Can NMT Understand Me? Towards Perturbation-based Evaluation of NMT Models for Code Generation

Pietro Liguori
University of Naples Federico II
Naples, Italy
pietro.liguori@unina.it

Cristina Improta
University of Naples Federico II
Naples, Italy
crist.improta@studenti.unina.it

Simona De Vivo
University of Naples Federico II
Naples, Italy
simona.devivo@unina.it

Roberto Natella
University of Naples Federico II
Naples, Italy
roberto.natella@unina.it

Bojan Cukic
University of North Carolina at Charlotte
Charlotte, North Carolina, USA
bcukic@uncc.edu

Domenico Cotroneo
University of Naples Federico II
Naples, Italy
cotroneo@unina.it

ABSTRACT

Neural Machine Translation (NMT) has reached a level of maturity to be recognized as the premier method for the translation between different languages and aroused interest in different research areas, including software engineering. A key step to validate the robustness of the NMT models consists in evaluating the performance of the models on adversarial inputs, i.e., inputs obtained from the original ones by adding small amounts of perturbation. However, when dealing with the specific task of the code generation (i.e., the generation of code starting from a description in natural language), it has not yet been defined an approach to validate the robustness of the NMT models. In this work, we address the problem by identifying a set of perturbations and metrics tailored for the robustness assessment of such models. We present a preliminary experimental evaluation, showing what type of perturbations affect the model the most and deriving useful insights for future directions.

CCS CONCEPTS

• **Computing methodologies** → *Machine translation.*

KEYWORDS

neural machine translation, robustness testing, code generation, adversarial inputs

ACM Reference Format:

Pietro Liguori, Cristina Improta, Simona De Vivo, Roberto Natella, Bojan Cukic, and Domenico Cotroneo. 2022. Can NMT Understand Me? Towards Perturbation-based Evaluation of NMT Models for Code Generation. In *The 1st Intl. Workshop on Natural Language-based Software Engineering (NLBSE'22), May 21, 2022, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3528588.3528653

NLBSE'22, May 21, 2022, Pittsburgh, PA, USA

ACM ISBN 978-1-4503-9343-0/22/05…$15.00
https://doi.org/10.1145/3528588.3528653

1 INTRODUCTION

As in many areas of artificial intelligence, deep neural networks have become the dominant paradigm for machine translation, bringing impressive improvements in the quality of the translation, and continuously moving forward the state-of-the-art performance [21].

Unlike traditional phrase-based translation, which consists of many small sub-components tuned separately, Neural Machine Translation (NMT) attempts to build and train a single, large neural network that reads a sentence and outputs a correct translation [4]. NMT has reached a level of maturity to be recognized as the premier method for the translation between different languages [48] and aroused interest in different research areas, including software engineering. In particular, the *code generation* task, also lately referred to as *semantic parsing* [49, 53], is an emerging and important application of NMT. It consists in the automatic translation of an *intent* in natural language (NL), such as the English language, into a *code snippet* written in a specific programming language. Indeed, NMT has been extensively used for generating programs (e.g., Python [51] and Java [29]), or to perform other programming tasks, such as code completion [9, 39], the generation of UNIX commands [27, 28], etc. Recently, NMT techniques have been also adopted to automatically generate code for software exploits starting from the description in natural language [24–26].

A common situation in any translation task from NL to programming language is the gap between the natural language used in the corpora and the natural language actually used by programmers. As a matter of fact, the corpora used for NMT models are often too "literal" and cumbersome to be realistically used by programmers. For example, in the *Shellcode_IA32* dataset [24, 25] used for the generation of assembly code from natural language, the *intent*, i.e., the natural language description, *"Push the contents of eax onto the stack"* takes longer than writing the assembly instruction "`push eax`". The *Django* dataset [36], which is widely used for evaluating neural machine translation task from English to Python [16, 50, 52], contains numerous Python code snippets that are relatively short (e.g., "`chunk_buffer = BytesIO(chunk)`") described with with English statements that are definitely longer than the snippets (*"evaluate the function BytesIO with argument chunk, substitute it for chunk_buffer."*). Again, in the *CoNaLa* dataset [50], we can find shortcode snippets (e.g., "`GRAVITY = 9.8`") described with longer English intents (*"assign float 9.8 to variable GRAVITY"*).

Since different users express the English intents in their own way, NMT models need to be robust against gaps between the actual intents and the ones in the corpora. A key approach typically used in machine learning research is to perform *robustness testing* of models, i.e., to evaluate the performance of the models when dealing with unexpected inputs, and to identify cases of misclassification. In particular, robustness testing has been adopted to identify security issues in machine learning models, by crafting *adversarial inputs* [55], i.e., inputs obtained from the original ones by adding small amounts of perturbation, which a malicious attacker may generate to mislead the model. These kinds of attacks on the inputs were first investigated for computer vision systems. Recent studies also addressed this problem in the context of language translation (e.g., from English to Chinese) by injecting noise in the input at different linguistic levels [5, 18, 23].

Given the gap discussed above, NMT models may not be robust to intents that are valid descriptions of the code, but that follow different styles or have different levels of detail compared to the training corpus. If NMT models are unable to handle this variability, they would be too inflexible and hamper the productivity of the programmers, hence limiting their usability in practice. Therefore, to evaluate the robustness of the NMT models, we aim to introduce non-arbitrary perturbations, e.g., variations from well-intentioned users. This is still an open research problem: while images can be easily perturbed without losing their original meaning and semantics, perturbing natural language can be much more challenging.

In light of these considerations, our work provides three key contributions:

- We propose a set of perturbations to evaluate the robustness of NMT models for the code generation task. The set includes both perturbations already used in previous studies and identified as suitable for the code generation task, and novel ad-hoc perturbations for the code generation task;
- We identify a set of metrics tailored for the robustness evaluation of NMT models under different levels of perturbations. Indeed, a significant aspect to take into account is that, while a perturbed intent may produce an output different from the original one, it may still preserve the semantic and syntactic correctness according to the target programming language;
- We present a preliminary experimental analysis to evaluate the robustness of an NMT model when dealing with perturbations in the intents. We show what perturbations affect the model the most and derive useful insights for future research.

