Improvement of Language Identification Processing Speed

Emma Bednaříková and Pavel Rychlý

Faculty of Informatics, Masaryk University
Botanická 68a, 602 00 Brno, Czech Republic
{536251, pary}@mail.muni.cz

Abstract. This paper presents the Langtok model developed for the purpose of identifying the language of each token in codemix texts, i.e. texts where words from multiple languages are used. It provides insight into the development process and evaluation. Furthermore, the paper describes the utilization of the model for the filtration of large amounts of Czech text for machine translation. The main focus of this paper is conducting a set of experiments to identify the optimal conditions for input processing with the aim of decreasing the process duration.

Keywords: Codemix Text, Processing Speed, Language Identification.

1 Introduction

The identification of the language used in a text is important in a number of contexts. The most commonly employed tools for this purpose, such as fasttext [5] [4], langdetect [8], or langid [6], are designed to detect the language of an entire sentence or paragraph. However, when analyzing codemix text, it is of particular importance to be able to determine the language of each word in a given sentence or paragraph. The model presented in this paper represents a potential solution for this task. When utilizing the model for the labeling of large amounts of data, it is vital to optimize the computation process. Therefore, a series of experiments was conducted to identify the best conditions for the processing of the inputs.

2 The Langtok Model

Langtok is a model designed to identify the language of each token in the input. Although the sentence on the input may contain words from different languages, the model should be able to distinguish them and return a list of language labels where each label corresponds to a token at the same position in the sentence.

The following example illustrates the desired behavior of the model. Due to the nature of the tokenizer used, the expected output would contain slightly more labels, as the tokenizer would probably split some of the words. This example is for demonstration purposes only.

A. Horák, P. Rychlý, A. Rambousek (eds.): Proceedings of Recent Advances in Slavonic Natural Language Processing, RASLAN 2024, pp. 129–138, 2024.

Input: This is a tool určený k zjištění jazyka von jedes Wortes in einem Satz.

Output: ['eng', 'eng', 'eng', 'eng', 'ces', 'ces', 'ces', 'ces', 'deu', 'deu', 'deu', 'deu', 'deu', 'deu']

3 Development

The first part of the model-building process was to create a dataset for the training. This also included choosing the languages that the model should be able to identify. Once the dataset was created, the model was trained on it.

3.1 Language Selection

This model includes 18 languages: Arabic, Czech, Danish, German, English, French, Haitian Creole, Italian, Japanese, Lingala, Dutch, Polish, Portuguese, Russian, Slovak, Spanish, Swedish, and Ukrainian.

3.2 Building of the Training Dataset

The training dataset consists of a set of modified sentences. Each sentence contains two snippets of text in a language other than the language of the sentence. The length of the snippets varies between two to six words. These snippets were inserted in a random position in the source sentence. The sentences and snippets were split by word_tokenize function from the nltk library[1], except for sentences and snippets in Japanese. In this case a different tokenizer[7] had to be utilized, due to the word_tokenize's incompetence to tokenize text in Japanese script.

Every such modified and tokenized sentence has a corresponding list of language labels of the same length as the sentence. Each label in the label list corresponds to a token in the same position in the list of tokens.

The raw text for the creation of the dataset was collected from Flores-200 [2]. The dataset consists of three parts: train, validate, and test. The size of the train split is ca 19250 input and output pairs, and the size of the validate and test split is in both cases ca 2150 input and output pairs. This contributes to the approximate ratio of 80:10:10.

3.3 Training

The base model used for the training was Google's BERT multilingual base model (cased) [3]. The training parameters were the same as the ones provided in the HuggingFace tutorial on training a model for token classification tasks.[1] The batch size was set to 16, the number of epochs was 3, and the learning rate was 2e-5. The training was conducted on a GPU of the Tesla T4 type, with a total training time of 28 minutes and 28 seconds.

