Code to Comment "Translation": Data, Metrics, Baselining & Evaluation

David Gros*, Hariharan Sezhiyan*, Prem Devanbu, Zhou Yu
University of California, Davis
{dgros,hsezhiyan,devanbu,joyu}@ucdavis.edu

ABSTRACT

The relationship of comments to code, and in particular, the task of generating useful comments given the code, has long been of interest. The earliest approaches have been based on strong syntactic theories of comment-structures, and relied on textual templates. More recently, researchers have applied deep-learning methods to this task—specifically, trainable generative translation models which are known to work very well for Natural Language translation (*e.g.,* from German to English). We carefully examine the underlying assumption here: that the task of generating comments sufficiently resembles the task of translating between natural languages, and so similar models and evaluation metrics could be used. We analyze several recent code-comment datasets for this task: CodeNN, DeepCom, FunCom, and DocString. We compare them with WMT19, a standard dataset frequently used to train state-of-the-art natural language translators. We found some interesting differences between the code-comment data and the WMT19 natural language data. Next, we describe and conduct some studies to calibrate BLEU (which is commonly used as a measure of comment quality). using "affinity pairs" of methods, from different projects, in the same project, in the same class, *etc*; Our study suggests that the current performance on some datasets might need to be improved substantially. We also argue that fairly naive information retrieval (IR) methods do well enough at this task to be considered a reasonable baseline. Finally, we make some suggestions on how our findings might be used in future research in this area.

ACM Reference Format:
David Gros*, Hariharan Sezhiyan*, Prem Devanbu, Zhou Yu. 2020. Code to Comment "Translation": Data, Metrics, Baselining & Evaluation. In *Proceedings of ACM Conference (ASE '20).* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Programmers add comments to code to help comprehension. The value of these comments is well understood and accepted. A wide variety of comments exist [42] in code, including prefix comments (standardized in frameworks like Javadocs [31]) which are inserted before functions or methods or modules, to describe their function. Given the value of comments, and the effort required to write them, there has been considerable interest in providing automated assistance to help developers to produce comments, and a variety of approaches have been proposed [38, 47, 48, 59]. [1]

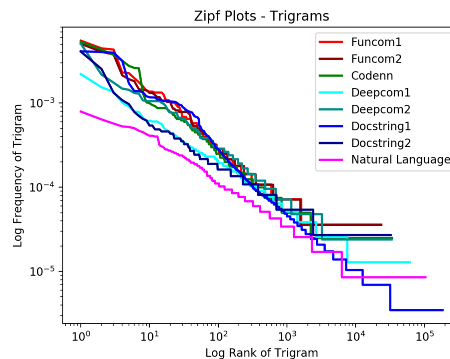

Figure 1: Distribution of trigrams in English (blue) in the WMT [10] German-English machine-translation dataset, and in English comments from several previously published Code-Comment datasets

Comments (especially prefix comments) are typically expected to be a useful summary of the function of the accompanying code. Comments could be viewed as a restatement of the semantics of the code, in a different and more accessible natural language; thus, it is possible to view comment generation as a kind of translation task, translating from one (programming) language to a another (natural) language. This view, together with the very large volumes of code (with accompanying comments) available in open-source projects, offers the very appealing possibility of leveraging decades of research in statistical natural language translation (NLT). If it's possible to learn to translate from one language to another from data, why not learn to synthesize comments from code? Several recent papers [22, 26, 33, 61] have explored the idea of applying Statistical Machine Translation (SMT) methods to *learn* to translate code to an English comments. But are these tasks really similar? We are interested to understand in more detail how similar the task of generating comments from code is to the task of translating between natural languages.

Comments form a domain-specific dialect, which is highly structured, with a lot of very repetitive templates. Comments often begin with patterns like *"returns the"*, *"outputs the"*, and *"calculates the"*. Indeed, most of the earlier work (which wasn't based on machine

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ASE '20, Sept 2020, Melbourne, Australia
© 2020 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

[1] * Authors contributed equally

learning) on this problem has leveraged this highly templated nature of comments [40, 48]. We can see this phenomenon clearly using Zipf plots. Figure 1 compares the trigram frequencies of the English language text in comments (from the datasets [22, 26, 33] that have been used to train deep-learning models for code-comment summarization) and English language text in the WMT German-English translation dataset: the x-axis orders the trigrams from most to least frequent using a log-rank scale, and the y-axis is the log relative frequency of the trigrams in the corpus. The English found in WMT dataset is the magenta line at the bottom. The comments from code show consistently higher slope in the (note, *log-scaled*) y-axis of the Zipf plot, suggesting that comments are far more saturated with repeating trigrams than is the English found in the translation datasets. This observation motivates a closer examination of the differences between code-comment and WMT datasets, and the implications of using machine translation approaches for code-comment generation.

In this paper, we compare code-comment translation (CCT) datasets used with DL models for the task of comment generation, with a popular natural translation (WMT) dataset used for training DL models for natural language translation. These were our results:

(1) We find that the desired outputs for the CCT task are much more repetitive.
(2) We find that the repetitiveness has a very strong effect on measured performance, much more so in the CCT datasets than the WMT dataset.
(3) We find that the WMT translation dataset has a smoother, more robust input-output dependency. Similar German inputs in WMT have a strong tendency to produce similar English outputs. However, this does appear to hold in the CCT datasets.
(4) We report that a naive Information retrieval approach can meet or exceed reported numbers from neural models.
(5) We evaluate BLEU *per se* as a measure of generated comment quality using groups of methods of varying "affinity"; this offers new perspectives on the BLEU measure.

Our findings have several implications for the future work in the area, in terms of technical approaches, ways of measurement, for baselining, and for calibrating BLEU scores. We begin below by first providing some background; we then describe the datasets used in prior work. We then present an analysis of the datasets and and an analysis of the evaluation metrics and baselines used. We conclude after a detailed discussion of the implications of this work.

But first, a disclaimer: this work does not offer any new models for or improvements on prior results on the CCT task. It is primarily retrospective, *viz*, a critical review of materials & evaluations used in prior work in CCT, offered in a collegial spirit, hoping to advance the way our community views the task of code-comment translation, and how we might together make further advances in the measurement and evaluation of innovations that are addressed in this task.

2 BACKGROUND & THEORY

The value of comments in code comprehension has been well-established [51]. However, developers find it challenging to create & maintain useful comments [17, 19]. This has sparked a long line of research looking into the problem of comment generation. An early line of work [11, 40, 48, 49] was rule-based, combining some form analysis of the source code to extract specific information, which could then be slotted into different types of templates to produce comments. Another approach was to use code-clone identification to produce comments for given code, using the comments associated with a clone [59]. Other approaches used keywords which programmers seem to attend to in eye-tracking studies [47]. Still other approaches use topic analysis to organize descriptions of code [37].

Most of the pioneeering approaches above relied on specific features and rules hand-engineered for the task of comment generation. The advent of large open-source repositories with large volumes of source-code offered a novel, general, statistically rigorous, possibility: that these large datasets be mined for code-comment pairs, which could then be used to train a model to produce comments from code. The success of classic statistical machine translation [30] offered a tempting preview of this: using large amounts of aligned pairs of utterances in languages A & B, it was possible to learn a conditional distribution of the form $p_t(b \mid a)$, where $a \in A$, and $b \in B$; given an utterance $\beta \in B$, one could produce a possible translation $\alpha \in A$ by simply setting

$$\alpha = \underset{a}{\operatorname{argmax}}\, p_t(a \mid \beta)$$

Statistical natural language translation approaches, which were already highly performant, were further enhanced by deep-learning (DL). Rather than relying on specific inductive biases like phrase-structures in the case of classical SMT, DL held the promise that the features relevant to translation could themselves be learned from large volumes of data. DL approaches have led to phenomenal improvements in translation quality [29, 52]. Several recent papers [24, 26, 33] have explored using these powerful DL approaches to the code-comment task.

Iyer *et al.* [26] first applied DL to this task, using code-English pairs mined from Stack Overflow—using simple attention over input code, and an LSTM to generate outputs. Many other papers followed, which are discussed below in section 3.2. We analyze the published literature, starting with the question of whether there are notable distributional differences between the code-comment translation (CCT) and the statistical machine translation (WMT) data. Our studies examine the distributions of the input and output data, and the dependence of the output on the input.

