Neural Machine Translation: Python-C++

Hugo Albert Bonet

1 Introduction

This section provides a brief description of the selected dataset and limitations adopted to run the experiments according to our computing capabilities.

1.1 Dataset Description

The selected dataset corresponds to a subset from XLCoST, a benchmark dataset for cross-lingual code intelligence. The original dataset contains 7 commonly used programming languages (C++, Java, Python, C#, Javascript, PHP, C), and natural language (English), from which we have selected the corresponding subset of Python translation to C++. Concretely, the final dataset used for this project is the code snippet dataset, which translates functions and small programs between those programming languages.

The dataset is composed of 10,675 code translations, divided into train (9308), validation (477), and test (890). The average token length of the complete snippets is 194.4 for Python and 199.6 for C++.

1.2 Limitations

The experiments have been carried out using Kaggle Notebooks environment, which offers a CPU with 29 GiB and two GPU T4 with 15 GiB VRAM each, restricted to 30 hours of computation a week. Therefore, some limitations have been mandatory for the proper development of the project.

Firstly, only snippets 128 tokens long or less have been taken into account, resulting in 1079 train samples, 56 for validation, and 133 for test. Moreover, Llama-related models have been trained with batch size of 1 and 8 gradient accumulation steps for 3 epochs, and a LoRA rank of 16. On the other hand, NLLB models have been trained for 10 epochs with a batch size of 4 and 4 accumulation steps.

Table 1 shows the percentage of trainable parameters during LoRA fine-tunning for each LLM trained in this project.

Model	Trainable%
CodeLlama 7b	0.1243
CodeLlama 3 8b	0.0848
Llama 2 7b	0.1243
NLLB 600M	0.2066
NLLB 1.3B	0.2483

Table 1: LLMs varied parameters

2 Results

This section shows the results of the baseline and fine-tuned models on various performance metrics.

The models trained on the Python-C++ translation task were CodeLlama (versions 2 and 3), LLaMa 2, and NLLB (version 600M).

Regarding the performance metrics employed, each model was evaluated using BLEU (B), COMET (C), and CODEBLEU (CB), which evaluates n-gram match (N), weighted n-gram match (WN), syntax match (S) and dataflow match(D). Figure 1 shows how CODEBLEU is computed:

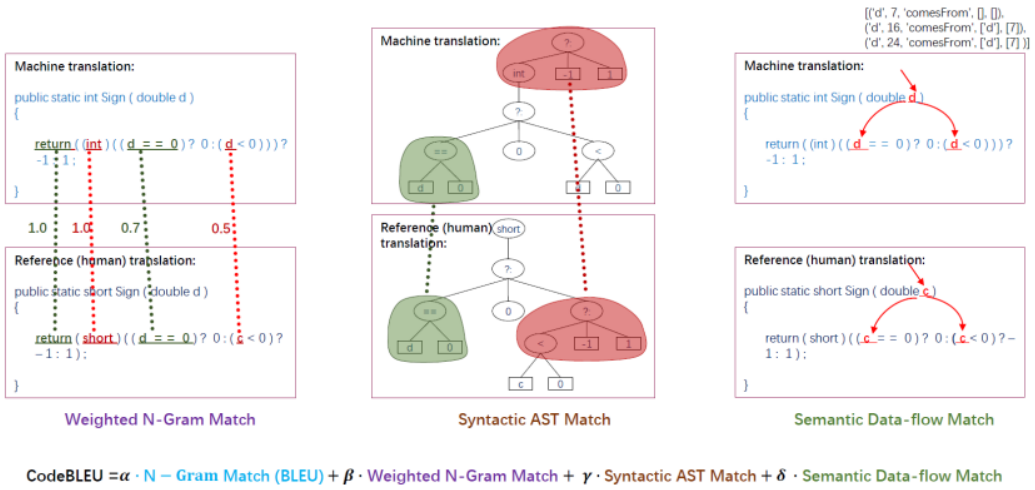

Figure 1: CODEBLEU computation

Moreover, as the code snippets used are competitive programming oriented, they all produce an output that can be checked for correctness. Two different programs can be valid methods to solve the same problem. Therefore, this paper computes BLEU and COMET metrics on the outputs of the generated programs, which will be called resBLEU (RB) and resCOMET (RC). Nonetheless, given the unlikeliness of NLLB models to produce a working program file, this metric has not being computed for those models.

That being said, Table 2 shows the results for each metric and model, both as baseline (bl) and fine-tuned (ft).

