Comparison of Sequence-to-Sequence and Retrieval Approaches on the Code Summarization and Code Generation Tasks

Nicolas Chausseau

A Thesis
in
The Department
of
Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements
For the Degree of Master of Computer Science (Computer Science) at
Concordia University
Montréal, Québec, Canada

March 2021

Concordia University

School of Graduate Studies

This is to certify that the thesis prepared

By: **Nicolas Chausseau**

Entitled: **Comparison of Sequence-to-Sequence and Retrieval Approaches on the Code Summarization and Code Generation Tasks**

and submitted in partial fulfillment of the requirements for the degree of

Master of Computer Science (Computer Science)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining commitee:

_____ Chair
Dr Leila Kosseim

_____ Examiner
Dr Leila Kosseim

_____ Examiner
Dr Jinqiu Yang

_____ Supervisor
Dr Peter C. Rigby

Approved by _____
Dr Hovhannes Harutyunyan, Graduate Program Director

13 April 2021 _____
Dr Mourad Debbabi, Dean
Faculty of Engineering and Computer Science

Abstract

Comparison of Sequence-to-Sequence and Retrieval Approaches on the Code Summarization and Code Generation Tasks

Nicolas Chausseau

In this study, we evaluate and compare state-of-the-art models on the code generation and code summarization tasks (English-to-code and code-to-English). We compare the performance of neural seq2seq BiLSTM [111] and attentional-GRU architectures [54], along with that of a semantic code search model reproduced from [90]. We compare these three models' BLEU scores (1) on their original study datasets as well as (2) on additional benchmark datasets [111, 67, 54], each time for translation and back-translation (i.e. English-to-code and code-to-English). We observe that, surprisingly, semantic code search performs best overall, surpassing the seq2seq models on 5 task-dataset combinations out of 8. We find that the seq2seq BiLSTM always outperforms the attentional-GRU, including on the relatively large (2M pairs) Javadoc-based dataset from the original attentional-GRU study, setting a new high score on that dataset, higher than four previous published studies.

However, we also observe that model scores remain low on several datasets. Some test-set questions are harder to answer due to a lack of relevant examples in the training-set. We introduce a new procedure for estimating the degree of novelty, and difficulty of any given test-set question. We use the BLEU score of the highest-scoring training-set entry as reference point for model scores on the question, a procedure which we call *BLEU Optimal Search*, or BOS. The BOS score (i) allows us to generate an information retrieval ceiling for model scores for each test-set question, (ii) can help to shed light on the seq2seq models' capacity to generalize to novel, unseen questions on any dataset, and (iii) helps to identify dataset-artifacts, by inspecting the rare model answers that score above it. We observe that the BOS is not reliably surpassed by the seq2seq models, except in the presence of dataset-artifacts (such as when the first words of the question contains the answer), and call for further empirical investigation.

Acknowledgments

I would like to take this opportunity to thank my supervisor Dr. Rigby for his guidance and insightful feedback. Without him, this research and thesis would not have been possible. I would also like to thank Dr. Kosseim, who is an expert in natural language processing, and who was instrumental in providing advice on the appropriateness of our models and outcome measures. Her courses were also extremely instructive and useful to me, and I recommend them to any new student. I would also like to thank Anunay Amar for taking the time to review this thesis in detail, and providing important comments, corrections and feedback throughout the text. Finally, I would like to deeply thank my parents and my brother for their love and constant support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Developers, beginners and experts alike, spend a significant amount of time searching for code, especially when navigating new languages and frameworks [106]. They usually search using natural language queries, describing a software engineering task; for example, looking for a code snippet for the task of *"determining variable type in Python"*, or *"filtering rows in pandas using regex"*. Several websites, *e.g.,* StackOverflow, are used today for this purpose and provide community-generated documentation and tutorials. In recent years, a number of systems were developed to make such recommendations of code snippets *automatically* to accelerate the search for relevant code snippets and to improve developer productivity [90, 47, 35, 81]. Systems take a natural language query as input (intent) and return a code snippet. Search-based systems use information retrieval techniques across a parallel corpus of English-code pairs to find the most relevant existing code snippet. In contrast to search-based systems, generative, translation-based systems use statistical or neural approaches, in particular sequence-to-sequence models, to produce a potentially new code snippet to solve the specific query. In this case, the models are trained to *generate* new code, instead of *retrieving* an existing code snippet. These neural translation systems can be trained on the same corpora of English-code pairs that are used by search-based systems.

Systems which generate or retrieve code snippets based on an English description can also be run in reverse taking a code snippet as input and summarizing it in natural language, *i.e.* performing the Code-to-English task, also called code summarization. Automating code summarization can be useful to, for example, generate descriptions for lengthy or complicated functions that are undocumented, or for machine-generated code. Since most search and translation systems were designed to be language agnostic [98], the code summarization task can be automated using the same search systems and translation systems that are used for the English-to-code task, and can also be trained on the same datasets of English-code pairs. Tables 1, 2, and 3 show some sample model outputs for three test-set

questions from one of the datasets we use in this study (CoNaLa challenge dataset).

Sample test-set question	Test-set answer	AttendGRU answer
send a signal signal sigusr1 to the current process	`os.kill(os.getpid(), signal.SIGUSR1)`	`print(`hello`, `hello.join(`\\ \\ python)`
decode a hex string 4a4b4c to utf 8	`bytes.fromhex('4a4b4c').decode('utf-8')`	`` `````````.decode(`utf-8`) ``
check if all elements in list mylist are identical	`all(x == myList[0] for x in myList)`	`all(list(range(a)), set(b))`

Table 1: Example outputs for the CodeSearch retrieval model on CoNaLa challenge test-set questions

Sample test-set question	Test-set answer	Seq2SeqLSTM answer
send a signal signal sigusr1 to the current process	`os.kill(os.getpid(), signal.SIGUSR1)`	`round('<unk><unk><unk>}'.format(h), -1)`
decode a hex string 4a4b4c to utf 8	`bytes.fromhex('4a4b4c').decode('utf-8')`	`urllib.parse.unquote('raw_unicode_escape')`
check if all elements in list mylist are identical	`all(x == myList[0] for x in myList)`	`all(x == 0 for x in L)`

Table 2: Example outputs for Seq2SeqLSTM sequence-to-sequence model on CoNaLa challenge test-set questions

Sample test-set question	Test-set answer	CodeSearch answer
send a signal signal sigusr1 to the current process	`s.kill(os.getpid(), signal.SIGUSR1)`	`subprocess.call(['/usr/bin/perl', './uireplace.pl', var])`
decode a hex string 4a4b4c to utf 8	`bytes.fromhex('4a4b4c').decode('utf-8')`	`\"\"\"\\\\xc3\\\\x85\u3042\"\"\".encode('utf-8').decode('unicode_escape')`
check if all elements in list mylist are identical	`all(x == myList[0] for x in myList)`	`len(set(mylist)) == 1`

Table 3: Example outputs for AttendGRU sequence-to-sequence model on CoNaLa challenge test-set questions

The goal of this work is to evaluate and compare the performance of a search model and neural translation models for code generation and summarization. In particular, we would like to compare the recent neural sequence-to-sequence translation models with a simpler code search model (document retrieval model).

We select three state-of-the-art models: a seq2seq BiLSTM model (abbreviated Seq2SeqLSTM throughout this text), replicated from Yin et al. [111], a seq2seq attentional-GRU model (abbreviated AttendGRU), replicated from Leclair et al. [54], and a semantic code search model (abbreviated CodeSearch), which is reproduced based on Sachdev et al. [90] (its code and dataset was not released at the time our study was conducted). We evaluate these models on the following datasets: (1) a dataset from Yin et al. [111] containing 3K manually annotated English-to-code pairs scraped from StackOverflow (i.e. the small "manually annotated" CoNaLa dataset) (2) a dataset again from Yin et al. [111] containing 600K English-to-code pairs scraped from StackOverflow (i.e. the large "mined" CoNaLa dataset), (3) the dataset from Leclair et al. [54] containing 2M Java docstring-code pairs from Github, and (4) the dataset from Sennrich et al. [67] which contains 143K Python docstring-code pairs also from Github. Each time, we train *in both translation directions*, that is, both for the English-to-code task and the code-to-English task. We answer the research questions described below, in Chapter 5.

To summarize, we evaluate three techniques: AttendGRU, Seq2SeqLSTM, and CodeSearch. We have four datasets: StackOverflow manually annotated (SOa), StackOverflow (SO), Pydoc, Javadoc. We have two tasks: code-to-English (C2E) and English-to-code (E2C). As a result, our study involves 24 *task-dataset-model* combinations, which forces us to use the following notation: Task-Dataset-Technique. For example, running the AttendGRU for the the code-to-English task on the manually annotated StackOverflow dataset is represented as C2E-SOa-AttendGRU.

Research Questions and Contribution Summary

RQ1: (English-to-Code) How well do the existing techniques perform for code generation?

In order to answer the research question described above, we start by replicating the Seq2SeqLSTM model on its original dataset, the CoNaLa challenge dataset [111] (E2C-SO). We then run all models on the datasets from all studies (novel task-dataset combinations).

E2C-SO (Yin et al. dataset [111], Seq2SeqLSTM replication). The first study replicated is Yin *et al.* [111]. The seq2seq BiLSTM with attention was originally trained for the English-to-code task, on the StackOverflow CoNaLa corpus (E2C-SO) and the CoNaLa challenge authors reported a BLEU4 score of 14.26 on the CoNaLa leaderboard [111]. In our replication we obtain a marginally higher score of 15.75. On this E2C-SO dataset from Yin *et al.* [111] the Seq2SeqLSTM remains the best performing model. The AttendGRU and CodeSearch models have a much lower score of 4.63 and 4.90, respectively.

We observe that the models score zero on the large majority of the test-set questions. The best-scoring Seq2SeqLSTM obtains a score of zero on 337 out of 500 test-set questions (67.40%). For the AttendGRU and CodeSearch models, the number of zeros is even higher: 448 (89.60%) and 444 (88.80%) respectively. For all three models, we find that the BLEU metric is skewed and misleading, as the median BLEU score *for individual test-set questions* is zero.

Finally we note that, since the authors of the CoNaLa challenge, Yin et al. [111], use the Seq2SeqLSTM to filter the larger E2C-SO (CoNaLa) dataset, it might have an unfair advantage on that dataset. This could be the cause of the unusually large lead of 10 BLEU4 points that the Seq2SeqLSTM obtains over the other two models on that task-dataset combination, by far the largest lead it has on the second-best scoring model on any dataset in our experiments (second largest lead it obtains is 1.96 BLEU4, C2E-Javadoc).

Novel task-dataset combinations (E2C-SOa, E2C-Pydoc, E2C-Javadoc). In theory the Seq2SeqLSTM, AttendGRU and CodeSearch models are all three *language agnostic*: they can be run on the other task-dataset combinations, and reversed for back-translation when necessary. We train the models on the three remaining task-dataset combinations for code generation (E2C), which have not been examined in prior work: E2C-SOa, E2C-Pydoc, and E2C-Javadoc.

On the three novel task-dataset combinations for E2C (code generation), surprisingly, CodeSearch performs the best every time. We also observe that the Seq2SeqLSTM outperforms the AttendGRU on all E2C task-dataset combinations, by a very large margin of at least 5 BLEU4 points.

Again here on novel task-dataset combinations, for all three models, we find that the average BLEU score for the entire test-set is misleading, as the median score is zero for individual questions. The median score for individual questions is zero on every task-dataset combination for E2C.

RQ2: (Code-to-English) How well do the existing techniques perform for code summarization?

In order to answer the research question described above, we start by replicating the AttendGRU model on its original dataset, the Leclair et al. dataset [55] (E2C-SO). We then run all models on the datasets from all studies (novel task-dataset combinations).

C2E-Javadoc (LeClair et al. dataset [54], AttendGRU replication). The second study replicated is LeClair et al. [54]. The AttendGRU model was trained for code summarization on the C2E-Javadoc corpus [55] and the authors reported a BLEU4 score of 19.4. In our replication, we obtain a score of 19.58 BLEU4.

Again for C2E, on the dataset from a previous study, we observe that the Seq2SeqLSTM is the best performing: the Seq2SeqLSTM scores 21.28, above their AttendGRU model. The Seq2SeqLSTM also outperforms three other recent studies on this same dataset: a neural seq2seq model using *attention to file context* [53], and graph neural networks [38], as well as two Transformer models in [36] and in [38]. CodeSearch scores below neural sequence-to-sequence models on this task-dataset combination from Leclair et al. [54], but is not very far below, with a BLEU4 of 17.12.

Again for C2E, we observe a large number of model answers with zero score. The best-scoring Seq2SeqLSTM obtains a score of zero on 73,338 (80.73%) out of the original 90,908 test-set questions from the dataset of Leclair et al. [54]. For the AttendGRU and CodeSearch models, the number of zeros is even higher: 75,158 (82.68%) and 78,435 (86.28%) respectively. For this reason, we believe that corpus-level BLEU scores are inappropriate as a measure when the data is skewed, since they

represent an average over the test-set as a whole. Also, for this C2E-Javadoc dataset, as we discuss in Section 5.3 in more detail, we identified a large number of dataset-artifacts (trivial questions) which inflate scores for the two neural sequence-to-sequence models, i.e. the Seq2SeqLSTM and AttendGRU, but not for the CodeSearch model.

Novel task-dataset combinations (C2E-SOa, C2E-SO, C2E-Pydoc). On the novel task-dataset combinations, again for C2E, CodeSearch performs the best. It surpasses the Seq2SeqLSTM and AttendGRU models on 2 out of 3 task-dataset combinations. CodeSearch is only surpassed on C2E-SO, by the Seq2SeqLSTM, and only by a very small margin of 1.05 BLEU points. This dataset is also exceptional in that all three models score extremely low on it, each having more than 97% answers with a BLEU4 score of zero.

We observe that the Seq2SeqLSTM outperforms the AttendGRU on all C2E task-dataset combinations, although this time the AttendGRU does better, and comes within 2 BLEU4 points of the Seq2SeqLSTM on 2 out of the 3 novel task-dataset combinations. On the two SO/SOa datasets, the AttendGRU performs better in the E2C direction than in the C2E direction, while on the Pydoc and Javadoc datasets it performs better in the C2E direction than in the E2C direction. This trend is observed for all models, although less pronounced, and it appears as if the docstring datasets are easier in the C2E direction.

These results lead us to a new question: why are scores so low on certain datasets, for all models, yet so high on other ones, again for all models? For our datasets' test-train splits, a large majority of questions do not have relevant, related answers in the train-set, unlike in the study from Sachdev et al. [90], where they tested the semantic search model only on answers which had a very similar answer in the training-set (duplicate or quasi-duplicate across test and train sets). To investigate this question, we examine, for each given test-set question, whether we have good, relevant training-examples. We do this using the BLEU Optimal Search (BOS) procedure described in the next section.

RQ3: (Generalization and Dataset-Artifacts) Can sequence-to-sequence models surpass the BOS ceiling for any particular test-set question? Under what circumstances?

We make a novel contribution by introducing the BLEU Optimal Search (BOS) score. For each document in the test-set (or for a random sample of them) we calculate the BLEU score against all documents in the training set and report the training document with the highest BLEU score. This result represents the best possible result that is contained in the training data for that test-set

question, *i.e.* that could be found by search. This BOS score is thus the effective ceiling score for the CodeSearch model and can be used as reference point for the neural sequence-to-sequence models.

Manually inspecting model answers that score above the BOS leads to discovery of dataset-artifacts. The first observation from BOS results is that sequence-to-sequence models rarely score above the BOS, on any individual test-set question and on the test-set as a whole. As can be seen in Figures 11 to 18, the BOS ceiling (retrieval ceiling) is rarely surpassed. On the docstring-derived datasets however, we can see that the sequence-to-sequence models are sometimes able to score above the BOS score for a greater number of individual test-set question. In a random sample of test-set questions for the C2E-Javadoc corpus from [54], we observe that for every one of these instances the high score is explained by the dataset-artifact described in section 1: the question contains the answer; that is, the function identifier split by underscore by the dataset preprocessing from [54, 90] contains almost all the words of the docstring and in the same order. This dataset-artifact, present mainly in the two docstring corpora (Pydoc, Javadoc), greatly inflates scores for both sequence-to-sequence models, as they learn to copy the first tokens of the "question" as "answer". For example, on the C2E-Javadoc task-dataset combination from the original AttendGRU study, both the AttendGRU and Seq2SeqLSTM are able to score above the BOS 15 and 22 times out of the 58 and 56 non-zero answers that they obtain in our sample, respectively. This constitutes more than 25% of the 58 non-zero answers for the AttendGRU, and more than 39% of the 56 non-zero answers of the Seq2SeqLSTM. Given that non-zero model scores are so rare to start with, these dataset-artifacts have a major influence on the sequence-to-sequence models' scores. CodeSearch cannot take advantage of such artifacts, since it can only return an existing training-set instance, intact.

The C2E-Javadoc original study dataset from Leclair et al. [54] is inherently easier for the two neural seq2seq models due to the dataset-artifacts. After examining BOS scores in relation to model scores for all datasets, we conclude that the original study dataset from [54] (C2E-Javadoc) inherently favours neural sequence-to-sequence models, because it contains by far the largest number of model answers that are affected by dataset-artifacts, across all task-dataset combinations examined in our experiments.

The conclusion from the analysis of BOS scores in relation to model scores is that the capacity to generalization is essentially absent in off-the-shelf neural seq2seq models tested, on our particular datasets. The vast majority of the time, the neural seq2seq models can only answer a test-set question as well as the best answer available in the training-set. Thus, in order to obtain high model scores without the presence of artifacts, a high BOS score is a necessary prerequisite, in all of our experiments over 24 different task-dataset-model combinations. This has important implications for

predicting seq2seq model scores on a new test-set, as well as for data augmentation approaches for practical applications.

This thesis is structured as follows. In Chapter 2, we perform a broad survey of previous works on code generation, code summarization and code search as well as evaluation methods and generalization in DNNs. In Chapter 3, we describe the three models evaluated in this study, the BLEU metric and the BOS procedure. In Chapter 4, we describe the datasets, preprocessing and training hyperparameters used in our experiments. In Chapter 5, we present results for each of our research questions, and discuss threats to validity. In Chapter 6, we discuss our results, and suggest avenues for future work. In Chapter 7, we conclude the thesis and highlight our contributions.