In the following, Section 2 discusses related work; Section 3 proposes a set of perturbations to evaluate the robustness of NMT models; Section 4 describes the metrics for the evaluation of the model robustness; Section 5 presents the preliminary evaluation; Section 6 concludes the paper.

2 RELATED WORK

State-of-the-art provides several recent works on adversarial natural language processing (NLP) covering different research topics such as sentiment analysis, toxic content detection, machine comprehension, and numerous similar contexts.

Previous works explored and analyzed noise generation at different linguistic levels, i.e., character, word, and sentence-level. At character-level, text can be perturbed by inserting, deleting, randomizing, or swapping characters to study the effects on natural language processing (NLP) tasks [5, 17, 23]; furthermore, homographic attacks can be employed to mislead models in question answering [47], and QWERTY character swapping can be used to reproduce keyboard typos [5]. At the word level, words in a sentence can be substituted with different random words, similar words in the word embedding space, or meaning-preserving words [18, 23, 30]. Regarding sentence-level manipulation, paraphrasing, back translation, and reordering are some of the approaches used to produce a syntactically and semantically similar phrase to fool the models [18].

Heigold *et al.* [17] studied the effects of word scrambling and random noise insertion in NLP tasks such as morphological tagging and machine translation, both regarding English and German languages. The perturbation strategies used include character flips and swaps of neighboring characters to imitate typos. Belinkov *et al.* [5] analyzed how natural noise, i.e., the natural occurring of errors from available corpora, and synthetic noise, i.e., character swaps aimed to reproduce misspellings and keyboard typos, affect character-based NMT models, focusing on machine translation from natural languages such as French, German and Czech to English. The authors used a black-box adversarial training setting and found that these architectures have a tendency to break when presented with noisy datasets.

In the context of the machine comprehension and question answering, Wu *et al.* [47] investigated what type of text perturbation leads to the most high-confidence misclassifications and which embeddings are more susceptible to adversarial attacks. They used homographic attacks, synonyms substitutions, and sentence paraphrasing to investigate models' performances in a perturbed context paragraph. Huang *et al.* [18] conducted the first empirical study to evaluate the effect of adversarial examples on SOTA neural semantic parsers by perturbing existing benchmark corpora with four different word-level operations and two sentence-level operations and applying meaning-preserving constraints.

Recent works introduced tools and frameworks for the generation of adversarial inputs. TextBugger [23] is a framework to efficiently generate utility-preserving adversarial texts under both white-box and black-box settings to evaluate the robustness of various popular, real-world online text classification systems. In the white-box scenario, the attacker is aware of the model's architecture and parameters, so they first find important words by computing the Jacobian matrix of the classifier, then choose an optimal perturbation from the generated five kinds of perturbations. In the black-box scenario, the attacker does not have information on the model's internals, so they first find the significant sentences and then use a scoring function to find the main words to manipulate. Specifically, their targets are sentiment analysis and toxic contents detection models. Gao *et al.* [13] presented DeepWordBug, an algorithm to effectively generate small text perturbations in a black-box setting. The authors use novel scoring and ranking techniques to identify the most important words that, if perturbed, lead the model to a misclassification. Concerning these perturbations, they apply character-level transformations such as swap, substitution, deletion, and insertion. Cheng *et al.* [6] proposed Seq2Sick, an optimization-based framework to generate adversarial examples for sequence-to-sequence neural network models. The authors implemented novel loss functions to conduct a non-overlapping attack and targeted keyword attack, to handle the almost infinite output space.

Our work can be considered complementary to the previous ones. Indeed, although the robustness evaluation of the deep learning models has been widely addressed by the previous research, to the best of our knowledge, the use of adversarial attacks has not been applied to validate the usability of the NMT models in the code generation task.

3 PERTURBATIONS IN CODE GENERATION

To measure the robustness of the NMT models in the code generation task, we are interested to analyze the models with respect to their inputs (i.e., intents in natural language). Indeed, the description of a natural language code snippet by different authors may be characterized by different writing styles and capabilities. A sentence may be rephrased through multiple synonyms, it may order words in different ways, it may lack some significant detail, or be too specific.

Therefore, although character-level perturbations may be meaningful to study the sensitivity of NMT models to human errors (e.g., typos), in this work we focus on perturbing words in a sentence but still preserving the original meaning of the intents. In particular, we focus on two types of perturbations: the *unseen synonyms*, and the *missing information*. The former can be used to evaluate the performance of the translation task when the intents significantly diverge from the terms used in the corpus (e.g., word synonyms). The latter, instead, is suitable to assess the models' performance when programmers may omit information that would be redundant, such as information implicitly contained in the sentence, or information already stated in previous intents. Both these aspects are important for the usability of NMT models.

3.1 Unseen Synonyms

A robust model should be resistant to noise caused by Unseen Synonyms and should produce the same output when presented with two semantically similar intents. Therefore, it is interesting to our cause to substitute words within an intent either with a synonym from a lexical database (e.g., WordNet [32]) or with their neighbor in the *word embedding* space (i.e., a numerical representation of the words) [31] and examine the model's response.

However, blindly replacing words with their synonym may lead to the loss of the sentence's original meaning since terms with small word embedding distance may belong to the same context but not be semantically similar (e.g., the words "*father*" and "*mother*"). Moreover, code generation is a highly specific domain, thus some words have a precise meaning and cannot simply be replaced with another. As a simple example, consider the intent "*clear the contents of the register*". A valid perturbation on the input can reasonably lead to the sentence "*empty the contents of the register*", but not to "*purify the contents of the register*" since the verb "*purify*" is clearly out of the programming context.

