Out of the BLEU: how should we assess quality of the Code Generation models?

Mikhail Evtikhiev
JetBrains Research
mikhail.evtikhiev@jetbrains.com

Egor Bogomolov
JetBrains Research
egor.bogomolov@jetbrains.com

Yaroslav Sokolov
JetBrains
yaroslav.sokolov@jetbrains.com

Timofey Bryksin
JetBrains Research
timofey.bryksin@jetbrains.com

ABSTRACT

In recent years, researchers have created and introduced a significant number of various code generation models. As human evaluation of every new model version is unfeasible, the community adopted automatic evaluation metrics such as BLEU to approximate the results of human judgement. These metrics originate from the machine translation domain and it is unclear whether they are applicable for the code generation tasks and how well do they agree with the human evaluation on this task. There also are two metrics, CodeBLEU and RUBY, that were developed to estimate the similarity of code and take into account the code properties. However, for these metrics there are hardly any studies on their agreement with the human evaluation. Despite all that, minimal differences in the metric scores are used to claim superiority of some code generation models over the others.

In this paper, we present a study on applicability of six metrics—BLEU, ROUGE-L, METEOR, ChrF, CodeBLEU, RUBY—for evaluation of the code generation models. We conduct a study on two different code generation datasets and use human annotators to assess the quality of all models run on these datasets. The results indicate that for the CoNaLa dataset of Python one-liners none of the metrics can correctly emulate human judgement on which model is better with $> 95\%$ certainty if the difference in model scores is less than 5 points. For the HearthStone dataset, which consists of classes of particular structure, the difference in model scores of at least 2 points is enough to claim the superiority of one model over the other. Using our findings, we derive several recommendations on using metrics to estimate the model performance on the code generation task.

ACM Reference Format:
Mikhail Evtikhiev, Egor Bogomolov, Yaroslav Sokolov, and Timofey Bryksin. 2022. Out of the BLEU: how should we assess quality of the Code Generation models?. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Code generation systems are a way to make the process of writing source code easier and more accessible. In a common formulation, such systems take an intent — description in a natural language — as an input and output a snippet of code that implements the intent. Proper code generation is a long-standing problem [5] that, if implemented well, would aid in education, simplify drafting program implementations for non-programmers, and attract new programmers who may have limited programming experience in a given language [10]. Therefore, having a strong code generation model could be very beneficial for the software development industry.

There currently are many various code generating models [10, 34, 41–43] and several datasets [3, 7, 18, 20, 25, 39, 44] on which these models are evaluated. Currently, the code generation models are assessed with either accuracy, BLEU metric [26], or CodeBLEU metric [32]. Originally, BLEU was created to evaluate the quality of the machine translation for the natural language processing, and it was empirically validated to be correlated with the human judgments of the translation quality for the natural language texts. However, no such validation exists for the code generation task. Moreover, for the closely related code migration problem Tran et al. [35] have shown that the BLEU results are only weakly correlated with the human judgment. For the related code summarization problem, Roy et al. [33] have shown that BLEU metric is a less reliable indicator of human judgement than other metrics such as METEOR or ChrF.

We identify three possible problems with the application of the BLEU metric for the code generation task, that, up to our knowledge, have hardly been addressed [32, 35]:

- It is unclear whether existing metrics are suitable for the assessment of the code generation models.
- It is unclear how significant are the metrics results and how big should be the difference in the scores to claim one model's supremacy over the other.
- It is unclear how well do the metrics scores for generated code datasets correlate with the human judgement.

In our study, we consider two different datasets. The CoNaLa dataset [39] is a dataset of questions posted on Stack Overflow[1] with the posted solutions in Python. The solutions are short and generally are one line-long. Card2code Hearthstone [18] is a dataset dedicated to generating classes that are descriptions of the cards used in the Hearthstone game. The classes are rigid and most of the

[1]Stack Overflow: https://stackoverflow.com/

class structure is identical for every snippet. Our choice of datasets was motivated by the following factors:

- Every snippet in the dataset is a self-contained piece of code (as compared to e.g. Django dataset [25]), which provides a more realistic code generation problem setup.
- The snippet descriptions are unambiguous enough to allow human annotators to assess whether the generated code corresponds to the description (which is not the case for e.g. Docstrings dataset [7]).
- The two datasets we chose provide two different formulations of the code generation problem that are relatively accessible for the existing code generation models.

For each of the datasets, we consider several machine learning models for code generation.

For the CoNaLa dataset, we compare the results of five different models: CoNaLa baseline [39], Codex [10], TranX without pretraining [42], and TranX with pretraining both with and without reranking [43]. While being publicly available, the selected models greatly vary in quality and complexity, which allows judgement on the relation between models' quality, metric values and human assessment.

For the Hearthstone dataset, we compare the results of two models that previously evaluated on this dataset: NL2Code [41] and GCNN [34]. We could not apply Codex to the Hearthstone dataset, as the model encountered the required code during training and simply remembered it. When we prompt Codex with a Hearthstone card summary, it generates correct code not only for the corresponding class, but also for several other cards one by one. It happens because the model trained on vast amount of code from GitHub, including the repository with Hearthstone dataset.

To address the problem of automated metrics applicability, we carry out paired bootstrap resampling [12]. We consider BLEU, METEOR, ROUGE-L, ChrF, CodeBLEU, and RUBY [6, 17, 26, 27, 32, 35] metric scores of the models.

To address the problem of correlation between human assessment and computer metrics scores, we carry out a human evaluation of the generated snippets. Developers evaluated whether the suggested snippets were helpful in solving the posed problem on the scale from 0 to 4. For the CoNaLa dataset, we get in average 4.5 grades from different developers per snippet. In total, 12 developers took part in the evaluation. For the Hearthstone dataset, there were 4 graders, and every grader evaluated the whole dataset.

We then aggregate the obtained grades according to the method of Ma et al. [22], yielding a human "ground truth". We use paired bootstrap resampling for the obtained scores to further study the results. For the CoNaLa dataset for any pair of evaluated models, the "ground truth" grades can be used to distinguish which model is better with $> 95\%$ confidence. For the Hearthstone dataset, human graders and ChrF, CodeBLEU, and RUBY metrics cannot decide with at least 95% confidence which of the NL2Code and GCNN models is better. Yet, ROUGE-L, METEOR, and BLEU metrics find that NL2Code model is better than GCNN with $> 95\%$ confidence.

The amount of grades per snippet we collect is not enough to analyze the metrics performance on the snippet level, as Mathur et al. [24] argue it is necessary to have 15 grades per snippet to provide a stable score. Thus, we focus on the comparison of models at the corpus level. The available set of ML models is not large enough to study the significance of difference in metric scores: for example, for CoNaLa dataset there only are 5 original models, and thus only 10 different pairs of models to compare. To provide a statistical analysis of corpus-level score differences, we augment the original set of models with a set of synthetic models. In it, we replace a part of some model predictions with predictions that have a higher or lower human assessment score, following Roy et al. [33].

We show that, depending on a task and a metric, in 8.5% to 62.5% cases the metrics predictions of relative model quality do not agree with the human assessment. We find that the bigger is the difference in metric scores of two models, the more likely human assessors are to agree with the model quality assessment according to the metric. However, it is not the case for smaller differences in metric scores. For the CoNaLa dataset, all automated metrics disagree with human evaluation in $> 5\%$ of the cases if the difference in metric scores is less than 5 points. As the human evaluation is the gold standard of quality assessment, automated metrics cannot be used to tell whether one model is better than the other on the CoNaLa dataset, if the difference in scores is less than 5 points. For the Hearthstone dataset, the automated metrics can on average be used to find which of the two models is better if the difference in scores is at least 2 points.

To sum up, our contributions are the following:

- We investigate the applicability of BLEU, ROUGE-L, METEOR, ChrF, CodeBLEU, and RUBY metrics for assessing the quality of generated code; up to the best of our knowledge, this is the first study that carries out such an assessment. We show that for both CoNaLa and Hearthstone datasets and every metric at the corpus level, metrics disagree with the human judgement in more than 5% of the cases, meaning that we are yet to find the most appropriate metric for code generation evaluation.
- We find that the BLEU and the CodeBLEU can disagree with the human judgement when assessing the non-synthetic models, while ChrF, ROUGE-L and METEOR always agree with humans. This observation is particularly important as BLEU was clearly the evaluation standard for code generation [34, 41] and the CodeBLEU rapidly becomes de-facto standard [20].
- We find that ChrF and ROUGE-L are the best-performing metrics for the code generation task. In particular, these metrics make fewer type-II errors and, unlike *e.g.,* BLEU, can be used to discriminate model performance if the difference in scores is large enough for both CoNaLa and HearthStone dataset.

2 BACKGROUND

2.1 Code Generation

Code generation is a long-standing problem [5], and a good code generation model could decrease the barrier for writing code, automate some of the routine tasks engineers have, and help non-programmers create programming solutions for their problems.

This problem is related to the other applications of machine learning to code. In the greater context of code-related tasks, code generation is a task dual to code summarization, and is closely related to code migration and code completion.

The development of deep learning has enabled the successful application of various neural models to the code generation problem. In particular, Ling et al. [18] suggested a sequence-to-sequence model to generate code from natural language descriptions. Yin et al. [41] and Rabinovich et al. [30] modified the standard decoder that generates a sequence of tokens to enforce grammar rules by first generating an abstract syntax tree and then converting it into code. Sun et al. [34] suggested replacing recurrent neural networks with grammar-based structural convolutional neural networks. Unlike recurrent neural networks, convolutional neural networks can track the context even between distant regions of the analyzed data. In contrast, recurrent neural networks are not capable of tracking the context when relevant pieces of information are far apart, which is called the long dependency problem [8, 14]. Wei et al. [37] suggested dual training of code generation and code summarization models to enhance the quality of both models.