Chapter 2

Survey of the Literature

The goal of this study is to evaluate systems that can find or generate a code snippet from an English description, and vice versa. These two tasks are sometimes called the English-to-code and code-to-English tasks. We break the related literature into the following categories:

1. Big Code and Naturalness (Machine Learning and Natural Language Processing in Software Engineering)
2. Code Generation From Natural Language Inputs (English-to-code)
3. Code Summarization (Code-to-English)
4. Evaluation Methods and their Criticisms

2.1 Big Code and Naturalness

Since the advent of large online repositories of code and question-answer websites, such as Github and StackOverflow, machine learning applications for source code have developed at a rapid pace. Allamanis et al. [2] survey the literature on machine learning for big code and naturalness (machine learning and statistics applied to source code datasets). They report that the interest in big code and naturalness started with frequent API usage pattern mining approaches, that used for example frequent itemset counting, clustering, retrieval approaches, while more recent approaches tend to use deep neural networks (DNNs). Allamanis et al. criticize some of the metrics used to evaluate the models on code-related tasks, such as the BLEU score: they note that those metrics were originally devised for NLP tasks, but are often not well adapted to source-code-related tasks. They reflect that the granularity over which the BLEU metric is computed (per-statement vs. per-token) can be controversial. They remark on another downside of the BLEU score: the possibility of several correct

answers for a given test-set question. Indeed, syntactically diverse answers may be semantically equivalent, potentially causing BLEU scores to be low on a significant portion of test-set questions. Allamanis et al. continue with noting that although DNNs have been shown to learn some aspects of compositionality, highly compositional objects (such as the nested combinations of API elements into abstract syntax trees present in code) remains a challenge. This is an aspect that is specific to source code: they note the "highly compositional nature of code", and its abstract syntax tree (AST) structure, often forming deeper trees than the ones seen in natural languages. They suggest that improving machine learning for big code, by improving representations of source code artifacts, could also improve several downstream tasks, such as bug fixing, code auto-completion and recommendation, and code transcompilation. This preoccupation with compositionality of source code is reflected in many studies in this area of research. For example, many studies [54, 6, 3, 53, 18, 82, 72, 64, 115, 45] have tried representing the source code processed by machine learning models as an abstract syntax tree (AST), instead of a sequence of tokens, aiming to increase performance and generalization capacity. Some AST-based approaches are shown to outperform non-AST based approaches [6], but this is not always the case [30]. Finally, Allamanis et al. note the difficulty of finding data of high quality for machine learning applications. They note the potential of data coming from coding teaching websites for future applications.

API Usage Pattern Mining

Usages of API elements from programming languages and frameworks are sometimes complex and often not well documented for corner use cases. To compensate for this lack of documentation, one line of work aims to mine API usage patterns in large online code repositories such as Github. These search approaches have aimed to identify and collect the different ways in which API elements are used for different tasks.

Michail et al. [68] release CodeWeb, a system that mines frequently reused API methods and classes from a library. Their tool can help developers learn new APIs, and facilitate code reuse. Zhong et al. [119] developed MAPO, a tool that mines API usage patterns from clustered search results, using a frequent subsequence mining technique. The results are presented to the developer performing the search, for inspection. The API usage patterns mined are more complex than those of previous works, and can involve multiple methods and temporal information. They find that their tool returns fewer and more relevant results than previous search approaches, and assess the effectiveness of their tool in an empirical study with developer users. Wang et al. [103] develop BIDE, a system that mines API usage patterns more efficiently than previous approaches, using less memory, and showing faster query speeds. Their approach relies on the BackScan pruning method and the ScanSkip optimization technique.

Code Retrieval for Source Code Recommendation, Auto-Completion

By using measures of similarity between user-generated code and source code in existing repositories, it is possible to make recommendations of code changes, or provide auto-completions of source code.

Holmes et al. [43] develop Strathcona (2005), a system that recommends code snippets based on their structural similarity with user-provided code. Their system is used as Eclipse plugin for source code recommendation: it returns a list of best code snippets matches for the query, for perusal by the developer. They conduct a human evaluation of their tool, and find that on 5 out of 8 development tasks of increasing difficulty, the developers could find a relevant code snippet in the results list.

Yang et al. [63], similarly use the structural context of a code query to find suitable code examples in a code repository. Cubranic et al. [23, 24] also develop and Ecplise plugin, HipiKat, which recommend artifacts relevant to the tasks that is performed by the developer, by searching project archives. Ying et al. [113] developed a source code recommender which suggests potentially relevant source code based on the current set of files being modified by the developer, and previous project history in version control. Sahavechaphan et al. [92, 71] also developed a combination of search, graph-based code mining algorithm to recommend relevant code snippets based on the context of code under development. Their tool provides both specialized, context-sensitive code recommendations as well as more general results which cover a larger number of scenarios.

N-gram and Neural Language Models for Source Code Auto-Completion, Program Repair

Hindle et al. [40] note that programming language present statistical regularities which make them predictable, like natural languages. They used an n-gram language model to build a code completion engine for Java. This work instead of relying on structural aspects of the code to search and provide code recommendations, is inspired by statistical language models used in the field of natural language processing (NLP). Several subsequent studies will adopt this approach and adapt machine learning models, originally developed in the field of NLP, to source code related tasks.

Nguyen et al. [76] develop SLAMC, a system for source code auto-completion. It improves on the simple n-gram language model proposed by Hindle et al. [40], by incorporating semantic information in n-grams, by specifying pairwise associations (e.g. `fopen`, `fclose`), and by using topic modelling, whereby a codebase can be charaterized by several "code topics". In essence, it aims to improve on the shortcomings of an n-gram model, for example (i) long dependencies (`try ... catch`), (ii) capturing the "context" or topic of the code to-be-completed (e.g. file i/o, database connection, etc.). They show very large improvements of 10 to 25% in top-k accuracy over an n-gram language model.

In [72], Nguyen et al. develop GRALAN, a graph-based statistical language model for source code auto-completion. They reason that frequently-used code templates being mined are best represented

as abstract syntax trees (AST). Moreover, unlike for natural languages, source code AST could be parsed unambiguously if the code compiles. Their system therefore aims to mine a frequent parent graph to the graph that was started by the developer, allowing for its completion. Matching a sub-graph with its parent is done by Bayesian inference (since it can belong to more than one parent graph). Their system outperforms an n-gram model, however, although not explicitly mentioned, it appears to score lower in top-k accuracy compared to their previous SLAMC model, which was based on a modified n-gram model, with no graph construct.

Maddison et al. [64] compare tree-traversal models with earlier off-the-shelf n-gram based language models on the code generation task, from code inputs (not English-to-code). They reason that using an AST representation of the source code is crucial in order to improve their system. They show that their Log-bilinear Tree-Traversal (LTT) system which combines n-grams and AST outperforms simple n-gram based models. They train their models on a code corpus scraped from TopCoder.com.

Liu et al. [59] use an LSTM for Javascript code completion. They approach the problem of code completion as an AST-traversal problem, and use AST as input to the model, without the type information. They train the model on a benchmark dataset of 100K Javascript functions from previous work, and show that it can surpass a decision tree baseline.

More recently, Svyatkovskiy et al. [99] used a Transformer architecture for code completion. Their model is based on the GPT-2 architecture, with a similar number of trainable parameters, and is trained on 1.2 billion lines of source code in Python, C#, JavaScript and TypeScript. They obtain a ROUGE-L precision and recall of 0.80 and 0.86, almost double the score of a 5-gram language model, for code completion in the Python language. They also provide an efficient implementation that meets the requirements for commercial applications in the Cloud and in IDEs such as Visual Studio.

Code search, in addition to helping software developers, can also be applied to improve automatic bug fixing [8] and program repair when used in conjunction with a genetic algorithm [107]. The search model complements the genetic algorithm by suggesting relevant candidate code modifications to apply.

Code Transcompilers

Also inspired by language models coming from the field of NLP, other approaches adapt translation models to the code transcompilation task. They use source code in one programming language as input to generate new code in another programming language.

Karaivanov et al. [50] used phrase-based machine translation to translate C# into Java. They train on a large C# and a Java parallel dataset. They observe that phrase-based machine translation (PBMT) does not produce syntactically valid code, and decide to extend the PBMT to accomodate the grammar of the language. Lachaux et al. [87] use neural translation models to translate between

programming languages. They show that a neural unsupervised approach significantly outperforms two rule-based baselines. They also show that the neural models are versatile and language-agnostic, they can be trained to translate between a variety of programming languages. In our study, we also use two neural machine translation (NMT) models, but instead of being trained on code as input and guide to generate the code snippet answer, our models are trained on an English intent / guide.

2.2 Code Generation From Natural Language Input (English-to-code)

One shortcoming of the first approaches for API usage pattern mining, code completion, and recommendation is that they do not allow to search code with an English description. This use case however is one of the most useful to developers, when they do not know in advance which API elements to use to accomplish a particular software engineering task. Sadowski et al. [91] find that 34% of internal code search queries at Google aim to find short code examples illustrating API element usage; it is the most frequent goal of a code search. Other frequent uses involve locating code to edit, understanding and debugging functions used, and finding code authors. They observe that code search is essential to developer workflow, with queries being made on average 12 times a day (with a median of 6). They note that code search tool usage has been increasing in recent years. Shi et al. [95] report that 1.75 million code search queries were performed at Baidu in a year. They estimate that using a new code search platform, iSearch, could save 7057 working hours per year from code search features. iSearch provides code fragments search, navigation features, and dependency analysis. Xia et al. [106] find that during the software development phase of software projects, developers may search online for examples of API usage or to debug errors. They find that developers spend a median of 15% of their time searching for code example online. Xia et al. [106], also find that finding reusable code snippets is the third most frequent goal of developer web searches.

Code search systems can be used to search for common ways to use API elements, which is useful to learn a new programming library or framework. Another important use of code search models in industry is to search for functions inside large codebases, to retrieve all functions related to a particular task or intent [90, 91, 95]. This is especially useful, to help find function usage examples, locate existing functions to facilitate code reuse, or instead to find where to implement a particular code change.

More recently generative models such as neural machine translation (NMT), or sequence-to-sequence models, have been tested on the task of providing code examples given an English query. The use case for these types of generative models would be more for code suggestion, or discovery of new APIs and frameworks, but does not directly apply to the code search and reuse use case

described above.

Search-Based Approaches

Several models were developed to search for code snippets using an English description, or intent.

A recent implementation a semantic code search system is Neural Code Search (NCS) by Sachdev et al. [90]. The NCS system is a kNN-based code retrieval system, which evaluates the semantic similarity between the user-provided English query and the English words associated with code snippets from the training-set and returns top-k best matches for the user (developer) to inspect. Sachdev et al. use as training-set a corpus of Github-scraped functions, and extract English words from the function by, for example, splitting function names by underscore or camel case, or extracting variable names, which will then be used together as "intent" for building the bag-of-word vectors to-be-matched. In other words, their preprocessing procedures allow to extract English intents from the code itself, creating a sort of parallel corpus. To evaluate the performance of the NCS model, Sachdev et al. use a list of StackOverflow questions which have an associated code snippet identical or very similar to an existing function available in the Github-extracted training set. They find that their model is able to retrieve the correct answer 43 times out of 100. They then improve the basic NCS model with query enhancement, which allows NCS to outperform other conventional retrieval system such as ElasticSearch and BM25 by a large margin. NCS is called a semantic search model because, by using Word2vec embeddings, it is able to match synonyms or semantically related words between query and stored documents, instead of the exact same terms as other retrieval approaches sometimes do. The CodeSearch model used in our study is reproduced from NCS, with minor differences, specified in Section 3.1.1. In their study, Sachdev et al. also show by ablation on their model that the use of TF-IDF weights on the words has a greater impact than the use of word embedding vectors on the score.

Gu et al. [33], apply a deep learning approach to code search, in a system called CODEnn (not to be confused with the LSTM-based CodeNN for generative source code summarization, from [48]). Unlike NCS, CODEnn uses two embeddings, one for code tokens and one for English tokens, that are unified into a single vector space. CODEnn uses recurrent neural networks (RNNs) for sequence embedding of the English queries and source code tokens. They then retrieve documents from the training set by matching these document embeddings with those from the training set, using cosine distance as similarity metric. Like in the NCS paper and as is frequent for code summarization and generation tasks, their preprocessing procedure splits functions and identifiers by underscores and following camel case. They train their RNN embeddings on a Java corpus extracted from Github. They report that their model outperforms information retrieval systems such as Lucene. One advantage of their model is that, like NCS, their model is semantic, since it can match queries

and documents *by meaning*, if they use semantically similar words (or code tokens).

Cambronero et al. [17] use a deep learning model for code search. They systematically evaluate code search approaches from previous studies along with theirs. Their model, UNIF, is a modified version of NCS from Sachdev et al. [90]. They use two embedding matrices, one for English query tokens and one for code tokens. NCS used only query tokens embeddings to match a query to a training-set document, and is thus considered unsupervised from their point of view. They also replace TF-IDF weights with *learned* attention weights. Their model is observed to outperform NCS, but not on all datasets tested. They test on both StackOverflow-derived and Github-derived (docstring) corpora.

Liu et al. [60] observe that, when queries are short, the Neural Code Search (NCS) model from Sachdev et al. [90] performs less well. To solve this problem, they developed NQE, a neural model which recommends additional English terms to enhance the initial, short user query. They observe that this neural query enhancement system is able to outperform the previous NCS model for very short queries.

One known problem with the English-to-code task is that there can be different ways to code the same software engineering task in a same programming language. For example in the Java language, there are three or more different APIs that allow to open and read a file (Scanner, BufferedReader, FileReader, etc.). Therefore, several different code snippets could be relevant for a given English query. Premtoon et al. [80] develop a semantic code search system which also allows to retrieve code snippets that are equivalent even if they use different API to accomplish the same task. They accomplish this by defining a set of known possible code transformations, or rewrite rules, which allow to compare and assess different code snippets for equality.

Statistical and Machine Translation Based Approaches For Code Generation

In contrast with search-based approaches, generative approaches for English-to-code generate code token by token, and can potentially create new sequences that are unseen in the training set. To do so, generative approaches use statistical and / or neural language models, that learn a probability distribution over sequences of vocabulary tokens for a given language.

Raghothaman et al. [81] develop SWIM, a search-based system for code generation. They use count-based, statistical translation inspired English to code token mappings to identify a list of unordered relevant API elements to use, in a first step. These API elements are then matched to existing, ordered call sequences through a bag-of-word word-to-vector approach and returned to the user. They train and evaluate on a C# dataset.

Nguyen et al. [75, 73] develop a hybrid system based on SMT and graph-matching for code generation. Their system is a response to the shortcoming observed in sequence-to-sequence translation

models, which often do not respect the syntax of programming languages. After the English to API SMT-based step, which is similar to the SWIM system, they then generate a code graph, or abstract syntax tree (AST), through a graph-matching step, with the GraSyn tool. They prepare a parallel corpus of English-code pairs extracted from StackOverflow, to train and evaluate their model.

Neural Generative Approaches For Code Generation

In addition to retrieval and SMT-inspired approaches, neural generative approaches were also extensively explored in previous studies since 2015. A common theme in these approaches is the preoccupation with compositionality and the use of abstract syntax trees (AST) as input and / or output, in order to improve the capacity of the model to generalize and improve the syntactic correctness of the generated code in the target programming language.

Gu et al. [34] develop DeepAPI, an RNN-based sequence-to-sequence approach for code generation. They train on Java projects scraped from Github repositories. Their model is trained on a parallel corpus built from the first line of a docstring and the list of API elements used in the body of its associated function. Their system can be considered an off-the-shelf NMT system, except that they weight the intput tokens using a count-based TF-IDF scheme, instead of with gradient descent trained attention weights as is usual in more recent neural models for this task. DeepAPI tends to produce very short API recommendations, which obtain a very high precision, but generally also low recall, compared to the previous SMT-based approaches.

Yin et al. [112], in a study preceeding their release of the CoNaLa challenge, develop a new seq2Tree neural system that learns to predict Python code from English descriptions. They use a model that transduces the English description for the code into an abstract syntac tree (AST) in the target programming language, and obtain better results than previous studies which were not AST-based. In their results, a retrieval model surpasses an off-the-shelf NMT system, which is also what we observe in our results. However, in their study, on their particular dataset, their best-performing model seq2Tree surpassses the retrieval model.

Allamanis et al. [5] develop a bimodal neural models for the English-to-code and code-to-English tasks, a model that uses structural information in both English and code token sequences. Ling et al. [58] use a novel system based on neural language modeling to generate sequences of API tokens for a game (Magic the Gathering and Hearthstone). Murali et al. [70] develop a system for API-Element-to-code; their model generates code not based on English but rather on a list of relevant API elements. Although it is not designed for English-to-code, their system has been used later on as a substep in an English-to-code system [78]. Nye et al. [78] also develop a hybrid system for English-to-code, which consists of a neural sketch generator (seq-to-seq RNN with attention). It outputs a distribution over sketches as a first step, followed by a program synthesizer, which searches

for full programs which satisfy the spec.

More recently, Feng et al. [29] release a pre-trained BERT-based model for programming and natural language. Hu et al. [44] develop code embeddings for overcoming the problem of alignment between English and code, which is not as exact as between two natural language pairs. Balog et al. [14] develop a neural network to augment search-based techniques, such as an SMT-based solver and enumerative search. For a range of inductive program synthesis (IPS) baselines, they improved the runtime by 1-3 orders using neural networks. In our research, we compare a search based model with neural generative approaches, but this study shows that the combination of the two can be fruitful for improving the runtime of the system.

2.3 Code Summarization (Code-to-English)

Zhu et al. [120] survey the literature on automated code summarization. They find that approaches fall in the following categories: Information Retrieval, Machine Learning, Stereotype Identification, Natural Language Processing, External Description Usage. They also note that evaluation approaches fall into the following categories: Manual Evaluation, Statistical Analysis, Gold Standard Summary (e.g. BLEU score, ROUGE score, F1 measure), Extrinsic Evaluation (effect of the summary on humans carrying out a particular task related to code comprehension), and None. In our study, we reproduce a Information Retrieval approach, as well as two neural sequence-to-sequence generative approaches. The evaluation technique, the BLEU score, falls under the Gold Standard Summary category.