To overcome these issues, a solution could be limiting the space of the possible words by creating a dictionary of words used to describe programming code (e.g., by using books and tutorials as reference). However, building a vocabulary from scratch containing only words used in the programming language context may be too time-consuming or, even worse, unfeasible. A more practical approach consists in applying *constraints* on the transformation method. An example of constraints for synonyms is to ensure that the words can be replaced only with one of its top k-nearest neighbors in the source embedding space before computing a similarity score to filter out dissimilar terms [18, 23].

The use of the constraints for the choice of synonyms also allows limiting situations in which the new word produces a different meaning from the original intent. Referring to the previous example "*clear the contents of the register*", a synonym without constraints for the verb "*clear*" is the verb "*shift*" [22], which is definitely used in the programming code context, but with a completely different purpose. Taking this into account, we identified three different types of constraints useful to perform word substitution in the intents:

- ***Word Embedding Distance***: It measures the value of the cosine similarity between word embeddings. The constraint-based on the word embedding distance performs the substitution of words only if the value of the cosine similarity between the replaced word and its synonym is higher than a specified value;
- ***BERT-score***: It measures token similarity between two texts using *contextual embedding* [54]. Contextual embeddings, such as BERT, can generate different vector representations for the same word in different sentences depending on the surrounding words, which form the context of the target word [8]. By using the constraint on the BERT-score, the substitution of the words is performed only if the score between the replaced word and its synonym is higher than a specified value;
- ***Part-of-Speech (POS) tag***: It is the process of marking up a word in a text as corresponding to a particular part of speech. The constraint using the POS tag allows the substitutions only if the replaced word and its synonym have the same POS tag (e.g., a verb should be replaced only with a verb, a noun with a noun, etc.).

3.2 Missing Information

In the context of code generation, the removal of information becomes of particular interest since the intents of the corpora are usually concise and detailed, thus they may completely lose their original meaning even if only a single word is omitted or removed. Nevertheless, this represents a common situation because users can inadvertently neglect some details, or avoid specifying information implicitly contained in the intent or included in the previous ones.

The action of removing information from the intents can be performed randomly [18] or following particular criteria. In our case, it is interesting to analyze how the model's behavior and text comprehension varies when important information is missing. This kind of perturbation is yet to be explored in the code generation task. For this reason, we first define what *important information* means

Table 1: Examples of omitted information on the same intent. ~~Slashed~~ text refers to the omitted words.

Perturbation	Intent
None (Original Intent)	Store the shellcode pointer in the ESI register.
Action-related words	~~Store~~ the shellcode pointer in the ESI register.
Language-related words	Store the shellcode pointer in the ESI ~~register~~.
Value-related words	Store the shellcode pointer in the ~~ESI~~ register.

in our context before removing one or more significant words from each intent. When commenting on a code snippet, there are two fundamental aspects to be considered: i) *what action the user aims to take*, and ii) *what is the target of the action*. For example, the simple intent "*call the myfunc function*" contains the action, i.e., the verb *call*, and the target, i.e., the *myfunc function*. The target of the action can be further divided in the value of the target (i.e., the name of the function), and the word specifying the type (i.e., the word "function"). Based on these assumptions, we identify three main categories of significant words in the intents:

- ***Action-related words***: Words containing the information related to the actions of the intent, which are usually specified by the verbs (e.g., jump, add, call, declare, etc.);
- ***Language-related words***: Words related to the target programming language (e.g., the words "class", "function", "variable", "register", "label", etc.);
- ***Value-related words***: They include the name or the values of the variables, the names of functions, classes and, where available (e.g., assembly language), the value of the memory addresses, and of name of registers or labels.

Table 1 shows the different types of word removal perturbations on the English intent "*Store the shellcode pointer in the ESI register.*", which is commonly used to decode shellcodes in assembly language for the IA-32 architecture [26]. The table shows examples in which the intent still preserves its meaning even without specifying the omitted words. The verb store and the keyword register are implicit (the pointer of the shellcode can be only moved to `ESI`, which is, in fact, a register), while the name of the register can be derived from the context of the program (the `ESI` register is commonly used to store the shellcode). However, this is not always the case. For example, a list can be created or deleted, therefore, not specifying the verb can imply an opposite action. A user can create non-primitive data structures, but the type of the structure (e.g., list, dictionary, etc.) has to be specified to perform a correct prediction. Finally, values and names have a broader range of meaning and usage, hence it might be more difficult for a model to learn and predict their behavior.

4 EVALUATION METRICS

When the input is perturbed, we need to assess if the output predicted by the model is *correct*, i.e., it is equivalent to the reference of the test set (i.e., the *ground-truth*). However, the robustness evaluation of the NMT models is not trivial in that we need to take into account different aspects.

The *ambiguity of the natural language* implies that the same sentence can have different meanings and, therefore, it can be translated into different and non-equivalent programming code snippets. This problem is further exacerbated by the introduction of perturbations on the intents (e.g., word synonyms, omitted words, etc.). A significant takeaway is that, although the model's prediction can be incorrect with respect to the reference, it can result in the right translation of the perturbed intent.

As well as in natural language we can express the same intents with different sentences (e.g., through the use of synonyms, sentence paraphrases, etc.), the *equivalence of code snippets* allows programmers to write different but equivalent programming code. This means that, even if the output predicted by the model differs from the ground truth, it can still be considered correct.

In the light of the above considerations, the choice of the right metrics is a key step to assess the robustness of the NMT models in the code generation task. In the remainder of this section, we describe a set of metrics suitable for this specific research problem.

4.1 Automatic Metrics

Automatic metrics are a valuable means to assess the quality of the code generation task since they are reproducible, easy to be tuned, and time-saving.

Among the most commonly used metrics in machine translation, we definitely find the ***Bilingual Evaluation Understudy*** (BLEU) score and the ***Exact Match Accuracy*** (EM) [3, 14, 29, 44, 51–53]. BLEU score [37] is based on the concept of *n-gram*, i.e., the adjacent sequence of *n items* (e.g., syllables, letters, words, etc.) from a given example of text or speech. This metric measures the degree of n-gram overlapping between the strings of words produced by the model and the references at the corpus level. BLEU measures translation quality by the accuracy of translating n-grams to n-grams, for n-gram of size 1 to 4 [15]. The Exact Match Accuracy, instead, measures the fraction of the exact match between the output predicted by the model and the reference in the test set.