In contrast to recurrent neural networks, models based on Transformer [36] process the whole sequence simultaneously, which is more efficient both in terms of computational speed and capturing the dependencies between distant tokens. Nowadays, we observe rapid progress in the quality of code generation models due to gigantic Transformer-based models such as Codex [10], AlphaCode [16], and CodeParrot[2].

The table 1 summarizes the types of neural networks and the metrics used by the researchers in the papers discussed above.

2.2 Evaluation of Code Generation Models

To be able to track the improvements of the model, it is necessary to evaluate its performance. Human assessment is the gold standard for most machine translation or machine generation problems. However, manual assessment is also very expensive and slow, and it is impractical to do human evaluation for each generated sample during the model development. Thus, it is crucial to have an easy to compute metric to evaluate the output of a model.

The code generation task is no different. The evaluation approaches for code generation can be split into three categories:

(1) Metrics from the machine translation domain;
(2) Metrics developed to compare code snippets;
(3) Running and testing the generated code.

Further, we discuss all three in detail.

2.2.1 Metrics from machine translation. As Table 1 shows, the quality of code generation models is typically assessed by the BLEU metric score [26] or accuracy. The BLEU (BiLingual Evaluation Understudy) metric is a corpus-level metric that was originally developed for the automatic quality evaluation of machine-translated texts. BLEU metric is a corpus-level metric based on the modified n-gram precision measure with a length penalization for the candidate sentences that are shorter than the reference ones.

Researchers also consider other machine translation metrics:

- ROUGE-L [17] is a recall-oriented metric that looks for the longest common subsequence between the reference and the candidate.
- METEOR [11] is a mixed recall-precision metric that also penalizes candidates for not having adjacent unigrams that are adjacent in the reference example.
- ChrF [27] is a character n-gram F-score metric, where precision and recall in the F-score computation are averaged over 1- to 6-grams of characters.

In addition to the aforementioned metrics, researchers often report Accuracy as an additional metric. While it supports the fact that one model is superior to another, it is rarely used as the primary metric in the generation tasks due to being too strict and less robust. Thus, we do not analyze Accuracy in our study, as we focus on metrics used for direct model comparison.

2.2.2 Metrics designed for code. Even though BLEU (like METEOR and ROUGE-L) was originally created for the assessment of machine translation models for natural languages, it is widely used for assessing code generation, code migration, and code summarization models. Tran et al. [35] conducted an empirical study on BLEU to check its suitability in the context of the code migration task. In their paper, they show that the BLEU metric has a rather weak correlation of 0.583 with the human assessment. The authors also construct a synthetic dataset to illustrate that BLEU may yield similar results for the models whose quality differs from the perspective of the human grader. To address this issue, the authors devised a new metric RUBY, which takes code structure into account. The metric compares the program dependency graphs (PDG) of the reference and the candidate; if the PDG is impossible to build, it falls back to compare AST, and if AST is also impossible to build, the metric compares the weighted string edit distance between the (tokenized) reference R and candidate sequence.

Ren et al. [32] suggested a new metric called CodeBLEU to evaluate the quality of generated code for code generation, code translation and code refinement tasks. CodeBLEU is a composite metric with the scores being weighted average of 4 different sub-metrics treating code differently: as a data-flow graph, as an abstract syntax tree, and as a text.

2.2.3 Test-based evaluation. The impressive performance of recent large-scale models [10, 16] allows the use of evaluation techniques which are closer to practical applications: actually running the generated code on pre-written unit-tests and checking whether it solves the posed problem. For example, the authors of Codex [10] also present a dataset called HumanEval which consists of programming tasks and tests validating the correctness of the generated code.

While this approach is reasonable, we argue that for now it will not fully replace existing evaluation techniques that rely on the usage of automated metrics. In order to apply test-based evaluation, researchers need both carefully created datasets for each particular code generation setting. Additionally, the studied models should pass large enough number of tests in order to robustly distinguish between them.

[2] CodeParrot model page on HuggingFace: https://huggingface.co/codeparrot/codeparrot

Paper	NN type	Metrics	Year
Barone et al. [7]	NMT	BLEU	2017
Chen et al. [10]	Transformer	BLEU, Pass@k	2021
CodeParrot	Transformer	Pass@k	2021
AlphaCode [16]	Transformer	Evaluation on Codeforces	2022
Ling et al. [18]	RNN	BLEU, Accuracy	2016
Lu et al. [20]	RNN, Transformer	BLEU, Accuracy, CodeBLEU	2021
Rabinovich et al. [30]	RNN	BLEU, Accuracy, F1	2017
Ren et al. [32]	PBSMT, Transformer	BLEU, Accuracy, CodeBLEU	2020
Sun et al. [34]	CNN, RNN	Accuracy, BLEU	2019
Wei et al. [37]	RNN	BLEU, Percentage of valid code	2019
Yin et al. [41]	RNN	BLEU, Accuracy	2017
Yin et al.[42]	RNN	Execution accuracy, exact match accuracy	2018
Yin et al.[43]	RNN	BLEU, Accuracy	2019

Table 1: Comparison of various code generation papers

3 MOTIVATION

Metrics are used during the validation phase of a machine learning pipeline and to compare different models. However, if human assessment is the golden standard, the metric should align with human judgement as closely as possible. For example, in machine translation, there is an annual contest between various metrics, with the best metric being the one that emulates human judgement the best [21, 23].

Even if some metric (such as BLEU) has been used in the past to emulate human judgement, it may be beneficial to consider other metrics which may have better correlation with human assessments. A similar situation has emerged in the natural language generation: even though BLEU was initially adopted to this domain, it was later shown [19] that word-overlap based metrics (such as BLEU) have very low correlation with human judgement in certain natural language generation tasks such as dialog response generation.

In the rest of this Section, we discuss in detail why studying the automated metrics for code generation is important and which questions we need to answer in this work.

3.1 Metrics and Test-based Evaluation

With the recent introduction of HumanEval [10], a dataset that allows running and testing generated Python code in a close-to-practical setting, it might seem that the usage of automated metrics will soon become obsolete. However, we think that it will not be the case in the near future.

Firstly, the collection of test-based evaluation datasets requires significant human effort to develop a set of tasks as well as cover them with tests. Given that the code generation task can be formulated differently and applied to different languages and domains, each particular case requires a separate manually crafted evaluation system. Thus, the usage of automated metrics is helpful when adopting code generation in new domains.

Secondly, training and inference of very large models like Codex are both costly and technically challenging [10]. For this reason, an important direction of research is the development of smaller code generation models which cannot yet achieve quality comparable to large Transformer-based counterparts. For smaller models, evaluation frameworks like HumanEval would lead to poor metric scores, and their robustness for model comparison in this case remains an open question.

Finally, even if two models generate code that does not pass any tests, it still might be possible to say which piece of code is closer to the correct solution. For example, for a problem "Get rid of None values in dictionary *d*"and two pieces of code presented below the first piece is much closer to the right solution, even though it still does not pass the tests.

```
1. print(dict((k,v) for k,v in d.items() if v)))
2. list(d.values())
```

It is important to be able to evaluate the quality of generated code snippets even if they do not pass the tests, as developers might find some generated snippets easier to fix and integrate in their code.

3.2 Are Existing Metrics Suitable for Code Generation?

Machine translation metrics were developed for natural languages and do not take into account properties of programming languages. Usage of such metrics might be sub-optimal for the code generation assessment due to several factors.

3.2.1 Differences between code and natural language. Programming languages have a strict syntactic structure, while the natural language structure is more relaxed. For example, while swapping two groups of tokens in a natural language sentence often does not strongly affect its meaning, such a transformation will often make a code snippet invalid. Secondly, machine translation (MT) metrics measure the lexical precision of the model output, while for the generated code we want to assess its functionality.

It is possible to make MT metrics somewhat more code-friendly. E.g., it is possible to rename all the variables in the candidate and the references according to their order of appearance, removing the spurious mismatch due to the different naming conventions. Yet, some issues cannot be apparently addressed without taking the code structure into account. It is therefore plausible that a metric that will take into account the code snippets' structure and syntax will be a better proxy of the human assessment.

3.2.2 BLEU has been outperformed in other tasks. It is unclear whether BLEU or any other metric scores are correlated well with the human assessment for the code generation task. The human judgement on whether the suggested code snippet is good at solving the problem is considered to be the ground truth. However, as human evaluation is very expensive, it is obviously impossible to have every new output of the model evaluated by a group of programmers. Original papers for machine translation metrics [6, 17, 26, 27] include studies that show a high correlation between the metrics scores and the human judgement for the machine translation task. However, a review by Reiter [31] shows that the BLEU–human correlations are poor for natural language generation tasks and BLEU should only be used to evaluate machine translation NLP systems.

For the closely related problem of code migration, it was shown [35] that the correlation between BLEU scores and human grades is 0.583, which is rather weak. There is a study on the metric-human correlation for BLEU, accuracy, and CodeBLEU metrics [32], which have shown that CodeBLEU metric is better correlated with human opinion, than accuracy or BLEU. However, this study didn't consider other metrics.

3.2.3 Translation from metrics to human assessment. It is unclear that an increase in a metric score is linearly related to the increase of the "true" quality of the code snippet. For an illustration, let us consider one of the tasks in the CoNaLa dataset:

```
Task: concatenate a list of strings ['a', 'b', 'c']
baseline model solution: set(['a','b','b'])
best-tranx-rerank solution: ''''''.join(['a','b','c'])
```

Even though the baseline snippet fails to solve the task question (and didn't even manage to reproduce the list of strings that need to be concatenated), it has a relatively high BLEU score of 48.09. The second snippet successfully solves the problem and has BLEU score of 100.

Now, let us consider hypothetical outputs of two different models A, B. Both outputs have BLEU 50, but for model A every candidate has BLEU 50 and is of quality similar to the one above, while for model B, half of the candidates have BLEU 0 and the other half have BLEU 100. In this case, it may be argued that model B is better than model A, even if they have close corpus-level BLEU scores: given the example above, model A can generate hardly relevant code snippets all the time, while model B generates perfect code in half of the cases.

