2022 IEEE/ACM 44th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion) | 978-1-6654-9598-1/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICSE-Companion55297.2022.9793747

CrystalBLEU: Precisely and Efficiently Measuring the Similarity of Code

Aryaz Eghbali
University of Stuttgart
Stuttgart, Germany
aryaz.eghbali@iste.uni-stuttgart.de

Michael Pradel
University of Stuttgart
Stuttgart, Germany
michael@binaervarianz.de

ABSTRACT

Recent work has focused on using machine learning to automate software engineering processes, such as code completion, code migration, and generating code from natural language description. One of the challenges faced in these tasks is evaluating the quality of the predictions, which is usually done by comparing the prediction to a reference solution. BLEU score has been adopted for programming languages as it can be easily computed for any programming language and even incomplete source code, while enabling fast automated evaluation. However, programming languages are more verbose and have strict syntax when compared to natural languages. This feature causes BLEU to find common n-grams in unrelated programs, which makes distinguishing similar pairs of programs from dissimilar pairs hard. This work presents *CrystalBLEU*, an evaluation metric based on BLEU, that mitigates the distinguishability problem. Our metric maintains the desirable properties of BLEU, such as handling partial code, applicability to all programming languages, high correlation with human judgement, and efficiency, in addition to reducing the effects of the trivially matched n-grams. We evaluate CrystalBLEU on two datasets from previous work and a new dataset of human-written code. Our results show that CrystalBLEU differentiates similar and unrelated programs better than the original BLEU score and also a variant designed specifically for source code, CodeBLEU.

ACM Reference Format:
Aryaz Eghbali and Michael Pradel. 2022. CrystalBLEU: Precisely and Efficiently Measuring the Similarity of Code. In *44th International Conference on Software Engineering Companion (ICSE '22 Companion), May 21–29, 2022, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3510454.3528648

1 INTRODUCTION

Machine learning models are being used for a variety of programming languages ranging from VHDL [3] to more conventional languages like Java [10] and Python [7]. Moreover, models have been developed to predict a variety of code sizes such as right-hand side of assignment [3], single line of code [2], sequence of API calls [8], block of code [10], full method [4], and full class [7]. Therefore, an ideal metric should be language-agnostic and be able to handle incomplete source code. Furthermore, the running time performance of a metric is an important factor as the metric might be used as an online metric [1]. Hence, the BLEU score [5] has been the dominating metric that provides all of these features.

Nevertheless, BLEU suffers from the problem of ignoring a property in programming languages that is not present in natural languages. This issue, that we call *trivially shared n-grams*, comes from the grammar and syntactic rules of programming languages alongside the coding conventions used by programmers. Some examples of trivially shared n-grams are "`for (`" in for loops, "`) {`" in many control flow statements followed by a code block, the main function definition in languages like C++ and Java, and commonly used APIs like "`console.log(`". The trivially shared n-grams appear in source code regardless of the semantics of the program, which means they increase the BLEU score without any implications on the similarity of two code pieces. In this work we quantify the magnitude of this problem using a novel meta-metric, called *distinguishability*. Distinguishability assigns a positive number to a metric, which shows how much the metric differentiates between similar pairs of programs and dissimilar pairs. Higher distinguishability is desirable in code similarity metrics, because the goal of these metrics is to detect semantic similarity.

To mitigate the problem of trivially shared n-grams and to increase distinguishability, we present a new metric, called *CrystalBLEU*. Our metric is language agnostic, works on partial and incomplete code, runs as fast as BLEU, correlates highly with human judgement, and achieves higher distinguishability than BLEU and CodeBLEU [6].

2 APPROACH

By analyzing two commonly used natural language[1] and source code[2] corpora, we observe that there exists a disparity within the frequency of the top occurring n-grams between natural and programming languages. Specifically, the most occurring n-grams in programming languages appear more frequently than the most occurring n-grams in natural languages, but the least occurring n-grams in programming languages appear fewer times than the least occurring n-grams in natural languages. This means that given a pair of random source code pieces in the same language, it is more likely to find more common n-grams than a pair of random English language texts. These results confirm our hypothesis about the existence of trivially shared n-grams.