Search-based Approaches for Code Summarization

Liu et al. [61] develop a simple kNN-based retrieval system for commit message generation (a task very similar to code summarization), and compare its BLEU score to those of NMT system from previous studies. They observe that the retrieval model inspired from NCS outperforms the NMT sequence-to-sequence generative models. The authors reach very similar conclusions to the ones we have observed in our study, and that are discussed in Chapters 5 and 6: (i) the high-scoring answers from neural machine translation (NMT) approaches generally are very similar to existing training-examples, a conclusion we also reached in our BOS analysis in Section 5.3, neural sequence-to-sequence models can only score high when the BOS score is high for that test-set question. (ii) Liu et al. find 16% of "trivial" training-examples, which they find are responsible for a large portion of the high-scoring NMT models' answers. (iii) the NMT models' scores decrease by a large amount after removing trivial training-examples from the dataset, which were boosting the NMT models' score. This study is an excellent confirmation of our results and conclusions.

Zhang et al. [116] use a hybrid retrieval-NMT approach to code summarization. Since neural sequence-to-sequence models have difficulty with low frequency words (out-of-distribution questions), they decide to enhance the model with the most similar code snippets retrieved from the training set. They compare the BLEU4 scores of a syntax-based retrieval model (20.2 BLEU4), a semantic-based retrieval model (20.1 BLEU4), pure-NMT model (14.2 BLEU4), as well as their three novel variants of hybrid models: NMT+semantic-retrieval (19.5 BLEU4), NMT+syntactic-retrieval (19.8 BLEU4), NMT+semantic-syntactic-retrieval (20.7 BLEU4). Although they conclude that their third hybrid model scores slightly above the two pure-retrieval models, we should note that this difference in score is small (0.5 BLEU4) and well-within the natural variation that models can obtain on two different test-train splits of the same datasets, or even on two different runs on the same test-train split. Also, it is interesting to observe that in their experiments, the pure-NMT system scored more than 5 BLEU4 points below both retrieval models, which is substantial. Finally, out of their three hybrid models, only one scored above pure retrieval models, and by a very small margin. These results are aligned with the scores that we observed in our study, where our replicated semantic retrieval model (CodeSearch) scored above neural NMT approaches on most datasets. Finally Zhang et al. note that the retrieval part of their system takes longer to complete at query time, but because of optimizations they are able to generate an answer in less than 89ms on their setup. In our study, we also note that a downside to vector-similarity based retrieval models is the increased query time, in comparison with neural sequence-to-sequence models.

Ye et al. [109] use another hybrid NMT-retrieval approach for code summarization, but instead of feeding the retrieved code snippets to enhance the NMT's inputs, they use the output of the NMT to enhance the inputs of the retrieval model.

Statistical and Machine Translation Based Approaches For Code Summarization

Guerrouj et al. [35] use an n-gram statistical model trained on StackOverflow language and code pairs to produce code summaries. They obtain an accuracy of 54% on a gold standard evaluation using R-precision score. In our study, we have not included a statistical model; we only evaluate an information retrieval model, and two neural generative models. It would be interesting in the future to compare statistical and neural approaches with search-based approaches, as it has been done in the natural translation domain [11].

Neural Generative Approaches For Code Summarization

A very large number of studies have experimented with neural sequence-to-sequence models for the code summarization task. As for the code generation task, several papers have focused on representing AST information in the neural input in order to improve scores, generalization and syntactic validity.

Leclair et al. [54] train three variants of attentional-GRU models on the code summarization task. They use as training-set a large dataset of 2M English-code pairs, scraped from Github, previously released in [55]. They postulate that combining the AST and sequential-token input from two separate encoders might lead to an increase in model performance, based on observations coming from the neural image-recognition domain, where convolution embeddings and image tags were similarly combined in some experiments. They flatten the AST, using a procedure called structure based traversal (SBT), and use it as input to an additional encoder dedicated to the AST input only. The embeddings from the AST and non-AST encoders are then concatenated and fed to the decoder. They train three variants: one that uses an additional AST encoder, another without AST, and a third that ensembles the two. They obtain BLEU4 score of 19.6 for the AST-based model, which outperforms the non-AST based model (19.4), and a single decoder LSTM baseline, CodeNN from Iyer et al. [48] (9.95). An ensemble of the AST and non-AST approaches obtains 20.9 BLEU4. This paper is the third replicated model in our study (AttendGRU), and we use the non-AST model. We also use their Java dataset, initially released in [55], to train all our models. In our study, we observe that the scores of a same model, from run to run, can diverge even more than the improvement in scores obtained from the AST over the non-AST version. With the second replicated model (Seq2SeqLSTM) we also obtained a score higher than all the previous published papers on this dataset to date.

Haque et al. [38] use a AttendGRU model to generate summaries of code functions, by training on the same Github-scraped parallel corpus of java functions and their docstrings previously released in [55]. Based on the assumption that function dependencies can help further understand a code snippet's intent, they use dependencies of a given function to-be-summarized, as additional input for the summarization model, in order to improve scores. They find that their model outperforms baselines from previous studies. In our study, which uses the same java docstring dataset form Leclair et al. [55], the LSTM model replicated from Yin et al. [111] scores above all previous models on this dataset, including this graph neural network.

Leclair et al. [53] reuse again the java docstring corpus from Leclair et al. [55] but this time they train a graph neural network for code summarization. Their best-performing combination is a Convolution graph neural network BiLSTM, which obtains 19.93 BLEU4. Just like in our study, it is frequent that the BiLSTM outperforms other models on code summarization and code generations tasks [53] [30].

Gupta et al. [36] use a Transformer architecture for the code summarization task. They train their model on the same Java dataset of doctstring-code pairs from Leclair et al. [54]. They obtain slightly lower scores (17.99 BLEU4) than the original Leclair et al. study [54] (19.6, 19.4 BLEU4). The paper is only published on arxiv for now, but is mentioned nonetheless since it is relevant to the

present study.

Thanks to the sustained effort of Leclair et al. in doing systematic comparisons and ablation testing on different variants for neural sequence-to-sequence architectures for code summarization, we can now better compare different models and sequence-to-sequence architectures on the task. Results from Leclair et al. in their three recent studies [54, 53, 38] and the additional one from Gupta on the same dataset [36], are extremely interesting as they show that a wide variety of neural architectures, attend-GRU (with and without AST as encoder input), sequence-to-sequence with attention to file context, graph neural networks, Transformer, as well as previous baselines from the literature such as CodeNN often obtain BLEU scores extremely close to one another, in the range one would obtain by re-training the same model twice on the same dataset and test-set, with identical hyperparameters (e.g. less than 0.5 BLEU4 difference). For example, their ast-attend-GRU model scores 19.6 BLEU4 in their first code summarization study [54], but 18.69 BLEU4 in their subsequent study [38], although they do not specify if the test-train split was exactly identical to the one in first study. Similarly, their attend-GRU model shows a change in score between the two studies, 19.4 and 18.22. When we replicated their attend-GRU on the same dataset and test-train split as [54], we obtained 19.58 BLEU4, in contrast with their 19.4 original score. We also ran the attend-GRU model several times to ensure we would obtain the highest possible score for that model, and the highest results from each run, with identical hyperparameters, ranged between 18.35 and 19.58 BLEU4 after at least 20 epochs.

One of the first approaches using sequence-to-sequence generative models on the code summarization task, in 2016, is that of Iyer et al. [48], who use an off-the-shelf LSTM-with-attention model, that they call CodeNN. Their model is trained on a dataset gathered from the StakOverflow website. They use both gold standard evaluation, with the METEOR and BLEU4 scores (see 2.4 and 3.2.1 for a definition of the BLEU score), as well as human evaluation, and find that their model outperforms an information retrieval based model, as well as a phrase-based machine translation system statistical based model (MOSES with a 3-gram language model), and an off-the-shelf approach from natural language summarization, from Rush et al. [88]. In our study we find, in contrast, that the information retrieval approach outperforms the neural ones on most datasets. We did not evaluate phrase-based translation system or SMT-based system.

Allamanis et al. [4] use an attentional convolution neural network for source code summarization. Their model generates function names, that can act as short descriptive summaries for the inputted code snippet. For evaluation, they use the F1 score, as well as precision and recall. They compare the summaries generated by their model with those of three baselines: TF-IDF retrieval model, Standard Attention sequence-to-sequence from [9], Interestingly, and in line with the results obtained in our study, they observe that the standard, off-the-shelf attention-based translation model from Bahdanau

et al. (2015) [9], used as baseline in their experiments, scores below a simple TF-IDF retrieval model.

Alon et al. [6] develop code2seq, an AST-based sequence-to-sequence LSTM architecture for code summarization (generating a function name from a function body) and code captioning (generating an English description from a code snippet). They reason that a *flattened* AST is not in an ideal form for use as training input. They decide to improve on the flattened AST approach, by instead exhaustively collecting all possible paths between terminal nodes (user-defined variables) in the AST, and using those paths as the representation of the code snippet, and as training input to the sequence-to-sequence model. Because of the large number of possible subpaths between terminal nodes, they use a sample of k paths as the representation of the code snippet. Each of the k AST-subpaths are fed to one of k encoders, and the embeddings from all k encoders are then averaged before being used as input to the decoder. They train their model on a large dataset of 16M C# functions for the code captioning task, and find that their model outperforms all previous baselines by a substantial margin. For exampe, their model obtains 23.04 BLEU, 2.51 BLEU points more than the next-best model, CodeNN (20.53) [48] on the code captioning task. They also perform ablation studies that clearly show improvements coming from the use of the AST subgraph inputs in their model. This study shows the advantage of AST-based approaches for source code summarization. In our study, we have not evaluated an AST-based approach. In future work, it would be interesting to examine the capacity that the AST-based approaches have in general. Code2seq is a good candidate for such an evaluation.

Fernandes et al. [30] use BiLSTM and graph neural network for code summarization, similarly to the approach of Leclair et al. [53]. They train their model on a dataset of C# function-docstring pairs, as well as on a Java dataset used in [6] for method naming. They compare a hybrid BiLSTM-GNN (22.5 BLEU4) approach with an attention-GNN approach (21.4 BLEU4), and the BiLSTM-GNN is the best performing model in their study. On the method naming task, their BiLSTM-GNN (51.4 F1) also outperforms the code2seq model (43.0 F1) from Alon et al. [6].

Zeng et al. [115] use an LSTM with attention sequence-to-sequence model for source code summarization. They train their model not on the source code itself, but rather on an AST representation of the code snippet. They use a Java dataset containing 396,184 English-code pairs scraped from Github. They compare approaches with and without AST, and observe that the AST improves scores significantly. They obtain a BLEU4 score of 52.80 for the model using AST, while the best of non-AST models scores 47.82 BLEU4.

In [69] Moore et al. use a convolution encoder and LSTM decoder architecture for source code summarization. They train their model on a large dataset of English-code pairs from Github. They forego using AST as training inputs, and observe that their model matches AST-focused models from a previous study, [45]. Their model obtains a BLEU4 score of 38.63, while the AST-driven model from

Hu et al. [45] obtained 38.17. This result confirms the observations we made in the present study: that improvement in BLEU4 scores coming from the use of AST-driven sequence-to-sequence models from previous studies are not always significant and are sometimes within the normal variation of a model from run to run.

In [19] Chen et al. use variational autoencoders (VAEs) to jointly model source code and natural language descriptions of source code. They then use the semantic vector representations produced by those VAEs to generate completely new descriptions for arbitrary code snippets. On the code summarization task, they obtain a similar performance to previous approaches. They use BLEU4 and METEOR score for evaluation. They obtain 20.9 (20.5) BLEU4 on a C# summarization dataset, and 19.7 (21.0) BLEU4 on a SQL summarization dataset, while previous approaches (CodeNN) obtained 20.5 BLEU4 and 18.4 BLEU4, respectively.

2.4 Evaluation Methods and Their Criticisms

The CoNaLa challenge [111], whose dataset we use to train and evaluate models in this study, is using BLEU4 score as evaluation metric, to rank models. This is not a surprising choice of metric, since the tasks of "code generation from a query" and "code snippet search", as well as their reverse, "code summarization" can all be understood and approached as translation problems. BLEU score was the first translation evaluation metric shown to correlate highly with human assessment on natural language translation tasks [79], but it remains an open question whether the BLEU score (see 3.2.1) is an adequate measure of the performance of code generation and summarization systems. On this topic, we now discuss the study from Stapleton et al. [97].

Criticisms of BLEU in the Software Engineering Domain

Stapleton et al. [97] evaluate the relevance of the BLEU metric in the context of code summarization task. They wanted to test the common assumption that higher BLEU scores for generated code summaries implies better quality comments. To do so, they examined the correlation between comments BLEU score and measures of code comprehension by humans, and find that there is only a weak correlation between the BLEU score of the generated summary and the correctness of answers on a code comprehension task based on the generated summary. They also found that human generated summaries are better at positively affecting correctness on the code comprehension task in comparison with machine generated summaries, even if the study participant is often unable to detect the human-generated summaries as being of higher quality, when they are asked to rate them. This would imply that machine generated comments are dangerously misleading, since they often do not appear as low-quality, or incorrect at first, yet they contribute to errors in code comprehension.

In our study, we also observe that neural generated summaries are too often not accurate: their BLEU scores are so low on average that they cannot be useful in their current state.

However, it is important to notice that the study *did* find a correlation between the BLEU score and comprehension outcomes, only that this correlation was very weak. This could be explained by the fact that, like in NLP, multiple solutions exist for a given question, or input, and therefore a large portion of the time, even a "perfect" model would score close to zero. As the average number of possible answers increases in a dataset, the correlation between BLEU scores and human evaluation will become weaker, eventually becoming non statistically significant, even if slightly positive. In sum, while the BLEU score *may be* positively correlated with human judgement, if too many correct answers exist for a given question, and a large portion of the correct answers from a model get a very low score, then their observations would be expected: the correlation BLEU and comprehension outcomes would be visible, but weak.

Criticisms of BLEU in the field of NLP

In the NLP literature, the BLEU score was shown to have several downsides, and is widely criticized as an imperfect measure. A first important problem with the BLEU is that incorrect answers can score high if they use the same words as the reference but in a different order, or if they add a few modifier words (e.g. use a negation to modify meaning). This of course is also problematic in the domain of code generation and summarization. On the code generation and summarization tasks, we can observe that the neural generative models often do not respect syntax, do not order the English or code tokens in a correct order, and frequently include irrelevant tokens as well, while still in some cases obtaining high BLEU scores.

We provide here a list (non-exhaustive) of criticisms of the BLEU score, from previous studies in the field of NLP:

1. BLEU correlates only partially with human evaluation [16].

2. Models can optimize specifically for BLEU score while bringing down the true quality of the translation [16]. It is possible to cheat the score by having shorter answers, only including words to which the model assigns high probability, since it is a precision based score. To mitigate this problem however, the brevity penalty was introduced.

3. Despite the brevity penalty, it is still shown that when recall is included, correlation with human performance increases [52]. The F1 score (harmonic mean of precision and recall) is shown to correlate better with human evaluation than the BLEU score.

4. BLEU scores are shown to favour SMT over neural translation, and rule-based over neural translation [16]. For this reason, human assessment is preferred over BLEU scores.

5. BLEU scores are hard to compare across different tasks and datasets [117]. High BLEU scores of 35-40 can sometimes correlate to very high human assessment on some tasks and datasets but not others.

6. BLEU scores are highly dependent on the preprocessing steps and tokenization, and often different steps are used making the score impossible to compare.

7. Using only one reference translation per question for BLEU score calculation can result in low BLEU scores; multiple accepted reference translations increase chance of matching with a given translation [39].

8. BLEU does not guarantee correct meaning of translation, or syntax, only measures presence of words, and n-gram overlap.

Alternatives to the BLEU score exist, but are not as widely used. The METEOR metric for translation is based on F1 score as well as accepting synonyms: [10]. Other metrics such as SacreBLEU, ROUGE, TER were also proposed [1]. Human assessment of machine translation outputs is generally considered the most meaningful and reliable metric. Different protocols exist for human evaluation of translation which are out of scope of this study.

Despite these criticism, because the BLEU score is widely used, in the CoNaLa chalenge and to compare performance with previous models from [54, 55, 54, 38, 53, 36], we adopt the BLEU score for evaluation of the models in the present study. The BOS score described in the following section aims to make BLEU score easier to interpret.

Benchmark Datasets

In an effort to assess fairly the progress of models on the English-to-code and code-to-English tasks, as well as on other related NLP tasks, several researchers have produced and released challenge datasets.

Sennrich et al. [67] release a Python code-docstring dataset for automated source code generation, automated source code documentation, and code search. They note recent breakthrough in machine translation with neural sequence-to-sequence models, and highlight the need for large amounts of parallel data to train on. They also note that one of the existing code datasets used in some early studies on code generation with neural sequence-to-sequence models [58] yield BLEU scores substantially higher than BLEU scores seen in natural language translation, sometimes twice as high. They conjecture that those datasets are too easy for the models, which motivates their work to produce a more realistic corpus that reflects industrial source code. In our study we use their dataset to train and evaluate all models (E2C-Pydoc and C2E-Pydoc).

Yin et al. [111] release a benchmark dataset for the English-to-code task, the CoNaLa dataset, which is extracted from StackOverflow. Because StackOverflow posts are noisy, with intents often not descriptive of the code they are associated to, they use a filtering procedure in an attempt to identify code-English pairs which are of high quality to train NMT systems with. In order to produce a high-quality dataset for English-to-code, they train a neural model on a manually annotated subset of a large training dataset scraped from StackOverflow; they then use this trained model to filter the rest of the dataset. More specifically, they use the next-token probabilities produced by the model during prediction on a new training example as a proxy for the quality of this training example. In our study, we use the CoNaLa benchmark dataset to train all three models replicated and reproduced (CodeSearch, Seq2SeqLSTM, AttendGRU). The CoNaLa baseline NMT model is also replicated and evaluated in our study (Seq2SeqLSTM) on all datasets examined.

LeClair et al. [55] note the difficulty of obtaining good training corpora for the summarization task and the need for a corpus which can be used as reference point between models. They remark that using different corpora makes models very hard to compare, and report swings of 33% in performance for certain models depending on the dataset they are trained on. They release a corpus which could become widely used in the community in future code summarization studies. Their corpus is one of the four corpora used in this study, and all three models are trained and tested on it. In addition to the quality of the dataset, the difficulty of the test-train split can produce swings in models performance [55]. In particular they note that in the case of code summarization datasets, which are often obtained from Github, it is important that a particular repository or codebase is not spread across the test set and the train set, since a repository is more likely to contain very similar functions and/or duplicate functions.