If the dependency between human assessment and metric values is not linear, we cannot simply average the metric values over all the snippets to reflect the human assessment of the model. In addition, there might be other reasons why BLEU scores and human scores might not correlate well, and it is necessary to study the correlation between the two to be able to infer the knowledge how to interpret BLEU scores and assess the models' quality from them.

3.3 Do We Use Automated Metrics Correctly?

The current way of using automated metrics to assess models is to report a single, corpus-level number for each model output on the test dataset. While this approach is simple and might be very practical during the training process, it is unclear how the raw difference in metric scores can be translated into statements on the statistical significance of the difference.

The currently accepted way (see *e.g.,* [20, 30, 41, 42]) of comparing different code generation models is by simply comparing their BLEU or CodeBLEU scores, averaged over the whole test dataset. However, when an improvement from *e.g.,* the BLEU score of 29 to the BLEU score of 30 is claimed, it is rarely supported by data on the statistical significance of the improvement. As Roy et al. [33] have shown, for the closely related code summarization task small difference in metric scores is statistically insignificant, it is possible that the same phenomenon exists for code generation.

Therefore, it is important to study how big the difference between the metric scores of two models for a particular dataset should be to claim that one of the models is better than the other with the desired confidence.

4 STUDY OF METRICS FOR CODE SUMMARIZATION

Recently, Roy et al. [33] studied the applicability of automated metrics for the code summarization task, which is closely related with code generation. For this task, metrics such as BLEU are also used widely as proxies of human evaluation. The authors have shown that there is no statistically significant difference between the models with corpus scores different by less than 1.5 points according to any of the considered metrics. Moreover, all the metrics the authors considered are not reliable proxies of human evaluation if the difference in corpus scores is less than 2 points according to the metrics. Of all the metrics considered in Roy et al., METEOR, ChrF, and BERTScore show the best agreement with the human judgement on the corpus level. As Roy et al. do an extensive study of the metric performance for a task that is closely related to the code generation, we adopt many of the methods they have employed in our research.

4.0.1 Dataset and labeling. Roy et al. use the Java code summarization dataset of LeClair et al. [15]. They randomly sample 383 snippets from it and generate 5 summaries with different models. The human annotators then evaluate the 5 generated summaries and the reference summary on a 5-point Likert scale to assess the conciseness, fluency, and content adequacy of each summary. They also assign a Direct Assessment (DA) score on a $0-100$ scale that reflects their opinion about the general quality of a summary. Only the Direct Assessment score is used to analyze the relative metric performance.

4.0.2 Corpus-level metric assessment. The corpus-level assessment of metrics applicability by Roy et al. pursues two slightly different goals. First, authors are interested in whether the metrics are capable of distinguishing the quality of the existing models. To do that, they carry out randomized significance testing on the 383-snippet dataset to find, that out of 5 models considered in the study the difference in scores of the best 4 models is not statistically significant. It is important to highlight that this lack of statistical difference was found solely from the metric scores and does not rely on human labelling.

The second goal for the corpus-level metric assessment is to find whether commonly used corpus-level metrics reflect human quality assessments of generated summaries. There is a relative dearth of the available machine learning models (Roy et al. used 5 code summarization models in their study). Thus, it is impossible to study directly, what difference in metric scores is necessary to claim that one model is better than the other according to the humans – there is not enough pairs of models to get enough data on differences in model scores. However, if there would be many more independent models, the researchers would have to label much more model outputs, increasing the costs and laboriousness of the study. In order to get more diversity in metric scores without increasing number of summaries to label, Roy et al. use synthetic models.

A synthetic model is a model that yields set of summaries based on one of the 5 original models, with a varying proportion of summaries replaced by the predictions of the other models. In particular, to create a synthetic model that improves the original model A by 1%, authors replace 1% of the summaries predicted by the model by the better predictions of other models. The quality of the prediction is assessed according to the human DA score. Roy et al. create a set of synthetic models and then select 100 of them. Then they add them to the five original models and do a pairwise comparison into several different buckets based on the statistical significance of the metric score difference as well as on the magnitude. The bucket can be defined for e.g. statistically significant metric differences between 2 and 5. For each of these pairs, Roy et al. also calculate the significance of the difference in their corresponding human DA scores. The effectiveness of a corpus-level metric can then be determined by looking at the agreement between the metric score and human assessment score. For a reliable automatic evaluation metric, one expects to find a one-to-one correspondence between significant differences in metric scores and human assessment scores.

Using pairwise comparison approach, Roy et al. are able to analyze the following:

- They find, for how many pairs in a given bucket the two models in the pair are significally different according to each metric. This allows to deduct, what difference in the metric scores of two models outputs is necessary to expect that the two models will also be significantly different from the metric point of view.
- For each bucket and for every metric they consider the group of pairs, in which one model is significantly better than the other according to the metric. Then for each pair they check whether the two models in it are also significantly different according to the human assessment. This allows them to study the Type-I error of each metric and check how it changes from bucket to bucket.
- For each bucket and for every metric they consider the group of pairs, in which the two models are not signifcantly different according to the metric. Then for each pair from this group they check whether the two models in it are significantly different according to the human assessment. This allows them to study the Type-II error of each metric and check how it changes from bucket to bucket.

From this analysis, Roy et al. have found that automatic evaluation metrics are not able to accurately capture differences in summarization quality between two approaches when the metric difference is less than 2 points. METEOR, BERTScore and chrF perform the best in terms of Type-I and Type-II error rate. BLEU has the highest Type-I error rate regardless of the magnitude of the difference.

4.0.3 Snippet-level analysis. Roy et al. also consider the metric performance for the snippet level. In principle, snippet-level metric result analysis can provide an advantage over corpus-level analysis by tracking fine-grained performance of the models. However, Mathur et al. [24] argue that it is necessary to collect at least 15 human assessments per snippet in order to provide stable score. To carry out the snippet-level analysis, Roy et al. use the Direct Assesment Relative Ranking technique, which compares the pairwise relative scores of two snippets [23]. This technique relies on the Direct Assessment scoring and cannot be applied to the the annotations on the 5-point scale, which is the scale we used for labeling the generated code snippets. As we were able to collect only 4 grades per snippet for the Hearthstone dataset and 4.5 grades per snippet for the CoNaLa dataset, we opted not to analyze metric performance on the snippet level.

5 METHODOLOGY

The problems we list in Section 3 have motivated us to pose the following research questions:

- RQ1 Does the performance of the considered models differ significantly on the corpus level?
- RQ2 How significant are the results of automated metrics and how big should be the difference in corpus-level metric scores of two models to claim that one model is better (according to the given metric) than the other with predefined significance?
- RQ3 How well do the corpus-level metric scores reflect the human assessment of generated code?

Inspired by the work of Roy et al. [33] described in detail in Section 4, the pipeline of our approach is as follows:

1. We collect the models' output on the datasets we consider and evaluate automated metrics on generated code snippets, getting every metric score for every generated snippet.
2. We carry out a human evaluation of the generated snippets (described below in more details), collecting a set of human grades for every generated snippet.
3. Using the obtained set of human grades, we get the "ground truth" human grade by aggregating the grades together with the M-MSR algorithm [22], getting a single grade for each snippet evaluated by experts.
4. Using the models' output, we create synthetic models by replacing some varying parts of the predictions with the predictions that received higher or lower human assessment score. For example, to get a synthetic `tranx-annot` model with 1% of predictions improved, we consider its outputs and replace 1% of its worst predictions with the best predictions available from other models. The quality of a prediction is derived from the human assessment score.
5. For every pair of both synthetic and non-synthetic models evaluated on the same dataset, we carry out paired bootstrap resampling. We do that to find the statistical significance

of the claim that one of the models is better than the other according to the metric scores. We use a 95% threshold to claim a statistically significant difference between the models.

6. For each dataset evaluated by humans and for every pair of models evaluated on it, we carry out paired bootstrap resampling on the ground truth grades to check with what statistical significance we can infer that one of the models is better than the other according to the human opinion.
7. Following [24] and [33], we carry out a pairwise model comparison of human assessment and corpus level metrics for CoNaLa and HearthStone datasets. We start by computing the difference in corpus-level metric scores for all pairs of models evaluated over the given dataset. We then divide these model pairs into several bins according to the difference in the metric scores; we also have an extra bin for the pairs which metrics cannot distinguish. For each of the pairs in every bin, we check if the human evaluation agrees with the metric evaluation, *i.e.,* do humans distinguish the pair of models or not.

5.1 Datasets and Models

In our study, we consider two different datasets: CoNaLa [39] and Card2code Hearthstone [18]. We focused on the datasets containing general Python code, leaving the non-Python datasets such as Spider (containing SQL) [44] and JuICe (containing Jupyter Notebooks) [4] out of the scope.

5.1.1 CoNaLa. The CoNaLa dataset was collected by Yin et al. [40] and consists of 2,879 examples (split into 2,379 training and 500 test examples), crawled from Stack Overflow and then manually curated by human annotators. In addition to the main dataset, Yin et al. also provide a large automatically-mined dataset that consists of Stack Overflow "how to" questions as training intents and contiguous lines from code blocks in answers as candidate implementations for the intent. This dataset has more than a hundred thousand examples. Some of the models that we consider use it for training. The CoNaLa dataset has the following features:

- The CoNaLa dataset has a sound variety of intents that cover many methods used in Python (as compared to *e.g.,* the Card2Code dataset [18], which is dedicated to the generation of classes with very rigid structure).
- Intents in the CoNaLa dataset are detailed and written in natural language, which distinguishes it from *e.g.,* the Docstrings [7] dataset, where the intents are rather short and in many cases a human programmer would have problems with writing the correct code given only the intent.
- There is a relatively rich choice of the publicly available models that were evaluated on this dataset (as compared to the other datasets), enabling us to have more comparisons.
- The best performing models, evaluated on the CoNaLa dataset, have BLEU scores around 30, allowing to have generated test snippets of both high and low quality. For example, the best model evaluated on the Docstrings dataset has BLEU 12.1, which corresponds to a majority of the snippets being low quality, making it harder for human graders to reliably distinguish between them.
- The CoNaLa snippets are generally very short, with the absolute majority of them being a single line of code. It limits the possible usability of the CodeBLEU and RUBY metrics that take code structure into account.