Distinguishability. The first step for mitigating against the problem of trivially shared n-grams is to define a meta-metric for measuring the effects of the noise caused by these n-grams on the output of the evaluation metric. We define the *distinguishability* meta-metric based on the problem we observe when using BLEU

[1]Brown dataset, http://www.nltk.org/nltk_data/
[2]Java small dataset, https://github.com/tech-srl/code2seq/#datasets

ICSE '22 Companion, May 21–29, 2022, Pittsburgh, PA, USA

ACM ISBN 978-1-6654-9598-1/22/05.
https://doi.org/10.1145/3510454.3528648

on source code. Given a dataset of source code pieces that are partitioned into equivalent classes, we can calculate the inter-class and intra-class similarities from a given metric. The meta-metric should assign higher scores for metric that achieve high intra-class similarity and low inter-class similarity. Therefore, we define the distinguishability of similarity metric m as the ratio of intra-class to inter-class similarity scores from m. Since calculating the inter-class and intra-class similarities for all possible combinations of solution and references pairs is practically expensive, a sample of these combinations can be used to calculate distinguishability. In our analyses we use same-sized samples from each equivalency class and the same number of pairs for inter- and intra-class pairs of solution and references. This is done to cancel the effects of imbalanced class sizes and larger number of pairs in the inter-class combinations. We calculate distinguishability of CrystalBLEU, BLEU, and CodeBLEU using a dataset from the set of programming challenges and their solutions, which are tested using several test cases. All submissions that pass all of the test cases for a particular task are considered to be semantically equivalent in our work.

CrystalBLEU. We propose a surprisingly simple, yet effective, two-phase algorithm to calculate similarity of a source code piece to one or more reference code pieces, which we call *CrystalBLEU*. First, in the preprocessing phase, based on the observation that trivially shared n-grams are not informative enough, we gather a set, S, of the most frequent n-grams from a code corpus in the same domain. For example, when applying CrystalBLEU to the code pieces from the Nexgen [10] dataset, which consists of catch blocks in Java, we use a corpus of Java catch blocks. Since the elements in S are the most frequent n-grams of the domain of interest, they are shared between many unrelated code pieces, and therefore can be considered as being the trivially shared n-grams. The second phase is the calculation of the CrystalBLEU score for a prediction against a set of references. In this phase we execute the BLEU score algorithm, except when the algorithm extracts the n-grams from the prediction and references, we remove all n-grams that are in S. For instance, if set S contains "`catch (`", our algorithm assumes that no such 2-gram exists in the prediction or the references. Hence, the modified precision computed for the n-grams only considers the precision of non-trivial shared n-grams. Another possible approach is to assign lower weights to the n-grams in set S, but for two reasons we do not use this approach. The first and main reason is that removing the n-grams in S makes the algorithm simple, while resulting in high distinguishability, and it does not add additional parameters that require tuning. The second reason is that removing them makes the calculations faster, which is of considerable importance.

3 EVALUATION

CrystalBLEU is as scalable as BLEU, with similar or even faster running times for calculating the metric. It only requires a one-time preprocessing phase over a typical corpus from the task's domain. In our experiments it runs in less than 20 seconds on a regular laptop over 1.8m tokens.

RQ1: Distinguishability of CrystalBLEU. Table 1 shows the inter- and intra-class similarity scores achieved by BLEU, CodeBLEU, and CrystalBLEU on the Java programs in the ShareCode dataset, alongside the respective distinguishabilities. For the C++ programs, we are not able to calculate CodeBLEU, as it does not support the language. However, for the C++ programs BLEU achieves 2.82 in distinguishability, while CrystalBLEU achieves 8.29.

Table 1: Similarity scores for ShareCode Java programs.