Further metrics useful in the context of the robustness evaluation are based on sub-string analysis [41]. For example, the ***LCS-based metric*** measures the normalized similarity by calculating the longest common sub-sequence between the translation to the output of the original input and the translation to the output of the mutated input, respectively. The ***Ed-based metric*** measures the edit-distance between two strings, where edit-distance is a way of quantifying dissimilarity between two strings (i.e., the minimum number of operations required to make two strings equal).

4.2 Manual Metrics

Although automatic metrics can evaluate the differences between the output predicted by the model and the reference of the test set, the automatic evaluation can not truly reflect the *correctness* of the predicted code when it differs from the reference of the test set [40].

Therefore, to properly assess the robustness of the models, we need to evaluate the quality of the code snippets by using manual metrics, i.e., metrics that are computed through human inspection. In the context of the code generation task, in order to estimate the correctness of the output, we need to look into the code with respect to i) *how the code is written*, i.e., the *code syntax* and ii) *what the code actually does*, i.e., the *code semantic*.

Therefore, a key step to evaluate the correctness of the model's output is to estimate both the ***Syntactic Accuracy*** and ***Semantic Accuracy*** (also ***Execution Accuracy***) [26, 45], which measure the fraction of syntactic and semantic correct predictions over all the predictions, respectively. While the former gives insights into whether the code is correct according to the rules of the target language, the latter indicates whether the output is the exact translation of the intent into the target programming language. The semantic correctness implies syntax correctness, while a snippet can be syntactically correct but semantically incorrect. Of course, the syntactic incorrectness also implies the semantic one [26].

Different from the syntax, the evaluation of the code snippet semantic depends by definition on the intent considered as reference. For example, semantic accuracy assesses if the prediction, after the perturbation, is correct according to the intent of the original test set. The ***Perturbation Accuracy*** [18], instead, computes the fraction of predictions considered correct with respect to the perturbed version of the intents, i.e., the output is considered correct if it is the exact translation of the perturbed input into the target programming language. This metric is of particular interest when the perturbation introduces ambiguity or, even worse, changes the semantic meaning of the intent. In this case, indeed, the output may be considered correct according to the perturbed version of the intent but incorrect when considering the intent in the original test set (not perturbed) as the reference, and vice-versa.

A further metric of interest in this context is the ***Robust Accuracy*** [18]. The metric focuses the evaluation on the intents of the original test set which are properly predicted by the model without any perturbations, discarding the ones mispredicted by the model. To assess the robustness, it computes the fraction of correct predictions under perturbations over the subset of the previous correct outputs. The metric is based on the assumption that to evaluate the model's robustness, it may be meaningless to include intents leading to the model's mispredictions, regardless of the perturbations.

5 PRELIMINARY EVALUATION

We performed a set of preliminary experiments to assess the model's ability to tolerate noise and still produce accurate outputs. We targeted the Seq2Seq model since it is widely used in a variety of neural machine translation tasks. In particular, we adopted the Seq2Seq model with Bahdanau-style attention mechanism [4]. We implemented the Seq2Seq model using `xnmt` [35]. We used an Adam optimizer [20] with $\beta_1 = 0.9$ and $\beta_2 = 0.999$, while the learning rate α is set to 0.001. We set all the remaining hyper-parameters in a basic configuration: layer dimension = 512, layers = 1, epochs (with early stopping enforced) = 200, beam size = 5. We did not use any pre-processing or post-processing steps to help the model in the generation of the output since we are interested in quantifying the impact of the noise rather than maximizing the performance.

To feed the model, we used the assembly dataset released by Liguori *et al.* [26] for automatically generating assembly from natural language descriptions. This dataset consists of assembly instructions, commented in English language, which were collected from shellcodes for *IA-32* and written for the *Netwide Assembler* (NASM) for Linux [10]. The dataset contains 3,715 unique pairs of assembly code snippets/English intents: 3105 pairs in the training set, 305 pairs in the dev set, and 305 pairs in the test set.

Our preliminary evaluation interested a subset of the perturbations described in § 3. In particular, we evaluated the robustness of the model by using three different types of perturbations:

- *Unseen synonyms with constraints using the BERT-score and POS tag*: We applied a transformation only when the synonym, chosen as a neighbor in the word embedding space, and the original word have a BERT-score similarity greater than 0.85 and the same POS tag. We empirically choose a high value for the BERT-score similarity to introduce diversity in the intent without losing the original meaning. We randomly replaced the 10% of the selected words within a single intent, ensuring that at least one word is swapped with its synonym in each intent.
- *Omission of the action-related words*: We removed the verbs from every intent in the test set using a POS tagger (e.g., "define", "add", etc.);
- *Omission of the language-related words*: We removed the words related to the assembly programming language from each intent (e.g., "register", "label", etc.) in the test set.

We used TextAttack [33], a Python framework for data augmentation in NLP, to replace words with synonyms and apply the constraints, and Flair POS-tagging model [2] as part-of-the speech tagger. The TextAttack framework implements the *word swap by embedding* transformation, i.e., a novel *counter-fitting* method for injecting linguistic constraints into word vector space representations, which post-processes word vectors to improve their usefulness for tasks involving the semantic similarity judgements [34].

We perturbed all the intents of the test set (i.e., the test set is 100% perturbed), while we did not add any noise in the training and dev sets. All experiments were performed on a Linux OS running on a virtual machine with 8 CPU cores and 8 GB RAM.

5.1 Automatic Evaluation

We first evaluated the performance of the code generation task in terms of automatic metrics both on the original and on the perturbed test set. The key idea is that, the more the performance decreases compared to the one of the original test set, the more the model is affected by the perturbation. As automatic metrics, we used the BLEU-4, the exact match accuracy (EM), the Ed-based metric (ED), and the LCS-based metric (LCS). Table 2 shows the results.