RQ1. What are the differences between the translation (WMT) data, and code-comment (CCT) data?

Next, we examine whether these differences actually affect the performance of translation models. In earlier work, Allamanis [3] pointed out the effects of data duplication on machine learning applications in software engineering. We study the effects of data duplication, as well as the effects of distributional differences on deep learning models. One important aspect of SMT datasets is *input-ouput dependence*. In translation *e.g.* from German (DE) to English (EN), similar input DE sentences will to produce similar output EN sentences, and less similar DE sentences will tend to

produce less similar EN sentences. This same correlation might not apply in CCT datasets.

RQ2. How the distributional differences in the SMT & CCT datasets affect the measured performance?

There's another important difference between code and natural language. Small differences, such as substituting $*$ for $+$ and a 1 for a 0, can make the difference between a *sum* and a *factorial* function; likewise changing one function identifier (*mean*, rather than *variance*). These small changes should result in a large change in the associated comment. Likewise, there are many different ways to write a sort function, all of which might entail the same comment. Intuitively, this would appear to be less of an issue in natural languages; since as they have evolved for consequential communication in noisy environments, meaning should be robust to small changes. Thus on the whole, we might expect that small changes in German should in general result in only small changes in the English translation. Code, on the other hand, being a *fiat* language, might not be in general as robust, and so small changes in code may result in unpredictable changes in the associated comment. Why does this matter? In general, modern machine translation methods use the generalized function-approximation capability of deep-learning models. If natural language translation (WMT) has a more functional dependency, and CCT doesn't, there is a suggestion that deep-learning models would find CCT a greater challenge.

RQ3. Do similar inputs produce similar outputs in both WMT and CCT datasets?

Prior work in natural language generation has shown that information retrieval (IR) methods can be effective ways of producing suitable outputs. These methods match a new input with semantically similar inputs in the training data, and return the associated output. These approaches can sometimes perform quite well [21] and has been previously applied successfully to the task of comment generation [14, 62]. Our goal here is to ask whether IR methods could be a relevant, useful baseline for CCT tasks.

RQ4. How do the performance of naive Information Retrieval (IR) methods compare across WMT & CCT datasets?

Finally, we critically evaluate the use of BLEU scores in this task. Given the differences we found between datasets used for training SMT translators and the code-comment datasets, we felt it would be important to understand how BLEU is used in this task, and develop some empirical baselines to calibrate the observed BLEU values in prior work. How good are the best-in-class BLEU scores (associated with the best current methods for generating comments given the source of a method)? Are they only as good as simply retrieving a comment associated with a random method in a different project? Hopefully they're much better. How about the comment associated with a random method from the same project? With a random method in the same class? With a method that could reasonably be assumed quite similar?

RQ5. How has BLEU been used in prior work for the code-comment task, and how should we view the measured performance?

In the next section, we review the datasets that we use in our study.

3 DATASETS USED

We examine the characteristics of four CCT data sets, namely CodeNN, DeepCom, FunCom, & DocString and one standard, widely-used machine-translation dataset, the WMT dataset. We begin with a description of each dataset. Within some of the CCT datasets, we observe that the more popular ones can include several different variations: this is because follow-on work has sometimes gathered, processed, and partitioned (training/validation/test) the dataset differently.

CodeNN Iyer *et al* [26] was an early CCT dataset, collected from StackOverflow, with code-comment pairs for C# and SQL. Stackoverflow posts consist of a title, a question, and a set of answers which may contain code snippets. Each pair consists of the *title* and *code snippet* from answers. Iyer *et al* gathered around a million pairs each for C# and SQL; from these, focusing on just snippets in *accepted* answers, they filtered down to 145,841 pairs for C# and 41,340 pairs for SQL. From these, they used a trained model (trained using a hand-labeled set) to filter out uninformative titles (*e.g.,* "How can make this complicated query simpler") to 66,015 higher-quality pairs for C# and 33,237 for SQL. In our analysis, we used only the C# data. StackOverflow has a well-known community norm to avoid redundant Q&A; repeated questions are typically referred to the earlier post. As a result, this dataset has *significantly less duplication*. The other CCT datasets are different.

DeepCom Hu et al. [22] generate a CCT dataset by mining 9,714 Java projects. From this dataset, they filter out methods that have Javadoc comments, and select only those that have at least one-word descriptions. They also exclude getters, setters, constructors and test methods. This leaves them with 69,708 method-comment pairs. In this dataset, the methods (code) are represented as serialized ASTs after parsing by Eclipse JDT.

Later, Hu et al. [23] updated their dataset and model, to a size of 588,108 examples. We refer to the former as DeepCom1 and obtain a copy online from followup work[2]. We refer to the latter as DeepCom2 and obtain a copy online[3]. In addition DeepCom2 is distributed with a 10-fold split in the cross-project setting (examples in the test set are from different projects). In Hu et al. [23] this is referred to the "RQ-4 split", but to avoid confusion with our research questions, we refer to it as DeepCom2f.

Funcom LeClair et al. [33] started with the Sourcerer [7] repo, with over 51M methods from 50K projects. From this, they filtered out methods with Javadoc comments in English, and then also the comments that were auto-generated. This leaves about 2.1M methods with patched Javadoc comments. The source code was parsed into an AST. They created two datasets, the *standard*, which retained the original identifiers, and *challenge*, wherein the identifiers (except for Java API class names) were replaced with a standardized token. They also made sure no data from the same project was duplicated across training and/or validation and/or test. Notably, the FunCom

[2] https://github.com/wasiahmad/NeuralCodeSum/tree/d563e58/data
[3] https://github.com/xing-hu/EMSE-DeepCom/tree/98bd6a

dataset only considers the first sentence of the comment. Additionally, code longer than 100 words and comments longer 13 words were truncated.

Like for DeepCom, there are several versions of this dataset. We consider a version from LeClair et al. [33] as FunCom1 and the version from LeClair and McMillan [34] as FunCom2. These datasets are nearly identical, but FunCom2 has about 800 fewer examples and the two versions have reshuffled train/test/val splits. The Funcom1[4] and Funcom2[5] datasets are available online.

Docstring Barone and Sennrich [8] collect Python methods and prefix comment "docstrings" by scraping GitHub. Tokenization was done using subword tokenization. They filtered the data for duplications, and also removed excessively long examples (greater than 400 tokens). However, unlike other datasets, Barone *et al.* do not limit to only the first sentence of the comments. This can result in relatively long desired outputs.

The dataset contains approximately 100k examples, but after filtering out very long samples, as per Barone *et al* preprocessing script[6], this is reduced to 74,860 examples. We refer to this version as DocString1.

We also consider a processed version obtained from Ahmad et al. [2] source[2] which was attributed to Wei et al. [58]. We refer to this version as DocString2. Due to the processing choices, the examples in DocString2 are significantly shorter than DocString1.

WMT19 News Dataset To benchmark the comment data with natural language, we used data from the Fourth Conference of Machine Translation (WMT19). In particular, we used the news dataset [9]. After manual inspection, we determined this dataset offers a good balance of formal language that is somewhat domain specific to more loose language common in everyday speech. In benchmarking comment data with natural language, we wanted to ensure variety in the words and expressions used to avoid biasing results. We used the English-German translation dataset, and compared English in this dataset to comments in the other datasets (which were all in English) to ensure differences in metrics were not a result of differences in language.

Other CCT Datasets We tried to capture most of the code-comment datasets that are used in the context of translation. However, there are some recent datasets which could be used in this context, but we did not explore [1, 25]. While doing our work we noticed that some prior works provide the raw collection of code-comments for download, but not the exact processing and evaluations used [39]. Other works use published datasets like DocString, but processing and evaluation techniques are not now readily available [56, 57]. As we will discuss, unless the precise processing and evaluation code is available, the results may be difficult to compare.

3.1 Evaluation Scores Used

A common metric used in evaluating text generation is BLEU score [43]. When comparing translations of natural language, BLEU score has been shown to correlate well with human judgements of translation quality [16]. In all the datasets we analyzed, the associated papers used BLEU to evaluate the quality of the comment generation. However, there are rather subtle differences in the way the BLEUs were calculated, which makes the results rather difficult to compare. We begin this discussion with a brief explanation of the BLEU score.