We evaluate five models on the CoNaLa dataset. One of the models we consider is the baseline CoNaLa model [39], and three others are Transformer-based `tranX` models. The `tranx-annot` model was trained on the main CoNaLa dataset; `best-tranx` was also pretrained on the automatically-mined CoNaLa; `best-tranx-rerank` is the enhanced version of the second model that uses reranking postprocessing (*i.e.,* reranking the *n*-best predictions to increase the quality of the output). Finally, we run Codex [10], specifically, its `davinci` version, in the Q&A mode. Following the authors' recommendations, we do not fine-tune Codex on the CoNaLa training part and rather provide it with three code snippets as examples. That is, each code snippet is generated via OpenAI Q&A API for Python code generation, and three intent-snippet pairs as the examples.

5.1.2 Card2Code Hearthstone. Card2Code is a pair of datasets derived from the collectible trading card games *Magic the Gathering* and *Hearthstone*; in our research, we focus on the Hearthstone dataset as it is more popular among the researchers. The Hearthstone dataset contains 665 pairs of Hearthstone card descriptions and corresponding Python snippets. Each snippet is a class implementation that can be used in the Hearthbreaker Hearthstone simulator [1] to describe the card's logic. The dataset is split into 533 training pairs, 66 validation pairs, and 66 test pairs. The Hearthstone dataset has the following features:

- As the intents are the descriptions of Hearthstone cards that should adhere to the Hearthbreaker notation, the generated code has a relatively rigid structure.
- The code generation problem is very peculiar: every task requires the model to generate a class. The snippets have very similar outline and the difference between various snippets is limited: each snippet is a class inherited from one of three parent classes (MinionCard, SpellCard and WeaponCard). Almost every snippet has exactly two methods: constructor and a method with name depending on the parent class (*use* for SpellCard, *create_weapon* for WeaponCard). Thus, the generality of the conclusions we may infer from the results is limited.
- The generated code is relatively long and complex, allowing application of the CodeBLEU and RUBY metrics that take the underlying code structure into account.

There are only two *publicly available* models that are evaluated on the Hearthstone dataset. One of the models is a syntactic neural model `NL2code` [41], and another is a grammar-based structural convolutional neural network `GCNN` [34]. The Codex model was evidently familiar with the dataset since it provided reference snippets as an output, so we did not consider it. In particular, without a tight limit on the number of generated tokens, Codex successfully generated several classes from the testing dataset in a single run. It suggests that Codex is capable of reproducing whole files that it has seen during training, including ones from the Hearthstone dataset.

To check the significance of the difference in the metric scores, we have considered paired bootstrap resampling [12] for the metric scores of the models evaluated on the test part of the dataset.

5.2 Corpus-level model performance

To address RQ1, we compare the significance of metric score differences on the corpus level. For the metrics which define corpus-level scores as an aggregate of snippet-level scores, it is possible to use techniques such as Wilcoxon sign-rank test [38] to compare the models. However, there are metrics like BLEU which are corpus-level by design, so that simple averaging of per-snippet scores over the corpus does not give corpus-level metric score (see appendix A.1 for more details). Thus, Wilcoxon test is not applicable in this case. This restricts us to using randomized significance testing for comparing corpus level scores, which is a common practice in the machine translation community [13]. According to Graham et al., there is little practical difference between using bootstrap, paired bootstrap and approximate randomization to test significance. We choose paired bootstrap resampling to test significance. To test for the statistical significance, we take 1000 bootstrap samples.

5.3 Automated Metric Scores Significance

To address RQ2, we consider the significance of difference in metric scores for various pairs of models. We expect that the significance of difference in metric scores will vary with the difference in scores (so that for a pair of models with BLEU scores 20 and 80 it is more likely one of the models will be better than the other, as compared to the pair of models with BLEU scores 30 and 29.5). Thus we follow Roy et al. [33] and split the pairs of models into the bins according to the difference in the scores. The bin composure ($[0, 1]$, $[1, 2]$ etc.) is slightly different for HearthStone and CoNaLa dataset. It was determined empirically to have similar number of pairs in every bin. We strived to have a comparable number of pairs in every bin in order to have a significant number of pairs in every bin, so that it is possible to draw statistically robust conclusions.

We augment our set of original ML models by the synthetic models built according to the approach of Roy et al. [33]. We build the synthetic models' outputs from the outputs of real models. There are several reasons why we use synthetic models:

(1) There is a relative scarcity of available models. In the best case of the CoNaLa dataset, we only have five models of various quality, which may not provide enough data to assess metrics applicability. The usage of syntactic models allows us to cover a much more diverse range of metric values without training many new models.
(2) Even if there was a great variety of models so that there would be enough data points for proper metric comparison, it would require immense investments in labeling the data. For example, in this research, we study outputs of 85 models in total just for the CoNaLa dataset, with each of the outputs consisting of 472 snippets. If all 85 models were independent, it would require us to have people experienced in Python labeling more than 40,000 snippets. As we deemed it necessary to collect at least three scores for every snippet, such a procedure would be prohibitively hard or expensive.
(3) Improving or worsening the model scores results in a set of synthetic models with the metric and human scores relatively close to each other. This allows us to compare many models with relatively close scores and check the significance of relatively small differences in their metric scores. This is relevant to the practitioners, since the improvements over the state-of-the-art models often come in small increments.

5.3.1 Building Synthetic Models. We create a synthetic model by starting with the outputs of some of the original models and replacing X% of its worst-rated snippets with the best-rated snippet for the problem. The quality of the snippet is assessed according to the human evaluation scores. If the picked snippet is already the best-rated snippet, it is skipped. The reverse procedure is applied for synthetically worsened models. We continue the replacement procedure until X% of snippets is changed or there are no more snippets left to change.

Following Roy et al., we consider eight different proportions for the replacements: replacing 1%, 3%, 5%, 10%, 15%, 20%, 25%, and 30% of the generated snippets. Our replacement proportions are identical to those of Roy et al. with a slight variation: we replace 3% of the dataset instead of 2% replaced by Roy et al. This procedure yields $5 \times 8 \times 2 = 80$ synthetic models for CoNaLa and $2 \times 8 \times 2 = 32$ synthetic models for the HearthStone dataset. Then, we add the original models and deduplicate them by throwing out models with fully identical outputs. This leaves us with 81 models for CoNaLa and 29 models for HearthStone that we use for our analysis in RQ2. We consider all pairwise combinations of the models (both synthetic and original), and do paired differences test for every metric.

5.4 Agreement Between Metrics and Human Evaluation

To address RQ3, we assess the degree of agreement between the human assessment and metric scores on the corpus level. In order to do so, we carry out corpus-level significance tests to check whether the metric and the human prediction agree for every pair of models. Similarly to the previous research question, we utilize both original and synthetic models we used in RQ2.

5.4.1 Bins for Corpus-level Assessment. There are several options for disagreement between human assessors and a metric for a given pair of models A, B:

- When A is better than B according to the metric, but the models are equivalent according to human assessors (Type-I error).
- When models A and B are equivalent according to the metric, but one of the models is better according to human assessors (Type-II error).
- When model A is better than model B according to the metric, but according to the human assessors, model B is better than model A (Type-I error).

We consider all pairwise combinations of the models (both synthetic and original), and do paired differences test for the human and the metric assessments. Using the aggregated human scores as ground truth, we quantify Type-I and Type-II errors of the metric. As we expect that the probability of a metric to make an error for a pair of models depends on the difference of the models' scores, we divide the data on the metric errors into several bins. The "NS" bin corresponds to the cases where the difference in the model scores is insignificant according to a given metric. All errors in this bin are Type-I errors. Other bins correspond to the cases where the difference in the model scores is significant according to the metric. The bin composure for the RQ3 is identical to the one we choose for the RQ2.

5.4.2 Human Evaluation. To get the human assessment of the considered models, we created a survey, in which we asked programmers to evaluate the presented snippets. The snippets were presented one by one and were randomly chosen out of the combined pool of snippets generated by the models and reference snippets. The graders did not know the origin of each snippet. The graders rated the snippets on the scale from 0 to 4, with the following grade descriptions:

0. Snippet is not at all helpful, it is irrelevant to the problem.
1. Snippet is slightly helpful, it contains information relevant to the problem, but it is easier to write the solution from scratch.
2. Snippet is somewhat helpful, it requires significant changes (compared to the size of the snippet), but is still useful.
3. Snippet is helpful, but needs to be slightly changed to solve the problem.
4. Snippet is very helpful, it solves the problem.

The graders did not have to evaluate all snippets in the dataset and could stop at any moment.

5.4.3 The CoNaLa Dataset. For the CoNaLa dataset, there were 2860 snippets to evaluate: 5×472 snippets generated by the models plus 500 reference snippets (for some of the intents dataset contains more than one reference snippet). 16 participants took part in our survey, and on average, we received 4.49 grades per model-generated snippet. Figure 1 shows the distribution of the number of grades. Three of the graders have less than 2 years of experience with Python, six have 2 to 3 years of experience, and seven are programming in Python for 4 or more years.