In [56], Chandra et al. release an a training dataset and separate evaluation dataset for code search. It is different from other corpora for code search and summarization in that it uses StackOverflow questions as test-set and Github functions as training set. It is also different from the Pydoc and Javadoc datasets used in our study, in that the training-set does not include docstrings. Instead, only the function name and body, url and filepaths are provided, and the features to-be-matched to the query must be extracted from them, for example extracting English words and code tokens from the function name, by splitting it by underscore or came case. Because this dataset was not released at the same time as the NCS model that used it for evaluation, it was not available at the time the experiments for the present study were conducted.

Husain et al. [47], release a training and evaluation corpus specifically designed for evaluating code search models. Like the CoNaLa challenge for the English-to-code task, the challenge was specifically designed to compare models and help assess better the state of the art in this area.

In the image-classification domain, Recht et al. [84] hypothesize that by repeatedly evaluating

models on very few benchmark datasets, for example CIFAR-10, keeping the test-train split constant over the years, the published models of gradually increasing scores over the years could be gradually overfitting those datasets, instead of being true advances. They find that on their new test-train split for CIFAR-10, all models score significantly lower, but that the ranking of the models relative to each other remains, surprisingly, roughly the same. This indicates, according to their analysis, that their new test-train split is more difficult than the original test-train split conventionally used by previous studies, but that model are not overfitting CIFAR-10, since the ranking remains approximately the same. In the present study, we have not examined the scores on different test-train split, so we cannot come to a conclusion on this topic. What we do observe, however, is that the ranking of models varies greatly across datasets for a same task.

Importance of using several datasets for a more fair assessment of models

Although the importance of using benchmark copora is well established in all domains of machine learning, it is rare to see researchers advocate for testing models over several datasets, to obtain a more complete picture of the comparative strengths and weaknesses of different models. Indeed, models can be more sensitive to a certain type of dataset-artifact in one dataset but not another, but also, there is the danger that models are, over time, selected to overfit one dataset in particular.

Some researchers have produced challenge datasets for NLP-related tasks. Goel et al. [32] develop a set of challenge datasets that test the robustness of neural language models, their capacity to generalize to new questions, bias and security. They want to make it easier for researchers to test the performance of their models, on different dimensions of performance (generalization, bias and security), using different datasets and tasks. In particular, they implement "five subpopulations that capture summary abstractiveness, content distillation, information dispersion, positional bias and information reordering". They use their evaluation framework in case studies, and find that the variety of challenges helps to obtain a more informative picture of the performance of the models, and helps to identify the particular strengths and weaknesses of models on the different dimensions mentioned.

In other application-domains of machine learning, researchers have also found that assessing models on more than one dataset or challenge is crucial to obtain a more precise assessment of their performance. Coleman et al. [22] develop an automated technique to generate evaluation tasks that vary in certain key characteristics. They hypothesize that in order to properly assess alternative neural models, to reach more robust conclusions about their relative performance, it is necessary to test all models on a wider variety of challenges, and environments. They test three different neural models in a variety of environments and find that different challenges, datasets and different test-train splits affect models differently, and report that model ranking is not consistent across

different environments. As we also observe in the present study, it is important to test models on a wide variety of datasets in order to reach more robust conclusions about their relative performance, as well as their respective strengths and weaknesses.

Dataset Difficulty in the field of NLP

In the field of NLP, several papers have tried to predict model performance, and / or assess dataset difficulty. It is generally observed that machine learning models, including neural models, do not perform well on test questions that are too different from examples in the training-set, also called "out-of-distribution" questions.

In [105] Xia, Neubig et al. seek to predict the performance of models on a variety of NLP tasks, in order to determine where current models can best be applied. Several models must be tested on several datasets, and the combinations become exponentially large, and the computations expensive. Is it possible to predict which dataset and model combination will yield high publishable results? Which are the fruitful problems to apply ML models to? They observe that a high performance of models on a subset of datasets for a task, say, translation, is not always a good predictor of performance for all datasets available for the task. They observe that commonly used datasets are largely not correlated in difficulty and taken individually are not representative of the difficulty of the NLP task. They develop a lean regression model that is able to predict model performance on a task and dataset with low computational complexity compared to training the full model. They produce a random forest classifier to predict performance of models on different NLP tasks, based on input features such as dataset size, vocabulary size, sentence length. They show that their predictor can be more accurate than human experts in predicting performance of the models. They find that the predictor when trained on a representative sample of datasets for a particular task, allows them to obtain plausible predictions of model performance on the rest of the dataset for the task. Their observations support the idea that evaluating models on only one particular benchmark dataset is unrepresentative of results over a large spectrum of datasets for a particular NLP task; differences in dataset difficulty can be large. Indeed, it is not because a model scores high on a dataset that the problem is solved. Any particular dataset might be easier than the average, for example, if it has artifacts, as we found in some of our experiments.

Similarly, Wang, Neubig et al. [104] develop an algorithm for selecting training data for neural translation models, that will maximize translation performance for a given target sentence to produce.

Bugliarello et al. [14] seek to identify the causes of natural language translation difficulty on different languages. They identify only two metrics that significantly correlate with dataset difficulty: *source-side type-token ratio* and *the distance between source and target languages*. They find no evidence that translating into morphologically rich languages is harder than into morphologically

impoverished ones.

In [94] Scheidegger et al. propose and test different measures of dataset difficulty for several of the most popular image classification challenge datasets (MNIST, CIFAR, etc.). They test three methods for assessing dataset difficulty: silhouette score which measures the separability of the classes as the proxy for dataset difficulty, k-means clustering, and finally probe nets which trains a small neural network on the task as a proxy for the performance of the bigger networks. They show that their dataset difficulty assessment approach using probe nets can run 27 times faster than a training run, for the common neural architectures, given the same resources. Probe nets is also the best performing approach for predicting the difficulty of a dataset, correlating best with the actual scores of the actual models networks on the datasets.

Evaluating Generalization in Neural Models

In [51], Lake et al. assess the capacity of neural sequence-to-sequence model to generalize on a particular set of questions, specifically designed to measure generalization: the SCAN dataset. They observe that sequence-to-sequence models do not generalize beyond very simple examples in their experiments, and postulate that these results might be observable with other models and datasets. In [89] Russin et al. develop a compositional sequence-to-sequence model which learns syntax and semantics separately, and test it on the SCAN generalization benchmark dataset [51]. They demonstrate results substantially higher than previous works, hinting that compositionality and modularity priors in neural model could allow for better generalization performance.

In [77] Nogueira et al. examine the ability of large pretrained language models to generalize on an arithmetic task. They show that big Transformer models can only perform arithmetic on small numbers, which are more likely to be present in the training set. Their work, despite some initially encouraging results when feeding modified number encodings to the model, is ultimately an indication that Transformer models are unable to generalize to symbolic tasks, and are mostly memorizing, overfitting the training data. Specifically, they observe that the model can only perform arithmetic on numbers that have the same length as the ones in the training data. Since programming code and an arithmetic computational graphs are similar in nature, their paper can help shed some light on the poor generalization capacity displayed by Transformer models in the code-to-English and English-to-code domains.

Again in the NLP domain, Jastrzebski et al. [49] evaluate a 2-layer DNN from previous work along with statistical count-based models (Bilinear, Factorized, Prototypical) on a task of knowledge-based extraction mining. They want to assess the degree of generalization of the models to test-set questions that they consider "novel", i.e. sufficiently different from the training-set. They observe that model scores degrade rapidly as the novelty of a test-set question increases. They also note that high scores

reported by previous studies relied on a high number of test-set questions that were quasi-duplicates (minor rewordings). Their work is similar to ours in that they develop a measure of novelty of a test-set question (how different is it from the closest training-set examples), and examine the performance of each model depending on the measured novelty of the test-set question. As measure of novelty between a test-set question and a training-set example, they compute the distance of word embeddings between corresponding items in a relation triple. For example, replacing an entity in the triple with a close synonym would not create significant novelty. They notice many trivial test-set questions in datasets of previous work, i.e. test-set questions that present a very small degree of novelty to a training-set example, according to their metric. To evaluate the models, they produce test-set question "bins" of increasing novelty, and observe that the performance of all models degrades rapidly as the questions become more novel. They observe that the deep neural network model (DNN), in their evaluation, shows the greatest degradation in performance as the novelty of the test-set questions increases in each test-set bin, compared to simpler count-based models. In addition to this comparative work and assessment of generalization, they observe that 60% of test-set questions from prior work under evaluation consist of minor rewordings of training-set examples. This is also another point of comparison with our study, since we also detected the presence of a large percentage of excessively easy, or "unfair" test-set questions, in particular on the C2E-Javadoc and E2C-Javadoc task-dataset combinations. In our case, the excessively easy or unfair questions were due to *the test-set question containing the answer* (as described in Section 5.3), whereas in their case, they observed a large number of test-set questions that were quasi-duplicates, i.e. minor rewordings of examples from the training-set. They conclude that larger datasets will be necessary to successfully mine novel common sense relation triples for knowledge bases; however, we seriously doubt that dataset size will be a sufficient condition for increasing generalization in machine learning models, We suggest thoroughly examining the effect of dataset size on capacity to generalize to questions with high measured novelty, in future work, as we discuss in Chapter 6.

Dataset Artifacts

Dataset-artifacts are spurious correlations between question and answer in machine learning datasets [65], which allow models to trivially answer the question, thereby inflating their score. In the field of NLP, several papers have studied dataset-artifacts, and how they affect neural model scores on different tasks ranging from text-generation to question answering.

McCoy et al. [65] observe that models can perform well on a dataset, yet their performance drops on other datasets relative to other models. They study how this problem affects neural sequence-to-sequence models, and observe that discrepancies in scores are frequently caused by the reliance of sequence-to-sequence models on spurious patterns and dataset-artifacts to obtain high scores on a

particular dataset. Wallace et al. [102] show that problematic dataset collection methodology can often lead to dataset biases, or "artifacts". They note that these spurious correlation between a hypothesis and the classification label can misrepresent the true performance of neural models on NLP tasks. Si et al. [96] examine what the pretrained BERT model can learn from multiple-choice reading comprehension datasets. They observe that BERT is exploiting dataset-artifacts and statistical regularities, which allow it to answer questions correctly without the full context. Dua et al. [25] observe that "dataset artifacts" can be used by neural models to increase their score without having to learn to reason about the question, in the way humans expect it to. Ross et al. [86] present MICE, or Minimal Contrastive Editing, a technique for generating contrastive explanations of model predictions, which can be used to uncover dataset-artifacts, and debug incorrect model predictions. In our study, we use a completely different procedure, the BOS, to uncover dataset artifacts, and our procedure also allows to debug low-scoring, or incorrect model answers. Trivedi et al. [101] ask whether there has been real progress made in the area of multi-hop question answering, and note that models often rely on dataset artifacts to produce correct answers, "without connecting information across multiple supporting facts".

Gururangan et al. [37] show that in a significant portion of datasets for natural language inference, it is possible to identify the label by looking only at the hypothesis. They suggest that due to inflated scores, caused by these dataset-artifacts, the success of natural language inference models in recent years has been overestimated.

Chapter 3

Background and Definitions

In this Chapter, we introduce the background on the three models being evaluated in this study: Semantic Code Search (CodeSearch), Sequence-to-sequence BiLSTM with Attention and Beam Search (Seq2SeqLSTM), Sequence-to-sequence Attentional-GRU (AttendGRU). We then describe the outcome measure used, BLEU4, and the BOS ceiling procedure, the retrieval ceiling, which is used as a reference point for scores of all models.

3.1 Models

3.1.1 Semantic Code Search

The semantic code search model reproduced in this study is based on a model from Sachdev et al. [90], called Neural Code Search (NCS). Vector space retrieval models such as semantic search models have been extensively used in the information retrieval domain for several decades [62, 15, 93]. The NCS model is a variant of the vector space retrieval model that uses neural word embeddings (Word2vec) as the vector space for matching documents. This reproduced model is used in our experiments, both for the English-to-code and code-to-English tasks, to retrieve code snippets, and English descriptions of code.

CodeSearch Description. The CodeSearch model runs a k-nearest-neighbour search on English intents from the training-set, which are represented using a weighted bag-of-words approach. More specifically, a high-dimensional vector is computed for each intent in the training-set, as the TF-IDF-weighted average of their Word2vec embedding vectors. At training-time, all training-set English-intent vectors are pre-computed and stored. When presented with a new query, the model vectorizes it using the same procedure, and searches the training corpus for the text-label (intent) that is most similar to the query, according to the Euclidean distance, to then return its associated

content (code snippet) as the answer. The CodeSearch model can be reversed for the code-to-English task, by instead vectorizing the code snippet for each training example, performing a Euclidean similarity search with a new query or test-set question (i.e. a code snippet), and returning its associated English intent, as the code summary.

NCS is called a *semantic* search model because, by using Word2vec embeddings, it is able to match test-set questions (or user queries) to a training-set intent even if that intent uses synonyms or semantically related words, instead of the exact same terms. The system is called a Neural Code Search because it relies on neural word embeddings. NCS retrieves an existing code snippet from the training-set, while neural seq2seq models generate a code snippet, token by token.

Although our CodeSearch model is mostly based on NCS [90], we changed the distance metric used from cosine distance (used in original paper) to Euclidean distance (ours). Cosine distance is often preferred for matching documents vectors, as it will be sensitive to *the ratios of words in the intent, and query*, rather than to their absolute numbers. This means that a short document will be matched to a very long document with an identical ratio of words used, before being matched to another short document of identical length with a very close, but different ratio of words. In our case, for the code generation and summarization tasks, since models are evaluated with the BLEU procedure, it is arguably more appropriate to use the Euclidean distance, as we want to retrieve documents not only with similar word usage and meaning (semantics) but also similar in length. We reason that a training-set intent with a similar word count to the query is more likely to have a code snippet similar in length to the reference, obtain a higher ratio of n-gram overlaps, and therefore a high BLEU score. If on average in a dataset intent length is not correlated with the length of its code snippet, then Euclidean distance might not increase BLEU scores. Future work should compare outcomes for the two distance metrics.

Only the highest-ranked recommendation is used in our experiments. Another important note and difference from the original NCS paper [90] is that, in all our experiments, only the top-1 recommendation is considered to produce the BLEU score for the code search model. In [90] the more lenient Mean Reciprocal Rank (MRR) metric is used, which assigns credit to the model when the best answer is not the first returned, but does appear in its top-k recommendations. In practice, if the user of an English-to-code system is able to look at a few answers, a code search model could become even more helpful than what our results could reflect. Additionally, as we pointed out in Chapter 1, the evaluation in [90] is done only on questions for which a perfect answer exists in the dataset, which is equivalent to testing on a test-train split where all answers have a BOS of 1.0, or very close to it (duplicate or quasi-duplicates). This is not the case in the present study: most test-set answers do not have a relevant examples in the train-set, as shown by the examination of BOS results in Section 5.3. Despite these limitations, the code search model is the best performing overall, as we will discuss in Chapter 5.

3.1.2 Seq2seq BiLSTM With Attention and Beam Search

The Seq2SeqLSTM model replicated in this study is based on the model used by Yin et al. [111], a sequence-to-sequence bidirectional LSTM with attention and beam search. Seq2seq LSTM models were originally developed for natural language translation [98] [9]. They were later applied to several other problems that make use of sequential data: music generation [100], speech recognition [74], text summarization [118], and more. In this study the seq2seq LSTM is used in both English-to-code and code-to-English tasks, to generate code and English descriptions of code.

LSTMs Description. The LSTM, or long short term memory network, is a recursive neural architecture developed by Hochreiter et al. in 1997 [42] for iterative sequence generation. In LSTMs, like in other Recurrent Neural Networks (RNNs), and unlike in regular feed-forward neural networks the inputs are fed *in sequence* (time-series): they are provided one at-a-time. LSTM cells process inputs in sequence and compute the next activations, which are *fed back to itself* and to the next layer or output, at the next time-step. Because of these recursive activations, backpropagation must happen *through time*, in addition to flowing back through the layers. The LSTM variant used in this study contains three memory gates: a forget gate, an input gate, an output gate. The internal gates of the LSTM cell, which are themselves feed-forward neural networks, learn to produce a compressed, synthetic representation of the sequence at each time step. They learn to selectively discard, or retain, information that is less predictive for the task during training, and keep track of the most predictive information from past events. LSTMs, with their particular architecture of recursive connections and gates (particularly the Constant Error Carousel) were aimed at overcoming the vanishing gradient problem during backpropagation-through-time, which was present in older recurrent neural network (RNN) architectures and made training on long sequences infeasible [41].

The equations for computing LSTM weights at each of its gate, during the forward pass, are as follows:

$$
\begin{aligned}
f_t &= sigmoid(W_f x_t + U_f h_{t-1} + b_f) \\
i_t &= sigmoid(W_i x_t + U_i h_{t-1} + b_i) \\
o_t &= sigmoid(W_o x_t + U_o h_{t-1} + b_o) \\
\tilde{c}_t &= tanh(W_c x_t + U_c h_{t-1} + b_c) \\
c_t &= c_{t-1} \circ f_t + g_t \circ \tilde{c}_t \\
h_t &= o_t \circ sigmoid(c_t)
\end{aligned}
$$

where

$$
\begin{aligned}
& x_t \in \mathbb{R}^d : \textit{input vector} \\
& f_t \in \mathbb{R}^h : \textit{forget gate activation vector} \\
& i_t \in \mathbb{R}^h : \textit{input gate activation vector} \\
& o_t \in \mathbb{R}^h : \textit{output gate activation vector} \\
& h_t \in \mathbb{R}^h : \textit{hidden state vector, i.e. output vector} \\
& \tilde{c}_t \in \mathbb{R}^h : \textit{cell input activation vector} \\
& c_t \in \mathbb{R}^h : \textit{cell state vector} \\
& W \in \mathbb{R}^{h \times d}, U \in \mathbb{R}^{h \times h} and b \in \mathbb{R}^h : \textit{weight matrices and bias vector}
\end{aligned} \tag{1}
$$

Seq2seq high-level description. The seq2seq architecture for translation [98] consists of an encoder and a decoder, which both are stacks of LSTM cells. The encoder takes as input the word-tokens from the sentence to translate and produces a thought-vector, which is generally considered to represent the "contextualized meaning" of the current word-token. The decoder takes as input (1) the thought-vector produced by the encoder at the current time step (bi-directional, contextualized meaning of current word in view of the whole sentence) and (2) the already-generated words in the translation (which can act as a second form of contextualized meaning, also predictive of the next token). Note that in general words are inputted as Word2vec embeddings, for both languages (English and code in our case).