Among the type of perturbations, the use of unseen synonyms with constraints less affect the performance of the model. The model, indeed, showed to be robust when dealing with word synonyms, also because the high BERT-score similarity set as constraint

Table 2: Automatic evaluation of different types of adversarial inputs. The worst performance is red/bold.

Test Set	BLEU-4 (%)	EM (%)	ED (%)	LCS (%)
Original (no perturbations)	17.39	19.67	62.48	64.70
Unseen synonyms with const.	16.03	18.11	59.53	62.66
Action-related words	13.45	**13.11**	**53.19**	**56.08**
Language-related words	**13.09**	16.39	56.09	58.48

limited the amount of diversity of the words. The explicit information removal from the intents, instead, negatively impacted the model's prediction. In particular, the removal of the action-related words implied the worst performance in terms of exact match accuracy, Ed-based metric, and the LCS-based metric, while the model shows the worst BLEU-4 when dealing with the removal of the language-related words.

5.2 Manual Evaluation

The previous metrics do not provide a complete and robust evaluation: EX only measures exact match and cannot thus give credit to semantically correct code that is different from the reference, while it is not clear whether BLEU provides an appropriate proxy for measuring semantics in the code generation task [51]. Therefore, we further studied the impact of perturbations on the code generation task by performing a manual evaluation. In particular, for each code snippet predicted by the model, all authors evaluated both the syntactic and semantic accuracy, independently. To reduce the possibility of errors in the manual analysis, multiple authors discussed cases of discrepancy, obtaining a consensus for the syntactic and semantic correctness. Table 3 shows the percentage of syntactically (SYN) and semantically (SEM) correct snippets over all the examples of the test set.

The table shows that the use of perturbations does not negatively impact the model's ability to predict syntactically correct code snippets. Even better, the removal of action-related words slightly increased the performance of the syntactical accuracy of the model. Through an in-depth analysis of the model's outputs, we found that the removal of verbs resulted in the prediction of relatively simple code snippets (in terms of length) and, thus, syntactically correct, but which do not represent the exact translation of the original intent. As a matter of fact, the removal of the action-related words resulted in the most significant dropping of the performance in terms of semantic accuracy. Similarly, the use of unseen synonyms and the removal of language-related words negatively affected the semantic accuracy of the model, but the dropping of semantic accuracy is more limited. In particular, the table shows that the semantic accuracy of the outputs achieved when the language-related words are omitted is close to the one of the original test set.

We conducted a *paired-sample T-test* to compare the syntactic and the semantic accuracy values of the code snippets predicted under perturbations with the ones of the original test set (given the same example). We found that the differences in the syntactic accuracy obtained under different types of perturbations are not statistically significant from the one of the original test set. Concerning the semantic accuracy, the hypothesis testing suggested

Table 3: Manual evaluation of different types of adversarial inputs. The worst performance is in red/bold ($*$ = p<0.01).

Test Set	SYN (%)	SEM (%)
Original (no perturbations)	88.52	22.95
Unseen synonyms with const.	**87.87**	18.36*
Action-related words	89.51	**14.75***
Language-related words	88.20	20.98

that the performance achieved with the use of unseen synonyms and the removal of the action-related words are statistically significant with $p < 0.01$. The difference of the performance achieved with the removal of the language-related words, instead, did not result in any statistical evidence.

A significant takeaway from this preliminary evaluation is that in the generation of assembly code from natural language, the NMT model: i) can deal with the use of synonyms in the intents and, therefore, different ways of describing the code by different users; ii) is very robust to non-explicit information on language-related words, such as keywords; iii) is hugely affected by intents where actions are non explicitly stated.

6 CONCLUSION AND FUTURE WORK

We addressed the problem of evaluating the robustness of the NMT models for the code generation task by proposing a set of perturbations and metrics to assess the impact of the models when dealing with different inputs. We performed a preliminary evaluation of the Seq2Seq model in the assembly code generation from natural language description and showed how different perturbations on the inputs affect the model's performance.

As future work, we aim to extend the robustness evaluation to different DL-based architectures [12, 42]. We are also investigating different solutions to make NMT models more robust. In particular, we foresee the use of the *adversarial training* (i.e., injecting perturbed inputs into training data to increase robustness) [7, 11, 19] and the development of solutions that help the models to derive the missing or implicit information from the context of the program [1, 38, 43, 46].

ACKNOWLEDGMENTS

This work has been partially supported by the University of Naples Federico II in the frame of the Programme F.R.A., project id OSTAGE.

REFERENCES

[1] Ruchit Rajeshkumar Agrawal, Marco Turchi, and Matteo Negri. 2018. Contextual handling in neural machine translation: Look behind, ahead and on both sides. In *21st Annual Conference of the European Association for Machine Translation*. 11–20.

[2] Alan Akbik, Duncan Blythe, and Roland Vollgraf. 2018. Contextual String Embeddings for Sequence Labeling. In *Proceedings of the 27th International Conference on Computational Linguistics, COLING 2018, Santa Fe, New Mexico, USA, August 20-26, 2018*, Emily M. Bender, Leon Derczynski, and Pierre Isabelle (Eds.). Association for Computational Linguistics, 1638–1649. https://aclanthology.org/C18-1139/

[3] Erfan Al-Hossami and Samira Shaikh. 2022. A Survey on Artificial Intelligence for Source Code: A Dialogue Systems Perspective. *CoRR* abs/2202.04847 (2022). arXiv:2202.04847 https://arxiv.org/abs/2202.04847

[4] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural Machine Translation by Jointly Learning to Align and Translate. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1409.0473

[5] Yonatan Belinkov and Yonatan Bisk. 2018. Synthetic and Natural Noise Both Break Neural Machine Translation. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net. https://openreview.net/forum?id=BJ8vJebC-

[6] Minhao Cheng, Jinfeng Yi, Pin-Yu Chen, Huan Zhang, and Cho-Jui Hsieh. 2020. Seq2Sick: Evaluating the Robustness of Sequence-to-Sequence Models with Adversarial Examples. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*. AAAI Press, 3601–3608. https://aaai.org/ojs/index.php/AAAI/article/view/5767

