Is the Corpus Ready for Machine Translation? A Case Study with Python to Pseudo-Code Corpus

Sawan Rai1 · Ramesh Chandra Belwal1 · Atul Gupta1

Received: 19 October 2021 / Accepted: 9 June 2022 / Published online: 19 July 2022
© King Fahd University of Petroleum & Minerals 2022

Abstract
The availability of data is the driving force behind most of the state-of-the-art techniques for machine translation tasks. Understandably, this availability of data motivates researchers to propose new techniques and claim about the superiority of their techniques over the existing ones by using suitable evaluation measures. However, the performance of underlying learning algorithms can be greatly influenced by the correctness and the consistency of the corpus. We present our investigations for the relevance of a publicly available python to pseudo-code parallel corpus for automated documentation task, and the studies performed using this corpus. We found that the corpus had many visible issues like overlapping of instances, inconsistency in translation styles, incompleteness, and misspelled words. We show that these discrepancies can significantly influence the performance of the learning algorithms to the extent that they could have caused previous studies to draw incorrect conclusions. We performed our experimental study using statistical machine translation and neural machine translation models. We have recorded a significant difference (∼10% on BLEU score) in the models’ performance after removing the issues from the corpus.

Keywords Parallel corpus · Python code · Pseudo-code · Statistical machine translation · Neural machine translation

1 Introduction
Recently, applications of machine learning and deep learning techniques have drawn a lot of attention in virtually every domain of research and software engineering is no exception. Learning algorithms like deep learning, require massive data [26]. Crowdsourcing is one of the ways to create or collect large corpora, and it is gaining momentum as the new online distributed problem solving and production model in which people complete a task in a collaborative manner [36].

Corpus annotation is one of the fields where crowdsourcing is a helpful technique for the research community. One of the most challenging tasks in the annotation process is the quality control to ensure the quality of annotated data because there is no guarantee that all workers have enough knowledge needed to complete the required tasks at a satisfactory level of quality [5]. Naively collecting the data through non-professional workers, results in low-quality data if no or little quality control is incorporated [40]. Annotating linguistic data is often a hard, time-taking, and expensive task. Even with strong annotation guidelines, human subjects often deviate in their process of annotation; each has different biases, interpretations of the task, and levels of consistency [12]. Therefore, a post-editing or cross-check step is essentially required to create a correct and consistent corpus that can be used for learning tasks.

In this paper, we present a case study on a parallel corpus developed using crowdsourcing and provided by Oda et al. [28] for the language translation task. A parallel corpus is a collection of actual text instances in one language and their translations in another language [21]. Though, Oda et al. [28] have constructed two parallel corpus: Python-English and Python-Japanese language corpus for language translation tasks. We focus on the Python-English language corpus only for this study, which we shall refer it as “V1” in the rest of the paper. The authors proposed a Statistical Machine Translation (SMT)-based approach for Python
code to Pseudo-code generation [28]. Subsequently, many research works have used the corpus for various purposes [2,3,24], including proposing improved algorithms for the documentation tasks.

1.1 Motivation and Objective

Arguably, we initially targeted the available corpus in enhancing the documentation task further as the reported performance so far suggested a BLEU score of 0.54 [1], which normally indicates the fair possibility of taking it further. After a few experiments, we become a little doubtful about the correctness and the consistency of the corpus. Subsequently, we analyzed the corpus for visible issues and found a few, significant of them. For instance, we observed the wrapping of text in the target language instances (Pseudo-code), causing incorrect translations in the corpus for more than 10% of the cases. Secondly, the annotator annotated the same code statement in different styles over the period of time, causing inconsistencies in the corpus. Aforementioned issues motivated us to investigate the effects of these issues on the translation performance of the model. A more detailed account of these discrepancies is included in the following sections.

This case study was carried out in two parts. In the first part, we verified and reproduced the results claimed by various studies on the original (i.e., version “V1”) corpus. Subsequently, we fixed the instance overlapping problem caused by the wrapping of the text in the target language of the corpus data and fixed the inconsistencies related to the different styles used by the annotator and again repeated the studies on the updated version of the corpus “V2”. We found a few incomplete and misspelled instances in the target language of the parallel corpus. Our main objective is to find the answer to the following research questions using experimental study:

- **RQ 1** How the distribution of the corpus get affected after updates?
- **RQ 2** How the performance of Statistical and Neural translation models get affected with the updated corpus?

In our analysis, we first applied Statistical Machine Translation (SMT) on two versions of the corpus. Statistical Machine Translation is a technique, which is used to find the relationship between two different natural languages [37]. In the previous work [28], Tree2SMT method provided the best translation among the other SMT techniques. Therefore, we have also used Tree2SMT to repeat the previous study with V1 and V2. We evaluated the performance on metrics BLEU [29], METEOR [7], and RIBES [17] to analyze the SMT model performance.

Next, we applied Neural Machine Translation on the two versions of the parallel corpus. In the recent past, Alhefdhi et al. [1], and Xu et al. [39] proposed Neural Machine Translation approaches to generate pseudo-code from the python source code. Our NMT model is inspired by the Architecture proposed by Bahdanau et al. [6]. We use LSTM and GRU variants to perform the NMT study. LSTM performed better than the GRU variant on both the versions of the corpus.