BiLSTM architecture. In the present study, the seq2seq LSTM model that we evaluate uses a BiLSTM architecture for its encoder. An LSTM encoder consists of LSTM cells, which are stacked on top of each other; the top cell (i.e. layer) produces the state-vector (thought-vector). In a BiLSTM encoder however, each layer consists of two cells: (1) one which does a forward pass on the words, *up to the current word*, outputting a vector (hidden state) at each time-step (each new word), and (2) another which does a backward pass on the words, *down to the current word* (starting from the end of the sentence), outputting a vector (hidden state) at each time-step (each new word in reverse).

The outputs from the two LSTM cells in each layer are concatenated and fed to the next layer (or to the decoder if the encoder has only one layer). BiLSTMs were shown to better capture context for a word than LSTMs; this is because they consider words both before *and* after the current word, at each time steps. Intuitively, it does make sense that future information in a sentence can be predictive of the current word-token, and the BiLSTM takes advantage of this.

Attention mechanism in Seq2SeqLSTM. Since earlier works on statistical machine translation (SMT), we know that information about word alignment can improve next token predictions and improve translation BLEU scores. We also know, from the use of TF-IDF in retrieval models, that focusing attention on certain words more than others, by specifying weights for each word, is a very effective technique for improving the quality of semantic vectors representations for documents or sequences of text-tokens. A few months after the original seq2seq architecture was proposed by Sutskever et al. [98], Bahdanau et al. [9] added an attention mechanism that improved the seq2seq's performance on translation tasks. This attention mechanism simply learns weights for each hidden state of the concatenated thought-vector described earlier. In other words, the attention mechanism learns *which time-steps* in the encoder's thought-vector are more "important" in order to predict the next word-token; i.e. it learns which contextualized-word-representation to focus on in the encoder's representation of the source sentence, at any given time-step.

Beam search. Beam search is applied to the output of the Seq2SeqLSTM model. Beam search is external to the neural network. Instead of keeping only the most likely token at each time-step (greedy approach), it keeps a "beam" of top-k probabilities at each time-step, and picks the combination of tokens that maximizes the *overall* probability of the sequence produced by the seq2seq's decoder. Beam search can help maintain coherence in the decoder's outputs, especially on tokens that are very frequent and can act undesirably as "pivots" to change the main topic of the generation. Beam search has been observed to sometimes improve scores substantially in translation tasks, for example, Huang et al. [46] report an average BLEU score increase of 4.2 on 4 benchmark datasets, while [83] report a 2.2 increase.

3.1.3 Attentional-GRU

The AttendGRU model replicated in this study is based on the model used by Leclair et al. [54], a sequence-to-sequence attentional GRU. In our study the sequence-to-sequence attentional GRU is used in both English-to-code and code-to-English tasks, to generate code and English descriptions of code.

Bahdanau, Cho et al. [20] develop the GRU as a slightly simplified, more computationally efficient alternative to the LSTM cell. The GRU is similar to an LSTM but has only two gates instead of three. The work from Cho et al. also introduced the idea of the sequence-to-sequence architecture,

which was then adapted to use the older LSTM unit, in subsequent studies [9, 98].

The equations for computing weights at each gate of the GRU, during the forward pass, are as follows:

$$
\begin{aligned}
z_t &= sigmoid(W_z x_t + U_z h_{t-1} + b_z) \\
r_t &= sigmoid(W_r x_t + U_r h_{t-1} + b_r) \\
\tilde{h}_t &= tanh(W_h x_t + U_h(r_t \circ h_{t-1}) + b_h) \\
h_t &= (1 - z_t) \circ h_{t-1} + z_t \circ h_t
\end{aligned}
$$

where

$$
\begin{aligned}
& x_t \in \mathbb{R}^d : \textit{input vector} \\
& z_t \in \mathbb{R}^h : \textit{input gate vector} \\
& r_t \in \mathbb{R}^h : \textit{reset gate vector} \\
& h_t \in \mathbb{R}^h : \textit{output vector} \\
& \tilde{h}_t \in \mathbb{R}^h : \textit{candidate vector} \\
& W \in \mathbb{R}^{h \times d}, U \in \mathbb{R}^{h \times h} and b \in \mathbb{R}^h : \textit{weight matrices and bias vector}
\end{aligned}
\tag{2}
$$

GRU sequence-to-sequence architecture with attention. In the same way the LSTM cells are used in the encoder of the Seq2SeqLSTM, the GRU cells are used for the encoder and decoder of the AttendGRU model. Similarly to the Seq2SeqLSTM, an attention mechanism is used in the AttendGRU model. The AttendGRU model does not use beam search in its implementation from Leclair et al. [54]. The authors mention that since the main aim of their study is to compare an AST model to a non-AST one, they decided to keep the models as simple as possible.

3.1.4 Reversal of Models For Back-Translation

The English-to-code (code generation) and code-to-English (code summarization) tasks are often approached as a translation problem. Statistical Machine Translation (SMT) models, text-retrieval models, or sequence-to-sequence translation models (such as the sequence-to-sequence BiLSTM and AttendGRU presented in 3.1.2 and 3.1.3) can be used to map and translate in both directions, on the two tasks. Translation models can, often with minimal changes, be retrained successfully on the reversed corpus to perform translation in both direction; most translation model architectures are essentially language-agnostic, and were designed for that purpose [98]. However, this reversal for back-translation might *not* be possible if the model uses a different representation and encoding for the code and the English; for example, using AST to represent code and word-tokens to represent

English. If the representation of the code and the English intent are different, the reversal for back-translation might require changes in the model's architecture and / or preprocessing. This study only uses published models that use word-tokens, since AST embeddings for code are more complex to run in reverse, and have been shown to provide only modest improvements [54], in the order of 1 BLEU4 point on a same corpus.

Reversal of retrieval model for docstring search. When approaching the task as a search problem, it is also frequent that the search model can be reversed, with minimal changes, to search in both directions (search for a code snippet given an intent, or search for an intent given a code snippet). Again here, reversal is possible only if the search model uses the same representation for code and English (for example word-tokens embedded as word2Vec vectors).

3.2 BLEU and BOS Scores

3.2.1 Background on BLEU score

The main purpose of the BLEU score is to evaluate translations automatically, without human intervention, which is costly and time consuming. The BLEU score was developed at IBM, by Papineni et al. [79], and was the first translation evaluation metric to be shown to correlate highly to human assessment. As we saw in Section 2.4, many other evaluation metrics are shown to correlate better with human assessment, including a simple F1 score [66] (harmonic mean of precision and recall of n-gram overlaps, for any desired n). However, the BLEU score rapidly gained popularity and became the dominant metric for ranking SMT models and later neural translation models. Often, in order to compare new scores with previous studies, the BLEU score must be used.

Computing BLEU scores. The BLEU score, in short, is a measure of the number of n-gram matches between two sentences. To compute it, the first step is to extract the tokens in both the reference-translation and the hypothesis generated by the model. In order to be consistent across evaluations, preprocessing steps must be identical: the same tokenization rules (e.g. whether punctuation are marks excluded from tokens), case insensitivity, stemming rules (if used), as well as the same n-gram order (4-gram in our case) must be used.

After preprocessing, matching n-grams from the reference-translation and the hypothesis are counted. For each n-gram order for *n = 1, 2, 3, ... k*, the percentage of matching n-grams is calculated:

$$P(n) = \frac{Matched(n)}{H(n)}$$

where $H(n)$ is the number of n-gram instances in the hypothesis. This value is the precision for the given n-gram level. In both the CoNaLa challenge [111] and the AttendGRU study [54], n is 4, leading to the BLEU4 score.

Matched(n) counts the number of n-gram matches in the hypothesis, but only up to the maximum number of instances of that n-gram in each reference (there can be more than one reference). Matched(n) is computed like so:

$$Matched(n) = \sum_{t_n} \min\{C_h(t_n), \max_j C_{hj}(t_n)\}$$

where t_n is an n-gram instance in hypothesis h; $C_h(t_n)$ is the number of times t_n occurs in the hypothesis; $C_{hj}(t_n)$ is the number of times t_n occurs in reference j.

Then, the geometric average of the precision scores is computed:

$$BLEU_a = \left\{ \prod_{n=1}^{K} P(n) \right\}^{1/K}$$

The BLEU score does not consider recall in its calculation. It is a precision-based metric. For this reason, it would be possible to cheat the score, by generating very short sentences that only consist of n-grams that the model is very sure about. To penalize such a strategy, the brevity penalty was introduced:

$$\rho = \exp\left\{ \min(0, \frac{n-L}{n}) \right\}$$
$$BLEU_b = \rho \; BLEU_a$$

Finally, since the precision of higher-order n-grams is considered more important than lower-order n-grams, some weights to the precision of different n-grams are introduced:

$$BLEU_c = \rho \prod_{n=1}^{K} P(n)^{w_n}$$

where $\sum_{n=1}^{K} w_n = 1$.

In the present study, all our BLEU score evaluations are computed with the CoNaLa challenge evaluation script from Yin et al. [111], available online [110].

We compute the BLEU score at the test-set level in order to comply with the CoNaLa challenge evaluation, and compare BLEU4 scores with previous studies. To compute the BLEU score at the test-set level for a model, we pass the complete list of model hypotheses to the CoNaLa evaluation script along with the test-set answers, to then obtain the test-set-level BLEU4 score.

BLEU4 scores on their own are misleading, as the score can be high despite a high number of answers scoring zero. In addition to the test-set-level BLEU4 score, we computed the sentence-level BLEU4 score, to observe the distribution of BLEU4 scores on a per-question basis (see Figures 2 to 8). To compute the BLEU4 for a given sentence (i.e. hypothesis for one given test-set question), we pass one hypothesis to the CoNaLa evaluation script along with its corresponding test-set answer, to obtain the sentence-level BLEU4 score for that question. Note that because of the peculiarities of the BLEU score procedure, in particular the brevity penalty, the test-set BLEU4 score is not exactly equivalent to the average of the sentence-level BLEU4 scores for the same test-set.

3.2.2 Introducing the BLEU Optimal Score (BOS)

We make a novel contribution by introducing the BLEU Optimal Search (BOS) score. To obtain the BOS score for a given test-set question, we simply search for the highest-scoring training-set example for that test-set question. We do so by exhaustively computing the BLEU score of every example in the train-set against the answer for that test-set question, and retaining the highest. The BOS is the theoretical ceiling score for a retrieval model on a given test-set question, and is used as reference point for all model scores in our results.

How is the BOS useful, and why do we use it? We find that the BOS is particularly effective to:

1. Debug any machine learning model, and inspect whether low-scoring answers are only due to a lack of relevant training-data or a problem with model training, or model code.

2. Examine the model's capacity to generalize, i.e. to successfully generate new, high-scoring sequences of tokens that are not already present in the training-set. In this sense it is a very good, and easy complement to generalization-specific benchmarks such as SCAN [51].

3. Help to identify hard-to-find dataset-artifacts beyond simple duplicates, as we do in section 5.3, given the knowledge that certain models generally do not surpass the BOS without the presence of a dataset-artifact.

4. Provide a reference point, and allow one to better compare scores across different datasets and test-train splits (since some can be substantially more difficult than others for models, as shown in [54, 85]). It could help identify which machine learning models do generalize occasionally on a certain task, even if those models are trained and benchmarked on different datasets.

5. Discover if some questions are likely unanswerable, and discover what the information retrieval ceiling is for the question, since models rarely surpass the BOS; in practical applications of machine learning, this knowledge can also help to discover which types of questions should be targeted for additional data collection, or for data augmentation for example.

When the training and test sets are too large to exhaustively calculate the top answer in the training set, we randomly sample test-set questions and calculate the BOS across this sample. The BOS BLEU score can be computed for each individual test-set question. The BOS BLEU score can also be computed at the test-set level, as is done for any model used on the task. The BOS procedure is described in Listing 1.

Algorithm 1: BOS procedure

```
initialization;
Set list_of_BOS_for_each_question to empty list;
foreach test_set_reference do
    max_BLEU ← −1;
    foreach training_set_answer do
        current_BLEU ← computeBLEU(test_set_reference, training_set_answer);
        if max_BLEU < current_BLEU then
            max_BLEU ← current_BLEU;
        end
    end
    list_of_BOS_for_each_question.append(max_BLEU);
end
return list_of_BOS_for_each_question;
```

In an idealized setting where all test-set and training-set entries have the same length, the time complexity of the naive, or brute-force BOS score calculation for a given test-set can be expressed as:

$$O(n \cdot l_h \cdot l_r \cdot S_{tr} \cdot S_{te})$$

where n is the order of the BLEU score (4 in the present study), l_r is the number of word-tokens in the reference (answer), l_h is the number of tokens in the hypothesis S_{tr} is the number of training-set entries and S_{te} is the number of test-set entries. BOS scores can be long to compute, especially on large datasets. On our datasets, the BOS took between approximately 15 minutes and 6 days to compute, for a set of 500 or 610 test-set questions extracted from the smallest and largest datasets we used (computation times were obtained on a standard personal computer with specifications described in section 4). We should note however that, during our BOS calculations, we calculated not only the BLEU4 BOS, but also the BLEU2 BOS, the BLEU1 BOS, and similarly the F1 (precision, recall and their harmonic mean). The time could have been substantially decreased if those four additional metrics were excluded from the script. Also, we should note that BOS calculations could be parallelized if ran on very large corpora (we did not parallelize them). Another possible optimization for the BOS calculation would be to initially index all dataset answers by 4-gram, and exclude, during the BOS computation, all training-examples that don't present at least one 4-gram match with the current test-set question; this optimization is possible because the BLEU score for a question is

zero if the model answer and the test-set answer (reference) have no 4-gram in common. Further optimization can be achieved by considering only training entries which have the greatest number of 4-gram matches in the dataset for the test-set question, since they are also guaranteed to have a higher BLEU score. However, we did not implement n-gram indexing and use instead the naive, or brute force implementation for the moment. We leave the optimizations of the BOS calculation for future work. Because of its long computation time on the larger datasets (Javadoc in particular with 90k test-set entries), we computed the BOS for a random sample of test-set questions for the following task-dataset combinations: C2E-Javadoc (610 randomly sampled questions among 90,908 test-set answers) and E2C-Javadoc (500 randomly sampled questions among 90,908 test-set answers), Pydoc-E2C (3882 questions out of 3885 test-set questions, due to a task hanging on the cluster toward the end). For the C2E-Javadoc, we later computed the BOS on a larger random sample of 10,399 out of 90,908 test-set questions, which further confirmed our results, as discussed in section 5.4. A summary of the random sample sizes for the BOS analysis for each dataset can be seen in section 4 in Table 9

We computed the BOS for the totality of test-set questions for the following task-dataset combinations: SOa-E2C (500 questions), SO-E2C (500 questions), SOa-C2E (500 questions), SO-C2E (500 questions), Pydoc-C2E (3885 questions). After obtaining a few hundred of those random samples, we can already obtain a very good picture of the model's performance on different types of questions. In light of its usefulness for assessing model performance and generalization, as well as its use for finding dataset-artifacts, visualize the number of test-train duplicates, close-duplicates, etc. we believe that the moderate amount of computation necessary to calculate the BOS is a relatively small downside.

Corpus-level BLEU score versus per-question BLEU score. BOS scores are computed for each test-set question, and compared, or correlated against model scores, which are also calculated on a per-question basis (as is done in the scatter-plot figures 9 to 18 in section 5.3). An overall BOS score at the test-set level can also be computed, by scoring the entire list of top-scoring training-set answers against the entire list of test-test answers, or reference (as can be seen in Tables 10 and 11 of section 5). Note that because of the particularities of the BLEU score calculation procedure, the average of *per question BLEU scores* for a test-set is *not* exactly equivalent to the BLEU score directly calculated on *the whole test-set*. As per the original BLEU metric procedure (Papineni et al. 2002) and as described previously in Section 3.2.1, the BLEU score for a test-set is not calculated by taking the average of each individual hypotheses' BLEU scores, but by considering *the whole set of model hypotheses for the test-set* as *a single hypothesis*, scored against the whole set of references.

Chapter 4

Experimental Setup

In this Chapter we describe the benchmark datasets and models from previous studies that we selected, as well as the preprocessing and training regimen used for them in our experiments. Finally we describe the evaluation procedure, and the collection of BOS scores.

4.1 Dataset Selection

CoNaLa Challenge. In 2018 a benchmark dataset for the English-to-code task was published, called the CoNaLa challenge [111]. Each competing model should submit an answer to every one of the 500 official test-set questions (the test-set is made public [110]). The BLEU4 score is then used to score the answer against the known code answer to evaluate the model's performance. This dataset uses exclusively Python-related StackOverflow questions, and all source code in it is in the Python language. Other Python code-English datasets are also allowed and recommended to be used for training, but the test-set, extracted from the filtered StackOverflow question-answer pairs remains the same for everyone. The CoNaLa dataset is separated into two different subset: a small, clean dataset called CoNaLa-annotated (noted SOa in our experiments), and a larger dataset that comprises both annotated dataset and additional English-code pairs (SO).

CoNaLa annotated (SOa, 3K English-code pairs). The CoNaLa "annotated" corpus is a small collection of 3K question-answer pairs from StackOverflow, curated by hand. The intents are re-written by hand by the study participant, to make them less ambiguous; they are meant by the authors to define the code snippet more accurately than the original StackOverflow intents, which are sometimes vague or not well formulated. They keep the original intent field, and add the disambiguated intent in a second field called rewritten intent.

CoNaLa mined (SO, 600K English-code pairs). The CoNaLa annotated data, because

Sample train-set question	Train-set answer
Generate all possible strings from a list of token	`print([''.join(a) for a in combinations(['hel', 'lo', 'bye'], 2)])`
trim whitespace	`s.strip()`
Delete a file or folder	`os.rmdir()`
How to convert strings numbers to integers in a list?	`changed_list = [(int(f) if f.isdigit() else f) for f in original_list]`
Calling an external command	`stream = os.popen('some_command with args')`

Table 4: CoNaLa annotated dataset (SOa): sample training-set examples

curated by humans, is of higher quality than the average SO post. As described in the study from Yin et al. [111], this cleaner, annotated data is used to condition a sequence-to-sequence model to later recognize good correlations between English and code. This model is then used to filter the larger set of SO posts, to produce the mined corpus, which contains more than 600K English-code pairs.