[7] Yong Cheng, Lu Jiang, and Wolfgang Macherey. 2019. Robust Neural Machine Translation with Doubly Adversarial Inputs. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, Anna Korhonen, David R. Traum, and Lluís Màrquez (Eds.). Association for Computational Linguistics, 4324–4333. https://doi.org/10.18653/v1/p19-1425

[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, Jill Burstein, Christy Doran, and Thamar Solorio (Eds.). Association for Computational Linguistics, 4171–4186. https://doi.org/10.18653/v1/n19-1423

[9] Ian Drosos, Titus Barik, Philip J. Guo, Robert DeLine, and Sumit Gulwani. 2020. Wrex: A Unified Programming-by-Example Interaction for Synthesizing Readable Code for Data Scientists. In *CHI '20: CHI Conference on Human Factors in Computing Systems, Honolulu, HI, USA, April 25-30, 2020*, Regina Bernhaupt, Florian 'Floyd' Mueller, David Verweij, Josh Andres, Joanna McGrenere, Andy Cockburn, Ignacio Avellino, Alix Goguey, Pernille Bjøn, Shengdong Zhao, Briane Paul Samson, and Rafal Kocielnik (Eds.). ACM, 1–12. https://doi.org/10.1145/3313831.3376442

[10] J. Duntemann. 2000. *Assembly Language Step-by-Step: Programming with DOS and Linux*. Wiley. https://books.google.it/books?id=7-h1RPbnTTAC

[11] Javid Ebrahimi, Daniel Lowd, and Dejing Dou. 2018. On Adversarial Examples for Character-Level Neural Machine Translation. In *Proceedings of the 27th International Conference on Computational Linguistics, COLING 2018, Santa Fe, New Mexico, USA, August 20-26, 2018*, Emily M. Bender, Leon Derczynski, and Pierre Isabelle (Eds.). Association for Computational Linguistics, 653–663. https://aclanthology.org/C18-1055/

[12] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020 (Findings of ACL, Vol. EMNLP 2020)*, Trevor Cohn, Yulan He, and Yang Liu (Eds.). Association for Computational Linguistics, 1536–1547. https://doi.org/10.18653/v1/2020.findings-emnlp.139

[13] Ji Gao, Jack Lanchantin, Mary Lou Soffa, and Yanjun Qi. 2018. Black-Box Generation of Adversarial Text Sequences to Evade Deep Learning Classifiers. In *2018 IEEE Security and Privacy Workshops, SP Workshops 2018, San Francisco, CA, USA, May 24, 2018*. IEEE Computer Society, 50–56. https://doi.org/10.1109/SPW.2018.00016

[14] Carlos Gemmell, Federico Rossetto, and Jeffrey Dalton. 2020. Relevance Transformer: Generating Concise Code Snippets with Relevance Feedback. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval, SIGIR 2020, Virtual Event, China, July 25-30, 2020*, Jimmy Huang, Yi Chang, Xueqi Cheng, Jaap Kamps, Vanessa Murdock, Ji-Rong Wen, and Yiqun Liu (Eds.). ACM, 2005–2008. https://doi.org/10.1145/3397271.3401215

[15] Lifeng Han, Gareth J. F. Jones, and Alan F. Smeaton. 2021. Translation Quality Assessment: A Brief Survey on Manual and Automatic Methods. *CoRR* abs/2105.03311 (2021). arXiv:2105.03311 https://arxiv.org/abs/2105.03311

[16] Shirley Anugrah Hayati, Raphael Olivier, Pravalika Avvaru, Pengcheng Yin, Anthony Tomasic, and Graham Neubig. 2018. Retrieval-Based Neural Code Generation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (Eds.). Association for Computational Linguistics, 925–930. https://doi.org/10.18653/v1/d18-1111

[17] Georg Heigold, Stalin Varanasi, Günter Neumann, and Josef van Genabith. 2018. How Robust Are Character-Based Word Embeddings in Tagging and MT Against Wrod Scramlbing or Randdm Nouse?. In *Proceedings of the 13th Conference of the Association for Machine Translation in the Americas, AMTA 2018, Boston, MA, USA, March 17-21, 2018 - Volume 1: Research Papers*, Colin Cherry and Graham Neubig (Eds.). Association for Machine Translation in the Americas, 68–80. https://aclanthology.org/W18-1807/

[18] Shuo Huang, Zhuang Li, Lizhen Qu, and Lei Pan. 2021. On Robustness of Neural Semantic Parsers. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume, EACL 2021, Online, April 19 - 23, 2021*, Paola Merlo, Jörg Tiedemann, and Reut Tsarfaty (Eds.). Association for Computational Linguistics, 3333–3342. https://aclanthology.org/2021.eacl-main.292/

[19] Yatu Ji, Hongxu Hou, Junjie Chen, and Nier Wu. 2020. Adversarial Training for Unknown Word Problems in Neural Machine Translation. *ACM Trans. Asian Low Resour. Lang. Inf. Process.* 19, 1 (2020), 17:1–17:12. https://doi.org/10.1145/3342482

[20] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1412.6980

[21] P. Koehn. 2020. *Neural Machine Translation*. Cambridge University Press. https://books.google.it/books?id=iRzhDwAAQBAJ

[22] Oxford Languages. 2022. Oxford Languages and Google - English. https://languages.oup.com/google-dictionary-en/.