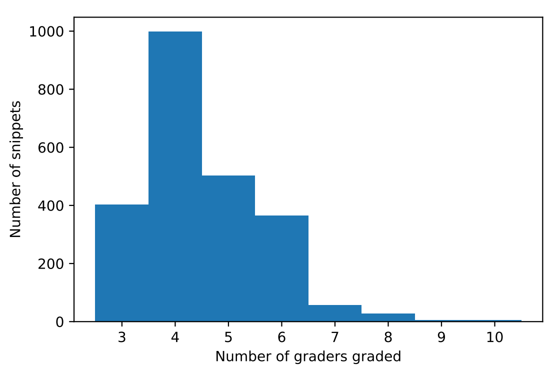

Figure 1: Distribution of the number of grades per snippet

5.4.4 The Hearthstone Dataset. Similarly to the CoNaLa dataset, we also ran a survey in which programmers evaluated the presented snippets. The snippets were presented one by one along with the Hearthstone card images, and the graders assessed whether the snippet represents the card correctly. Figure 2 shows an example of a card image along with the corresponding code snippet.

```
class Archmage(MinionCard ) :
    def __init__ (self) :
        super().__init__("Archmage", 6, CHARACTER_CLASS.ALL,
        CARD_RARITY.COMMON)
    def create_minion (self, player) :
        return Minion(4, 7, spell_damage = 1)
```

Figure 2: The Archmage card and the corresponding code snippet

There were 198 snippets to evaluate: 2×66 snippets generated by the models plus 66 reference snippets. 4 participants took part in the survey, every participant has graded all the snippets. Two of the participants had 3+ years experience of playing Hearthstone, and two other participants have studied the rules through videos and manuals. One of the graders has 1.5 years of experience with Python, two have 2 years of experience, and one is programming in Python for 4 years.

6 RESULTS

6.1 Corpus-level Model Performance

6.1.1 CoNaLa Dataset. The test part of the CoNaLa dataset consists of the 500 reference snippets, but some of the intents appear more than once, so in total, there are 472 unique intents. Different references, corresponding to the same intent, were accounted for as parts of the references corpus. We consider 5 different models trained on the CoNaLa dataset: baseline CoNaLa (`baseline`), tranX trained on the main dataset (`tranx-annot`), best version of tranX by [42] with pretraining and without reranking (`best-tranx`), best version of tranX with pretraining and reranking (`best-tranx-rerank`) [43], and Codex [10]. We compute BLEU, ROUGE-L, METEOR, ChrF, CodeBLEU, and RUBY scores for the outputs of these models (getting scores for each of the test snippets). Table 2 shows metric values for all the models on the CoNaLa dataset.

Alongside the automated metrics, we report the aggregated assessor scores. We convert all the metrics to 0-100 scale by multiplying with an appropriate factor: we multiply assessor scores by 25 and multiply automated metric scores by 100, if the metric scores are in

	baseline	tranx-annot	best-tranx	best-tranx-rerank	Codex
BLEU	$12.37^{+1.59}_{-1.46}$	$28.58^{+3.18}_{-3.06}$	$31.48^{+3.01}_{-2.98}$	$33.14^{+2.91}_{-2.94}$	$33.04^{+3.24}_{-3.14}$
ROUGE-L	$36.51^{+1.41}_{-1.46}$	$49.22^{+1.79}_{-1.69}$	$51.47^{+1.87}_{-1.90}$	$52.83^{+1.96}_{-1.84}$	$56.52^{+2.25}_{-2.29}$
chrF	$17.51^{+1.26}_{-1.26}$	$28.30^{+1.66}_{-1.79}$	$31.14^{+1.89}_{-1.85}$	$32.67^{+2.10}_{-1.95}$	$42.84^{+2.68}_{-2.54}$
METEOR	$28.43^{+1.54}_{-1.54}$	$44.03^{+2.18}_{-2.03}$	$46.55^{+2.28}_{-2.30}$	$48.32^{+2.43}_{-2.38}$	$50.66^{+2.66}_{-2.49}$
RUBY	$43.32^{+1.99}_{-1.84}$	$43.52^{+1.92}_{-1.93}$	$44.81^{+2.00}_{-2.00}$	$46.26^{+2.03}_{-1.97}$	$57.70^{+2.24}_{-2.24}$
CodeBLEU	$30.97^{+1.67}_{-1.47}$	$33.02^{+1.57}_{-1.62}$	$34.07^{+1.67}_{-1.63}$	$34.33^{+1.69}_{-1.65}$	$46.58^{+2.64}_{-2.47}$
Human	$8.74^{+1.80}_{-1.64}$	$26.69^{+3.44}_{-2.91}$	$35.22^{+3.23}_{-3.28}$	$40.10^{+3.70}_{-3.39}$	$59.85^{+3.50}_{-3.50}$

Table 2: Metric results for the CoNaLa dataset.

	Delta: [0, 2)		Delta: [2, 5)		Delta: [5, 10)		Delta: [10, 100)	
	Significant	Not significant	Significant	Not significant	Significant	Not significant	Significant	Not significant
BLEU	192	398	732	42	893	0	1064	0
ROUGE-L	252	296	736	0	1023	0	1014	0
chrF	253	212	633	1	922	0	1300	8
METEOR	195	324	699	7	914	0	1182	48
RUBY	235	437	828	9	972	0	840	0
CodeBLEU	382	474	895	6	857	0	707	0

Table 3: Corpus-level metrics score difference significance on the CoNaLa dataset

$[0, 1]$ span. Together with the scores, we report confidence intervals for each of the metrics. The confidence intervals were computed with the aid of bootstrap over 1000 resamplings; X^{+Y}_{-Z} should be read as "95% of the resampled models yielded score in the $[X - Z, X + Y]$ range".

The BLEU metric failed to recognize the difference in quality between Codex and `best-tranx-rerank`, and between Codex and `best-tranx`. The RUBY metric failed to recognize the difference in quality between any of the following three models: baseline, `tranx-annot`, and `best-tranx`. The CodeBLEU metric failed to recognize the difference in quality between any of the two models from the following ones: `tranx-annot`, `best-tranx`, and `best-tranx-rerank` models.

6.1.2 HearthStone Dataset. For the Hearthstone dataset, we only evaluate two different models available: a syntactic neural model NL2Code [41] and a grammar-based structural convolutional neural network (GCNN) [34]. We compute BLEU, ROUGE-L, METEOR, ChrF, CodeBLEU, and RUBY scores for the outputs of these models, getting scores for each of the test snippets. The format in which we report the scores is the same as the format in which we presented CoNaLa scores. We trained the GCNN model for 30 epochs, as there was no recommended number of epochs in the original paper [34], and the default value of 1000 epochs is unfeasible. This may be the reason why the GCNN model we trained performs relatively worse than NL2Code contrary to the results of the original paper [34].

According to the ROUGE-L, METEOR, and BLEU metrics, the NL2Code model is better than GCNN with $> 95\%$ confidence Table 5.

6.2 Automated Metric Scores Significance

6.2.1 CoNaLa Dataset. In Table 3, we present the data on the significance of differences in model scores. For every pair of models, we compute the difference in their scores according to each of the metrics we consider, and check whether the difference is significant according to the paired bootstrap resampling procedure. We split the possible scores into 4 different bins — $[0, 2)$, $[2, 5)$, $[5, 10)$, $[10, 100)$ — and put every pair of models into the corresponding bin. The results show that with the exception of BLEU metric, if the difference in metric scores of two models is larger than 2 points, then it is possible to claim with at least 95% confidence that the difference is significant. The results also show that if the difference in scores of two models is less than 2 points, it is impossible to claim that one of the models is better without carrying out additional statistical tests. This means that even if metrics emulate human opinion perfectly well, the difference in scores of less than 2 points may still be insignificant and should be reported together with the statistical tests that prove its significance. Moreover, if the difference in BLEU scores is less than 5 points, additional statistical tests are necessary to claim that the difference is significant.

6.2.2 HearthStone Dataset. Table 6 presents the dependence between the difference in model scores according to the metrics and their ability to determine which model is better with at least 95% confidence. The results show, that for the Hearthstone dataset difference in scores of less than 2 points according to any metric makes it impossible to claim that one of the models is significantly better without additional statistical tests. For the adopted by the community BLEU and CodeBLEU metrics — and only for them – it is impossible to claim that one of the models is significantly better

	Delta: [0, 2)		Delta: [2, 5) [a]		Delta: [5, 10) [b]		Delta: [10, 100)		Delta: NS [c]		Total [d]
	Mismatches	Pairs	Mismatches	Pairs	Mismatches	Pairs	Mismatches	Pairs	Mismatches	Pairs	mismatch
BLEU[e]	2.7%	187	15.1%	747	12.0%	890	0.6%	1070	85.5%	427	17.95%
ROUGE-L	6.7%	254	12.0%	740	3.7%	1016	0	1018	72.0%	293	10.69%
chrF	5.2%	248	16.2%	627	2.8%	923	0	1305	64.7%	218	8.49%
METEOR	4.7%	190	14.8%	694	9.3%	914	0	1187	81.5%	336	14.18%
RUBY	6.6%	213	21.0%	837	4.7%	965	0	838	85.9%	468	19.21%
CodeBLEU	6.0%	382	9.4%	896	5.8%	842	0	715	80.9%	486	16.53%

Table 4: Corpus-level metrics disagreement rate on the CoNaLa dataset

	gcnn	nl2code
BLEU	$69.20^{+6.52}_{-6.29}$	$74.52^{+6.20}_{-6.03}$
ROUGE-L	$84.71^{+3.53}_{-3.50}$	$86.54^{+3.05}_{-3.18}$
chrF	$80.76^{+4.21}_{-4.35}$	$80.60^{+3.87}_{-3.78}$
METEOR	$75.18^{+5.72}_{-5.60}$	$79.64^{+5.35}_{-4.98}$
RUBY	$85.82^{+3.71}_{-3.74}$	$85.56^{+3.48}_{-3.69}$
CodeBLEU	$71.59^{+6.24}_{-5.84}$	$72.35^{+5.73}_{-5.57}$
Human	$65.53^{+6.44}_{-6.44}$	$68.18^{+5.68}_{-5.68}$

Table 5: Metric results for the Hearthstone dataset

if the difference in model scores is less than 4 points. Similarly to our results on CoNaLa dataset, this finding highlights that the small difference in the metric scores should be reported together with the statistical tests that prove the significance of the difference.