Our results indicate the influence of correctness and consistency of the corpus on the performance of algorithms used. SMT model performed better than the NMT models for both the versions of the corpus. Alhefdhi et al. [1] claimed a BLEU score of 54.78 on V1. Whereas Xu et al. [39] claimed 38.11 BLEU on V1, we got a BLEU score of 33.24. This suggests that the discrepancies in the corpus may lead to drawing incorrect conclusions.

Based on our investigation of the parallel corpus, we conclude:

- The incorrectness and the inconsistencies in the corpus for various machine learning tasks (like language translation) can lead to drawing incorrect conclusions. This, in turn, can result in a cascading effect misleading the research community.
- Corpus obtained from the manual annotation process must be post-processed for avoidable issues, and authors should provide a detailed account of it, including a suitable disclaimer before sharing this corpus.

The rest of the paper is organized as follows: We discuss the background and related work in Sect. 2. In Sect. 3, we provide the various details of the analysis and removal of discrepancies from the corpus. SMT- and NMT-based empirical studies are given in Sects. 4 and 5, respectively. Next, we discuss the threats to validity in Sect. 6. Finally, we provide our conclusions and future directions in Sect. 7.

2 Related Work

In 2005, Voormann and Gut [35] acknowledged the quality of corpus-based research and theories influenced by the quality of the corpora, not only in terms of their content and volume but particularly the accuracy and richness of the annotations are concerned. They proposed a novel agile corpus creation method that aims the problem of maximizing corpus size as well as the quality and quantity of manual and automatic annotations while reducing the time and cost associated with corpus creation. The significant aspects of agile corpus creation lie in the reorganization of corpus design, data collection, data annotation, corpus analysis, and the identification of potential sources of errors during corpus creation.
Zaidan and Burch [40] proposed an idea to achieve professional-level translations with the help of non-professional translators on the crowdsourcing platforms. They proposed a set of features that model both the translations and the translators. They were able to discriminate between acceptable and unacceptable translations by the model trained on the features set. Evaluation is performed with the NIST 2009 Urdu-English corpus to show that their models can select translations within the limit of quality that is expected from the professional translators.

Fort et al. [14] analyzed many annotation campaigns and proposed an analytical framework, a grid of analysis, to recognize the complexity of annotation tasks. They performed a decomposition of annotation tasks and complexities into elementary ones. The complexity of each elementary task is calculated separately, and the final complexity of annotation campaign is calculated as the combination of the complexity of the elementary tasks.

Baba et al. [5] proposed an unsupervised statistical quality estimation method for general crowdsourcing tasks with unstructured response formats like article writing, program coding, and logo design, which cover the majority on most crowdsourcing market. Their model is a two-stage generative model. The first stage is the creation stage, which provides a model to assess the effects of the annotator’s ability and performance on the artifacts. The next stage is Reviewing, where reviewers provide a quality score to the artifacts.

In 2015, Kajimura et al. [19] addressed the problem of Quality control in the POI (Point Of Interest) collection task. They proposed a two-stage quality control mechanism consisting of an answer clustering stage and a reliability estimation stage. Standard clustering techniques applied to cluster the answers of POI, and the cluster center is chosen as the representative of the cluster. Next, each cluster representative is reviewed and analyzed to estimate the reliability of the answer. They performed experiments on the POI tasks posted to the Lancers crowdsourcing marketplace.

If we look at the few recent works, Roitero et al. [31] applied a crowd-sourcing approach to study whether it is reliable to verify the truthfulness of statements during the COVID-19 pandemic. They asked crowd workers to evaluate the truthfulness of statements and provide corresponding evidence (URLs or text justification) to justify their assessments. An et al. [4] addressed the problem of unfair payment, negative work of participants, cooperative cheating, data quality in crowd-sensing. They proposed a Blockchain model for data quality assessment. They applied DPoR (delegated proof of reputation) consensus mechanism to handle the computing power issue in crowd-sensing networks.

In 2022, Soprano et al. [32] developed Crowd Frame, a software framework that provides services to design and deploy different types of crowd-sourcing tasks. The user does not require deep knowledge of programming, unlike Amazon Mechanical Turk or any other alternatives.

The parallel corpus of Python–Pseudo-code used in our study is given by Oda et al. [28]. Oda proposed a hierarchical statistical machine translation method to generate Pseudo-code for the given Python code instance. Python code is used in the parse tree format and pseudo-code in the tokenized format. In addition to the SMT method, deep learning algorithms are also applied to this parallel corpus by alhefdhi et al. [1] and Xu et al. [39]. After updating the parallel corpus, we apply previous studies on versions (V1, V2) to highlight the impact of corrections performed on the corpus.

3 Parallel Corpus Analysis

In this section, we explain about our detailed analysis to identify and remove the visible issues in the parallel corpus. As a result, we get the updated version of the corpus named V2.