[23] Jinfeng Li, Shouling Ji, Tianyu Du, Bo Li, and Ting Wang. 2019. TextBugger: Generating Adversarial Text Against Real-world Applications. In *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019*. The Internet Society. https://www.ndss-symposium.org/ndss-paper/textbugger-generating-adversarial-text-against-real-world-applications/

[24] Pietro Liguori, Erfan Al-Hossami, Domenico Cotroneo, Roberto Natella, Bojan Cukic, and Samira Shaikh. 2021. Shellcode_IA32: A Dataset for Automatic Shellcode Generation. In *Proceedings of the 1st Workshop on Natural Language Processing for Programming (NLP4Prog 2021)*. Association for Computational Linguistics, Online, 58–64. https://doi.org/10.18653/v1/2021.nlp4prog-1.7

[25] Pietro Liguori, Erfan Al-Hossami, Domenico Cotroneo, Roberto Natella, Bojan Cukic, and Samira Shaikh. 2022. Can we generate shellcodes via natural language? An empirical study. *Automated Software Engineering* 29, 1 (05 Mar 2022), 30. https://doi.org/10.1007/s10515-022-00331-3

[26] Pietro Liguori, Erfan Al-Hossami, Vittorio Orbinato, Roberto Natella, Samira Shaikh, Domenico Cotroneo, and Bojan Cukic. 2021. EVIL: Exploiting Software via Natural Language. In *2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE)*. 321–332. https://doi.org/10.1109/ISSRE52982.2021.00042

[27] Xi Victoria Lin, Chenglong Wang, Deric Pang, Kevin Vu, Luke Zettlemoyer, and Michael D. Ernst. 2017. *Program synthesis from natural language using recurrent neural networks*. Technical Report UW-CSE-17-03-01. University of Washington Department of Computer Science and Engineering, Seattle, WA, USA.

[28] Xi Victoria Lin, Chenglong Wang, Luke Zettlemoyer, and Michael D. Ernst. 2018. NL2Bash: A Corpus and Semantic Parser for Natural Language Interface to the Linux Operating System. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation, LREC 2018, Miyazaki, Japan, May 7-12, 2018*, Nicoletta Calzolari, Khalid Choukri, Christopher Cieri, Thierry Declerck, Sara Goggi, Kôiti Hasida, Hitoshi Isahara, Bente Maegaard, Joseph Mariani, Hélène Mazo, Asunción Moreno, Jan Odijk, Stelios Piperidis, and Takenobu Tokunaga (Eds.). European Language Resources Association (ELRA). http://www.lrec-conf.org/proceedings/lrec2018/summaries/1021.html

[29] Wang Ling, Phil Blunsom, Edward Grefenstette, Karl Moritz Hermann, Tomás Kociský, Fumin Wang, and Andrew W. Senior. 2016. Latent Predictor Networks for Code Generation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*. The Association for Computer Linguistics. https://doi.org/10.18653/v1/p16-1057

[30] Paul Michel, Xian Li, Graham Neubig, and Juan Miguel Pino. 2019. On Evaluation of Adversarial Perturbations for Sequence-to-Sequence Models. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, Jill Burstein, Christy Doran, and Thamar Solorio (Eds.). Association for Computational Linguistics, 3103–3114. https://doi.org/10.18653/v1/n19-1314

[31] Tomás Mikolov, Wen-tau Yih, and Geoffrey Zweig. 2013. Linguistic Regularities in Continuous Space Word Representations. In *Human Language Technologies:*

Conference of the North American Chapter of the Association of Computational Linguistics, Proceedings, June 9-14, 2013, Westin Peachtree Plaza Hotel, Atlanta, Georgia, USA, Lucy Vanderwende, Hal Daumé III, and Katrin Kirchhoff (Eds.). The Association for Computational Linguistics, 746–751. https://aclanthology.org/N13-1090/

[32] George A. Miller. 1995. WordNet: A Lexical Database for English. *Commun. ACM* 38, 11 (1995), 39–41. https://doi.org/10.1145/219717.219748

[33] John X. Morris, Eli Lifland, Jin Yong Yoo, Jake Grigsby, Di Jin, and Yanjun Qi. 2020. TextAttack: A Framework for Adversarial Attacks, Data Augmentation, and Adversarial Training in NLP. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, EMNLP 2020 - Demos, Online, November 16-20, 2020*, Qun Liu and David Schlangen (Eds.). Association for Computational Linguistics, 119–126. https://doi.org/10.18653/v1/2020.emnlp-demos.16

[34] Nikola Mrkšić, Diarmuid Ó Séaghdha, Blaise Thomson, Milica Gašić, Lina Rojas-Barahona, Pei-Hao Su, David Vandyke, Tsung-Hsien Wen, and Steve Young. 2016. Counter-fitting Word Vectors to Linguistic Constraints. In *Proceedings of HLT-NAACL*.

[35] Graham Neubig, Matthias Sperber, Xinyi Wang, Matthieu Felix, Austin Matthews, Sarguna Padmanabhan, Ye Qi, Devendra Singh Sachan, Philip Arthur, Pierre Godard, John Hewitt, Rachid Riad, and Liming Wang. 2018. XNMT: The eXtensible Neural Machine Translation Toolkit. In *Proceedings of the 13th Conference of the Association for Machine Translation in the Americas, AMTA 2018, Boston, MA, USA, March 17-21, 2018 - Volume 1: Research Papers*, Colin Cherry and Graham Neubig (Eds.). Association for Machine Translation in the Americas, 185–192. https://aclanthology.org/W18-1818/

[36] Yusuke Oda, Hiroyuki Fudaba, Graham Neubig, Hideaki Hata, Sakriani Sakti, Tomoki Toda, and Satoshi Nakamura. 2015. Learning to Generate Pseudo-Code from Source Code Using Statistical Machine Translation (T). In *30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015, Lincoln, NE, USA, November 9-13, 2015*, Myra B. Cohen, Lars Grunske, and Michael Whalen (Eds.). IEEE Computer Society, 574–584. https://doi.org/10.1109/ASE.2015.36

[37] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a Method for Automatic Evaluation of Machine Translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, July 6-12, 2002, Philadelphia, PA, USA*. ACL, 311–318. https://doi.org/10.3115/1073083.1073135