Sample train-set question	Train-set answer
Python regular expression matching a multiline block of text	`re.compile('^(.+)\\\\n((?:\\\\n.+)+)', re.MULTILINE)`
How to convert a date string to different format	`datetime.datetime.strptime('2013-1-25', '%Y-%m-%d').strftime('%-m/%d/%y')`
format strings and named arguments in Python	`\"\"\"{1} {ham} {0} {foo} {1}\"\"\".format(10, 20, foo='bar', ham='spam')`
check for valid arguments	`print(valid(y, (), {'a': 'hello', 'b': 'goodbye', 'c': 'what'}))\nprint(valid(y, ('hello', 'goodbye'), {'c': 'what?'}))`
Deleting already printed in Python	`if os.name == 'posix':\n  os.system('clear')\nelif os.name in ('nt', 'dos', 'ce'):\n  os.system('CLS')\nelse:\n  print('\\n' * numlines)`

Table 5: CoNaLa mined dataset (SO): sample training-set examples

Java-Javadoc corpus from Leclair et al. 2019 [54] (Javadoc, 2M code-English pairs). Leclair et al. released an English-Javadoc corpus [55], before their AttendGRU study examining the effect of AST representations of code in attentional-GRU models [54]. This corpus is based on pairs of Javadoc docstrings and their associated Java code, extracted from open source repositories. The parallel corpus consists of Javadoc and function bodies in the Java language (2.1M functions). Most docstrings present in this corpus are for internal documentation of software projects. The docstrings in that corpus generally do not describe low level API usages like SO posts. This corpus is used in the original AttendGRU study, and we train all three models on it to compare results.

Sample train-set question	Train-set answer
`registers an instance for a given class`	protected void register instance class test class t instance instances put test class instance
`return the selected button`	public jtoggle button get selected return selected button
`the actual worker method of the thread`	protected void do run throws throwable
`append an int value`	public jsonarray put final int value this put new integer value return this
`returns the button onclick attribute value or null if not defined`	public string get on click if attributes null return attributes get onclick else return null

Table 6: Leclair et al. 2019 code summarization dataset [55] (Javadoc): sample training-set examples

Python-Docstring corpus from Sennrich et al. 2018 [67] (Pydoc, 143K English-code pairs). In addition to the CoNaLa and English-Javadoc corpus we add a Python docstring corpus, from [67]. This corpus has 143K docstring-code pairs. We wanted to use this second docstring corpus to further confirm the scores and get a better idea of the behaviour of the models on this type of corpus (English-docstring corpus). This corpus is one of the additional corpora recommended by the CoNaLa challenge authors.

Sample train-set question	**Train-set answer**
upgrade the given database to revision	`with _temp_alembic_ini db_url as alembic_ini check_call alembic c alembic_ini upgrade revision`
returns a random item from the list	`return random_module choice value`
delete tarball	`with lcd logdir local rm le tar gz`
shelve a server	`_find_server cs args server shelve`
set the given properties on a group snapshot and update it	`return impl group_snapshot_update context group_snapshot_id values`

Table 7: Sennrich et al. 2018 code generation dataset [67] (Pydoc): sample training-set examples

Table 8 shows statistics for the datasets used in our study.

4.2 Model Selection

For our study, we decided to replicate two published state-of-the-art sequence-to-sequence translation models [111, 55] that have been developed for the English-to-code task and the code-to-English task, respectively. We settled on these models because (1) we wanted to compare the sequence-to-sequence LSTM and AttendGRU with the semantic search model, (2) they had both reached high BLEU scores compared to other similar models evaluated on the same datasets, for English-to-code and code-to-English respectively, on different datasets and (3) their code and dataset had been released.

The CodeSearch model reproduced from Sachdev et al. [90] uses the Gensim library for its Word2vec word embeddings, and the sklearn library for its TF-IDF vectorizer and kNN search.

The sequence-to-sequence BiLSTM with attention and beam search is a common off-the-shelf neural machine translation model by its authors, and is considered a baseline for the English-to-code by the authors of the CoNaLa challenge. Its implementation relies on the Xnmt-DyNet framework. Beam Search is used on sequential outputs of size 5.

The AttendGRU model from LeClair et al. [54] uses a Keras / Tensorflow implementation. The model comes in two variants: the first variant's encoder it is modified to concatenate 1) flattened-AST and 2) programming-tokens; the second version only receives programming-tokens, as in a standard encoder-decoder architecture. The AST structure is flattened into a string sequence during preprocessing. Leclair et al. find that using a combination of AST and words-from-code provides a

	CoNaLa-annot	CoNaLa-mined	Pydoc	Javadoc
Dataset size (number of English-code pairs)	2,379	593,891	143,125	2,059,080
Test-set size	500	500	4K	90K
Provenance	StackOverflow-derived	StackOverflow-derived	Github-derived	Github-derived
Programming Language	Python	Python	Python	Java
Non-alphanumeric whitespace and underscore tokens?	Yes	Yes	No	No
Median English length (tokens)	8	8	10	9
Median snippet length (tokens)	17	14	31	20
Purpose	code generation	code generation	code generation and summarization	code summarization

Table 8: Statistics of datasets used in this study

0.2 BLEU4 increase over using only one or the other alone, however, because the AST and non-AST models obtain very similar scores we choose to replicate the model that does not parse AST, for its simplicity and reversibility on the Englih-to-code task.

4.3 Preprocessing

We keep the studies' original preprocessing pipeline intact when possible. We note that the processing scripts from Leclair et al. [54] remove all non-alphanumeric tokens from the code and docstrings, while this is not the case for the preprocessing of the CoNaLa challenge dataset for the Seq2SeqLSTM [110]. We make both docstring-derived corpora (Javadoc and Pydoc) use the processing scripts from [54], so they both exclude non-alphanumeric tokens, both in the train-set and test-sets. This preprocessing was done in the Leclair et al. [54] study on code summarization, and is more appropriate for code summarization, because it provides new English word tokens from the source code, for the models to learn from. For example, a code snippet to summarize appears as such:

```
6313385, public void call throws exception point cost pair pair optimizer
minimize cost function max evaluations checker vertex a vertex b queue in
x add pair get point terminated true it is the same condition used to say
that new x values are available but now they are null return void null
```

and the intent or docstring to generate appears similar to this:

```
6313385, <s> called by the thread executor when the thread is started </s>
```

When training the models on a new corpus, we use the original study's preprocessing pipeline also for the two other models. For example, when training the code search model and the seq2seq LSTM on the Javadoc code-to-English, we used the original preprocessed data from [54] since it

is, and simply re-formatted it in a jsonl, so it can be read by the two other models. That way, all models train on an identical parallel corpus.

The preprocessing from Leclair et al. filters entries that are longer than 13. We had to disable this option before training the AttendGRU on the CoNaLa dataset, since all other models have to train on an intact version of the data, for a fair comparison for the challenge. This option was only enabled on the C2E-Javadoc corpora: we simply used the intact Javadoc corpus from LeClair et al., and trained all models on it. On the Pydoc dataset, we also used the preprocessing from LeClair et al., for all models, but removing the limit of 13 tokens for translation.

4.4 Training

We run and collect these two models' BLEU scores on all the task-dataset combinations we selected: (1) first, on their original study datasets then available, to confirm we obtain the same scores (2) on the remaining datasets from other studies [111, 67, 55]. We train them and collect BLEU4 scores each time for translation and back-translation (i.e. English-to-code and code-to-English). The two sequence-to-sequence models, Seq2SeqLSTM and AttendGRU were trained with Nvidia Tesla V100 series GPUs with 16GBs of RAM, from a GPU cluster. The CodeSearch model was run for training and retrieval on a cluster with access to high RAM: we used 80 GBs for all task-dataset combinations.

For the CodeSearch model, we used the Gensim library for the Word2vec model, and sklearn for the TF-IDF vectorizer and kNN model, with the following parameters: Word2vec embedding size 800, min_count 1, epochs 100; kNN algorithm "brute" i.e. exhaustive search.

For the Seq2SeqLSTM and AttendGRU, we kept hyperparameters identical to the original studies as much as possible. Vocabulary sizes for the Seq2SeqLSTM on each corpus are already adapted to the size of the dataset in the open-source Seq2SeqLSTM repository: 4K for SOa (original study value), 16K for SO (original study value), Pydoc and Javadoc. Vocabulary sizes for the AttendGRU model were set to 44707 for English tokens and 75000 for code tokens (also original study value). For the Seq2SeqLSTM: activation ReLU, dropout 0.3, default_layer_dim 512, trainer AdamTrainer with learning of alpha 0.001, avg_batch_size 32, beam search size 5. AttendGRU: activation ReLU, dropout (none), embdims 256, learning rate (unspecified), batch_size 200, beam search (none). Note: the batch size for AttendGRU had to be reduced to 32 and 16 on the two larger docstring datasets in the E2C direction, in order to fit into GPU RAM.

Leclair et al. [54] used early stopping to train their models. We also used the weights from the epoch that had the best accuracy (next-token prediction accuracy) on the validation-set after 100 epochs, and this this is similar equivalent to the procedure followed by the Seq2SeqLSTM from [111]. On all datasets, the Seq2SeqLSTM is run for 30 epoch, and the best scoring epoch on the

validation-set is used for generations. The AttendGRU is run for more epochs (100), since it was not performing as well, we wanted to maximize its chance of obtaining a high score. In general, however, the AttendGRU's validation-set loss plateaued after 10-20 epochs.

All three models can be run in reverse for back-translation. We use the version of the AttendGRU from [54] that does not use an abstract syntax tree (AST) representation of code inputs, but rather sequences of tokens, both for code and English words. The BiLSTM also handles words and code tokens indiscriminately for both encoder input and decoder output. For this reason, these models can be reversed for back-translation, and, as we will show, with minimal modifications, still generate state-of-the-art scores on new datasets (as the Seq2SeqLSTM does on C2E-Javadoc), or in other cases scores comparable to the other two models that were originally designed for the task. The code search model can also be reversed successfully to search for English intents, given a code snippet. All three models can be reversed for back-translation without any significant modification aside from vocabulary size.

4.5 BLEU Evaluation Metric

The BLEU4 score is used to score the model's answer (hypothesis) against the known code answer (reference) to evaluate the model's performance. To evaluate all models, we use the CoNaLa challenge BLEU4 evaluation script.

4.6 BLEU Optimal Score (BOS)

We also provide the BOS score as a reference point for individual test-set questions. To obtain the BOS for each test-set question and for the test-set as a whole, we use the CoNaLa BLEU score evaluation script [110], and calculate the BLEU score obtained by train-set entry when scored against every test-set answer, as described in section 3.2.2. Because the BOS computation can be long for corpora that have long English intents or code snippets and millions of training-set entries, we computed the BOS scores only for a random sample of total test-set questions on the following task-dataset combinations: C2E-Javadoc (610 randomly sampled questions among 90,908 test-set answers) and E2C-Javadoc (500 randomly sampled questions among 90,908 test-set answers), E2C-Pydoc (3882 questions out of 3885 test-set questions, due to a task hanging on the cluster toward the end). We calculated the BOS on the entire test-set for the following task-dataset combinations: E2C-SOa (500 questions), E2C-SO (500 questions), C2E-SOa (500 questions), C2E-SO (500 questions), C2E-Pydoc (3885 questions).

The time it takes to compute the BOS scores varies widely from corpus to corpus, depending

Task-dataset combination	Size of test-set	Size of BOS random-sample as percentage of the test-set
E2C-SOa	500 code-English pairs	100%
E2C-SO	500 code-English pairs	100%
E2C-Pydoc	3,885 code-English pair	99.92%
E2C-Javadoc	90,908 code-English pairs	0.55%
C2E-SOa	500 code-English pairs	100%
C2E-SO	500 code-English pairs	100%
C2E-Pydoc	3,885 code-English pairs	100%
C2E-Javadoc	90,908 code-English pairs	0.67% (in section 5.3), 11.44% (in addendum 5.4)

Table 9: Size of random sample used for BOS analysis, on each task-dataset combination

on the average length of the English intent and code snippets. We collected the following BOS computation times on a personal computer with a 2.5 GHz Intel Core i7 processor and 16 GBs of 1600 MHz DDR3 RAM. They are not exactly the same completion times on the cluster with high RAM, but do correlate approximately. On the small E2C-SOa task-dataset combination (500 test-set questions, 3K training-examples), we observe that it takes on average 1.63 sec per test-set questions, and approximately 13.58 min for the entire test-set. On the larger E2C-SO task-dataset combination (500 test-set questions, 600K training-examples), we observe that it takes on average 7 min 42 s s per test-set questions, and approximately 61 hr 49 min 39 sec for the entire test-set. On the C2E-Javadoc task-dataset combination, which is the largest dataset we use in our experiments (610 test-set questions examined, 2M training-examples), we observe that it takes on average 13 mins 43 sec per test-set question, and approximately 5 days 19 hr 25 min to obtain our 610 random sample of the test-set. These times are for the simultaneous calculation of the BLEU4, BLEU2 and BLEU1, and F1 scores. The BLEU2, BLEU1, and F1 scores were computed for all test-sets, but we did not end up using them since results are extremely similar to those of the BLEU4 score, and most previous studies use BLEU4 for evaluation. If exclusively the BLEU4 had been computed in our BOS calculations, the time to complete would be reduced. We did not parallelize the BOS computation, and did not implement 4-gram indexing of training-examples as described earlier in 3.2.2, but those two optimizations could be done in order to reduce computation time for computing the BOS.

Chapter 5

Results

To evaluate how well models perform on the code-to-English and English-to-code tasks:

- We first replicate existing models on their original task and dataset, to ensure they obtain a score similar to the original study, and are well-functioning.
- We then train the models on the six remaining task-datasets combinations, which have not been examined in prior work. As mentioned in Section 4, the Seq2SeqLSTM, AttendGRU, and CodeSearch models are all *language agnostic*: they can be run on the other task-dataset combinations, and reversed for back-translation when necessary.

We report two outcome measures: the BLEU4 score at the test-set-level, and the percentage of answers with a BLEU4 score of zero. Tables 10 and 11 show the results for each outcome. In Figures 2 to 8, we show the distribution of BLEU scores per question, for each approach on each dataset and include the BOS score (retrieval ceiling) as reference point. **Note**: The BLEU score generally ranges from zero to one hundred. It can theoretically allow scores above 100 BLEU, due to the particularities of the brevity penalty, but BLEU scores are informally considered to be a 0-100 range.

5.1 RQ1: (English-to-code) How well do the existing techniques perform for code generation?

To assess approaches on the code generation task, we start by replicating the Seq2SeqLSTM model from Yin et al. [111], published on the CoNaLa leaderboard for the CoNaLa dataset (E2C-SO). We then train and test the models on the task-dataset-model combinations not examined in prior work.

	E2C-SOa		E2C-SO [111]		E2C-Pydoc		E2C-Javadoc	
	BLEU4	% B4=0	BLEU4	% B4=0	BLEU4	% B4=0	BLEU4	% B4=0
AttendGRU	8.66	81.28%	4.63	89.60%	0.54	83.56%	2.56	89.58%
Seq2SeqLSTM	14.15	70.20%	15.75	67.40%	1.67	93.77%	13.99	71.94%
CodeSearch	16.47	66.80%	4.89	88.80%	21.13	64.57%.	15.11	74.75%
BOS	40.33	8.79%	58.21	1.6%	31.70	26.96%	25.27	7.41%

Table 10: Code generation (E2C) model results with BLEU4, and the percentage of zero BLEU4 answers.

	C2E-SOa		C2E-SO		C2E-Pydoc		C2E-Javadoc [54]	
	BLEU4	% B4=0	BLEU4	% B4=0	BLEU4	% B4=0	BLEU4	% B4=0
AttendGRU	0.54	99.80%	0.77	99.00%	13.26	85.92%	19.58	90.49%
Seq2SeqLSTM	8.19	83.60%	2.01	97.40%	16.79	88.31%	21.28	90.82%
CodeSearch	8.27	83.20%	0.79	99.40%	36.91	66.54%	17.12	89.18%
BOS	25.44	23.0%	4.41	28.79%	38.78	40.87%	31.10	31.80%

Table 11: Code summarization (E2C) model results with BLEU4, and the percentage of zero BLEU4 answers.

E2C-SO (Yin et al. [111] Seq2SeqLSTM replication). The Seq2SeqLSTM was originally trained by Yin et al. [111] for the English-to-code task, on the large version of the CoNaLa corpus (E2C-SO) and the authors reported a BLEU4 score of 14.26. In our replication we obtain a marginally higher score of 15.75. On E2C-SO, the BOS score at the test-set level is 58.21 BLEU4. The Seq2SeqLSTM remains the best performing model on this task-dataset combination. The AttendGRU and CodeSearch models have a much lower score of 4.63 and 4.90, respectively.

Most importantly, we observe that the models score zero on a large majority of the questions. The best-scoring Seq2SeqLSTM obtains a score of zero on 337 out of 500 test-set questions (67.40%). For the AttendGRU and CodeSearch models (second and third-scoring), the number of zeros is even higher: 448 (89.60%) and 444 (88.80%) respectively. For all three models, we find that the overall (corpus-level) BLEU score is misleading, as the median BLEU score per question is zero. We show the distribution of model scores for the E2C task, in Figures 2 to 4.