[1] *https://huggingface.co/docs/transformers/tasks/token_classification*

4 Evaluation

In order to evaluate the performance of the model, a confusion matrix was constructed based on the model's results for the inputs from the test dataset. Figure 1 illustrates the results for a subset of the languages. To facilitate comprehension, the values in the matrix were converted to a percentage scale.

	ces	deu	eng	por	rus	slk	spa	ukr
ces	98,11%	0,10%	0,06%	0,06%	0,00%	1,44%	0,00%	0,01%
deu	0,00%	99,48%	0,09%	0,00%	0,00%	0,02%	0,07%	0,00%
eng	0,07%	0,00%	99,12%	0,04%	0,00%	0,18%	0,02%	0,00%
por	0,04%	0,04%	0,16%	98,42%	0,00%	0,11%	0,93%	0,00%
rus	0,00%	0,02%	0,00%	0,02%	99,48%	0,00%	0,00%	0,49%
slk	1,00%	0,01%	0,01%	0,01%	0,00%	98,56%	0,03%	0,00%
spa	0,10%	0,02%	0,07%	0,57%	0,02%	0,00%	98,87%	0,00%
ukr	0,00%	0,00%	0,00%	0,00%	0,30%	0,00%	0,00%	99,70%

Fig. 1: The evaluation results for eight of the languages

The model achieved a score of 98% or above for all languages. The highest scores the model reached with Arabic and Japanese. Given the fact that both of these languages use their unique scripts that significantly differ from the scripts of other languages within the model this is not surprising.

Another unsurprising, yet nevertheless interesting phenomenon is the fact that the language pairs in which the model is observed to produce the highest error rates are in most cases two languages from the same language family, namely Czech and Slovak, Spanish and Portuguese, or Russian and Ukrainian.

5 Practical Applications

The model can be employed in a similar manner to that of other language identification tools. The two main domains are data labeling and data filtering.

5.1 Filtering Data for Machine Translation

One of the possible practical applications of the model is the filtering of data for machine translation. To build a machine translation system, a large amount of training data is needed. As these data are frequently obtained through mechanical means with minimal supervision, there is a tendency for them to contain text that is not written in the desired language. The filtration of such data may prove beneficial in enhancing the performance of the translation system.

The advantage of the Langtok model compared to traditional language identification tools is that it can identify and subsequently filter sentences that are mostly written in one language, yet contain words of another. In order to avoid the inclusion of a codemix text in the monolingual training data, the use of a token-level language identifier represents an effective solution.

However, utilization of the Langtok model may also produce certain issues. When the model was first tested for this task on Czech data, it was observed that the model filtered sentences that did not present any problems. For each sentence, the filtering algorithm calculated the distribution of the language labels. The language with the highest number of labels was selected as the language of the sentence. In instances where the language of the sentence was not identified as Czech, said sentence got filtered. In several cases, the sentence was falsely labeled as Slovak, leading to its potential removal from the training data. Furthermore, there have been instances where the language identified for all the tokens within a given sentence was completely different from Czech or Slovak. Table 1 presents the sentences that were incorrectly selected as suitable for filtration and also the language label that was used for most of the tokens of the sentence instead.

Table 1: Sentences incorrectly selected as suitable for filtration (non-Czech) from the machine translation data.

Jamesi, já nestojím o hrdinu. (slk)	To je ono. (slk)
To je v poho, no tak. (slk)	Já ti ukážu, co je to pravý muž. (slk)
Zrzek. (pol)	No a? (por)
Hele. (dan)	Omdlela. (swe)

6 Optimizing Input Processing

Since many applications of the model often require the analysis of large amounts of data, it is important to keep the processing time per input sentence to a minimum. In this section, three different strategies that show a possible improvement in reducing the process duration are presented.

6.1 Structural Modifications in Input Processing Workflow

Firstly, it is important to keep the code that provides the inference with the model well structured in order to avoid unnecessary repetition of actions that could be simplified. Secondly, the deployment of different methods should be considered. The following paragraphs present three distinct strategies that can be employed independently.

The most straightforward approach to implementation is to have a function that loads both the model and the model's tokenizer and then processes a single

sentence. However, this would require loading the model every time results for a single sentence are computed. Therefore, this strategy represents a viable yet inherently time-consuming solution. Single model and tokenizer loading would lead to a significant decrease in the time spent obtaining the results for a set of sentences.