BLEU (as do related measures) indicates the closeness of a candidate translation output to a "golden" reference result. BLEU *per se* measures the *precision* (as opposed to *recall*) of a candidate, relative to the reference, using constituent n-grams. BLEU typically uses unigrams through 4-grams to measure the precision of the system output. If we define *:*

$$p_n = \frac{\text{number of } n\text{-grams in both reference and candidate}}{\text{number of } n\text{-grams in the candidate}}$$

BLEU combines the precision of each n-gram using the geometric mean, $\exp(\frac{1}{N}\sum_{n=1}^{N}\log p_n)$. With just this formulation, single word outputs or outputs that repeat common n-grams could potentially have high precision. Thus, a *"brevity penalty"* is used to scale the final score; furthermore each n-gram in the reference can be used in the calculation just once. [18] These calculations are generally standard in all BLEU implementations, but several variations may arise.

Smoothing: One variation arises when deciding how to deal with cases when $p_n = 0$, *i.e.*, an n-gram in the candidate string is not in the reference string [12]. With no adjustment, one has an undefined $\log 0$. One can add a small epsilon to p_n which removes undefined expressions. However, because BLEU is a geometric mean of $p_n, n \in \{1, 2, 3, 4\}$ if p_4 is only epsilon above zero, it will result in a mean which is near zero. Thus, some implementations opt to smooth the p_n in varying ways. To compare competing tools for the same task, it would be preferable to use a standard measure.

Corpus vs. Sentence BLEU: When evaluating a translation system, one typically measures BLEU (candidate *vs* reference) across all the samples in the held-out test set. Thus another source of implementation variation is when deciding how to combine the results between all of the test set scores. One option, which was proposed originally in Papineni *et al.* [43], is a "corpus BLEU", sometimes referred to as C-BLEU. In this case the numerator and denominator of p_n are accumulated across every example in the test corpus. This means as long as at least one example has a 4-gram overlap, p_4 will not be zero, and thus the geometric mean will not be zero An alternative option for combining across the test corpus is referred to as "Sentence BLEU" or S-BLEU. In this setting BLEU score for the test set is calculated by simply taking the arithmetic mean the BLEU score calculated on each sentence in the set.

Tokenization Choices: A final source of variation comes not from how the metric is calculated, but from the inputs it is given. Because the precision counts are at a token level, it has been noted that BLEU is highly sensitive to tokenization [44]. This means that when comparing to prior work on a dataset, one must be careful not only to use the same BLEU calculation, but also the same tokenization and filtering. When calculating scores on the datasets, we use the tokenization provided with the dataset.

Tokenization can be very significant for the resulting score. As a toy example, suppose a reference contained the string "`calls function foo()`" and an hypothesis contained the string "`uses`

[4]http://leclair.tech/data/funcom/
[5]https://s3.us-east-2.amazonaws.com/icse2018/index.html
[6]https://bit.ly/2yDnHcS

`function foo()`". If one chooses to tokenize by spaces, one has tokens [`calls`, `function`, `foo()`] and [`uses`, `function`, `foo()`]. This tokenization yields only one bigram overlap and no trigram or 4-gram overlaps. However, if one instead chooses to tokenize this as [`calls`, `function`, `foo`, `(`, `)`] and [`uses`, `function`, `foo`, `(`, `)`] we suddenly have three overlapping bigrams, two overlapping trigrams, and one overlapping 4-gram. This results in a swing of more than 15 BLEU-M2 points or nearly 40 BLEU-DC points (BLEU-M2 and BLEU-DC described below).

We now go through BLEU variants used by each of the datasets and assign a name to them. The name is not intended to be prescriptive or standard, but instead just for later reference in this document. All scores are the "aggregate" measures, which consider up to *4*-grams.

BLEU-CN This is a Sentence BLEU metric. It applies a Laplace-like smoothing by adding 1 to both the numerator and denominator of p_n for $n \geq 2$. The CodeNN authors' implementation was used [7].

BLEU-DC This is also a Sentence BLEU metric. The authors' implementation is based off NLTK [36] using its "method 4" smoothing. This smoothing is more complex. It only applies when p_n is zero, and sets $p_n = 1/((n-1) + 5/\log l_h)$ where l_h is the length of the hypothesis. See the authors' implementation for complete details[8].

BLEU-FC This is an unsmoothed corpus BLEU metric based on NLTK's implementation. Details are omitted for brevity, and can be found in the authors' source[9].

BLEU-Moses The Docstring dataset uses a BLEU implementation by the Moses project[10]. It is also an unsmoothed corpus BLEU. This is very similar to BLEU-FC (though note that due to differences in tokenization, scores presented by the two datasets are not directly comparable).

BLEU-ncs This is a sentence BLEU used in the implementation[11] of Ahmad et al. [2]. Like BLEU-CN, it uses an add-one Laplace smoothing. However, it is subtly different than BLEU-CN as the add-one applies even for unigrams.

SacreBLEU The SacreBLEU implementation was created by Post [44] in an effort to help provide a standard BLEU implementation for evaluating on NL translation. We use the default settings which is a corpus BLEU metric with an exponential smoothing.

BLEU-M2 This is a Sentence BLEU metric based on nltk "method 2" smoothing. Like BLEU-CN it uses a laplace-like add-one smoothing. This BLEU is later presented in plots for this paper.

We conclude by noting that the wide variety of BLEU measures used in prior work in code-comment translation carry some risks. We discuss further below. table 3 provide some evidence suggesting that the variation is high enough to raise some concern about the true interpretation of claimed advances; as we argue below, the field can benefit from further standardization.

[7] https://github.com/sriniiyer/codenn/blob/0f7fbb8b298a8/src/utils/bleu.py
[8] https://github.com/xing-hu/EMSE-DeepCom/blob/98bd6aac/scripts/evaluation.py
[9] https://github.com/mcmillco/funcom/blob/41c737903/bleu.py#L17
[10] https://bit.ly/2YF0hye
[11] https://github.com/wasiahmad/NeuralCodeSum/blob/b2652e2/main/test.py#L324

3.2 Models & Techniques

In this section, we outline the various deep learning approaches that have been applied to this code-comment task. We note that our goal in this paper is not to critique or improve upon the specific technical methods, but to analyze the data *per se* to gain some insights on the distributions therein, and also to understand the most comment metric (BLEU) that is used, and the implications of using this metric. However, for completeness, we list the different approaches, and provide just a very brief overview of each technical approach. All the datasets used below are described above in section 3.

Iyer *et al* [26] was an early attempt at this task, using a fairly standard seq2seq RNN model, enhanced with attention. Hu *et al* [22] also used a similar RNN-based seq2seq model, but introduced a "tree-like" preprocessing of the input source code. Rather than simply streaming in the raw tokens, they first parse it, and then serialize the resulting AST into a token stream that is fed into the seq2seq model. A related approach [5] digests a fixed-size random sample of paths through the AST of the input code (using LSTMs) and produces code summaries. LeClair *et al* [33] proposed an approach that combines both structural and sequential representations of code; they have also suggested the use of graph neural networks [32]. Wan *et al* [54] use a similar approach, but advocate using reinforcement learning to enhance the generation element. More recently, the use of function context [20] has been reported to improve comment synthesis. Source-code vocabulary proliferation is a well-known problem [28]; previously unseen identifier or method names in input code or output comments can diminish performance. New work by Moore *et al* [39] approaches this problem by using convolutions over individual *letters* in the input and using subtokens (by camel-case splitting) on the output. Very recently Zhang et al. [62] have reported that combining sophisticated IR methods with deep-learning leads to further gains in the CCT task. For our purposes (showing that IR methods constitute a reasonable baseline) we use a very simple, vanilla, out-of-box Lucene IR implementation, which already achieves nearly SOTA performance in many cases.

There are tasks related to generating comments from code: for example, synthesizing a commit log given a code change [15, 27, 35], or generating method names from the code [4, 5]. Since these are somewhat different tasks, with different data characteristics, we don't discuss them further. In addition code synthesis [1, 60] also uses matched pairs of natural language and code; however, these datasets have not been used for generating English from code, and are not used in prior work for this task; so we don't discuss them further here.