6.3 Agreement Between Metrics and Human Evaluation

6.3.1 CoNaLa Dataset. We also carry out human evaluation of the CoNaLa dataset and compare it with the results of automated metrics Table 4.

For the corpus-level metrics disagreement rate with the aggregated human scores, we can see the following:

1. The metrics are not reliable in determining that the difference between the models is not significant with the error rate being above 60% for every metric we consider, see column `Delta: NS` c.
2. When the difference in metric scores is less than 5 points, no metric is reliable enough to emulate the human judgement with at least 95% precision.
3. For the $[5, 10)$ metric scores difference bin only RUBY, ChrF and ROUGE-L metrics are able to emulate the human judgement with at least 95% precision, see column `Delta: [5, 10)` b.
4. It is possible to argue that out of the metrics we consider BLEU is the worst in emulating human judgement: even though it has second-highest total mismatch rate, it is the worst-performing metric for the models with score difference more than 5 points, see row `BLEU` e. It is also the only metric that sometimes disagrees with the human judgement for the pair of models that have score difference more than 10 points.
5. RUBY and CodeBLEU metrics, which were developed for assessing code, do not perform significantly better than the metrics originating from the machine translation domain. Moreover, they are among the least reliable in terms of total mismatch rate, see column `Total mismatch` d.
6. All metrics have the highest incidence of Type-I errors for the $[2, 5)$ bin, that then decreases with the increase in scores difference, see column `Delta: [2, 5)` a. This can be explained by the high mismatch rate in the NS bin, which consists of pairs of models with generally small difference in scores. If we do not consider the NS bin separately and aggregate the results according to the difference in pair of models scores, the highest error rate is for the $[0, 2)$ bin, similarly to the results of Roy et al. [33].

The general recommendation for the practitioners based the results of our comparison, is that difference of metric scores of at least 5 points is necessary to claim that one model is better than the other on the CoNaLa dataset with at least 95% certainty, if the human judgement is considered to be the golden truth. The ChrF and ROUGE-L are the best-performing metrics for the assessment of code generation models among the metrics we consider.

6.3.2 HearthStone Dataset. We also conducted the human assessment of the Hearthstone dataset. 4 graders labeled the outputs of the models, with every grader evaluating every snippet. Afterwards, we computed the "ground truth" human grade as it was done by Ma et al. [22].

For the corpus-level metrics disagreement rate with the aggregated human scores, we can see the following

1. The metrics are not reliable in determining that the difference between the models is not significant. The relative error rate, however, is slightly better than the one observed for the CoNaLa dataset: ChrF and ROUGE-L exhibit error rate less than 60%, see column `Delta: NS` g.
2. The total mismatch rate for the HearthStone dataset is worse than the one observed for CoNaLa dataset, see column `Total mismatch` h. The reason for that may be that we only have two models available for the dataset, and their metric scores is relatively close. As all synthetic models were generated from these two, it is not surprising the synthetic models scores are also rather close and it is hard for the metrics to discriminate between models.

	Delta: [0, 1)		Delta: [1, 2)		Delta: [2, 4)		Delta: [4, 100)	
	Significant	Not significant	Significant	Not significant	Significant	Not significant	Significant	Not significant
BLEU	30	91	16	56	98	16	128	0
ROUGE-L	58	138	67	35	137	0	0	0
chrF	71	134	90	33	99	0	8	0
METEOR	21	144	22	33	159	8	48	0
RUBY	60	164	76	73	62	0	0	0
CodeBLEU	31	243	22	102	24	13	0	0

Table 6: Corpus-level metrics score difference significance on the Hearthstone dataset

	Delta: [0, 1)		Delta: [1, 2) [f]		Delta: [2, 4)		Delta: [4, 100)		Delta: NS [g]		Total [h]
	Mismatches	Pairs	Mismatches	Pairs	Mismatches	Pairs	Mismatches	Pairs	Mismatches	Pairs	mismatch
BLEU[i]	3.7%	27	0.0%	16	24.4%	98	27.3%	128	81.9%	166	45.1%
ROUGE-L	1.6%	64	7.7%	65	0.0%	137		0	50.3%	169	20.9%
chrF	1.4%	73	27.2%	92	0.0%	99	0.0%	8	59.5%	163	28.3%
METEOR	5.9%	17	0.0%	22	18.9%	159	22.9%	48	74.6%	189	42.1%
RUBY	2.6%	39	4.5%	110	3.0%	66		0	62.7%	220	33.6%
CodeBLEU	15.2%	33	0.0	22	0.0%	24		0	75.0%	356	62.5%

Table 7: Corpus-level metrics disagreement rate on the HearthStone dataset

3. None of the metrics is reliable enough to discriminate between models with score difference less than 2 points with $> 95\%$ precision, see column `Delta: [1, 2)` f.
4. Once again, BLEU metric performs poorly: its total mismatch rate is among the worst, and, together with METEOR, these are the two only metrics which failed to discriminate well between models with score difference more than 2 points, see row `BLEU` i.
5. RUBY and CodeBLEU, metrics developed for assessing code, do not perform significantly better than the metrics originating from the machine translation domain. Moreover, they are among the worst metrics in terms of total mismatch rate.
6. There is no clear trend for the Type-I error incidence across all the metrics, unlike it is for the CoNaLa dataset. This can be explained by the bin selection that is different from the one chosen for CoNaLa dataset. Unfortunately, the bin selection similar to the one done for CoNaLa dataset would be even less informative: for most of the metrics the bins $[5, 10)$ and $[10, 100)$ would be virtually empty as the two non-synthetic models available for this dataset have relatively close quality.

The general recommendation for the practitioners based on the collected results is that the difference of metric scores of at least 2 points is necessary to claim that one model is better than the other on the Hearthstone dataset with at least 95% certainty, if the human judgement is considered to be the golden truth. The ROUGE-L metric is the best-performing metrics for the assessment of code generation models on this dataset, with ChrF being the second best.

7 THREATS TO VALIDITY

There are several possible threats to validity of our study. First of all, our research is based on two datasets: a dataset of Python one-liners, a peculiar Card2Code [18] dataset, for which the models are supposed to generate classes with very particular structure. It would be interesting to explore other datasets; unfortunately, there is a limited choice of datasets available, and very few models that can be run on a dataset are usually publicly available. The most interesting dataset that was left outside the scope of this paper is the Docstrings dataset [7]. Unfortunately, the models currently available perform on it rather poorly. In particular, the best available model has a BLEU score of 12.1 [35], which means that the expected human grades for it would be rather poor.

The dataset selection issue is closely related to the model selection issue. For every dataset we looked over, except for CoNaLa, there is a relative dearth of available models; in particular, we ran all models that were publicly available for the Hearthstone dataset. We contacted the authors of the models that were not open-sourced, but unfortunately we had no reply. It is possible that different model selection would yield different results.

All the datasets we use have code snippets written in Python. While most of the existing public datasets for code generation indeed have code in Python, generation of code in other languages is an important task and choice of the language might affect the results of a study like ours.

Another possible threat to validity is the small average number of grades available per snippet. It is possible that due to the limited number of developers who have participated in the evaluation, the human grade we derived is different from the "true" human grade a snippet has. Unfortunately, evaluating the examples is very time-consuming, so it is hard to recruit more graders with expertise in Python.

A related issue is biased graders. A grader may have their own preference in coding style or usage of particular technologies that may affect the grade they give to the snippets. Moreover, a "learning effect" might be present: it is possible that after a certain number of assessed snippets, the grader's perception will adjust according to the quality of the presented code. To ameliorate this problem, we shuffled the presented snippets, and added the correct snippets, so that the graders did not know which snippet is correct or not, in order to smear the possible learning effect across the outputs of different models.

8 CONCLUSION

In this work, we study the applicability of various computer metrics: BLEU, ROUGE-L, METEOR, chrF, RUBY, and CodeBLEU for evaluation of the code generation results. We employ 2 different datasets used in code generation, CoNaLa [39], and Hearthstone [18], and evaluate different models on them. Based on the results, we get to the following conclusions:

1. From the metrics point of view, the difference of less than 2 points in the two metrics scores of two models is not enough to claim that one of the models is better than the other with $> 95\%$ confidence without additional statistical tests. BLEU metric requires an even bigger difference: it should be at least 4 points for Hearthstone dataset and at least 5 points for CoNaLa dataset. This means that simply comparing *e.g.*, BLEU scores of two different models may be not enough to find that one model generates code significantly better than the other. We suggest authors of future works on code generation share a detailed models' output on the testing data to make comparison of different models to each other simpler and more precise.
2. The metric scores should be reported together with the data on the significance of the difference in scores. For the metrics that produce summary-level scores that can be aggregated to yield a corpus-level score (every metric except for BLEU, out of the metrics we consider), approaches like paired t-test or Wilcoxon sign-rank test can be recommended. When this is not possible, randomized significance tests such as paired bootstrap resampling or approximate randomization can be considered.
3. The results of human evaluation show that for the CoNaLa dataset of Python one-liners difference in model scores of less than 5 points does not guarantee a difference in human assessment. For the Hearthstone dataset difference of more than 2 points ensures for some metrics that a human will assess one of the models to be significantly better than the other.
4. The results of human evaluation show that ChrF and ROUGE-L metrics are better at emulating human judgement than BLEU or CodeBLEU.