As was previously noted in Chapter 4, the CoNaLa dataset was filtered using the Seq2SeqLSTM, which can give an unfair advantage to the Seq2SeqLSTM. The Seq2SeqLSTM was first trained on the smaller, high-quality CoNaLa-annotated corpus (E2C-SOa, vetted for having English intents very

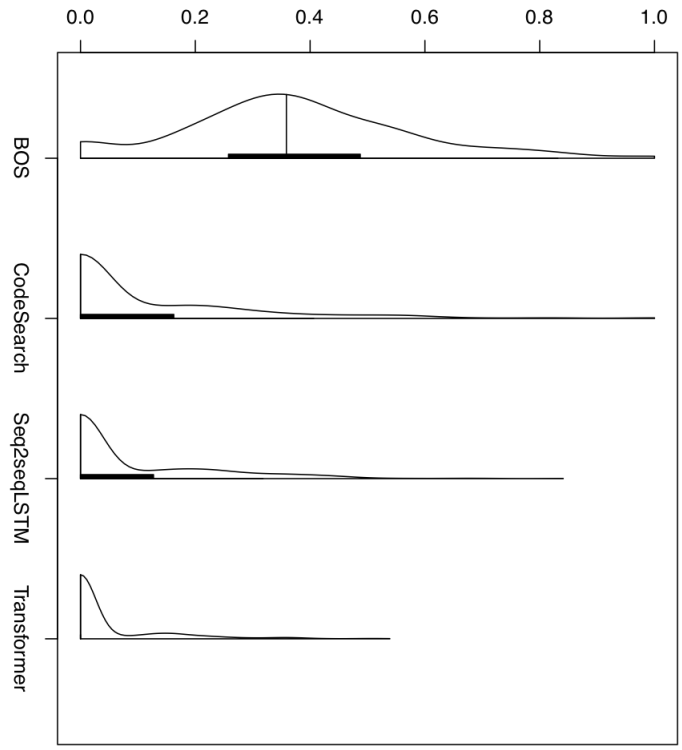

Figure 1: Distribution of per-question BLEU4 Scores for E2C-SOa

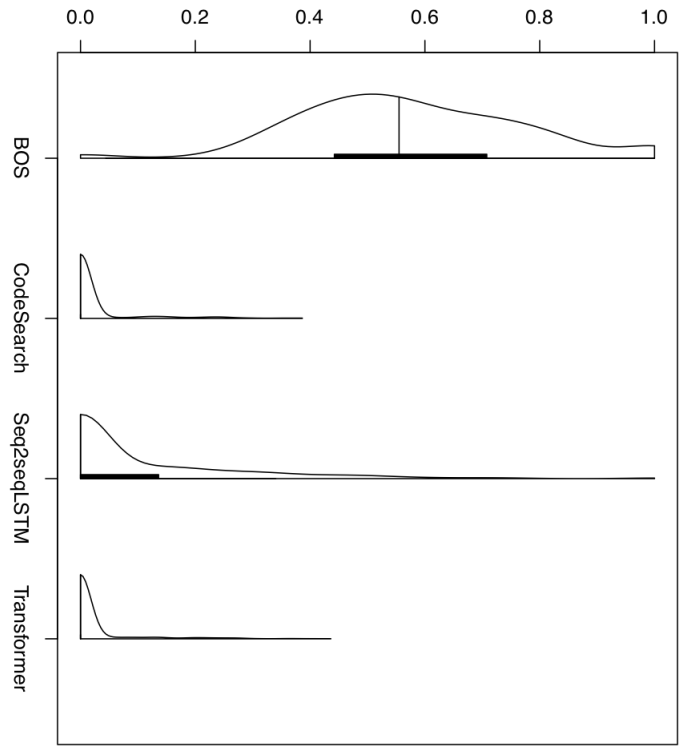

Figure 2: Distribution of per-question BLEU4 Scores for E2C-SO

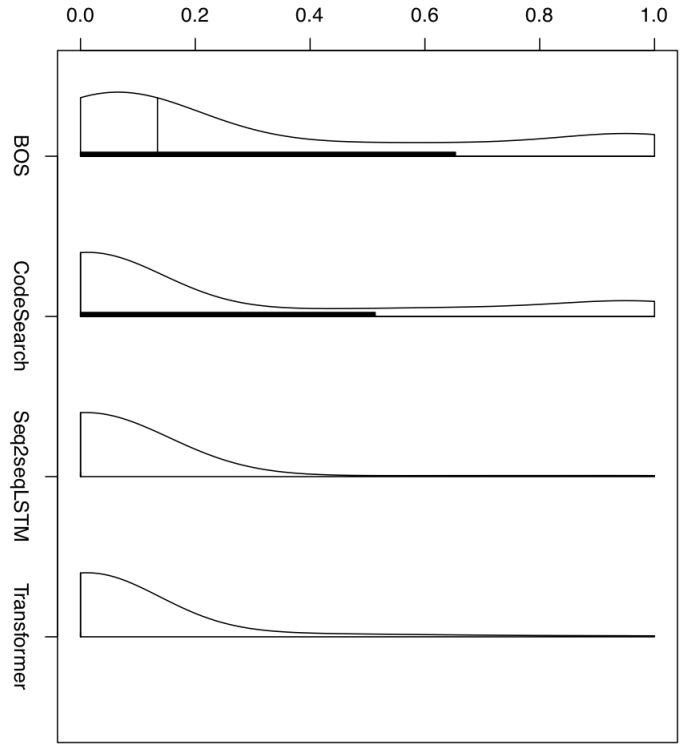

Figure 3: Distribution of per-question BLEU4 Scores for E2C-Pydoc

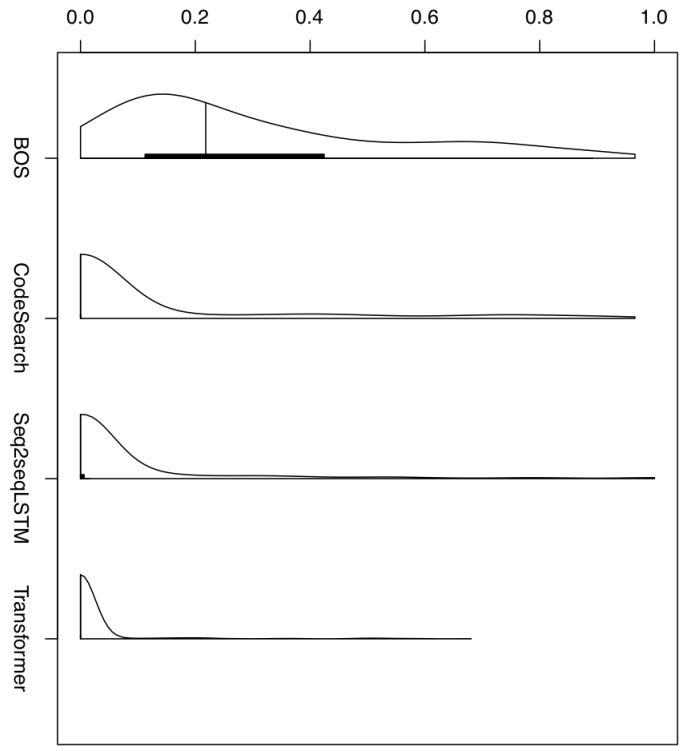

Figure 4: Distribution of per-question BLEU4 Scores for E2C-Javadoc

descriptive of their associated code). Then the Seq2SeqLSTM model was used to predict answers on new data, the mined data (not manually vetted for quality because too large), and the certainty of the Seq2SeqLSTM model's next-token predictions were used as part of a procedure to assess the quality of the new, StackOverflow mined data. This filtering of the E2C-SO could give an unfair advantage to the Seq2SeqLSTM on that dataset, and could explain why the Seq2SeqLSTM scores approximately 10 BLEU4 points above the two other models, which is the largest lead in score that the Seq2SeqLSTM has over other models in our experiments. The second largest lead in score that the Seq2SeqLSTM has over other models is on the C2E-Javadoc where it scores only 1.96 BLEU4 points above the next best-scoring model, the AttendGRU. This surprising result is an anomaly in our experiments, which seems to clearly correlate with the filtering procedure. The potential effect of dataset filtering was not mentioned in by the authors of the CoNaLa challenge [111]. Since the Seq2SeqLSTM does appear on the CoNaLa challenge leaderboard, we do include it in results.

Also, it is interesting to note that when we train the CodeSearch model on the smaller, manually annotated corpus (C2E-SOa-CodeSearch, but test it on the E2C-SO test, it surpasses the best-scoring Seq2SeqLSTM, with a score of 16.47 (the E2C-SOs and E2C-SO test-sets are identical, but the train-sets are different, E2C-SO being larger but also noisier). The AttendGRU also benefits from training on the cleaner E2C-SOa corpus even if it is smaller: it scores 8.66 instead of 4.63. The Seq2SeqLSTM is the only model to score higher on E2C-SO than on E2C-SOa (15.75 versus 14.15), which is also an indication that its use for filtering of the E2C-SO corpus could be problematic.

E2C-SOa (novel task-dataset combination). On E2C-SOa, the small, manually curated version of the CoNaLa corpus, the AttendGRU obtains 8.66 BLEU4, the Seq2SeqLSTM 14.15, and CodeSearch 16.47. The percentage of answers with a BLEU4 score of zero is 81.20%, 70.20% and 66.80% for the AttendGRU, Seq2SeqLSTM and CodeSearch respectively. The BOS at the test-set level is 40.33.

The test-set for this corpus is identical to that of E2C-SO dataset. Here it is interesting to note that the CodeSearch model is able to surpass the E2C-SO-Seq2SeqLSTM (from [111]), when it is trained on the smaller but cleaner E2C-SOa dataset (manually annotated subset of E2C-SO). It shows that the performance of the Seq2SeqLSTM on E2C-SO is still below what a search model can produce.

E2C-Pydoc (novel task-dataset combination). On E2C-Pydoc, the AttendGRU obtains 0.54 BLEU4, the Seq2SeqLSTM 4.21, and CodeSearch 21.13. The percentage of answers with a BLEU4 score of zero is 83.56%, 93.77% and 64.57% for the AttendGRU, Seq2SeqLSTM and CodeSearch respectively. The BOS score at the test-set level is 31.70 BLEU4.

E2C-Javadoc (novel task-dataset combination). On E2C-Javadoc, the AttendGRU obtains

2.56, Seq2SeqLSTM 13.99, and CodeSearch 15.11. The percentage of answers with a BLEU4 score of zero is 89.58%, 71.94% and 74.75% for the AttendGRU, Seq2SeqLSTM and CodeSearch respectively. The BOS score at the test-set level is 25.27 BLEU4.

CodeSearch outperforms the neural sequence-to-sequence models, Seq2SeqLSTM and AttendGRU, but only on novel task-dataset combinations. We observe that CodeSearch outperforms neural seq2seq models on all 3 novel task-dataset combinations. The Seq2SeqLSTM is able to come very close to it on both E2C-SOa and E2C-Javadoc, scoring just 1.32 and 1.12 BLEU4 points below, respectively.

Seq2SeqLSTM outperforms AttendGRU every time. We also observe that the Seq2SeqLSTM outperforms the AttendGRU on all C2E task-dataset combinations, by a very large marging of at least 5 BLEU4 points. The AttendGRU is observed to be hard to train on the small and mid-sized datasets used in our experiments and in previous studies (up to to 2M code-English pairs). The AttendGRU is a model that was explicitly designed to be computationally more efficient, with performance observed to be on par, or slightly lower than its LSTM counterparts. In our experiments, we observe that the AttendGRU performs significantly below the Seq2SeqLSTM.

We successfully replicate Yin et al. [111]. When we test all technique data-set combinations, we find that the CodeSearch model outperforms the AttendGRU and Seq2SeqLSTM on all but the original E2C dataset. The BLEU metric provides misleading results because, depending on the technique, between 64.57% and 93.77% of the answers have BLEU scores of zero.

5.2 RQ2: (Code-to-English) How well do the existing techniques perform for code summarization?

To assess approaches on the C2E task, we start by replicating the AttendGRU model on the Javadoc dataset from Leclair et al. [55]. We then train and test all models on the task-dataset-model combinations not examined in prior work.

C2E-Javadoc (LeClair et al. [54] AttendGRU replication). The study replicated is LeClair *et al.* [54]. The AttendGRU model was trained for code summarization on the English-Javadoc corpus [55] and the authors reported a BLEU4 score of 19.4. In our replication, we obtain a score of 19.58 BLEU4.

The Seq2SeqLSTM scores 21.28, 1.70 BLEU4 above the AttendGRU model. Seq2SeqLSTM also outperforms three recent studies using attention to function context [38], graph neural networks [53], and two Transformers [36, 38]. The CodeSearch model has a BLEU4 of 17.12.

For all three models, again in the C2E direction, we find that the BLEU metric that is usually reported is misleading, as the per-question median score is zero. The best-scoring Seq2SeqLSTM obtains a score of zero on 73,338 out of 90,908 test-set questions (80.73%). For the AttendGRU and CodeSearch models, the number of zeros is similar: 75,158 (82.68%) and 78,435 (86.28%) respectively. On the subset of test-set questions considered for comparison with the BOS score in Tables 10 and 11 (500 in total), the number of zeros is higher than on the overall (90,908 entry) test-set: 90.49%, 90.82%, and 89.18% for the AttendGRU, Seq2SeqLSTM, and CodeSearch respectively. We show the distribution of model scores for the C2E task, in Figures 6 and 8.

We also observe that several test-set questions contain the answer, that is, the function identifier contains almost all the words of the docstring in the same order. This dataset-artifact often leads to high scores for both Seq2SeqLSTM and AttendGRU on those types of test-set questions, as they learn to copy the first tokens of the "question" as "answer". This issue is discussed in more detail in Section 5.3. Examples of the dataset-artifact can be seen in Table 5.3.

C2E-SOa (novel task-dataset combination). On C2E-SOa, the AttendGRU obtains 0.54, Seq2SeqLSTM 8.19, and CodeSearch 8.27. The percentage of answers with a BLEU4 score of zero is 99.80%, 83.60% and 83.20% for the AttendGRU, Seq2SeqLSTM and CodeSearch respectively. The BOS score at the test-set level is 25.44 BLEU4.

Here, and as is the case on several of our dataset, Seq2SeqLSTM and CodeSearch are near in BLEU4 score, with CodeSearch slightly above Seq2SeqLSTM, and finally the AttendGRU much lower than both.

We observe that, on the two CoNaLa datasets, all model scores are significantly lower than in the E2C direction than in the C2E direction. This could be due to the great diversity of English descriptions for a code snippet, which means that when provided with a code snippet as input, English-to-code model can have difficulty deciding between several possible answers, when, in the reverse direction, the number of possible code snippets is more restrained.

C2E-SO (novel task-dataset combination). On C2E-SOa, the AttendGRU obtains 0.77, Seq2SeqLSTM 2.01, and CodeSearch 0.79. The percentage of answers with a BLEU4 score of zero is 99.00%, 97.40% and 99.40% for the AttendGRU, Seq2SeqLSTM and CodeSearch respectively. The BOS score at the test-set level is very low, at 4.41 BLEU4.

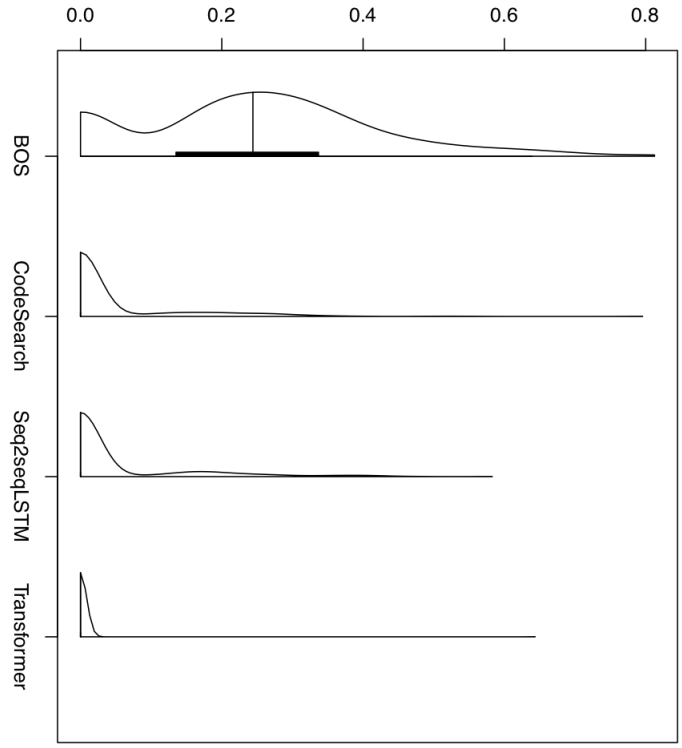

Figure 5: Distribution of per-question BLEU4 Scores for C2E-SOa

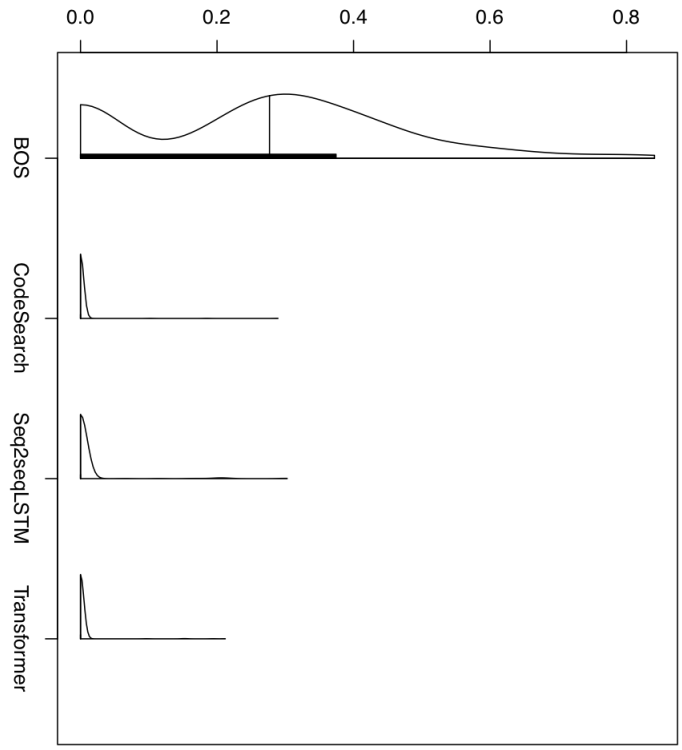

Figure 6: Distribution of per-question BLEU4 Scores for C2E-SO

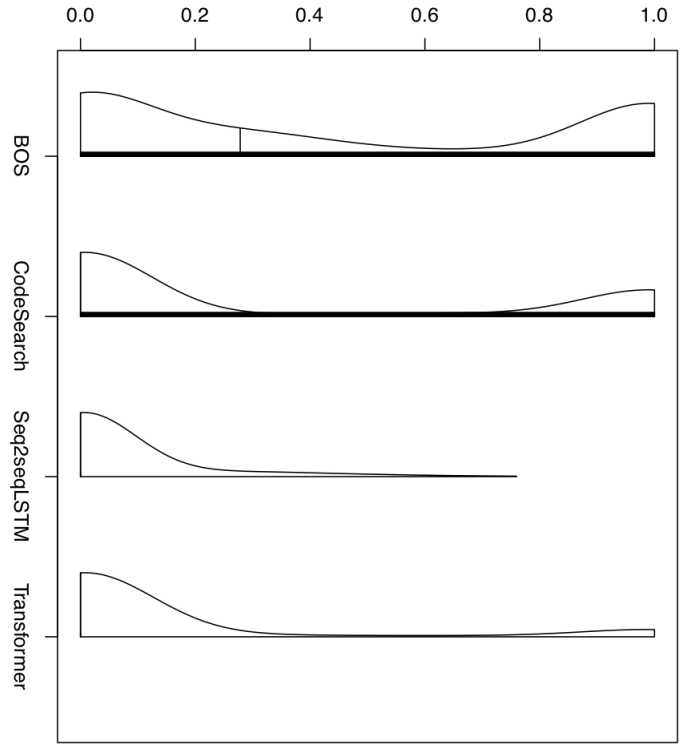

Figure 7: Distribution of per-question BLEU4 Scores for C2E-Pydoc

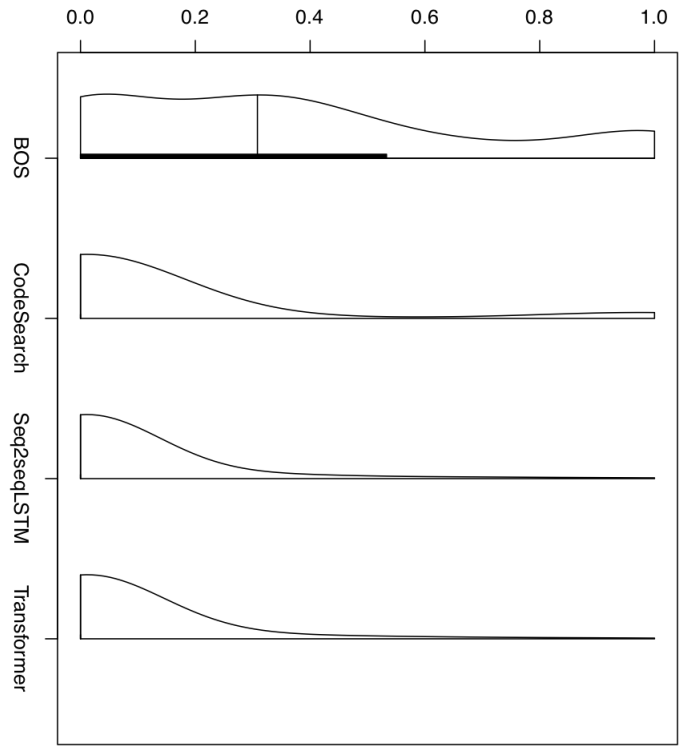

Figure 8: Distribution of per-question BLEU4 Scores for C2E-Javadoc

C2E-SO is the most difficult task-dataset combination in our experiments. All three models without exception score extremely low on it. This dataset is the only novel task-dataset combination on which CodeSearch is surpassed by a neural sequence-to-sequence model. We can observe that in addition to having one of the lowest BOS scores, C2E-SO presents at the same time a very low CodeSearch score, which could indicates that code snippets are not good predictors of their intents, i.e. that two very similar code snippets may have very, different, potentially irrelevant or unpredictable intents. Another possible cause for low scores is if test-set answers are much shorter than the ones found in the training-set on average. This could result in a heavy brevity penalty during the BLEU computation, as described in 3.2.1.