4 METHODS & FINDINGS

In the following section, we present our methods and results for each of the RQs presented in § 2. In each case, we present some illustrative plots and (when applicable) the results of relevant statistical tests. All p-values have been corrected using family-wise (Benjamini-Hochberg) correction. To examine the characteristics of each dataset, we constructed two types of plots: zipf plots and bivariate BLEU plots.

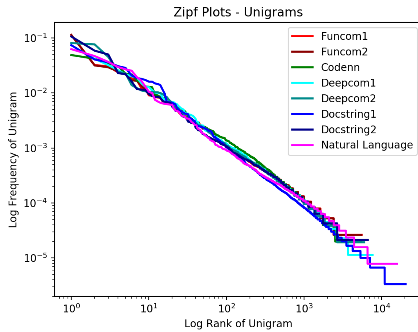

Figure 2: Unigram (Vocabulary) distribution. Zipf plot for all datasets look similar. Difference from trigram Zipf plot in Fig 1 suggests greater repetitiveness in code comments.

4.1 Differences between CCT and WMT data

The Zipf plots are a useful way to visualize the skewness of textual data, where (in natural text) a few tokens (or ngrams) account for a large portion of the text. Each plot point is a (rank, relative-frequency) pair, both log-scaled. We use the plot to compare the relative skewness of the (English) comment data in the CCT data and the desired English outputs in the WMT NLT data. Examining the unigram Zipf plot above, it can be seen in both code comments and natural English, a few vocabulary words do dominate. However, when we turn back to the trigram Zipf plots in Figure 1, we can see the difference. One is left with the clear suggestion that: while the vocabulary distributions across the different datasets aren't that different, the ways in which these vocabulary words are *combined into trigrams* are much more stylistic and templated in code comments.

> **Result 1:** *Code comments are far more repetitive than the English found in Natural Language Translation datasets*

Given this relatively greater repetitive structure in code comments, we can expect that the performance of translation tools will be strongly influenced by repeating (and/or very frequent) trigrams. If a few frequent *n*-grams account for most of the desired output in a corpus, it would seem that these trigrams would play a substantial, perhaps misleading role in measured performance. Figure 3 supports this analysis. The right-hand side plot shows the effect on BLEU-4 of replacing single words (unigrams) with random tokens in the "Golden" (desired) output in the various datasets. The left-hand plot shows the effect of replacing trigrams. The index (1 to 100) on the x-axis shows the number of most frequent n-grams replaced with random tokens. The y-axis shows the decrease in measured BLEU-4 as the code is increasingly randomized.

The Unigrams plot suggests that the effect on the desired Natural language ("nl") output, as measured by BLEU is relatively greater when compared to most of the comment datasets. This effect is reversed for Trigrams; the "nl" dataset is not affected as much by the removal of frequent Trigrams as the comment datasets. This analysis suggests that a tool that got the top few most frequent trigrams wrong in the code-comment generation task would suffer a larger performance penalty than a tool that got the top-few *n*-grams wrong in a natural language translation task. This visual evidence is strongly confirmed by rigorous statistical modeling, please see supplementary materials, bleu-cc.Rmd for the R code. Frequent trigrams that have a big effect on the code comment BLEU include *e.g.,* "`factory method for`", "`delegates to the`", and "`method for instantiating`". To put it another way, one could boost the performance of a code-comment translation tool, perhaps misleadingly, by getting a few such *n*-grams right.

> **Result 2:** *Frequent n-grams could wield a much stronger effect on the measured BLEU performance on code-comment translation tasks than on natural language translation*

4.2 Input-Output Similarity (RQ3)

An important property of natural language translation is that there is a general *dependence of input on output*. Thus, similar German sentences should translate to similar English sentences. For example two German sentences with similar grammatical structure and vocabulary should in general result in two English sentences whose grammatical structure and vocabulary resemble each other; likewise, in general, the more different two German sentences are in vocabulary and grammar, the more difference we expect in their English translations. Exceptions are possible, since some similar constructions have different meanings: (*kicking the ball* vs. *kicking the bucket*[12]). However, on average in large datasets, we should expect that more similar sentences give more similar translations.

When training a translation engine with a high-dimensional non-linear function approximator like an encoder-decoder model using deep learning, this monotonic dependence property is arguably useful. We would expect similar input sentences to be encode into similar points in vector space, thus yielding more similar output sentences. How do natural language translation (German-English) and code-comment datasets fare in this regard? To gauge this phenomenon, we sampled 10,000 *random pairs of input fragments* from each of our datasets, and measured their similarity using BLEU-M2, as well as the similarity of the corresponding Golden (desired) output fragments. We then plot the *input* BLEU-M2 similarity for each sampled pair on the x-axis, and the BLEU-M2 similarity of the corresponding pair of *outputs* on the y-axis. We use a kernel-smoothed 2-d histogram rather than a scatter plot, to make the frequencies more visible, along with (we expect) an indication suggesting that similar inputs yield similar outputs. Certainly, most inputs and outputs are different, so we expect to find a large number of highly dissimilar pairs where input and output pair BLEUs are virtually zero. So we considered our random samples, with and without input and output similarities bigger than $\epsilon = 10^{-5}$. The highly dissimilar input-output pairs are omitted in the plot. However, as an additional quantitative test of correlation, we also considered Spearman's ρ. both with and without the dissimilar pairs.

The bivariate plots are shown in Fig 4 and the Spearmans ρ in table 2. We have split the 10,000 random samples into two cases:

[12]The latter idiom indicates death; Some automated translation engines (*e.g.* Bing) seem to know the difference when translating from English

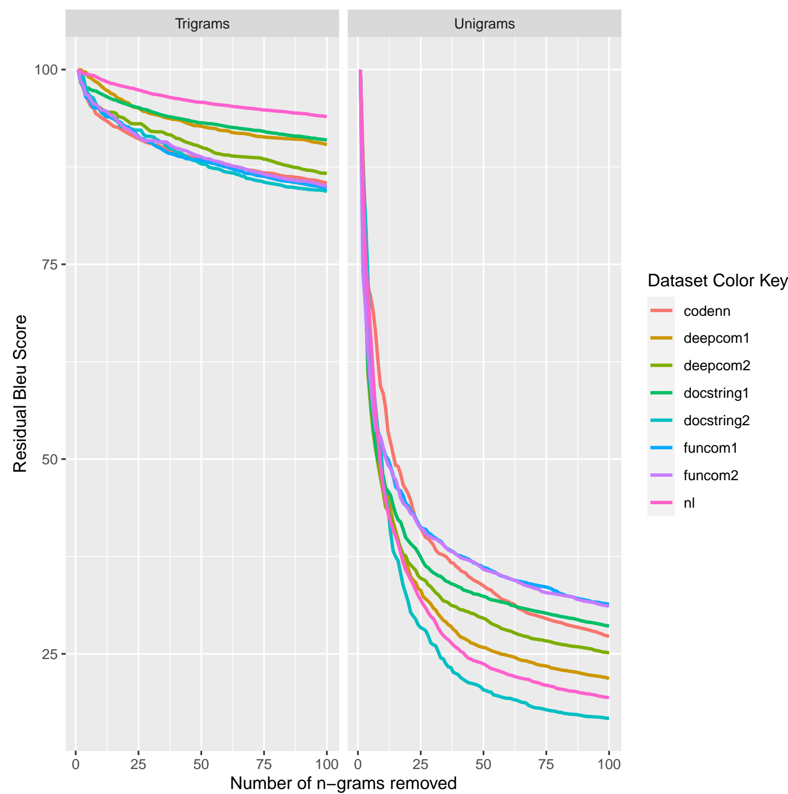

Figure 3: The effect of most frequent unigrams (left) and trigrams (right) on measured (smoothed) BLEU-4 performance. BLEU-4 was calculated after successive removals of of most frequent unigrams (right) and trigrams (left). The effect of removing frequent *Unigrams* is by and large greater on the natural language dataset ("nl"). However, the effect of removing frequent *trigrams*, on the comment datasets is generally stronger than on the "nl" dataset due to high degree of repetition in the comment datasets. These apparent visual differences are decisively confirmed by more rigorous statistical modeling.