Our work suggests that BLEU, despite being the most popular metric to evaluate quality of the generated code, is less reliable compared to other options. We recommend usage of ChrF as a standard metric for the code generation tasks. It is better correlated with human judgements and is more sensitive. Also, it is arguably easier to use since it is character-based, does not need tokenization and has a reference Python realization in `sacrebleu` package.

Since ChrF does not address the specifics of working with source code, the problem of developing a good automated metric tailored for the code generation task remains open. In order to support the development of such a metric, we make the collected human assessment scores open-source for both studied datasets and encourage other researchers to use them in their work.

REFERENCES

[1] [n.d.]. earthbreaker – open source Hearthstone simulator. https://github.com/danielyule/hearthbreaker. Accessed: 2022-06-06.
[2] [n.d.]. Rouge 1.5.5 Perl script. https://github.com/andersjo/pyrouge/tree/master/tools/ROUGE-1.5.5. Accessed: 2022-05-19.
[3] Rajas Agashe, Srinivasan Iyer, and Luke Zettlemoyer. 2019. Juice: A large scale distantly supervised dataset for open domain context-based code generation. *arXiv preprint arXiv:1910.02216* (2019).
[4] Rajas Agashe, Srinivasan Iyer, and Luke Zettlemoyer. 2019. JuICe: A Large Scale Distantly Supervised Dataset for Open Domain Context-based Code Generation. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Association for Computational Linguistics, Hong Kong, China, 5436–5446. https://doi.org/10.18653/v1/D19-1546
[5] Robert Balzer. 1985. A 15 year perspective on automatic programming. *IEEE Transactions on Software Engineering* 11 (1985), 1257–1268.
[6] Satanjeev Banerjee and Alon Lavie. 2005. METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*. 65–72.
[7] Antonio Valerio Miceli Barone and Rico Sennrich. 2017. A parallel corpus of Python functions and documentation strings for automated code documentation and code generation. *arXiv preprint arXiv:1707.02275* (2017).
[8] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. 1994. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks* 5, 2 (1994), 157–166.
[9] Boxing Chen and Colin Cherry. 2014. A systematic comparison of smoothing techniques for sentence-level bleu. In *Proceedings of the ninth workshop on statistical machine translation*. 362–367.
[10] 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* (2021).
[11] Michael Denkowski and Alon Lavie. 2014. Meteor universal: Language specific translation evaluation for any target language. In *Proceedings of the ninth workshop on statistical machine translation*. 376–380.
[12] Bradley Efron. 1983. Estimating the error rate of a prediction rule: improvement on cross-validation. *Journal of the American statistical association* 78, 382 (1983), 316–331.
[13] Yvette Graham, Nitika Mathur, and Timothy Baldwin. 2014. Randomized significance tests in machine translation. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*. 266–274.
[14] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.
[15] Alexander LeClair and Collin McMillan. 2019. Recommendations for datasets for source code summarization. *arXiv preprint arXiv:1904.02660* (2019).
[16] Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals. 2022. Competition-Level Code Generation with AlphaCode. https://doi.org/10.48550/ARXIV.2203.07814
[17] Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*. 74–81.
[18] Wang Ling, Edward Grefenstette, Karl Moritz Hermann, Tomáš Kočiskỳ, Andrew Senior, Fumin Wang, and Phil Blunsom. 2016. Latent predictor networks for code generation. *arXiv preprint arXiv:1603.06744* (2016).
[19] Chia-Wei Liu, Ryan Lowe, Iulian V Serban, Michael Noseworthy, Laurent Charlin, and Joelle Pineau. 2016. How not to evaluate your dialogue system: An empirical study of unsupervised evaluation metrics for dialogue response generation. *arXiv preprint arXiv:1603.08023* (2016).
[20] Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin Clement, Dawn Drain, Daxin Jiang, Duyu Tang, et al. 2021. CodeXGLUE: A Machine Learning Benchmark Dataset for Code Understanding and Generation. *arXiv preprint arXiv:2102.04664* (2021).
[21] Qingsong Ma, Ondřej Bojar, and Yvette Graham. 2018. Results of the WMT18 metrics shared task: Both characters and embeddings achieve good performance. In *Proceedings of the third conference on machine translation: shared task papers*. 671–688.
[22] Qianqian Ma and Alex Olshevsky. 2020. Adversarial crowdsourcing through robust rank-one matrix completion. *arXiv preprint arXiv:2010.12181* (2020).

[23] Qingsong Ma, Johnny Wei, Ondřej Bojar, and Yvette Graham. 2019. Results of the WMT19 metrics shared task: Segment-level and strong MT systems pose big challenges. In *Proceedings of the Fourth Conference on Machine Translation (Volume 2: Shared Task Papers, Day 1)*. 62–90.
[24] Nitika Mathur, Timothy Baldwin, and Trevor Cohn. 2020. Tangled up in BLEU: Reevaluating the evaluation of automatic machine translation evaluation metrics. *arXiv preprint arXiv:2006.06264* (2020).
[25] Yusuke Oda, Hiroyuki Fudaba, Graham Neubig, Hideaki Hata, Sakriani Sakti, Tomoki Toda, and Satoshi Nakamura. 2015. Learning to generate pseudo-code from source code using statistical machine translation (t). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 574–584.
[26] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*. 311–318.
[27] Maja Popović. 2015. chrF: character n-gram F-score for automatic MT evaluation. In *Proceedings of the Tenth Workshop on Statistical Machine Translation*. 392–395.
[28] Maja Popović. 2016. chrF deconstructed: beta parameters and n-gram weights. In *Proceedings of the First Conference on Machine Translation: Volume 2, Shared Task Papers*. 499–504.
[29] Matt Post. 2018. A call for clarity in reporting BLEU scores. *arXiv preprint arXiv:1804.08771* (2018).
[30] Maxim Rabinovich, Mitchell Stern, and Dan Klein. 2017. Abstract syntax networks for code generation and semantic parsing. *arXiv preprint arXiv:1704.07535* (2017).
[31] Ehud Reiter. 2018. A structured review of the validity of BLEU. *Computational Linguistics* 44, 3 (2018), 393–401.
[32] 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 preprint arXiv:2009.10297* (2020).
[33] Devjeet Roy, Sarah Fakhoury, and Venera Arnaoudova. 2021. Reassessing automatic evaluation metrics for code summarization tasks. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1105–1116.
[34] Zeyu Sun, Qihao Zhu, Lili Mou, Yingfei Xiong, Ge Li, and Lu Zhang. 2019. A grammar-based structural cnn decoder for code generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 7055–7062.
[35] Ngoc Tran, Hieu Tran, Son Nguyen, Hoan Nguyen, and Tien Nguyen. 2019. Does BLEU score work for code migration?. In *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*. IEEE, 165–176.
[36] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems*, I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf
[37] Bolin Wei, Ge Li, Xin Xia, Zhiyi Fu, and Zhi Jin. 2019. Code generation as a dual task of code summarization. In *Advances in Neural Information Processing Systems*. 6563–6573.
[38] Frank Wilcoxon. 1992. Individual comparisons by ranking methods. In *Breakthroughs in statistics*. Springer, 196–202.
[39] Pengcheng Yin, Bowen Deng, Edgar Chen, Bogdan Vasilescu, and Graham Neubig. 2018. Learning to Mine Aligned Code and Natural Language Pairs from Stack Overflow. In *International Conference on Mining Software Repositories (MSR)*. ACM, 476–486. https://doi.org/10.1145/3196398.3196408
[40] Pengcheng Yin, Bowen Deng, Edgar Chen, Bogdan Vasilescu, and Graham Neubig. 2018. Learning to mine aligned code and natural language pairs from stack overflow. In *2018 IEEE/ACM 15th international conference on mining software repositories (MSR)*. IEEE, 476–486.
[41] Pengcheng Yin and Graham Neubig. 2017. A syntactic neural model for general-purpose code generation. *arXiv preprint arXiv:1704.01696* (2017).
[42] Pengcheng Yin and Graham Neubig. 2018. Tranx: A transition-based neural abstract syntax parser for semantic parsing and code generation. *arXiv preprint arXiv:1810.02720* (2018).
[43] Pengcheng Yin and Graham Neubig. 2019. Reranking for neural semantic parsing. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. 4553–4559.
[44] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. *arXiv preprint arXiv:1809.08887* (2018).

A METRICS COMPUTATION

A.1 BLEU

BLEU metric is based on the modified n-gram precision measure with a length penalization for the candidate sentences that are shorter than the reference ones. The BLEU score is determined by the following formula:

$$BLEU = BP \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right); \quad BP = \min(1, e^{1-r/c}), \quad (1)$$

where BP is the brevity penalty with r being the length of the reference and c being the candidate translation length. p_n corresponds to the weighted overlap between the bag of n-grams (repeated terms are allowed up to the maximal number of repeats across the references). If S^n_{ref} and S^n_{can} are bags of n-grams for the reference and the candidate correspondingly, then

$$p_n = \frac{\left|S^n_{ref} \cap S^n_{can}\right|}{\left|S^n_{ref}\right|} \quad (2)$$

Finally, w_n are the weights for various n-gram contributions; the standard weights are $w_1 = \ldots = w_4 = \frac{1}{4}$, $w_{n>4} = 0$. BLEU original realization [26] is a corpus-level metric, as it accounts for the micro-average precision. That is, to compute the precision one has to sum the numerators and the denominators for each hypothesis-reference pair before the division. It is possible to define sentenceBLEU metric to score individual hypothesis (as is done by e.g. Roy et al. [33]) by considering each hypothesis and references as an independent corpus. One, however, has to remember that the average of sentenceBLEU over the whole dataset is not necessarily equal to the BLEU evaluated on the dataset.

