27.3 | 22.3    | -15.3**              | [-26.0, -5.7]           | -5.0                   | [-10.0, -1.0]             |
| DeepSeek-Coder-6.7B   | 57.0 | 47.3 | 46.0    | -9.7                 | [-19.3, -1.0]           | -1.3                   | [-8.7, 6.0]               |
| Qwen2.5-Coder-7B      | 65.0 | 51.0 | 50.7    | -14.0***             | [-22.3, -6.7]           | -0.3                   | [-7.3, 6.3]               |
| Qwen2.5-Coder-14B     | 72.7 | 60.7 | 61.7    | -12.0***             | [-20.0, -5.3]           | 1.0                    | [-3.3, 6.0]               |
| Qwen2.5-Coder-32B     | 74.0 | 71.7 | 71.7    | -2.3                 | [-9.7, 5.0]             | 0.0                    | [-4.7, 5.0]               |

Table 3: Overall  $\text{pass}@1$  accuracy (%) across prompt-language conditions.  $\Delta\text{TR-EN}$ : prompt-language effect;  $\Delta\text{Loc.-TR}$ : identifier-localization effect. \* $p < 0.05$ , \*\* $p < 0.01$ , \*\*\* $p < 0.001$ .

glish, but the success rate dropped to zero in Turkish. To examine the source of performance differences, the failures of the code generated by code LLMs were separated into syntax errors, runtime errors, and logic errors, as shown in Table 5.

| Model                 | Language          | SE | RE  | LE  | Success |
|-----------------------|-------------------|----|-----|-----|---------|
| CodeLlama-7B-Instruct | English           | 0  | 72  | 100 | 128     |
|                       | Turkish           | 8  | 124 | 86  | 82      |
|                       | Turkish-localized | 4  | 120 | 109 | 67      |
| DeepSeek-Coder-6.7B   | English           | 0  | 33  | 96  | 171     |
|                       | Turkish           | 0  | 42  | 116 | 142     |
|                       | Turkish-localized | 0  | 47  | 115 | 138     |
| Qwen2.5-Coder-7B      | English           | 0  | 31  | 74  | 195     |
|                       | Turkish           | 2  | 39  | 106 | 153     |
|                       | Turkish-localized | 1  | 51  | 96  | 152     |
| Qwen2.5-Coder-14B     | English           | 0  | 11  | 71  | 218     |
|                       | Turkish           | 0  | 37  | 81  | 182     |
|                       | Turkish-localized | 0  | 27  | 88  | 185     |
| Qwen2.5-Coder-32B     | English           | 0  | 16  | 62  | 222     |
|                       | Turkish           | 0  | 12  | 73  | 215     |
|                       | Turkish-localized | 4  | 12  | 69  | 215     |

Table 5: Model error types and success counts across languages. (SE: syntax errors, RE: runtime errors, LE: logic errors)

When the distribution of error types was examined, the lowest number of syntax errors was observed in the generated code. This indicates that syntactic validity is largely preserved across scale and prompt language, though a small increase in syntax errors under Turkish conditions is observed for CodeLlama-7B-Instruct and Qwen2.5-Coder-7B, and for Qwen2.5-Coder-32B only under the Turkish-localized condition. However, the runtime and logic error rates differed noticeably for the Turkish prompt language compared to English. The error analysis shows that the performance loss caused by the prompt language occurs not at the level of producing syntactically valid code, but at the level of correctly executing the solution.

| Tokenizer      | English | Turkish |
|----------------|---------|---------|
| CodeLlama      | 1.34    | 3.35    |
| DeepSeek-Coder | 1.34    | 3.65    |
| Qwen2.5-Coder  | 1.24    | 2.53    |

Table 6: Tokenization fertility behavior across English and Turkish prompts.

The fertility analysis performed on the tokenizers of the compared LLM families is given in Table 6. The results show that Turkish requires considerably more tokens per word compared to English. CodeLlama, which has high fertility, also

shows the largest performance difference in Table 3. However, DeepSeek-Coder, which has the highest fertility, shows a smaller performance difference than Qwen2.5-Coder-7B. This suggests that fertility may contribute to the performance difference but is not its sole determinant.

## 5 Conclusion

This study examined the cost of prompt language on code generation performance for an agglutinative and morphologically rich language like Turkish. For this purpose, a parallel English-Turkish code generation benchmark consisting of 60 problems and 811 unit tests was created. This benchmark was used to evaluate five open-source code LLMs, spanning three model families, one of which was further evaluated across three parameter scales, under English, Turkish, and Turkish-localized conditions. Switching the prompt language from English to Turkish reduced pass@1 for all five models, and this reduction remained statistically significant after Holm-Bonferroni correction for three of them. These findings also showed that the performance difference between English and Turkish prompts mainly occurred at the runtime and logic levels, not at the syntax level. Although the effect of the Turkish-localized condition, where the identifiers were also localized into Turkish, varied from model to model, these differences did not reach statistical significance. Future studies could extend this benchmark to other low-resource languages, include large closed-source language models in the comparison, and retest the impact of identifier localization with a larger sample size.