An additional potential improvement in accelerating the computation is enabling the model to process multiple inputs concurrently. In this approach, the weights within the model are calculated for all inputs simultaneously. The deployment of this strategy has proven to be efficient. Nevertheless, concurrent processing may introduce a decrease in speed when working with large batches of inputs. This issue is addressed in the following part of this paper.

The last technique was the utilization of graphical processing units (GPUs). These provide faster computation when working with neural networks. This method requires transferring the model and the inputs to the GPU which results in a delay to the start of the calculation. However, the sole processing occurs in a shorter period than on the CPU.

In order to obtain an accurate representation of the average duration for each case, ten test runs were conducted. The values presented in Figure 2 represent the average duration of these runs.

Each run was tested on two input files; a short one containing 20 input sentences and a longer one containing 200 sentences. As anticipated, the instances where GPU was utilized exhibited inferior performance than the same cases with GPU utilization. However, it is noteworthy, that the transfer of the model to the GPU averaged 0,835 seconds. For cases 5 and 7, and 6 and 8, the results demonstrate a roughly similar duration, as in all of these cases the model and tokenizer were always loaded only once, due to the concurrent processing of inputs. The tests were conducted on a CPU device comprising 80 processors (apollo) and the GPU utilized was of the Tesla T4 type.

case	using GPU	loading model only once	simultaneous processing	duration of processing [s]	
				short file	long file
1	0	0	0	14,421	107,867
2	1	0	0	14,493	128,564
3	0	1	0	1,886	5,892
4	1	1	0	3,905	5,018
5	0	0	1	1,582	3,670
6	1	0	1	3,241	4,591
7	0	1	1	1,616	3,848
8	1	1	1	3,152	4,566

Fig. 2: Average durations of test runs

6.2 Finding the Optimal Batch Size For Input Processing

The simultaneous processing of multiple input sentences results in increased efficiency. When computing with multiple inputs, the model requires that the inputs be of the same length; thus, all inputs are padded to the length of the longest sentence. The complexity of the calculation of the result is dependent on the input length; the longer the input, the larger the matrix that will be processed during the computation. It is therefore desirable to feed the model with the inputs in batches to prevent the computation of unnecessarilly large matrices.

To estimate the ideal batch size value a series of tests was conducted. The evaluation file consisted of 2000 sentences that were processed by the model in batches. Each batch size was tested 4 times. The average duration of these tests is shown in Table 2. The same experiment was also conducted on a GPU device. The results of these tests are in Table 3. Both experiments were conducted on a CPU device comprising 32 processors (epimetheus2 and epimetheus4) and the GPU utilized was of the Tesla T4 type.

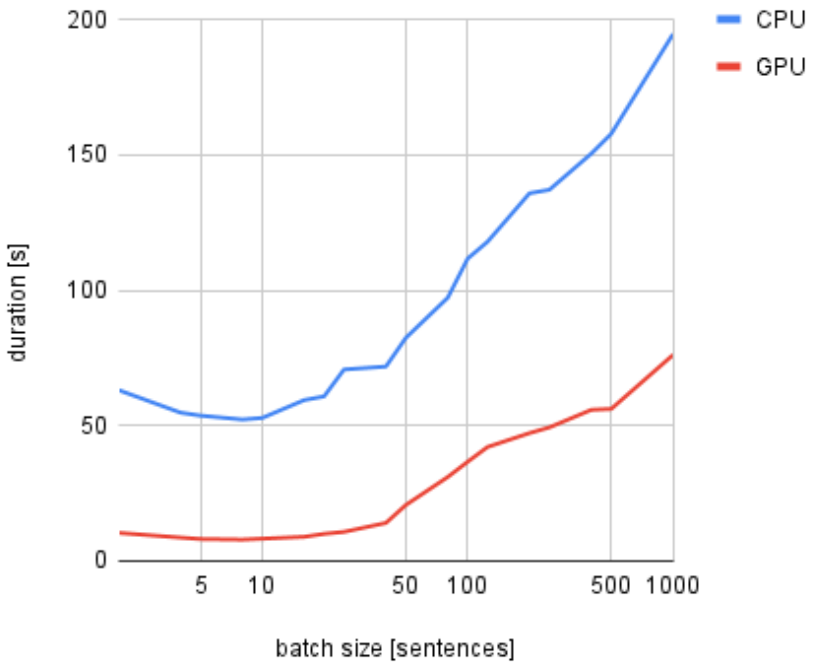

Fig. 3: Comparison of values measured on CPU and GPU

As illustrated in Figure 3, the batch sizes that offer the shortest duration for both CPU and GPU are those within the range of 4 to 10.