3.1 Base Version of the Corpus (V1)

The original corpus of Python–Pseudo-code is provided by Oda et al. [28]. We named it as base version: V1. Corpus contains 18805 parallel instances of Python code and corresponding Pseudo-codes in two separate files: all.code & all.anno. Corpus is publicly available at http://ahclab.naist.jp/pseudogen/. Few examples from the parallel corpus are shown in Table 1. Python code is taken from the Django1 web development framework, which is already in executable condition. Respective Pseudo-codes are generated by the human annotator, hired by the author. Pseudo-code is a representation of actual source code in a more human-understandable form. Pseudo-codes are not executable. There is no deterministic way to check the correctness of pseudo-codes. Hence,

1 https://www.djangoproject.com/.
Table 1  Python versus pseudo-code

| Python code                                      | Pseudo-code                                      |
|-------------------------------------------------|--------------------------------------------------|
| from django.core import signals                 | From django.core import signals into default name space |
| self.set (key, new_value, version = version)    | Call the self.set method with key, new_value and version set to version as arguments |
| def __init__ (self, host, *args, **kwargs):      | Define initialization method __init__ with 4 arguments: self, host, list of arguments args and dictionary of arguments kwargs |
| for post_callback in self._post_render_callbacks: | For every post_callback in self._post_render_callbacks |
| class WSGIHandler (base.BaseHandler):           | Derive the WSGIHandler class from the base class base.BaseHandler |

| Python Code                                      | Pseudo Code                                      |
|-------------------------------------------------|--------------------------------------------------|
| 235. cursor.execute("DELETE FROM %s WHERE code_key = %s and table_key = %s", [key]) | 235. call the method cursor.execute with string 'DELETE FROM %s WHERE code_key = %s and table_key = %s' as argument |
| 236. return default                              | 236. substitute the %s with table and list containing key, respectively. return default |
| 237. value = connection[db].ops.process.show (1)  | 237. call the method ops.process.show with second element of row as argument, on the object under the key of connections |
| 238. return value = base64.b64encode (value)     | 238. call the function byte with argument value, use the result as an argument for the call to the method base64.encode |
| 239. def set (self, key, value, version = 'DEFAULT', timeout = version): | 239. use the result as the argument for the function to the pickle.dumps, return the result. define the method set with 5 args |
| 240. for post_callback in self._post_render_callbacks: | For every post_callback in self._post_render_callbacks |
| class WSGIHandler (base.BaseHandler):           | Derive the WSGIHandler class from the base class base.BaseHandler |

3.2 Updated Version (V2)

Here, we discuss our analysis, where we found instance overlapping issues in the Pseudo-code. Some part of a pseudo-code from one instance is wrapped with another pseudo-code. In Fig. 1, at line no. 234 of pseudo-code, contains a sub-part of previous pseudo-code with the return statement pseudo code. At line no. 237 method definition pseudo-code is also preceded by the previous pseudo-code. The general interpretation of the overlapping in our context is as follows:

Required alignment between Python and Pseudo-code:

\[ py_i \leftrightarrow ps_j \text{ here, } i \in [1 \ldots 18805] \]

\[ py_i \text{ and } ps_j \text{ represent } i^{th} \text{ parallel instance in the corpus.} \]

But, there is an overlapping of Pseudo-codes as given below:

\[ \left| py_i \right| = n, \left| ps_j \right| = x, \left| py_{i+1} \right| = m, \text{ and } \left| ps_{i+1} \right| = y \]

Actual alignment with overlapping:

\[ py_1^i, py_2^i, \ldots , py_n^i \leftrightarrow ps_1^i, ps_2^i, \ldots , ps_y^i \]
In our manual analysis, we found more than 10% pseudo-codes were overlapped with their next instance. Overlapping of instance causes severe problems when the corpus is used for the translation task. In SMT, word alignment is performed to create a mapping b/w the source and target instances. With overlapped instances, SMT calculates incorrect word alignments, which will generate incorrect translations. We have removed all the overlapping in the first stage. The average word length remains between 14 and 15. 99.7% and 97.96% instances fall under the word length of 100 and 50, respectively (Fig. 2).

Annotators must follow the standard conventions and instructions while creating the corpus, to avoid inconsistency and incorrectness in the corpus. It is tough to achieve an ideal corpus with the help of human annotators. There are various factors like state of mind, human nature, contextual environment, etc., which affect the human annotator performance. Therefore, a critical post-check is required to control the quality of the corpus [5]. After removing the overlapping among pseudo-code instances, we analyze the pattern followed by the human annotator to write pseudo codes for the given python code. We kept our focus on the most frequently used Python constructs, e.g., Loops, Conditions, Method and Class definitions, Exceptions, Method calling and assignments, Return statements. We found that the annotator used different styles of the template to annotate the same type of Python code instances over the period of time.

Let us consider the simple example of continue statement, which has four different types of pseudo-codes (Table 2). Hence, there is a “one-to-many” mapping between Python and Pseudo-code instances, which will not provide the correct performance measures of the corpus when used under the translation task. This “one-to-many” is perfectly fine if we look in the context of natural language. But, when we consider the pseudo-codes or algorithmic statements, there is a set of rules to write them for a particular code statement. We follow this notion in our work to rectify the inconsistencies in pseudo-codes. For the except construct, the annotator used different phrases (exception occurred, exception raised, exception caught). We have shown various examples of inconsistent pseudo-codes in Table 2, grouped by the constructs. Line_no indicates the location of the instance in the corpus file (V1 version). In addition to the examples given in Table 2, we have found many pseudo-code instances were incomplete and misspelled. We tried our best to remove typo and incompleteness errors from the pseudo-code.