	BLEU	CodeBLEU	CrystalBLEU
Intra-class	0.79	0.52	0.65
Inter-class	0.32	0.36	0.10
Distinguishability	2.47	1.44	**6.50**

RQ2: Correlation with human judgment. We conduct a small human study to show that the increased distinguishability of CrystalBLEU does not cost the correlation with human judgment. In the human study we present two code pieces to the subject and ask to give a similarity score from 0 to 5, based on the human study done by Tran et al. [9]. The code pieces are selected from the Nexgen dataset and the code-to-code translation tasks of the CodeXGLUE dataset. In our experiment CrystalBLEU achieves 0.85 in Spearman's rank correlation in the Nexgen dataset, compared to 0.77 for CodeBLEU and 0.84 for BLEU. In the same experiment both CrystalBLEU and BLEU have 0.93 in Pearson correlation, compared to 0.86 for CodeBLEU. For the CodeXGLUE dataset, both CrystalBLEU and BLEU achieve 0.96 in Spearman's rank correlation (0.98 Pearson correlation), compared to 0.88 for CodeBLEU (0.93 Pearson correlation).

4 CONCLUSION

CrystalBLEU is a scalable metric, that provides the benefits of BLEU score, in addition to higher distinguishability. This means that using CrystalBLEU to evaluate code-generating models can better differentiate poor quality predictions from the higher quality ones.

5 ACKNOWLEDGMENTS

This work was supported by the European Research Council (ERC, grant agreement 851895), and by the German Research Foundation within the ConcSys and Perf4JS projects.

REFERENCES

[1] Saikat Chakraborty, Miltiadis Allamanis, and Baishakhi Ray. 2018. Tree2Tree Neural Translation Model for Learning Source Code Changes. http://arxiv.org/abs/1810.00314
[2] Yangruibo Ding, Baishakhi Ray, Premkumar Devanbu, and Vincent J Hellendoorn. 2020. Patching as Translation: the Data and the Metaphor. *arXiv:2008.10707*.
[3] Jaeseong Lee, Pengyu Nie, Junyi Jessy Li, and Milos Gligoric. 2020. On the Naturalness of Hardware Descriptions. In *ESEC/FSE*. 530–542.
[4] Anh Tuan Nguyen, Trong Duc Nguyen, Hung Dang Phan, and Tien N. Nguyen. 2018. A Deep Neural Network Language Model with Contexts for Source Code. In *SANER*.
[5] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: a method for automatic evaluation of machine translation. In *ACL*. 311–318.
[6] Shuo Ren, Daya Guo, Shuai Lu, Long Zhou, Shujie Liu, Duyu Tang, Ming Zhou, Ambrosio Blanco, and Shuai Ma. 2020. CodeBLEU: a Method for Automatic Evaluation of Code Synthesis. *arXiv:2009.10297* (2020).
[7] Zeyu Sun, Qihao Zhu, Lili Mou, Yingfei Xiong, Ge Li, and Lu Zhang. 2019. A Grammar-Based Structural CNN Decoder for Code Generation. In *AAAI*. 7055–7062. https://doi.org/10.1609/aaai.v33i01.33017055
[8] Yanfei Tian, Xu Wang, Hailong Sun, Yi Zhao, Chunbo Guo, and Xudong Liu. 2018. Automatically Generating API Usage Patterns from Natural Language Queries. In *APSEC*. IEEE, 59–68.
[9] Ngoc M. Tran, Hieu Tran, Son Nguyen, Hoan Nguyen, and Tien N. Nguyen. 2019. Does BLEU score work for code migration?. In *ICPC 2019*. IEEE / ACM, 165–176. https://doi.org/10.1109/ICPC.2019.00034
[10] Jian Zhang, Xu Wang, Hongyu Zhang, Hailong Sun, Yanjun Pu, and Xudong Liu. 2020. Learning to Handle Exceptions. In *ASE*.