Table 2: Statistics from test runs on CPU

batch size [*sentence*]	overall duration [*s*]	min batch length [*token*]	avg batch length [*token*]	avg memory [*GB*]	avg %CPU
2	66,605	4	29,644	1,071	1602,000
4	57,127	5	38,246	1,150	1607,417
5	55,910	5	41,823	1,150	1607,111
8	54,458	5	49,368	1,167	1608,667
10	54,729	6	54,515	1,221	1607,526
16	62,256	7	63,896	1,265	1605,706
20	63,307	12	71,890	1,294	1604,944
40	74,205	20	91,520	1,538	1602,615
80	99,797	58	121,080	1,464	1583,880
200	138,399	119	156,900	2,511	1501,474
500	160,357	119	180,500	2,800	1443,053
1000	197,280	204	216,500	5,558	1384,526

Table 3: Statistics from test runs on GPU

batch size [*sentence*]	overall duration [*s*]	min batch length [*token*]	avg batch length [*token*]	GPU memory [*MB*]	avg %GPU
2	13,244	4	29,644	1313	52,2
4	10,605	5	38,246	1345	55,5
5	10,138	5	41,823	1361	57,2
8	10,073	5	49,368	1399	63,7
10	10,354	6	54,515	1421	69,9
16	11,022	7	63,896	1525	76,9
20	12,059	12	71,890	1571	77,1
40	15,944	20	91,520	1853	78,1
80	33,277	58	121,080	2429	77,6
200	49,428	119	156,900	3719	86,4
500	58,797	119	180,500	7333	82,4
1000	78,708	204	216,500	13373	84,7

6.3 Increasing the Number of Processes Running Simultaneously

The last strategy for optimizing the process duration was evaluating more input files concurrently. By utilizing this, efficiency can be improved, however only to a certain extent. Once the number of processes operating concurrently exceeds the available resource capacity of the device, the computational speed declines markedly as the active processes must compete for the limited available resources.

In order to ascertain the optimal number of processes that can be run simultaneously to facilitate the most efficient input processing, an experiment was conducted. The objective was to process 24 files, each of which contained 400 sentences. Initially, the files were processed one by one, and subsequently, the number of files processed simultaneously was increased. Table 4 presents the results of conducting this experiment on a CPU device comprising 32 processors (epimetheus1).

As demonstrated in Table 4, the shortest processing time was observed when two files were processed simultaneously. A mere increase of one in the number of concurrent processes resulted in a notable increase in processing time. The same task was repeated on a GPU device of type Tesla T4. The results are shown in Table 5. Figure 4 illustrates the comparative results obtained from the CPU and GPU. It demonstrates that when a GPU device was utilized, an increase in the number of processes up to 12 did not result in an overall duration increase.

7 Future Work

This work primarily covers the description of the model and the search for the optimal processing conditions. Given the existence of alternative strategies for optimizing the computational process, such as sorting the inputs by length first and subsequently dividing them into batches or limiting the number of threads to one and increasing the number of processes, further research on this topic is anticipated.

Another further objective is to evaluate the performance of the Langtok model in comparison with other language identification tools, such as langdetect [8], fasttext [5] [4], or langid [6].