BLEU values range from 0 to 1, with higher scores corresponding to better n-grams precision. However, practitioners often multiply BLEU scores by a factor of 100 in their model quality reports. The default implementation of the BLEU metric gives zero score to the candidates which have zero overlap in 4-grams with the reference. This restriction may penalize the candidate sentences of mediocre quality too hard (e.g. for a seven-token reference a candidate that guessed 6 tokens right, but missed the token #4 will get score zero). Several smoothing algorithms have been suggested to avoid these situations, a systematic comparison of smoothing techniques for the sentence-level BLEU for the machine translation task can be found in the paper of Chen et al. [9]

In our study, we use the reference BLEU implementation from `sacrebleu` package [29].

A.2 ROUGE-L

ROUGE-L is a metric from the ROUGE family of metrics first suggested by Lin [17]. It was originally suggested for assessing quality of short text summaries, but then was adopted for other tasks. The basic notion for the ROUGE-L computation is the longest common subsequence (of hypothesis and reference). The common subsequence between two sequences $X = [x_i], Y = [y_j]$ is a sequence $[z_l]$ that is a subsequence of both X, Y. The longest common subsequence is then simply a common subsequence of maximal length. This allows us to define precision, recall and the ROUGE-L metric for hypothesis H and reference R as

$$R_{lcs}(H,R) = \frac{LCS(H,R)}{len(R)}$$
$$P_{lcs}(H,R) = \frac{LCS(H,R)}{len(H)}$$
$$ROUGE_L(H,R) = \frac{(1+\beta^2)P_{lcs}R_{lcs}}{R_{lcs}+\beta^2 P_{lcs}}$$

β is the parameter that determines recall weight, in our evaluation we use $\beta = 1$ (equal weight of precision and recall). The possible values range from 0 to 1, but similarly to BLEU and other metrics the corpus-level scores are often multiplied by 100 to simplify the perception. The ROUGE-L is commonly used as a snippet-level metric [33]. This means that to obtain corpus-level ROUGE-L score one has to average snippet-level scores. For a simple example, let us consider a reference and two hypothesis:

R : police killed the gunman

H_1 : police kill the gunman

H_2 : the gunman killed police

The longest common subsequence between R, H_1 is 3 tokens long (first, third and fourth token), and the longest common subsequence between R, H_2 is 2 tokens long (either first and second, or third and fourth token). Thus $ROUGE_L(H_1, R) = 0.75$, $ROUGE_L(H_2, R) = 0.5$.

We use the implementation of ROUGE-L from the `rouge-score` package, which yields results identical to the original Perl script [2].

A.3 ChrF

ChrF is a F-measure character-based metric first suggested by Popovic [27]. It was originally proposed for automatic evaluation of machine translation output. As a character-based metric, ChrF doesn't depend on tokenization rules. It takes every character into account, except for spaces. To compute ChrF in its standard definition, one has first to compute character-level precision and recall $chrP_k$, $chrR_k$ for character k-grams, where $1 \le k \le 6$. The total n-gram precision and recall $chrP$, $chrR$ is arithmetical average of $chrP_k$, $chrR_k$ respectively. Finally, ChrF is computed as

$$ChrF\beta = \frac{(1+\beta^2)ChrPChrR}{ChrR+\beta^2 ChrP} \tag{3}$$

Standard ChrF definition that we use sets $\beta = 2$, as this choice of β yields the best results in the machine translation tasks [28].

We use the reference implementation of ChrF from the `sacrebleu` package.

A.4 METEOR

METEOR was created as a metric for machine translation evaluation [6]. There are several versions of the metric that have slightly different computation rules. In our computations we have used the latest version of the metric – METEOR 1.5 [11]. Its computation consists of the following steps:

- Creating alignment between the hypothesis and the reference strings. The alignnment between the hypothesis and the reference strings is a mapping between the unigrams of these strings, such that every unigram in each string maps to zero or one unigrams in the other string. The alignment is created in several stages with different rules for the unigram matching in each stage. In the first stage, two words are matched if and only if they are identical. In the second stage, they are matched if they are identical after Porter stemming. In the third stage, two words are matched if they are synonyms according to the WordNet database. Finally, two phrases are matched if they are listed as paraphrases in the corresponding language table. The mappings are applied iteratively, and the final alignment is the largest subset of all matches built using the beam search. To determine the final alignment, the following criteria in order of the importance are applied:
  – The number of covered words across the both sentences should be maximized.
  – The number of *chunks* should be minimized. A *chunk* is a contiguous series of matches that has identical ordering in both sentences.
  – The sum of absolute distances between match start indices in the two sentences should be minimized. This is to break ties by preferring to align phrases that occur at similar positions in both sentences.
- After the alignment has been built, the words in the hypothesis and reference are split into content and function words according to a special function word list. For each of the applied matchers one should count the number of content and function words covered by matches of this type. Then one calculates weighted precision and recall P, R using matcher weights and content-function word weight. From P, R one then computes the weighted harmonic mean F_{mean}. Finally, to penalize gaps and differences in the word order, one computes a fragmentation penalty using the total number of matched words and number of chunks. The METEOR score is finally computed from F_{mean} and fragmentation penalty.

We use the implementation of METEOR from the `sacrerouge` package, which makes use of the original script and provides a Python wrapper for it.

A.5 RUBY

The metric is defined as

$$RUBY(R,C) = \begin{cases} GRS(R,C) & \text{if PDGs are applicable,} \\ TRS(R,C) & \text{if ASTs are applicable,} \\ STS(R,C) & \text{otherwise} \end{cases} \tag{4}$$

Here PDG stands for the program dependence graph and AST stands for the abstract syntax tree, R corresponds to the reference and C corresponds to the candidate. Here PDGs stand for the program dependence graphs and AST stand for the abstract syntax trees, R corresponds to the reference and C corresponds to the candidate. $GRS(R,C)$ measures the similarity between two program dependence graphs for R, C as

$$GRS(R,C) = 1 - \frac{GED(PDG_R, PDG_C)}{size(PDG_R) + size(PDG_C)}, \tag{5}$$

where $GED(PDG_R, PDG_C)$ is the edit distance between PDG of the reference code and PDG of the candidate code. $size(g)$ is the sum of number of vertices and edges of the graph g. $GED(a, b)$ is computed as the minimum number of graph edit operations to transform one graph into another with the allowed graph edit operations on vertexes and edges being insertion, deletion, and substitution.

In the case the PDG is not available for the candidate snippet, the next fallback option is $TRS(R, C)$, which measures the similarity between the ASTs for the reference and the candidate snippet as

$$TRS(R, C) = 1 - \frac{TED(AST_R, AST_C)}{size(AST_R) + size(AST_C)}, \quad (6)$$

where $size(T)$ is the number of nodes in the AST, and $TED(a, b)$ is the edit distance between the ASTs of the reference code AST_R and the candidate code AST_C. TED is given by the minimum number of the editing operations on the AST nodes (that include addition, deletion, replacement and movement) that make AST_R and AST_C identical.

Finally, the last fallback option for RUBY, that can always be computed, is the string similarity function $STS(R, C)$ that is defined as

$$STS(R, C) = 1 - \frac{SED(S_R, S_C)}{max(length(S_R), length(S_C))}, \quad (7)$$

where $SED(S_R, S_C)$ is the string edit distance between the reference sequence S_R and candidate sequence S_C. It measures the number of token deletion/addition actions the user must make to transform the candidate code into the reference one; $length(t)$ is the length of the sequence t. Tran et al. motivate this choice of metric by the observation that the more abstract metrics have better correlation with the human judgement. As Tran et al. do not provide a reference implementation of RUBY, in our study we use our own implementation of the RUBY metric.

A.6 CodeBLEU

The CodeBLEU metric as suggested by Ren et al. [32] is given by

$$CodeBLEU = 0.1 \cdot BLEU + 0.1 \cdot BLEU_w + \quad (8)$$

$$+ 0.4 \cdot Match_{ast} + 0.4 \cdot Match_{df}, \quad (9)$$

where:

- BLEU is the usual BLEU metric.
- $BLEU_w$ is the BLEU metric computed over unigrams only with keywords given 5 times higher weights. In another words, $BLEU_w$ is a precision for unigrams with BLEU brevity penalty. For example, for Python reference `for x in lst` and hypothesis `for x of` $BLEU_w = e^{-1/3}\frac{6}{12}$.
- $Match_{ast}$ is the syntactic AST match. To compute this sub-metric, one first has to build the AST for both reference and hypothesis, and extract all sub-trees from both ASTs. To track the syntactic structure, authors disregard the values in the leave nodes. $Match_{ast}$ is then given by $Match_{ast} = Count_{clip}(T_{hyp})/Count(T_{ref})$, where $Count(T_{ref})$ is the total number of sub-trees in reference AST and $Count_{clip}(T_{cand})$ is the number of sub-trees in hypothesis AST that are matched by sub-trees in the reference.
- $Match_{df}$ is the semantic data-flow match that considers the semantic similarity between the hypothesis and the reference by comparing the data-flow graphs of the reference and the hypothesis. The sub-metric is computed in several steps as follows:
  1. Build the data-flow graph for the reference and the hypothesis. To do that, one first has to get the variable sequence $V = \{v_0, v_1, \ldots, v_m\}$ from the AST. Each variable then becomes a node of the data-flow graph, and directed edges $\epsilon = \langle v_i, v_j \rangle$ signify that the value of j-th variable comes from the i-th variable. The graph $G = (V, E)$ is the data-flow graph.
  2. Normalize data-flow items. To do that, one has to collect all the variables in the data-flow items and rename them var_i, where i is the order of the variable appearance in all data-flow items.
  3. Calculate the semantic data-flow match score as $Match_{df} = Count_{clip}(DF_{hyp})/Count(DF_{ref})$, where $Count(DF_{ref})$ is the total number of the reference data-flows and $Count_{clip}(DF_{cand})$ is the number of the matched candidate data-flows.

Ren et al. compared CodeBLEU with BLEU and accuracy. As CodeBLEU was not compared to other automatic metrics apart from BLEU and accuracy, we need to carry out further assessment. In our study, we use our own implementation of the CodeBLEU metric.