Histograms of V1 and V2 are shown in Fig. 3. 98.67% instances in V1, and 97.99% instances in V2 fall inside the word length of 50. Therefore, for better visualization of Histograms, we have considered the instances having word length less or equal to 50. V2 contains more instances in the range of 1 to 10, compared to V1. V2 contains more instances above the range of 40 compared to V1. We have calculated the Chi-square (CS) and Bhattacharyya distance (BD) metrics for V2 with respect to V1. For the two similar distributions, CS and BD will be very close to zero. CS ranges between $[0 - \infty)$, whereas BD ranges between [0 and 1]. V1–V2 scored 786.10 and 0.433 for CS and BD, respectively. These values indicate that the distributions of V1 and V2 have a considerable difference, which answers our first research question (RQ 1).
| Line no | Python code | Pseudo-code |
|---------|-------------|-------------|
| 95      | def __init__(self, params): | Define function__init__ with self-class instance and params as arguments |
| 155     | def __contains__(self, key): | Define the private method__contains__ with self-class instance and key as arguments |
| 491     | def get(self, key, default=None, version=None): | Define the method get with 4 arguments, self-class instance, key, default set to None and version set to None |
| 626     | def get_many(self, keys, version=None): | Define the get_many method with self-class instance, keys and version set to None as arguments |
| 13360   | touch_import('django.utils.encoding', 'python_2_unicode_compatible', decorated) | Call the function touch_import with 3 arguments: string ‘django.utils.encoding’, string ‘python_2_unicode_compatible’ and decorated |
| 13524   | self._archive.printdir (*args, **kwargs) | Call the method self._archive.printdir with 2 arguments: unpacked list args and unpacked dictionary kwargs |
| 13937   | cache.set (cache_key, [], cache_timeout) | Call the method cache.set with cache_key, an empty list and cache_timeout as arguments |
| 56      | class DefaultCacheProxy (object): | Derive the class DefaultCacheProxy from the base class object |
| 79      | class InvalidCacheBackendError (ImproperlyConfigured): | Derive class InvalidCacheBackendError from the ImproperlyConfigured base class |
| 16101   | class HTMLParser (_html.parser.HTMLParser): | Derive the class HTMLParser from the base class HTMLParser located in the module _html.parser |
| 16556   | class AdminEmailHandler (logging.Handler): | Derive the class AdminEmailHandler from logging.Handler base class |
| 134     | raise NotImplementedError (‘subclasses of BaseCache must provide a set() method’) | Raise an exception of class NotImplementedError, with string ‘subclasses of BaseCache must provide a set() method’ as argument |
| 1316    | raise Not ImplementedError (‘subclasses of Storage must provide a delete() method’) | Raise an NotImplementedError exception with argument string ‘subclasses of Storage must provide a delete() method’ |
| 5867    | raise StopIteration | Raise an StopIteration exception |
| 17374   | raise value | Raise an exception value |
| 37      | return backend_cls (location, params) | Evaluate the function backend_cls with location and params as arguments, return the result |
| 181     | return self.incr_version (key, -delta, version) | Call the method self.incr_version with key, negative delta and version as arguments |
| 35      | except ImportError as e: | If ImportError, renamed to e, is caught |
| 44      | except AttributeError: | Except if AttributeError is caught |
| 1233    | except ImportError: | If ImportError exception occurred |
| 1462    | except (OSError): | If OSError exception was raised |
| 11611   | @ register.filter (“escapejs”) | Decorator function register.filter with an argument string “escapejs” |
| 3676    | @ total_ordering | Total_ordering decorator |
| 1884    | @ cached_property | Decorator cached_property |
| 823     | @ register (Tags.models, Tags.signals) | Decorator register with Tags.models and Tags.signals as arguments |
Table 2 continued

| Line_no | Python code | Pseudo-code |
|---------|-------------|-------------|
| Break statement |
| 1069    | break       | Break from the smallest enclosing loop |
| 1717    | break       | Break from the smallest enclosing loop execution |
| 12928   | break       | Break from the loop execution |
| 17867   | break       | Break the execution of the smallest enclosing loop |
| Continue statement |
| 1670    | continue    | Skip this iteration of the smallest enclosing loop |
| 2348    | continue    | Skip this loop execution |
| 3156    | continue    | Skip this loop iteration |
| 13931   | continue    | Continue with the next iteration of the for loop |
| Elif statement |
| 120     | elif timeout == 0: | Else if timeout equals to integer 0 |
| 256     | elif settings.USE_TZ: | Otherwise is settings.USE_TI is true |
| If statement |
| 1606    | if self.activated: | If self.activated is boolean True |
| 1743    | if settings.DEBUG: | If settings.DEBUG is True |
| Incomplete pseudo-code example |
| 2305    | def __init__(self, subject = ", body = ", from_email = None, to = None, bcc = None, connection = None, attachments = None, headers = None, cc = None): | Define the method __init__ with 10 arguments: self, subject set to an empty string, body set to an empty string |

Fig. 3 Pseudo-code length distribution

Previous research works [1,28,39] used the V1 version of the corpus. We have found two major flaws in the corpus, overlapping and Heterogeneous pseudo-code templates for the same type of python code. When the base version (V1) is used for the translation task, the model would not learn the correct mappings between Python and Pseudo-code. Therefore, we have updated the base version and come up with a new version V2. In Sects. 4, 5, we have shown the effects of corrections on the translation performance.

4 Statistical Machine Translation Based Analysis

Here, we analyze the performance of the statistical machine translation model (proposed by [28]) on both versions of the parallel corpus. We have used the implementation code\(^2\) of the previous work [28] to perform the translation.

\(^2\) https://github.com/delhiros/pseudogen.
4.1 Statistical Machine Translation

Statistical Machine Translation (SMT) has a very long history. SMT was first introduced in the 1950s [37]. Statistical Machine Translation is a technique, which is used to find the relationship between two different natural languages termed as “S (source or input)” and “T (Target or output)”. In our case, Python is the source, and Pseudo-code is the target language. SMT automatically extracts the relationship rules by mapping the S data to the T data. These rules are used to generate output for the given new data of the input language [22,25].