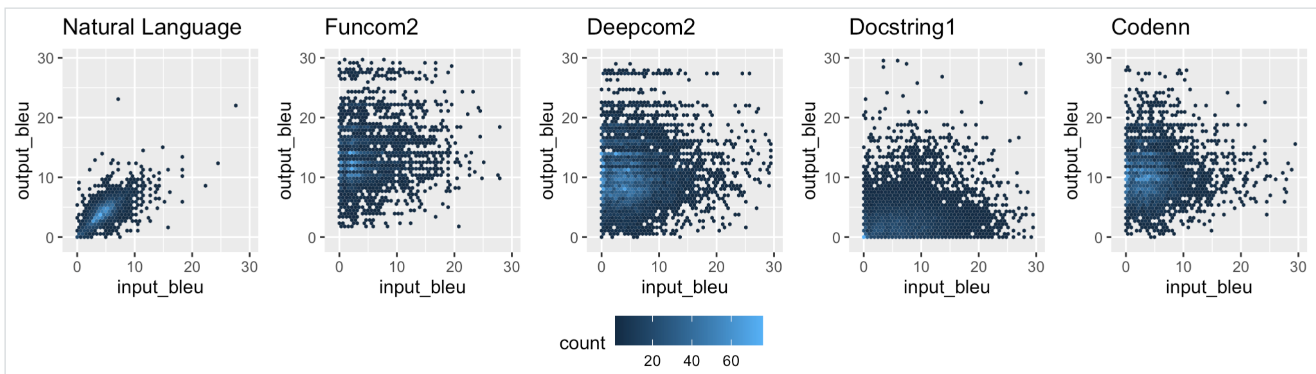

Figure 4: Bivariate plots showing dependency of input-similarity to output-similarity. Pairs with similarity (BLEU-4) less than 10^{-5} are omitted. The natural language translation data have the strongest dependency: similar inputs have the strongest tendency to provide similar outputs

one in which we do no further processing and one where both input/output BLEU similarities are both $> 10^{-5}$. Spearman's ρ, significance, and sample sizes are shown for non-zeroes in the columns (the numbers within the parentheses include the highly dissimilar ones). From the table (last column) we see that about 25-96% of the pairs have some similarity on both inputs and outputs, depending on the dataset. The table also shows the Spearman's ρ (first column) and significance (second column). Each subplot in

Dataset	*Spearman's ρ Correlation* ρ, *BLEU* $> \epsilon$ (ρ,all)	Significance *p-value* p-value, *BLEU* $> \epsilon$ (p-value, all)	*Number of pairs with* BLEU $> \epsilon$
NL	0.70 (0.02)	0.0 (0.055)	2568
Deepcom1	0.056 (0.057)	1.0e-6 (2.2e-8)	7811
Deepcom2	0.036 (0.045)	1.5e-3 (1.1e-5)	7966
Docstring1	0.147 (0.16)	6.7e-42 (4.6e-59)	8585
Docstring2	0.041 (0.047)	9.5e-5 (3.7e-6)	9585
Funcom1	0.124 (0.083)	1.30e-18 (3.4e-16)	5026
Funcom2	0.122 (0.079)	3.03e-18 (6.7e-15)	5082
Codenn	0.012 (0.0012)	0.409 (0.904)	2532

Table 1: Correlation values (Spearman's ρ, and significance, p-value, for the plots in Figure 4. Values outside paranthesis are calculated with only the pairs having pairwise BLEU $> 10^{-5}$; values in paranthesis include all pairs. p-values are adjusted with Benjamini-Hochberg familywise correction. In all cases, we chose 10,000 random pairs

Fig 4 shows one dataset, where x-axis is the BLEU similarity of a pair's inputs, and y-axis is that of outputs. The plot is a binned 2D histogram, using colored hexagons to represent counts in that bin. A representative variant of each dataset is plotted (as applicable); the omitted ones are visually very similar.

We can clearly see a stronger relationship between input and output BLEUs in the natural language setting. Particularly for natural language data, this is further evidenced by the rather high Spearman correlation for the non-zero BLEU pairs (*0.70!!*), and the evident visual dependence between input-input similarity and output-output similarity is note worthy; this indicates that there is strong, fairly monotonic relationship in natural language translation: the more similar the source, the more similar the translation!

This analysis suggests that natural language data has a stronger, more specific input-output dependence; this also suggests that translation between languages is more amenable to learnable function-approximators like deep learners; this appears to be *substantially less true* for code-comment data. This gives us the following conclusion with reference to RQ3.

> **Result 3:** *The natural language translation (WMT) shows a stronger input-output dependence than the CCT datasets in that similar inputs are more likely to produce similar outputs.*

4.3 Information Retrieval Baselines

As can be see in fig. 4 and table 2, datasets for the natural language translation task show a smoother and more monotonic input-output dependence; by contrast, code-comment datasets seem to have little or no input-output dependence. This finding casts some doubt on the existence of a general sequence-to-sequence *code* $\rightarrow$ *comment* function that can be learned using a universal function approximator like a deep neural network. However it leaves open the possibility that a more data-driven approach, that simply memorizes the training data in some fashion, rather than trying to generalize from it, might also work. Thus, given a code input, perhaps we can just try to find similar code in the training dataset, and retrieve the comment associated with the similar code. This is a simple and naive information-retrieval (IR) approach. We then compare this to the IR performance on NL translation.

4.3.1 Method. We use Apache Solr Version 8.3.1[13] to implement a straightforward IR approach. Apache Solr is a open source document search engine based on Apache Lucene. We simply construct an index of over the code parts of the relevant datasets; given a code input, we use that as a "query" over the index, find the closest match, and return the comment associated with the closest matching code as the "generated comment".

We used the default parameters of Solr without tuning. This includes the default BM25 scoring function [46]. For each dataset, we use always the same tokenization procedure used by authors. In addition, we perform some additional pre-processing on the code, that is typically required for IR approaches. For example, we remove highly frequent stop words from the code. Additionally, for datasets do not provide a tokenization phase that actually splits cammelCaseWords or snake_case_words, we include terms for indexing and searching which includes the split form of these words. However, we note that the processing of stop words and word splitting only effects a minor change in performance.

4.3.2 IR Results. We find that on most datasets the simple IR baseline approaches the neural models, and exceeds it for DeepCom1, DocString1, and DocString2. However, IR does poorly on the WMT translation dataset, and also on CodeNN. In both cases, we speculate that this may reflect the relative level of redundancy in these datasets. CodeNN is drawn from StackOverflow, which tends to have fewer duplicated questions; in the case of WMT, which is hand-curated, we expect there would be fewer duplications.

Prior work [14, 62] has used very sophisticated IR methods. We cannot claim to supersede these contributions; but will point out that a very naive IR method does quite well, in some cases better than very recently published methods on datasets/dataset-variations which currently lack IR baselines. We therefore view IR baselines as important calibration on model performance; by trying such a simple baseline first one can help find pathologies in the model or dataset which require further exploration.

We also note that there is variation results. In DeepCom2f, which includes 10 cross-project folds, we observe a wide range results ranging from a BLEU-DC of **20.6** to **48.4**! This level of variation across folds is a cause for concern...this suggests depending on the split, a model with higher capacity to memorize the training data might do better or worse, potentially muddling the results if only doing one split. Similarly we notice that between different versions of FunCom scores vary quite a bit; this variation may confound measurement of actual differences due to technical improvements.