## 6 Limitations

While the 60-problem benchmark used in this study is sufficient to identify the main effect of prompt language, it limits the statistical power for more subtle effects such as identifier localization. Indeed, the identifier localization effect observed in this study did not reach statistical significance at the current sample size. A larger benchmark might reveal this effect more clearly. In addition, the evaluation is limited to only five open-source models ranging from 6.7B to 32B parameters; whether much larger, closed-source models would show a similar language-related performance loss is outside the scope of this study. Furthermore, the study covers only one language pair and one programming language. The generalizability of the findings

to other agglutinative or low-resource languages, or to different programming languages, should also be tested.

## References

- Orevaoghene Ahia, Sachin Kumar, Hila Gonen, Jungo Kasai, David R Mortensen, Noah A Smith, and Yulia Tsvetkov. 2023. Do all languages cost the same? tokenization in the era of commercial language models. In *Proceedings of the 2023 Conference on empirical methods in natural language processing*, pages 9904–9923.
- 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*.
- M Ali Bayram, Ali Arda Fincan, Ahmet Semih Gümüş, Banu Diri, Savaş Yıldırım, and Öner Aytas. 2024. Setting standards in Turkish NLP: TR-MMLU for large language model evaluation. *arXiv preprint arXiv:2501.00593*.
- M Ali Bayram, Ali Arda Fincan, Ahmet Semih Gümüş, Sercan Karakaş, Banu Diri, and Savaş Yıldırım. 2025. Tokenization standards for linguistic integrity: Turkish as a benchmark. *arXiv preprint arXiv:2502.07057*.
- Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.
- Yakup Abrek Er, Ilker Kesen, Gözde Gül Şahin, and Aykut Erdem. 2026. Cetvel: A unified benchmark for evaluating language understanding, generation and cultural capacity of llms for turkish. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1052–1085.
- Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yifan Wu, YK Li, et al. 2024. DeepSeek-Coder: when the large language model meets programming—the rise of code intelligence. *arXiv preprint arXiv:2401.14196*.
- Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, et al. 2024. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*.
- Mingda Li, Abhijit Mishra, and Utkarsh Mujumdar. 2025. Bridging the language gap: Enhancing multilingual prompt-based code generation in llms via zero-shot cross-lingual transfer. In *2025 International Joint Conference on Neural Networks (IJCNN)*, pages 1–9. IEEE.
- Qiwei Peng, Yekun Chai, and Xuhong Li. 2024. Humaneval-xl: A multilingual code generation benchmark for cross-lingual natural language generalization. In *Proceedings of the 2024 joint international conference on computational linguistics, language resources and evaluation (LREC-COLING 2024)*, pages 8383–8394.
- Aleksandar Petrov, Emanuele La Malfa, Philip Torr, and Adel Bibi. 2023. Language model tokenizers introduce unfairness between languages. *Advances in neural information processing systems*, 36:36963–36990.
- Nishat Raihan, Antonios Anastasopoulos, and Marcos Zampieri. 2025. mhumaneval-a multilingual benchmark to evaluate large language models for code generation. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 11432–11461.
- Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, et al. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.
- Cagri Toraman, Eyup Halit Yilmaz, Furkan Şahinuç, and Oguzhan Ozelik. 2023. Impact of tokenization on language models: An analysis for turkish. *ACM Transactions on Asian and Low-Resource Language Information Processing*, 22(4):1–21.
- Arda Yüksel, Abdullatif Köksal, Lütfi Kerem Senel, Anna Korhonen, and Hinrich Schütze. 2024. TurkishMMLU: Measuring massive multitask language understanding in Turkish. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 7035–7055.

## A System Prompts

### English System Prompt:

You are an expert Python developer. Solve the task as a single Python function. Return ONLY one ``python code block. No explanation, no examples, no tests. The function name must match exactly. Put any imports inside the code block.

### Turkish and Turkish-Localized System Prompt:

Sen uzman bir Python geliştiricisin. Görevi tek bir Python fonksiyonu olarak çöz. SADECE tek bir ``python kod bloğu döndür. Açıklama, örnek veya test yazma. Fonksiyon adı tam olarak istendiği gibi olmalı. Gerekli import'ları kod bloğuna koy.