4.1.1 BLEU Score

We use the BLEU score as our metric for the tuning process in the SMT. BLEU (Bilingual Evaluation Understudy) [29] is an automatic evaluation metric to measure the goodness of machine translation models. Value of BLEU score ranges [0–1]. If the generated text \( \hat{T} \) from a particular MT method is exactly the same as reference text T, then BLEU will be 1. BLEU is defined in terms of “Modified n-gram Precision (\( p_n \))” and “Brevity Penalty (BP)”. Formulation of BLEU is given as follows:

\[
p_n = \frac{\sum_{n \text{-gram} \in T} \text{count}_{\text{clip}}(n\text{-gram})}{\sum_{n \text{-gram} \in \hat{T}} \text{count}(n\text{-gram})}
\]

Here, \( \text{count}_{\text{clip}} = \text{Min} \left( \text{count}, \text{max count in reference text} \right) \)

\[
BP = \begin{cases} 
1 & \text{if } c > r \\
\exp(1 - (r/c)) & \text{else} 
\end{cases}
\]

In Eq. 2, \( r \) and \( c \) denote the length of reference text and generated text, respectively.

\[
\text{BLEU} = BP \times \exp \left( \frac{1}{N} \sum_{n=1}^{N} \log p_n \right)
\]

\[
\text{BLEU} = BP \times \exp \left( \sum_{n=1}^{N} w_n \log p_n \right)
\]

Equation 4 is the generalized version of Eq. 3. In general, uniform weights are assigned for each modified n-gram precision. Therefore, \( \sum_{n=1}^{N} w_n = 1 \).

4.2 Process Flow

We have provided a brief detail of all the steps performed in the SMT to learn from the parallel corpus data. Figure 4 shows the execution flow of SMT. The following steps are performed on the Python source code.

**Tokenization** Python code instances are tokenized with the help of tokenize library.

**AST generation** The \( \text{ast} \) module of python is used to generate raw AST data. Oda et al. [28] proposed three models based on the AST data. (1) T2SMT (Tree2SMT) using raw AST, (2) T2SMT using Head inserted AST, (3) T2SMT using Reduced AST. Reduced-T2SMT performed best among all of them. Therefore we use Reduced-T2SMT in our study.

**Head Insertion** “ast” library of python uses the abstract parser, which is defined on runtime semantics. AST uses keywords and operators as internal nodes. We use the GHKM algorithm to extract tree-to-string rules [15]. GHKM searches only on the leaf nodes of the tree. Hence, we need to bring internal nodes at the leaf nodes through the Head insertion process.

**Pruning and Simplification** The number of leaf nodes becomes larger than words in the target sentence, which could lead to noise and extra computation time. To fix this issue, the simplification of head-inserted AST is performed. Oda et al. derived twenty rules to reduce or simplify the AST.

On the target side, the following steps are performed on the Pseudo-code data:

**Tokenization** Pseudo-code instances are considered as simple English language statements for the tokenization process.

**Language Model Generation** For the fluent target sentence generation, SMT model needs a language model built on the target language tokens. For the given target sentence, language model probability is calculated by the product of the probability of every word token in target, given the previous words.

After preprocessing the source and target code instances, alignment calculation is performed with the help of the pialign library. Simplified AST data and the pseudo-code tokens are fed as input to the pialign. Resultant alignment data are further used for rule extraction. We have used the travavatar library for the Tree2String rule extraction [27]. The travavatar internally uses GHKM algorithm [15] for rule extraction. GHKM splits the parse tree into sub-trees, based on the alignment information and extracts the pairs of sub-tree and

---

3 https://docs.python.org/3/library/tokenize.html.
4 https://docs.python.org/3/library/ast.html.
5 https://www.taus.net/technologies/189-pialign.
its related word tokens in the target sentence to form minimal rules. Larger rules are generated using these minimal rules, and original parse tree structure. Pseudo-code language model and extracted rules are fed into the SMT model for training.

After training the SMT model, parameter tuning is performed to optimize the model. BLEU score is used for the tuning process. At last, the model is tested on the test data, and performance parameters are calculated for the comparison. In addition to the BLEU score, METEOR (Metric for Evaluation of Translation with Explicit ORdering) [7], and RIBES (Rank-based Intuitive Bilingual Evaluation Score) [17] are also calculated while testing the model. In 2005, Banerjee et al. [7] highlighted the weaknesses of the BLEU metric and proposed METEOR to overcome the weakness. METEOR got a high correlation value with human evaluation compared to BLEU. Isojaki et al. [17] proposed RIBES for distinct languages having different word ordering. Conventional evaluation metrics do not significantly penalize word ordering mistakes. Therefore, we calculate the METEOR and RIBES also for the test data.

4.3 Analysis of Model Performance

Train, Test, and Validation splits are the same as mentioned in the previous study [28]. First 16000 for training, next 1000 instances for the validation and rest are used in testing. We have repeated the SMT experiment ten times with both versions of the parallel corpus. We shuffle the parallel corpus before each SMT experiment. Boxplots for all three performance measures are shown in Fig. 5, concerning the two versions of the corpus. From the analysis of the boxplots, it is evident that the SMT model performs better on the updated versions V2, compared to base version V1. SMT model trained on V2 surpassed the SMT model trained on V1 for all three measures. The above-mentioned experiments
SMT experiments were performed on CPU-based machine with the 32 GB memory.