> **Recommendation:** Since even naive IR methods provide competitive performance in many CCT datasets, they can be

[13] https://lucene.apache.org/solr/
[14] Value is for 2019 Model but on the 2018 test split

Dataset	Method Used	Score	Score Method
DeepCom2f	IR-Baseline	32.7	BLEU-DC
-	DeepCom (SBT) [22]	38.2	-
-	Seq2Seq [50]	34.9 [22]	-
DeepCom1	IR-Baseline	**45.6**	BLEU-ncs
-	Transformer [2, 53]	44.6	-
FunCom1	IR-Baseline	18.1	BLEU-FC
-	astattend-gru [33]	19.6	-
FunCom2	IR-Baseline	18.2	BLEU-FC
-	astattend-gru [33]	18.7 [32]	-
-	code2seq [6]	18.8 [32]	-
-	code+gnn+BiLSTM[32]	19.9	-
CodeNN	IR-Baseline	*7.6*	BLEU-CN
-	IR Iyer *et al.*	13.7 [26]	-
-	CodeNN [26]	20.4	-
DocString2	IR-Baseline	**32.6**	BLEU-ncs
-	Transformer [2, 53]	32.5 [2]	-
DocString1	IR-Baseline	**27.7**	BLEU-Moses
-	Seq2Seq	14.0 [8]	-
NL de-en	IR-Baseline	*2.2*	SacreBLEU
-	FAIR Transformer[41]	42.7[14]	-

Table 2: Measurements of a simple information retrieval baseline compared to various neural machine translation based methods. The scoring method we use mirrors the one used on the dataset (see Section 3.1).

an important part for checking for issues in the new collection and new processing of CCT datasets.

4.4 Calibrating BLEU Scores

We now return our last research question, RQ 5. How should we interpret the BLEU results reported in prior work, and also the information retrieval BLEU numbers that we found (which are in the same range, see table 1)?

To calibrate these reported BLEU scores, we conducted an observational study, using *affinity groups* (AGs) of methods that model different levels of expected similarity between the methods. For example, consider a random pair of methods, so that both elements of the pair are *methods from a different project*. This is our lowest-affinity group; we would expect the comments to have very little in common, apart from both being utterances that describe code. The next higher affinity group is a random pair of *methods from the same project*. We would expect these to be a bit more similar, since they are both concerned with the same application domain or function. The next higher level would *methods in the same class* which presumably are closer, although they would be describing different functions. By taking a large number random pairs from each of these affinity groups, and measuring the BLEU for pairs in each group, we can get an estimate of BLEU for each group. For our experiment, we picked the 1000 largest projects from Github, and then chose 5000 random pairs from each of the affinity groups. For each pair, we randomly picked one as the "reference" output, and the other as the "candidate" output, and the BLEU-M2 score. We report the results in two different ways, in fig. 5 and in table 3. For intraclass, we do not take more than six random pairs from a single class. In all AGs, we removed all but one of the overloaded methods, and all getters and setters before our analysis. Without this filtering we see a difference of around 1-3 points.

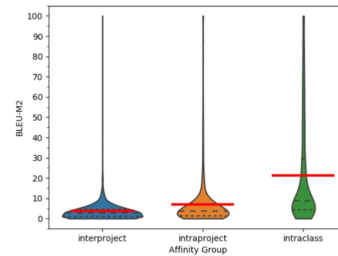

Figure 5: The distribution of BLEU scores between affinity groups. Red lines represent the means (i.e. the Sentence BLEU), and the dashed lines represent quartiles.

First we describe fig. 5 which shows the distribution of the BLEU scores in the 3 AGs. As might be expected, the inter-project AG shows a fairly low mean, around 3. The intra-project AG is a few BLEU points higher. Most notably, the intraclass AG has a BLEU score around 22, which is close to the best-in-class values reported in prior work for some (but not all) datasets.

Note with the implemented experiment we cannot exactly compare these numbers, as each dataset is drawn from a different distribution. Most CCT datasets provide the data in the traditional translation format of (source, target) pairs, making it difficult to recover the other affinity group pairs of a given dataset example. This is why created our own new sampling based off of 1000 large Github repos. While not exactly comparable to existing datasets, new creations of CCT data could start with this simple affinity group baseline to calibrate the reported results.

Another, stronger, affinity grouping would be methods that are semantically equivalent. Rather than trying to identify such an affinity group ourselves by hand (which might be subject to confirmation bias) we selected matched API calls from a recent project, SimilarAPI [15] by Chen [13] which used machine learning methods to match methods in different, but equivalent APIs (e.g., `Junit` vs. `testNG`). We extracted the descriptions of 40 matched pairs of high scoring matches from different APIs and computed their BLEU. We found that these BLEU scores are on average about 8 points higher, with a mean around 32. This number should be taken with caution, however, since comments in this AG sample are substantially shorter than in the other groups.

[15]http://similarapi.appspot.com/

Recommendation: Testing affinity groups can provide a baseline for calibrating BLEU results on a CCT dataset, as the simple trick of generating comments for a given method simply by retrieving the comments of a random other method in the same class possibly can approach SOTA techniques.

Postscript: (*BLEU Variability*) We noted earlier in Section 4.3.2, page 8, that there was considerable intrinsic variation within a dataset, simply *across different folds*; we reported that measured BLEU-DC in DeepCom2f ranged from 20.6 to 48.4; similar results were noted in the different variants of FunCom. This above affinity group experiment with IR provided an opportunity to calibrate another BLEU variability, *across different ways of calculating BLEU.*

Function	intraclass
BLEU-FC	24.81
BLUE-CN	21.34
BLEU-DC	23.5
SacreBLEU	24.81
BLEU-Moses	24.6
BLEU-ncs	21.49
BLEU-M2	21.22

Table 3: The scores of samples of Java methods from the same class.

We took the 5000-pair sample from the intraclass sample, and measured the sentence BLEU for these pairs using the BLEU implementation variations used in the literature. The results are shown in table 3. The values range from around 21.2 to around 24.8; this range is actually rather high, compared to the gains reported in recently published papers. This finding clarifies the need to have a standardized measurement of performance.

Observation: Measurements show substantial variation. The version of BLEU chosen, and sometimes even the folds in the training/test split, can cause substantial variation in the measured performance, that may confound the ability to claim clear advances over prior work.

5 DISCUSSION

We summarize our main findings and their implications.

Comment Repetitiveness Our findings presented in Figure 1 show that comments in CCT datasets are far more repetitive than the English found in the WMT dataset; figure 2 suggests that this not merely a matter of greater vocabulary in distribution in comments, but rather a function of how words are combined in comments. The highly-prevalent patterns in comment have a substantially greater impact on the measured BLEU performance of models trained with this CCT data, as shown in Figure 3. A closer look at the CCT datasets shows that trigrams such as `creates a new`, `returns true if`, `constructor delegates to`, `factory method for` are very frequent. Getting these right (or wrong) has a huge influence on performance.

Implications: These findings suggest that getting just a few common patterns of comments right might deceptively affect measured performance. So the actual performance of comment generation might deviate a lot from measured values, much more so relative to natural language translation. Repetition in comments might also mean that fill-in-the-blanks approaches [49] might be revisited, with a more data-driven approach; classify code first, to find the right template, and then fill-in-the-blanks, perhaps using an attention- or copy-mechanism.

Input/Output Dependence When translating from one language to another, one would expect that more similar inputs produce more similar outputs, and that this dependence is relatively smooth and monotonic. Our findings in figure 4 and table 1, indicate that this property is indeed very strongly true for general natural language outputs, but not as much for the comments.

Implications: Deep-learning models are universal high-dimensional continuous function approximators. Functions exhibiting a smooth input-output dependency, could be reasonably expected to be easier to model. BLEU is a measure of lexical (token sequence) similarit); the rather non-functional nature of the dependency suggested by figure 4 and table 1 indicate that token-sequence models that work well for Natural language translation may be less performant for code; it may be that other, non-sequential models of code, such as tree-based or graph-based, are worth exploring further [32, 56]

Baselining with IR Our experience suggests that simple IR approach provides BLEU performance that is comparable to current state of the art.

Implications Our findings suggest that a simple, standard, basic IR approach would be a useful baseline for approaches to the CCT task. Especially considering the range of different BLEU and tokenization approaches, this would be a useful strawman baseline.

Interpreting BLEU Scores BLEU, METEOR, ROGUE etc are measures that have been developed for different task in natural language processing, such as translation & summarization, often after extensive, carefully designed, human subject studies. Since BLEU is most commonly used in code-comment translation, we took an observational approach calibrate the BLEU score. Our results, reported in fig. 5 and table 3 indicate that the reported BLEU scores are not that high.