Model	B	C	CODEBLEU					RB	RC
			CB	N	WN	S	D		
CodeLlama 7b (bl)	23.44	67.30	43.92	25.39	44.90	40.98	64.39	0.0	58.71
CodeLlama 7b (ft)	**65.06**	**89.13**	**78.53**	**66.81**	**82.49**	**75.77**	**89.02**	**4.57**	***82.89***
CodeLlama 3 8b (bl)	30.62	73.75	43.37	20.34	30.76	49.48	72.90	0.0	60.33
CodeLlama 3 8b (ft)	38.48	**79.28**	**64.11**	40.15	**77.1**	**67.02**	72.13	**4.57**	***82.89***
Llama 2 7b (bl)	19.22	42.25	19.19	15.53	15.54	15.61	30.09	0.0	58.71
Llama 2 7b (ft)	**60.09**	***87.40***	***74.17***	***63.15***	***78.82***	***72.65***	***82.08***	***4.10***	**84.31**
NLLB 600M (bl)	30.10	25.36	29.70	11.59	11.48	21.52	74.23	-	-
NLLB 600M (ft)	56.98	25.36	52.99	51.32	51.19	43.52	65.91	-	-
NLLB 1.3B (bl)	29.38	25.36	28.95	10.23	10.23	20.78	**74.58**	-	-
NLLB 1.3B (ft)	***64.32***	25.36	59.08	**56.66**	56.64	50.87	72.15	-	-

Table 2: LLMs varied parameters

Table shows the three best results for each metric, the best is bold and underlined, the second is best and italic, and the third is just bold. We can observe that LLaMa models generally obtain better results, being CodeLLaMa 7b the absolute winner, although NLLB 1.3B enters the podium on some metrics. Another interesting observation is that CodeLLaMa models provide the vest baseline results, or that LLaMa 2 7b shows the greatest performance increase after fine-tuning.

3 Conclusions

By analysing the results and translation examples from the different models, we can draw various significant conclusions. This section details the most relevant ones.

First, it is important to highlight the increase of performance of every model after the fine-tuning process. Every model got better at this specific task even though we only used around 1000 samples and NLLB models were not trained over code samples. In fact, those models were fine-tuned as if it was an English-English translation task (*eng_Lang-eng_Lang*).

Secondly, COMET score is not trained with code snippets. This fact brings some problems to the table at evaluation time, as it does not properly calculate outputs similarity. For example, the mean COMET score for all NLLB models is exactly the same. More over, let us examine a few concrete examples where COMET's flaws can be noticed:

```
MODEL: CODELLAMA 3 8B
COMET of snippet 30: 0.74267

REAL:
void reverse ( string str ) { if ( str . size ( ) == 0 ) { return ; }
 reverse ( str . substr ( 1 ) ) ; cout << str [ 0 ] ; }
int main ( ) {
    string a = " Geeks \_ for \_ Geeks " ; reverse ( a ) ;
    return 0 ; }

PRED:
void reverse ( string string ) { if ( string . length ( ) == 0 ) return ;
  char temp = string [ 0 ] ; reverse ( string . substr ( 1 , string . length
  ( ) - 1 ) ) ; cout << temp << " " ; }
int main ( ) {
    string string = " Geeks   for   Geeks " ; reverse ( string ) ;
```

```
    return 0 ; }
```

For the previous snippet, the COMET score is 74.27, higher than the mean COMET score of most models. However, there are some serious problems when it comes to code that are less significant for natural language machine translation, and COMET does not penalize them properly. For example, the model output is calling a variable of type *string* with the name *string*, which is obviously a reserved keyword, which is not allowed. Another example is the call to the method *.length()* instead of *.size()*, which semantically is not a big deal but regarding code snippets it is enough to raise an error and avoid compilation. There are also some minor flaws, such as not following the instruction to use the character "_" instead of " ".

There we have another example:

```
MODEL: LLAMA 2 7B
COMET of snippet 129: 0.6839482188224792

REAL:
 float findVolume ( float a ) {
if ( a < 0 ) return -1 ;
float r = a / 2 ;
float h = a ;
float V = 3.14 * pow ( r , 2 ) * h ; return V ; }
int main ( ) { float a = 5 ; cout << findVolume ( a ) << endl ; return 0 ; }

PRED:
  nobody = int ( findVolume ( a ) ) ; cout << nobody << endl ; return 0 ; }
```

There is not much to analyse here, as the snippet tries to use a function that assumes exists rather than developing it, and COMET gives a score of 68.4 anyway.

Therefore, code-related metrics such as CODEBLEU are crucial for evaluating these type of tasks, as it considers aspects often overlooked by common machine translation metrics. RB and RC metrics are also interesting approaches but obviously have some problems. For example, if the result should be "9", RB score is the same for "8" as for "2". Nevertheless, for functions whose output is text, those metrics provide more valuable information than B and C, as the output is simpler than the program itself.