[38] Yves Scherrer, Jörg Tiedemann, and Sharid Loáiciga. 2019. Analysing concatenation approaches to document-level NMT in two different domains. In *Proceedings of the Fourth Workshop on Discourse in Machine Translation, DiscoMT@EMNLP 2019, Hong Kong, China, November 3, 2019*, Andrei Popescu-Belis, Sharid Loáiciga, Christian Hardmeier, and Deyi Xiong (Eds.). Association for Computational Linguistics, 51–61. https://doi.org/10.18653/v1/D19-6506

[39] Kensen Shi, David Bieber, and Rishabh Singh. 2020. TF-Coder: Program Synthesis for Tensor Manipulations. *CoRR* abs/2003.09040 (2020). arXiv:2003.09040 https://arxiv.org/abs/2003.09040

[40] Amanda Stent, Matthew Marge, and Mohit Singhai. 2005. Evaluating Evaluation Methods for Generation in the Presence of Variation. In *Computational Linguistics and Intelligent Text Processing, 6th International Conference, CICLing 2005, Mexico City, Mexico, February 13-19, 2005, Proceedings (Lecture Notes in Computer Science, Vol. 3406)*, Alexander F. Gelbukh (Ed.). Springer, 341–351. https://doi.org/10.1007/978-3-540-30586-6_38

[41] Zeyu Sun, Jie M. Zhang, Mark Harman, Mike Papadakis, and Lu Zhang. 2020. Automatic testing and improvement of machine translation. In *ICSE '20: 42nd International Conference on Software Engineering, Seoul, South Korea, 27 June - 19 July, 2020*, Gregg Rothermel and Doo-Hwan Bae (Eds.). ACM, 974–985. https://doi.org/10.1145/3377811.3380420

[42] Zeyu Sun, Qihao Zhu, Yingfei Xiong, Yican Sun, Lili Mou, and Lu Zhang. 2020. TreeGen: A Tree-Based Transformer Architecture for Code Generation. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*. AAAI Press, 8984–8991. https://ojs.aaai.org/index.php/AAAI/article/view/6430

[43] Jörg Tiedemann and Yves Scherrer. 2017. Neural Machine Translation with Extended Context. In *Proceedings of the Third Workshop on Discourse in Machine Translation, DiscoMT@EMNLP 2017, Copenhagen, Denmark, September 8, 2017*, Bonnie L. Webber, Andrei Popescu-Belis, and Jörg Tiedemann (Eds.). Association for Computational Linguistics, 82–92. https://doi.org/10.18653/v1/w17-4811

[44] Ngoc M. Tran, Hieu Tran, Son Nguyen, Hoan Nguyen, and Tien N. Nguyen. 2019. Does BLEU score work for code migration?. In *Proceedings of the 27th International Conference on Program Comprehension, ICPC 2019, Montreal, QC, Canada, May 25-31, 2019*, Yann-Gaël Guéhéneuc, Foutse Khomh, and Federica Sarro (Eds.). IEEE / ACM, 165–176. https://doi.org/10.1109/ICPC.2019.00034

[45] Chenglong Wang, Kedar Tatwawadi, Marc Brockschmidt, Po-Sen Huang, Yi Mao, Oleksandr Polozov, and Rishabh Singh. 2018. Robust text-to-sql generation with execution-guided decoding. *arXiv preprint arXiv:1807.03100* (2018).

[46] Longyue Wang, Zhaopeng Tu, Andy Way, and Qun Liu. 2017. Exploiting Cross-Sentence Context for Neural Machine Translation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*, Martha Palmer, Rebecca Hwa, and Sebastian Riedel (Eds.). Association for Computational Linguistics, 2826–2831. https://doi.org/10.18653/v1/d17-1301

[47] Winston Wu, Dustin Arendt, and Svitlana Volkova. 2020. Evaluating Neural Machine Comprehension Model Robustness to Noisy Inputs and Adversarial Attacks. *CoRR* abs/2005.00190 (2020). arXiv:2005.00190 https://arxiv.org/abs/2005.00190

[48] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *CoRR* abs/1609.08144 (2016). arXiv:1609.08144 http://arxiv.org/abs/1609.08144

[49] Frank F. Xu, Zhengbao Jiang, Pengcheng Yin, Bogdan Vasilescu, and Graham Neubig. 2020. Incorporating External Knowledge through Pre-training for Natural Language to Code Generation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel R. Tetreault (Eds.). Association for Computational Linguistics, 6045–6052. https://doi.org/10.18653/v1/2020.acl-main.538

[50] 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 *Proceedings of the 15th International Conference on Mining Software Repositories, MSR 2018, Gothenburg, Sweden, May 28-29, 2018*, Andy Zaidman, Yasutaka Kamei, and Emily Hill (Eds.). ACM, 476–486. https://doi.org/10.1145/3196398.3196408

[51] Pengcheng Yin and Graham Neubig. 2017. A Syntactic Neural Model for General-Purpose Code Generation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, Regina Barzilay and Min-Yen Kan (Eds.). Association for Computational Linguistics, 440–450. https://doi.org/10.18653/v1/P17-1041

[52] Pengcheng Yin and Graham Neubig. 2018. TRANX: A Transition-based Neural Abstract Syntax Parser for Semantic Parsing and Code Generation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, EMNLP 2018: System Demonstrations, Brussels, Belgium, October 31 - November 4, 2018*, Eduardo Blanco and Wei Lu (Eds.). Association for Computational Linguistics, 7–12. https://doi.org/10.18653/v1/d18-2002

[53] Pengcheng Yin and Graham Neubig. 2019. Reranking for Neural Semantic Parsing. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, Anna Korhonen, David R. Traum, and Lluís Màrquez (Eds.). Association for Computational Linguistics, 4553–4559. https://doi.org/10.18653/v1/p19-1447

[54] Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. 2020. BERTScore: Evaluating Text Generation with BERT. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net. https://openreview.net/forum?id=SkeHuCVFDr

[55] Xinze Zhang, Junzhe Zhang, Zhenhua Chen, and Kun He. 2021. Crafting Adversarial Examples for Neural Machine Translation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli (Eds.). Association for Computational Linguistics, 1967–1977. https://doi.org/10.18653/v1/2021.acl-long.153