5 Neural Machine Translation Based Analysis

References [1,39] applied Neural Machine Translation approach for the pseudo-code generation on the V1 dataset. Here, we analyze the performance of the NMT model on both versions of the parallel corpus. We use the PyTorch implementation (with required modifications) to perform the NMT task, which is influenced by the Bahdanau et al. [6] model architecture. Previous studies have also chosen the same NMT architecture for the implementation. We experiment with GRU [10] and LSTM [16] variants of the model to gain more understanding of the corpus.

5.1 Neural Machine Translation

Neural Machine Translation (NMT) approach has emerged in the last few years in the field of Machine Translation, proposed by Kalchbrenner and Blunsom [20], Sutskever et al. [33], and Cho et al. [9]. A typical NMT system is composed of an encoder-decoder Architecture [33]. The source sentence is fed into the encoder neural network, which encodes the source sentence into a fixed-length context vector. A decoder uses the context vector to output the translation sequence. The entire encoder-decoder system is jointly trained to increase the probability of the correct translation given the source sentence.

5.2 Process Flow of NMT Model

Figure 6 represents the architecture of the NMT model. At first, source and target code statements are tokenized into words. If we consider all the instances, the number of encoder-decoder units and computation will also increase. Therefore, we filter out parallel instances if either source or target exceeds the length of 100 tokens. 99.82% instances of V1 and 99.7% instances of V2 have a maximum length of 100 tokens. After the filtration, dictionaries are created for the source and target codes. The training data are fed into the encoder-decoder network for the training.

Due to the large size of dictionaries, a word-embedding layer is applied before passing the data into the encoder-decoder network. Word-embedding size and hidden state vector size are kept 256. We have applied Stochastic gradient descent for the optimization with Negative Log-Likelihood Loss. Model is learned with the learning rate of 0.01 and dropout 0.1. Teacher Forcing technique [18,30,38] is employed for efficient learning. NMT model is implemented using the PyTorch7 framework. NMT experiments are repeated with GRU and LSTM as the encoder-decoder units for both versions of the corpus. NMT experiments are performed on the Google colaboratory8 with the GPU runtime environment.

5.3 Model Performance Analysis

Figure 7 shows the training and validation loss information with respect to the epoch. We trained NMT models for 80 iterations with LSTM and GRU variants. Both GRU and LSTM units have an additional memory unit to memorize long data sequences. One feature of the LSTM unit, which is missing from the GRU, is the controlled exposure of the memory unit data. In the LSTM, the amount of the memory unit data is controlled by the output gate. On the other hand, the GRU exposes the full memory data without any control [11]. When we trained the GRU model, after ten iterations, loss value increases abruptly for both versions of the corpus. Loss score increase more with version V2 compared to V1. On the other hand, training loss for the LSTM model con-

---

6 http://www.statmt.org/moses/.
7 https://pytorch.org/.
8 https://colab.research.google.com.
continuously decreases with respect to epoch. Both GRU and LSTM achieved the lowest loss values on the version V2 of the corpus. We repeated this experiment ten times. Boxplots for all three performance measures are shown in Fig. 8 with respect to the two versions of the corpus. From the analysis of the boxplots, it is evident that both GRU and LSTM performed better on the updated versions V2, compared to base version V1. If we compare the performance of GRU and LSTM, the LSTM model has achieved higher scores for all three measures.

Table 3 shows the mean values of measures for all the experiments grouped by versions of the parallel corpus. Both SMT and NMT models performed better with the updated corpus, which answers our second research question (RQ 2). SMT model gains more than 12% increase in the performance with V2, compared to V1 in terms of BLEU score. BLEU score of GRU and LSTM increased by 17.8% and 26.5%, respectively. Corpus updates affected the LSTM model performance most.

It is a well-known fact that deep learning algorithms require a huge amount of data to learn [23], and the corpus size is 18805 in this study. Therefore, the NMT models scored less than the SMT model, which is contradicting the results claimed by the previous works [1,39]. Our objective for this work is not to improve the performance of already available research works like [1,28,39]. We have highlighted the issues in the parallel corpus and tried to correct them. Implementation related code, corpus versions (V1, V2), and predicted outputs are available at the private link of figshare: https://figshare.com/s/432c596faf1d52978f87.

6 Threats to Validity

In this section, we discuss potential threats to validity of our research and experimental design.

6.1 Construct Validity

Construct validity is the metric’s overall ability to measure the construct of interest. It is evaluated in various ways, most basically by aggregating the outcomes from all other validity types. We have used BLEU [29], METEOR, [7] and RIBES [17] metrics to evaluate our results. The same measures are
also used in the previous works [1,28]. We repeated each experiment for ten times with random shuffling to mitigate the element of bias or patterns in the split of corpus before training the model.

6.2 Internal Validity

Internal validity addresses the validity of the research outcomes. It is mainly focused on controlling the extraneous variables and outside influences that may impact the outcome. There might be a chance that some inconsistencies skipped by the annotator. Therefore, we employ another annotator to verify the changes.

6.3 External Validity

External validity talks about the generalizability and repeatability of the results. It is a case study on a parallel corpus. We are not making any generalized claims regarding outcomes. But, the same type of manual analysis would help others to find various types of inconsistencies in corpora.