8 Conclusion

The objective of this paper was to introduce the Langtok model and illustrate its potential applications. Additionally, a series of experiments was conducted with the aim to identify the optimal processing parameters. The experiments have demonstrated that the optimal batch size for the processing of multiple inputs in a simultaneous manner is within the range of four to ten. Finally, it has been found that when processing multiple input files simultaneously on a CPU device, it is extremely important to ensure that the number of computing resources is not exceeded. Otherwise, the efficiency of the processing will

Table 4: Statistics from test runs on CPU

number of processes	duration [s]	CPU memory / process	%CPU values / process
1	413	1,0g - 1,3g	1300 - 1620
2	242	1,0g - 1,1g	1560 - 1606
3	2764	970m - 1,1g	950 - 1160
4	2398	914m - 1,1g	750 - 840
6	2334	880m - 1,0g	507 - 572
12	1872	959m - 1,0g	256 - 277

Table 5: Statistics from test runs on GPU

number of processes	duration [s]	GPU memory [MB]	avg %GPU
1	181	1291,8	34,2
2	103	2232,5	58,4
3	72	2455,8	57,8
4	68	3896,4	79,7
6	54	4614,5	75,2
12	47	10053,9	92,0

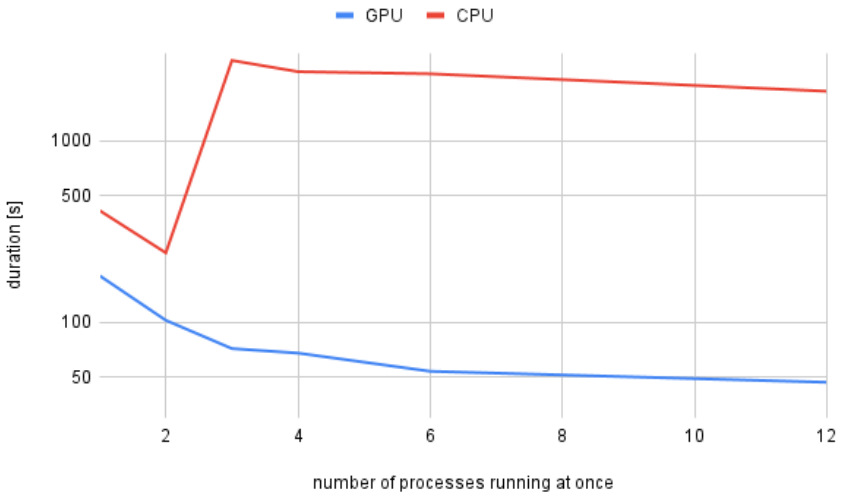

Fig. 4: Comparison of values measured on CPU and GPU

decrease significantly. Processing on a GPU device, however, did not pose such a threat, and the sole constraint that was identified was the limitation of memory resources.

Acknowledgements. The work described herein has been supported by the Ministry of Education, Youth and Sports of the Czech Republic, Project No. LM2023062 LINDAT/ CLARIAH-CZ

References

1. Bird, S., Klein, E., Loper, E.: Natural language processing with Python: analyzing text with the natural language toolkit. " O'Reilly Media, Inc." (2009)
2. Costa-jussà, M.R., Cross, J., Çelebi, O., Elbayad, M., Heafield, K., Heffernan, K., Kalbassi, E., Lam, J., Licht, D., Maillard, J., et al.: No language left behind: Scaling human-centered machine translation. arXiv preprint arXiv:2207.04672 (2022)
3. Devlin, J.: Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805 (2018)
4. Joulin, A., Grave, E., Bojanowski, P., Douze, M., Jégou, H., Mikolov, T.: Fasttext.zip: Compressing text classification models. arXiv preprint arXiv:1612.03651 (2016)
5. Joulin, A., Grave, E., Bojanowski, P., Mikolov, T.: Bag of tricks for efficient text classification. arXiv preprint arXiv:1607.01759 (2016)
6. Lui, M., Baldwin, T.: langid. py: An off-the-shelf language identification tool. In: Proceedings of the ACL 2012 system demonstrations. pp. 25–30 (2012)
7. McCann, P.: fugashi, a tool for tokenizing Japanese in python. In: Proceedings of Second Workshop for NLP Open Source Software (NLP-OSS). pp. 44–51. Association for Computational Linguistics, Online (Nov 2020), https://www.aclweb.org/anthology/2020.nlposs-1.7
8. Shuyo, N.: Language detection library for java (2010), http://code.google.com/p/language-detection/