Implications: The best reported BLEU scores for the German-English translation tasks are currently are in the low 40's. Our affinity group calibration suggests that on some datasets, the performance of models are comparable on average to retrieving the comment of a random method from the same class. While this conclusion can't be explicitly drawn for a specific dataset without using the exact examples and processing from that specific dataset, but comparing results at an affinity group level can provide insight into minimum expected numbers for a new CCT datset.

Learning From NLP Datasets We find that the current landscape of CCT datasets to be rather messy. There are often several different versions of the same dataset with different preprocessing, splits, and evaluation functions which all seem equivalent in name, but unless extra care is taken, might not be comparable.

However, some tasks in NLP do not seem to observe such variance within a task. We postulate this could be due to several reasons. For one, with the popularity of large open source repositories, it has become cheap easy for a software engineering researcher to collect a large number of pairs of code and comments. This does not require hiring a human to label properties of text, and thus less effort might be taken on quality control. Because researchers are domain experts in the datasets, they might be also more willing to apply their own version of preprocessing.

In addition, there are a wider array of tools to enforce consistency on various NLP tasks. For example the WMT conference on translation, a competition is ran with held out data and human evaluation. Other tasks, such as SQuAD[45] for reading comprehension and GLUE[55] for multitask evaluation allow for uploading code to a server which runs the proposed model on held out data. This ensure consistency in evaluation metrics and data.

We view adapting some these techniques as an interesting avenue for future work.

6 THREATS TO VALIDITY

Our paper is a retrospective, and doesn't propose any new tools, metrics,*etc*, still some potential threats exist to our findings.

Fold Variance With the exception of DeepCom2f we did not run measures over multiple folds or samples of the data. This makes it possible that there is variance in some of our reported numbers.

The Affinity Benchmarks When collecting affinity groups, we collect full methods and process them using a set of filters. This means that when comparing these numbers, they might not be dirrectly comparable to a specific dataset. The numbers are presented only as estimate of similarity of the affinity groups.

Replication Threat Whenever we had to , we did our best to replicate, and measure the quantities we reported using the same code as the previous work. Still, it is possible that we failed to comprehend some subtleties in the provided code, and this may be a threat to our findings.

Generalizability We covered all the commonly used datasets and literature we could find. However, it may be that we have missed some where cases our findings don't hold.

7 CONCLUSION

In this paper, we described a retrospective analysis of several research efforts which used machine learning approaches, originally designed for the task of natural language translation, for the task of generating comments from code. We examined the datasets, the evaluation metrics, and the calibration thereof. Our analysis pointed out some key differences between general natural language corpora and comments: comments are a lot more repetitive. We also found that a widely used natural language translation dataset shows a stronger, smoother input-output relationships than natural language. Turning then to a popular evaluation metric (BLEU score) we found considerable variation based on the way it's calculated; in some cases this variation exceeded claimed improvements. Looking at calibration of the reported BLEU scores, first, we found that simple off-the-shelf information retrieval offers performance comparable to that reported previously. Second, we found that the simple trick of retrieving a comment associated with a method in the same class as a given method achieves an average performance comparable to current state-of-the-art. Our work suggests that future work in the area would benefit from a) other kinds of translation models besides sequence-to-sequence encoder-decorder models b) more standardized measurement of performance and c) baselining against Information Retrieval, and against some very coarse foils (like retrieving a comment from a random other method in the same class).

Funding for this research was provided by National Science Foundation, under grant NSF 1414172, *SHF: Large: Collaborative Research: Exploiting the Naturalness of Software.*

Source code and data will be made available at https://bit.ly/3lBDegY

REFERENCES

[1] 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).

[2] Wasi Uddin Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. 2020. A Transformer-based Approach for Source Code Summarization. arXiv:cs.SE/2005.00653

[3] Miltiadis Allamanis. 2019. The adverse effects of code duplication in machine learning models of code. In *Proceedings of the 2019 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*. 143–153.

[4] Miltiadis Allamanis, Hao Peng, and Charles Sutton. 2016. A convolutional attention network for extreme summarization of source code. In *International conference on machine learning*. 2091–2100.

[5] Uri Alon, Shaked Brody, Omer Levy, and Eran Yahav. 2018. code2seq: Generating sequences from structured representations of code. *arXiv preprint arXiv:1808.01400* (2018).

[6] Uri Alon, Omer Levy, and Eran Yahav. 2018. code2seq: Generating Sequences from Structured Representations of Code. *CoRR* abs/1808.01400 (2018). arXiv:1808.01400 http://arxiv.org/abs/1808.01400

[7] Sushil Bajracharya, Trung Ngo, Erik Linstead, Yimeng Dou, Paul Rigor, Pierre Baldi, and Cristina Lopes. 2006. Sourcerer: a search engine for open source code supporting structure-based search. In *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*. ACM, 681–682.

[8] Antonio Valerio Miceli Barone and Rico Sennrich. 2017. A parallel corpus of Python functions and documentation strings for automated code documentation and code generation. *CoRR* abs/1707.02275 (2017). arXiv:1707.02275 http://arxiv.org/abs/1707.02275

[9] Loïc Barrault, Ondřej Bojar, Marta R. Costa-jussà, Christian Federmann, Mark Fishel, Yvette Graham, Barry Haddow, Matthias Huck, Philipp Koehn, Shervin Malmasi, Christof Monz, Mathias Müller, Santanu Pal, Matt Post, and Marcos Zampieri. 2019. Findings of the 2019 Conference on Machine Translation (WMT19). In *Proceedings of the Fourth Conference on Machine Translation (Volume 2: Shared Task Papers, Day 1)*. Association for Computational Linguistics, Florence, Italy, 1–61. https://doi.org/10.18653/v1/W19-5301

[10] Ondrej Bojar, Christian Buck, Christian Federmann, Barry Haddow, Philipp Koehn, Johannes Leveling, Christof Monz, Pavel Pecina, Matt Post, Herve Saint-Amand, et al. 2014. Findings of the 2014 workshop on statistical machine translation. In *Proceedings of the ninth workshop on statistical machine translation*. 12–58.

[11] Raymond PL Buse and Westley R Weimer. 2008. Automatic documentation inference for exceptions. In *Proceedings of the 2008 international symposium on Software testing and analysis*. Citeseer, 273–282.

[12] Boxing Chen and Colin Cherry. 2014. A Systematic Comparison of Smoothing Techniques for Sentence-Level BLEU. In *WMT@ACL*.

[13] Chunyang Chen. [n.d.]. SimilarAPI: Mining Analogical APIs for Library Migration. ([n. d.]).

[14] Qingying Chen and Minghui Zhou. 2018. A Neural Framework for Retrieval and Summarization of Source Code. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE 2018)*. Association for Computing Machinery, New York, NY, USA, 826–831. https://doi.org/10.1145/3238147.3240471

[15] Luis Fernando Cortés-Coy, Mario Linares-Vásquez, Jairo Aponte, and Denys Poshyvanyk. 2014. On automatically generating commit messages via summarization of source code changes. In *2014 IEEE 14th International Working Conference on Source Code Analysis and Manipulation*. IEEE, 275–284.

[16] Deborah Coughlin. 2003. Correlating automated and human assessments of machine translation quality. In *Proceedings of MT summit IX*. 63–70.

[17] Sergio Cozzetti B de Souza, Nicolas Anquetil, and Káthia M de Oliveira. 2005. A study of the documentation essential to software maintenance. In *Proceedings of the 23rd annual international conference on Design of communication: documenting & designing for pervasive information*. ACM, 68–75.

[18] Jacob Eisenstein. 2018. Natural language processing.

[19] Beat Fluri, Michael Würsch, and Harald C. Gall. 2007. Do Code and Comments Co-Evolve? On the Relation between Source Code and Comment Changes. *14th Working Conference on Reverse Engineering (WCRE 2007)* (2007), 70–79.

[20] Sakib Haque, Alexander LeClair, Lingfei Wu, and Collin McMillan. 2020. Improved Automatic Summarization of Subroutines via Attention to File Context. *arXiv preprint arXiv:2004.04881* (2020).

[21] Matthew Henderson, Ivan Vulić, Iñigo Casanueva, Paweł Budzianowski, Daniela Gerz, Sam Coope, Georgios Spithourakis, Tsung-Hsien Wen, Nikola Mrkšić, and Pei-Hao Su. [n.d.]. PolyResponse: A Rank-based Approach to Task-Oriented Dialogue with Application in Restaurant Search and Booking. In *Proceedings of EMNLP 2019*.