7 Conclusion and Future Work

In this case study, we have analyzed the Python–Pseudo-code parallel corpus. We have identified overlapped pseudo-code instances through manual analysis. Next, we found that the annotator used different styles of pseudo-code template for the same type of python code. We focused on the most frequent python code constructs to remove the heterogeneity for the same type of python code. Table 2 provides a few examples extracted from the corpus through our detailed analysis.

After updating the pseudo-code, we named it as version V2. After updating the parallel corpus, we applied the Statistical Machine Translation method for the Python–Pseudo-code proposed by Oda et al. [28] on both versions of the Parallel corpus. We found that SMT performs better on the version V2 compared to V1. Next, we applied Neural Machine Translation also, as defined by [6] with LSTM and GRU variants. Both LSTM and GRU models, performed better on the V2 version.

The corpus used in this study is unique of its own kind, which makes it a vital and useful resource for the research community. From this case study, we can conclude that cor-

Table 3: Test results

| Model       | Corpus version | BLEU   | RIBES  | METEOR  |
|-------------|----------------|--------|--------|---------|
| SMT         | V1             | 0.54527| 0.88770| 0.43191 |
|             | V2             | 0.61405| 0.91391| 0.48867 |
| NMT-GRU     | V1             | 0.28106| 0.76147| 0.29336 |
|             | V2             | 0.33117| 0.79104| 0.31496 |
| NMT-LSTM    | V1             | 0.33268| 0.78312| 0.3115  |
|             | V2             | 0.42086| 0.84346| 0.35261 |

Best scores are highlighted from bold text for each model.
pus creation is a critical and complex process when the human factor is involved. Post-verification of the corpus is essential for Quality control. We have provided the updated version of the parallel corpus for future studies. The same case study can be repeated with the other available corpora, which will help the research community to make correct inferences from the corpora.

We have analyzed the effect of all the inconsistencies in one go. In future, we would analyze the impact of each type of inconsistencies separately to gain more detailed insights. This kind of analysis can be performed on small and moderate size corpora. In the current scenario, researchers apply transfer learning techniques on low resource data (moderate size corpora). This type of detailed analysis would be effective for constructing an efficient model. In recent times, researchers have focused on the use of transformers \cite{13,34} for source code modeling and proposed new variants of transformers to model the source code \cite{8}. Therefore, we can repeat this study with the recently proposed transformer based models to identify new insights. Tool support can be developed to automate the manual analysis as performed in this study. The tool should be able to identify various avoidable issues and provide options to rectify the issues customarily. This would be helpful for the research community to validate the other corpus before using it.