[22] Xing Hu, Ge Li, Xin Xia, David Lo, and Zhi Jin. 2018. Deep code comment generation. In *Proceedings of the 26th Conference on Program Comprehension*. ACM, 200–210.

[23] Xing Hu, Ge Li, Xin Xia, David Lo, and Zhi Jin. 2019. Deep code comment generation with hybrid lexical and syntactical information. *Empirical Software Engineering* (2019), 1–39.

[24] Xing Hu, Yuhan Wei, Ge Li, and Zhi Jin. 2017. CodeSum: Translate program language to natural language. *arXiv preprint arXiv:1708.01837* (2017).

[25] Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. 2019. CodeSearchNet Challenge: Evaluating the State of Semantic Code Search. arXiv:cs.LG/1909.09436

[26] Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. 2016. Summarizing source code using a neural attention model. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2073–2083.

[27] Siyuan Jiang, Ameer Armaly, and Collin McMillan. 2017. Automatically generating commit messages from diffs using neural machine translation. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 135–146.

[28] Rafael-Michael Karampatsis, Hlib Babii, Romain Robbes, Charles Sutton, and Andrea Janes. [n.d.]. Big Code!= Big Vocabulary: Open-Vocabulary Models for Source Code.

[29] Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander M Rush. 2017. Opennmt: Open-source toolkit for neural machine translation. *arXiv preprint arXiv:1701.02810* (2017).

[30] Philipp Koehn. 2009. *Statistical machine translation*. Cambridge University Press.

[31] Douglas Kramer. 1999. API documentation from source code comments: a case study of Javadoc. In *Proceedings of the 17th annual international conference on Computer documentation*. 147–153.

[32] Alexander LeClair, Sakib Haque, Lingfei Wu, and Collin McMillan. 2020. Improved Code Summarization via a Graph Neural Network. arXiv:cs.SE/2004.02843

[33] Alexander LeClair, Siyuan Jiang, and Collin McMillan. 2019. A neural model for generating natural language summaries of program subroutines. In *Proceedings of the 41st International Conference on Software Engineering*. IEEE Press, 795–806.

[34] Alexander LeClair and Collin McMillan. 2019. Recommendations for Datasets for Source Code Summarization. *CoRR* abs/1904.02660 (2019). arXiv:1904.02660 http://arxiv.org/abs/1904.02660

[35] Zhongxin Liu, Xin Xia, Ahmed E Hassan, David Lo, Zhenchang Xing, and Xinyu Wang. 2018. Neural-machine-translation-based commit message generation: how far are we?. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. 373–384.

[36] Edward Loper and Steven Bird. 2002. NLTK: The Natural Language Toolkit. In *Proceedings of the ACL Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics*. Somerset, NJ: Association for Computational Linguistics, 62–69. http://arXiv.org/abs/cs/0205028.

[37] Paul W McBurney, Cheng Liu, Collin McMillan, and Tim Weninger. 2014. Improving topic model source code summarization. In *Proceedings of the 22nd international conference on program comprehension*. ACM, 291–294.

[38] Paul W McBurney and Collin McMillan. 2014. Automatic documentation generation via source code summarization of method context. In *Proceedings of the 22nd International Conference on Program Comprehension*. ACM, 279–290.

[39] Jessica Moore, Ben Gelman, and David Slater. 2019. A Convolutional Neural Network for Language-Agnostic Source Code Summarization. *CoRR* abs/1904.00805 (2019). arXiv:1904.00805 http://arxiv.org/abs/1904.00805

[40] Laura Moreno, Jairo Aponte, Giriprasad Sridhara, Andrian Marcus, Lori Pollock, and K Vijay-Shanker. 2013. Automatic generation of natural language summaries for java classes. In *2013 21st International Conference on Program Comprehension (ICPC)*. IEEE, 23–32.

[41] Nathan Ng, Kyra Yee, Alexei Baevski, Myle Ott, Michael Auli, and Sergey Edunov. 2019. Facebook FAIR's WMT19 News Translation Task Submission. arXiv:cs.CL/1907.06616

[42] Yoann Padioleau, Lin Tan, and Yuanyuan Zhou. 2009. Listening to programmers taxonomies and characteristics of comments in operating system code. In *Proceedings of the 31st International Conference on Software Engineering*. IEEE Computer Society, 331–341.

[43] 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 on association for computational linguistics*. Association for Computational Linguistics, 311–318.

[44] Matt Post. 2018. A Call for Clarity in Reporting BLEU Scores. arXiv:cs.CL/1804.08771

[45] Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know What You Don't Know: Unanswerable Questions for SQuAD. *CoRR* abs/1806.03822 (2018). arXiv:1806.03822 http://arxiv.org/abs/1806.03822

[46] Stephen E Robertson and Steve Walker. 1994. Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In *SIGIR'94*. Springer, 232–241.

[47] Paige Rodeghero, Collin McMillan, Paul W McBurney, Nigel Bosch, and Sidney D'Mello. 2014. Improving automated source code summarization via an eye-tracking study of programmers. In *Proceedings of the 36th international conference on Software engineering*. ACM, 390–401.

[48] Giriprasad Sridhara, Emily Hill, Divya Muppaneni, Lori Pollock, and K Vijay-Shanker. 2010. Towards automatically generating summary comments for java methods. In *Proceedings of the IEEE/ACM international conference on Automated software engineering*. ACM, 43–52.

[49] Giriprasad Sridhara, Lori Pollock, and K Vijay-Shanker. 2011. Automatically detecting and describing high level actions within methods. In *Proceedings of the 33rd International Conference on Software Engineering*. ACM, 101–110.

[50] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to Sequence Learning with Neural Networks. *CoRR* abs/1409.3215 (2014). arXiv:1409.3215 http://arxiv.org/abs/1409.3215

[51] Armstrong A Takang, Penny A Grubb, and Robert D Macredie. 1996. The effects of comments and identifier names on program comprehensibility: an experimental investigation. *J. Prog. Lang.* 4, 3 (1996), 143–167.

[52] Ashish Vaswani, Samy Bengio, Eugene Brevdo, Francois Chollet, Aidan N Gomez, Stephan Gouws, Llion Jones, Łukasz Kaiser, Nal Kalchbrenner, Niki Parmar, et al. 2018. Tensor2tensor for neural machine translation. *arXiv preprint arXiv:1803.07416* (2018).

[53] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. arXiv:cs.CL/1706.03762

[54] Yao Wan, Zhou Zhao, Min Yang, Guandong Xu, Haochao Ying, Jian Wu, and Philip S Yu. 2018. Improving automatic source code summarization via deep reinforcement learning. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. 397–407.

[55] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2018. GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding. *CoRR* abs/1804.07461 (2018). arXiv:1804.07461 http://arxiv.org/abs/1804.07461

[56] W. Wang, Y. Zhang, Y. Sui, Y. Wan, Z. Zhao, J. Wu, P. Yu, and G. Xu. 2020. Reinforcement-Learning-Guided Source Code Summarization via Hierarchical Attention. *IEEE Transactions on Software Engineering* (2020), 1–1.

[57] Wenhua Wang, Yuqun Zhang, Zhengran Zeng, and Guandong Xu. 2020. $TranS^3$: A Transformer-based Framework for Unifying Code Summarization and Code Search. arXiv:cs.SE/2003.03238

[58] Bolin Wei, Ge Li, Xin Xia, Zhiyi Fu, and Zhi Jin. 2019. Code Generation as a Dual Task of Code Summarization. arXiv:cs.LG/1910.05923

[59] Edmund Wong, Taiyue Liu, and Lin Tan. 2015. Clocom: Mining existing source code for automatic comment generation. In *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, 380–389.

[60] 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.

[61] Annie TT Ying and Martin P Robillard. 2013. Code fragment summarization. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*. ACM, 655–658.

[62] Jian Zhang, Xu Wang, Hongyu Zhang, Hailong Sun, and Xudong Liu. 2020. Retrieval-based Neural Source Code Summarization. In *Proceedings, 42nd International Conference on Software Engineering*.