**Declarations**

**Conflict of interest** The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

**References**

1. Alhefedi, A.; Dam, H.K.; Hata, H.; Ghose, A.: Generating pseudo-code from source code using deep learning. In: 2018 25th Australasian Software Engineering Conference (ASWEC), pp. 21–25. IEEE (2018). https://doi.org/10.1109/ASWEC.2018.00011
2. Allamanis, M.; Peng, H.; Sutton, C.: A convolutional attention network for extreme summarization of source code. In: International Conference on Machine Learning, pp. 2091–2100 (2016)
3. Alon, U.; Brody, S.; Levy, O.; Yahav, E.: code2seq: Generating sequences from structured representations of code. arXiv preprint arXiv:1808.01400 (2018)
4. An, J.; Cheng, J.; Gui, X.; Zhang, W.; Liang, D.; Gui, R.; Jiang, L.; Liao, D.: A lightweight blockchain-based model for data quality assessment in crowdsensing. IEEE Trans. Comput. Soc. Syst. 7(1), 84–97 (2020). https://doi.org/10.1109/TCSS.2019.2956481
5. Baba, Y.; Kashima, H.: Statistical quality estimation for general crowdsourcing tasks. In: Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 554–562. ACM (2013). https://doi.org/10.1145/2487575.2487600
6. Bahdanau, D.; Cho, K.; Bengio, Y.: Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473 (2014)
7. Banerjee, S.; Lavie, A.: 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, pp. 65–72 (2005)
8. Chirkova, N.; Troshin, S.: Empirical study of transformers for source code. In: Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 703–715. https://doi.org/10.1145/3468264.3468611 (2021)
9. Cho, K.; Van Merriënboer, B.; Bahdanau, D.; Bengio, Y.: On the properties of neural machine translation: encoder–decoder approaches. In: Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation, Association for Computational Linguistics, pp. 103–111. https://doi.org/10.3115/v1/W14-4012 (2014)
10. Cho, K.; Van Merriënboer, B.; Gulcehre, C.; Bahdanau, D.; Bougares, F.; Schwenk, H.; Bengio, Y.: Learning phrase representations using RNN encoder–decoder for statistical machine translation. arXiv preprint arXiv:1406.1078 (2014)
11. Chung, J.; Gulcehre, C.; Cho, K.; Bengio, Y.: Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555 (2014)
12. Cohn, T.; Specia, L.: Modelling annotator bias with multi-task Gaussian processes: an application to machine translation quality estimation. In: Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), vol. 1, pp. 32–42 (2013)
13. Devlin, J.; Chang, M.W.; Lee, K.; Toutanova, K.: Bert: pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805 (2018)
14. Fort, K.; Nazarenko, A.; Rosset, S.: Modeling the complexity of manual annotation tasks: a grid of analysis. In: Proceedings of COLING 2012, The COLING 2012 Organizing Committee, pp. 895–910 (2012)
15. Galley, M.; Hopkins, M.; Knight, K.; Marcu, D.: What’s in a translation rule. Technical report. Columbia University New York, Department of Computer Science. https://doi.org/10.21236/ADA460212 (2004)
16. Hochreiter, S.; Schmidhuber, J.: Long short-term memory. Neural Comput. 9(8), 1735–1780 (1997). https://doi.org/10.1162/neco.1997.9.8.1735
17. Isozaki, H.; Hirao, T.; Duh, K.; Sadoh, K.; Tsukada, H.: Automatic evaluation of translation quality for distant language pairs. In: Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, pp. 944–952 (2010)
18. Jordan, M.I.: Attractor dynamics and parallelism in a connectionist sequential machine. In: Artificial Neural Networks: Concept Learning, pp. 112–127 (1990)
19. Kajimura, S.; Baba, Y.; Kaijno, H.; Kashima, H.: Quality control for crowdsourced POI collection. In: Pacific-Asia Conference on Knowledge Discovery and Data Mining, pp. 255–267. Springer. https://doi.org/10.1007/978-3-319-18032-8_20 (2015)
20. Kalchbrenner, N.; Blunsom, P.: Recurrent continuous translation models. In: Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, pp. 1700–1709 (2013)
21. Kenning, M.M.: What are parallel and comparable corpora and how can we use them. In: The Routledge Handbook of Corpus Linguistics, pp. 487–500 (2010)
22. Koehn, P.: Statistical Machine Translation. Cambridge University Press, Cambridge (2009). https://doi.org/10.1017/CBO9780511815829
23. Koehn, P.; Knowles, R.: Six challenges for neural machine translation. In: Proceedings of the First Workshop on Neural Machine Translation, Association for Computational Linguistics, pp. 28–39. https://doi.org/10.18653/v1/W17-3204 (2017)
24. LeClair, A.; Jiang, S.; McMillan, C.: A neural model for generating natural language summaries of program subroutines. In: Proceedings of the 41st International Conference on Software Engineering, pp. 95–86. IEEE Press. https://doi.org/10.1109/ICSE.2019.00087 (2019)
25. Lopez, A.: Statistical machine translation. ACM Comput. Surv. (CSUR) 40(3), 8 (2008). https://doi.org/10.1145/1380584.1380586
26. Marcus, G.: Deep learning: a critical appraisal. arXiv preprint arXiv:1801.00631 (2018)
27. Neubig, G.: Travatar: a forest-to-string machine translation engine based on tree transducers. In: Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics: System Demonstrations, pp. 91–96 (2013)
28. Oda, Y.; Fudaba, H.; Neubig, G.; Hata, H.; Sakai, S.; Toda, T.; Nakamura, S.: 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), pp. 574–584. IEEE. https://doi.org/10.1109/ase.2015.36 (2015)
29. Papineni, K.; Roukos, S.; Ward, T.; Zhu, W.J.: 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, pp. 311–318. https://doi.org/10.3115/1073083.1073135 (2002)
30. Pineda, F.J.: Dynamics and architecture for neural computation. J. Complex. 4(3), 216–245 (1988). https://doi.org/10.1016/0885-064X(88)90021-0
31. Roitero, K.; Soprano, M.; Portelli, B.; Spina, D.; Della Mea, V.; Serra, G.; Mizzaro, S.; Demartini, G.: The Covid-19 infodemic: can the crowd judge recent misinformation objectively? In: Proceedings of the 29th ACM International Conference on Information & Knowledge Management, pp. 1305–1314. https://doi.org/10.1145/3340531.3412048 (2020)
32. Soprano, M.; Roitero, K.; Bombassei De Bona, F.; Mizzaro, S.: Crowd_frame: a simple and complete framework to deploy complex crowdsourcing tasks off-the-shelf. In: Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining, pp. 1605–1608. https://doi.org/10.1145/3488560.3502182 (2022)
33. Sutskever, I.; Vinyals, O.; Le, Q.V.: Sequence to sequence learning with neural networks. In: Advances in Neural Information Processing Systems, pp. 3104–3112 (2014)
34. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, Ł.; Polosukhin, I.: Attention is all you need. In: Advances in Neural Information Processing Systems, pp. 5998–6008 (2017)
35. Voormann, H.; Gut, U.: Agile corpus creation. Corpus Linguist. Linguist. Theory 4(2), 235–251 (2008). https://doi.org/10.1515/CLLT.2008.010
36. Vulovic, M.: Crowdsourcing for enterprises. In: Congress on Services-I, pp. 686–692. IEEE. https://doi.org/10.1109/SERVICES-I.2009.56 (2009)
37. Weaver, W.: Translation. In: Proceedings of the Conference on Mechanical Translation (1952)
38. Williams, R.J.; Zipser, D.: A learning algorithm for continually running fully recurrent neural networks. Neural Comput. 1(2), 270–280 (1989). https://doi.org/10.1162/neco.1989.1.2.270
39. Xu, S.; Xiong, Y.: Automatic generation of pseudocode with attention seq2seq model. In: 2018 25th Asia-Pacific Software Engineering Conference (APSEC), pp. 711–712. IEEE. https://doi.org/10.1109/APSEC.2018.00101 (2018)
40. Zaidan, O.F.; Callison-Burch, C.: Crowdsourcing translation: professional quality from non-professionals. In: Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1, Association for Computational Linguistics, pp. 1220–1229 (2011)