C2E-Pydoc (novel task-dataset combination). On C2E-Pydoc, the AttendGRU obtains 13.26 BLEU4, Seq2SeqLSTM 16.79, and CodeSearch 36.91. The percentage of answers with a BLEU4 of zero score is 85.92%, 88.31% and 66.554% for the AttendGRU, Seq2SeqLSTM and CodeSearch respectively. The BOS score is 38.78 BLEU4.

CodeSearch performs best on novel task-dataset combinations. On the novel task-dataset combinations, again for C2E, CodeSearch performs the best. It surpasses both the Seq2SeqLSTM and AttendGRU models on 2 out of 3 task-dataset combinations. CodeSearch is only surpassed on C2E-SO, by the Seq2SeqLSTM, and only by a very small margin of 1.05 BLEU point. This dataset is also exceptional in that all three models score extremely low on it, each having more than 97% answers with a BLEU4 score of zero.

Seq2SeqLSTM outperforms the AttendGRU every time. We also observe that the Seq2SeqLSTM outperforms the AttendGRU on all C2E task-dataset combinations, although this time it performs better, and comes within 2 BLEU4 points of the Seq2SeqLSTM on 2 out of 3 novel task-dataset combinations (and 3 out of the 4 C2E task-dataset combinations). The AttendGRU performs better on the E2C direction for the two SO/SOa datasets, while it performs better in the C2E direction for the Pydoc and Javadoc datasets. This trend is also observed with Seq2SeqLSTM scores, but is less pronounced.

Trends in datasets. We observe that, on the docstring-derived datasets, scores are lower in the E2C direction for the sequence-to-sequence models, Seq2SeqLSTM and AttendGRU. Functions are longer in the docstring-derived datasets than in the StackOverflow-derived CoNaLa corpus dataset. As we discuss in Section 6, this can make the task harder for sequence-to-sequence models. CodeSearch on the other hand is not affected by the length of the generation because it returns intact functions from the training-set. On the Stackoverflow-derived datasets however, this trend is reversed, with sequence-to-sequence models performing better in the C2E direction.

For the C2E task, we successfully replicate the LeClair et al.'s [54] AttendGRU on the original C2E-Javadoc dataset. However, we find that Seq2SeqLSTM outperforms AttendGRU on this original dataset. For novel datasets, Seq2SeqLSTM scores the highest on C2E-SO. On the two remaining datasets, C2E-SOa and C2E-Pydoc, CodeSearch scores the highest. For the C2E datasets, the proportion of answers with zero BLEU4 scores is even more problematic ranging from 64.57% and 99.80% depending on the model.

5.3 RQ3: (Generalization and Dataset-Artifacts) Can sequence-to-sequence models surpass the BOS ceiling for any particular test-set question? Under what circumstances?

BOS as a diagnostic tool. The initial intent in using the BLEU4 optimal search score, BOS, was to get an idea of how well models were performing in comparison with the available, relevant training data on a per-question basis. We wanted to understand why model scores were so low on some test-set questions, and on some datasets. Because we were able to re-train the Seq2SeqLSTM on the [54] dataset (C2E-Javadoc task-dataset combination) and obtain a state-of-the-art score for it, we suspected that the training-data was at least one of the reasons for the large fluctuation in scores across datasets and task-dataset combinations. To do this, we produced *BOS to model-score scatter-plots*, for all task-dataset combinations. These scatter-plots can be seen in Figures 11 to 18.

The BOS (retrieval ceiling) is observed to act as a ceiling also for the sequence-to-sequence models as well, with some rare exceptions. The first observation from the scatter-plots was that the BOS score, which is the theoretical ceiling for the CodeSearch model, is also usually not surpassed by the sequence-to-sequence models, on the large majority of test-set questions. This is true for the BLEU4 scores (4-grams) but also for BLEU2 and BLEU1 (single token matching). This is in itself an important finding, because it shows that sequence-to-sequence model cannot generalize often, if at all, on our datasets for the tasks of code generation and code summarization.

On some particular test-set questions however, the Seq2SeqLSTM and AttendGRU models can occasionally surpass the BOS. This could mean that they were able to generalize successfully on that particular test-set question. We wanted to examine such questions, to see what was actually happening, and see if the model successfully generalized. On the scatter-plots, data points represent

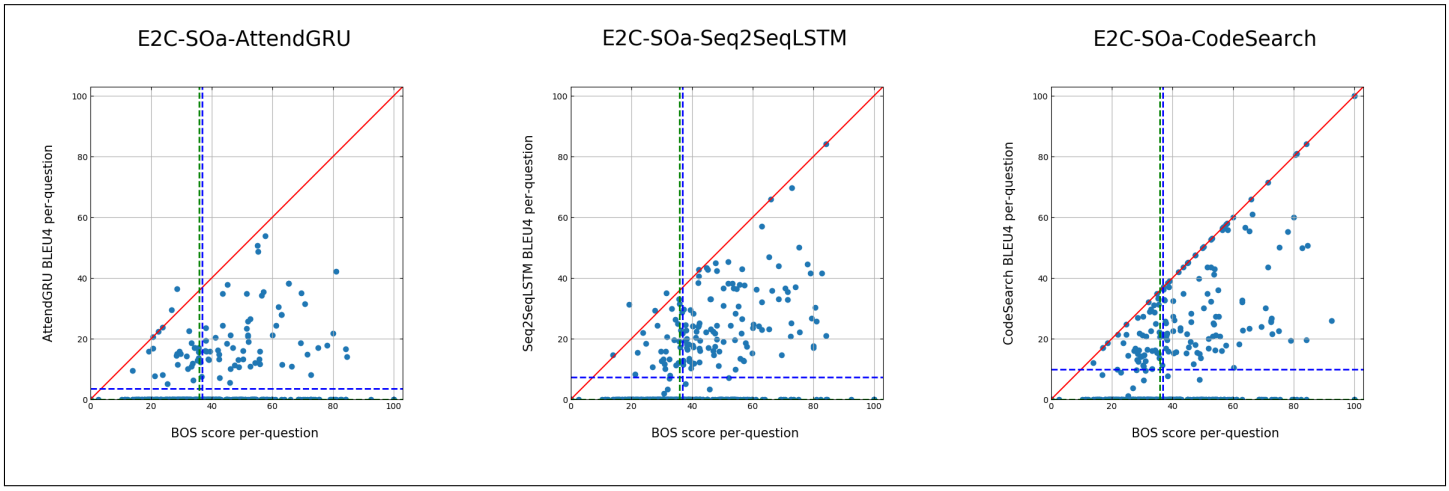

Figure 9: Scatter-plot of BOS scores versus model scores, for the small CoNaLa challenge annotated dataset from Yin et al. [111] (E2C-SOa). The BOS score (BLEU) is measured along the X axis, while the models' BLEU scores are measured along the Y axis. Each blue dot on the scatter-plots represents one test-set question answer by the model in question. The BOS score (retrieval ceiling) is represented as the red line. Any dot above this red line represents a model answer that obtained a BLEU score higher than that of any existing answer in the training-set. We see that the retrieval model, CodeSearch, is strictly bounded by the BOS ceiling, which is expected if the model and the BLEU evaluation are well-functioning. We can also observe that the two sequence-to-sequence models, AttendGRU and Seq2SeqLSTM can occasionally score above the BOS score, but that it is rare. In other words, the sequence-to-sequence model BLEU scores are also mostly bounded by the BOS score, in a way similar to a retrieval model.

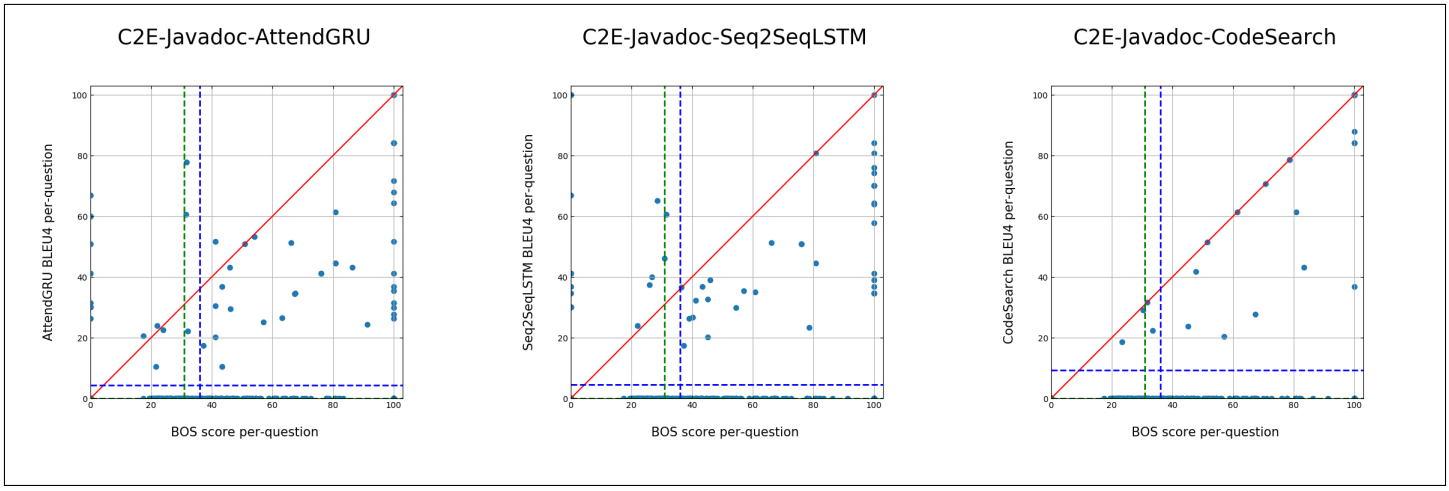

Figure 10: Scatter-plot of BOS scores versus model scores, for the large Javadoc dataset from Leclair et al. [54] (C2E-Javadoc). The BOS score (BLEU) is measured along the X axis, while the models' BLEU scores are measured along the Y axis. Each blue dot on the scatter-plots represents one test-set question answer by the model in question. The BOS score (retrieval ceiling) is represented as the red line. Any dot above this red line represents a model answer that obtained a BLEU score higher than any existing answer in the training-set. On this dataset, we see that, again, the retrieval model, CodeSearch, is strictly bounded by the BOS ceiling. However, this time, the sequence-to-sequence model scores are able to surpass the BOS score more frequently, and by a a large margin compared to what we saw on other datasets. We wanted to investigate to understand whether models were successfully generalizing, or whether something else was happening on this dataset. We manually examined every sequence-to-sequence model answer scoring above the BOS in our random sample, to discover that, in every case, these answers corresponded to a trivial test-set question (dataset-artifact) which gave away the tokens for the answer. The sequence-to-sequence models learned to copy the question as answer.

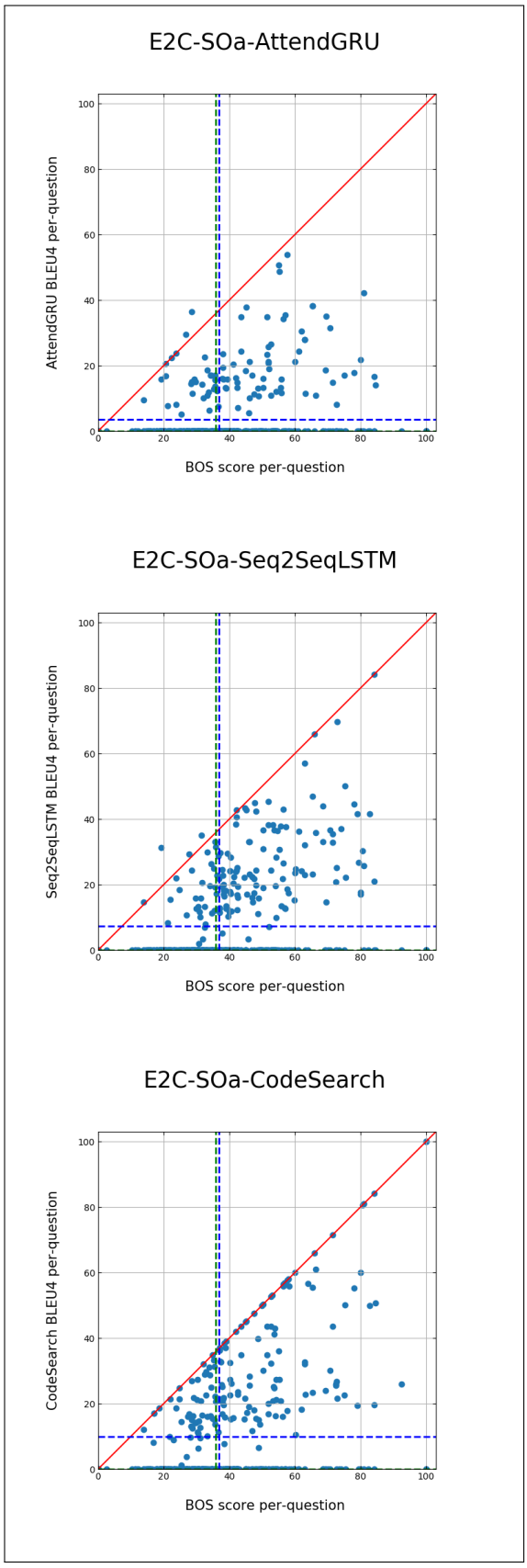

Figure 11: BOS versus Model score, per-question BLEU4, E2C-SOa.

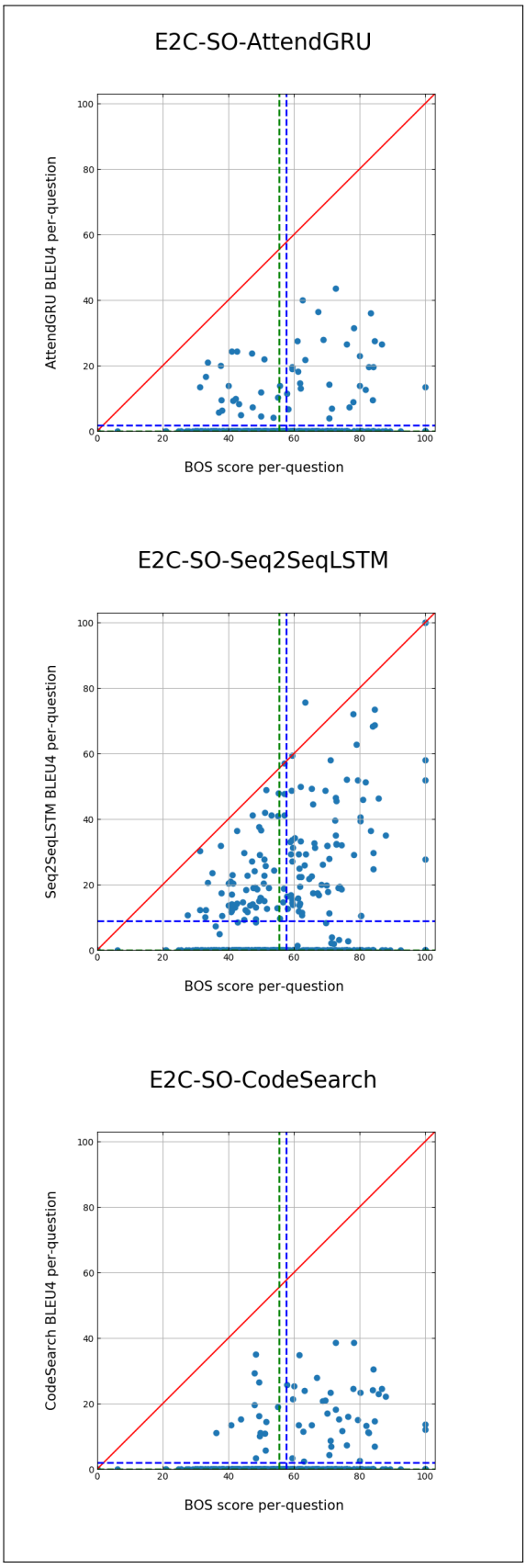

Figure 12: BOS versus Model score, per-question BLEU4, E2C-SO.

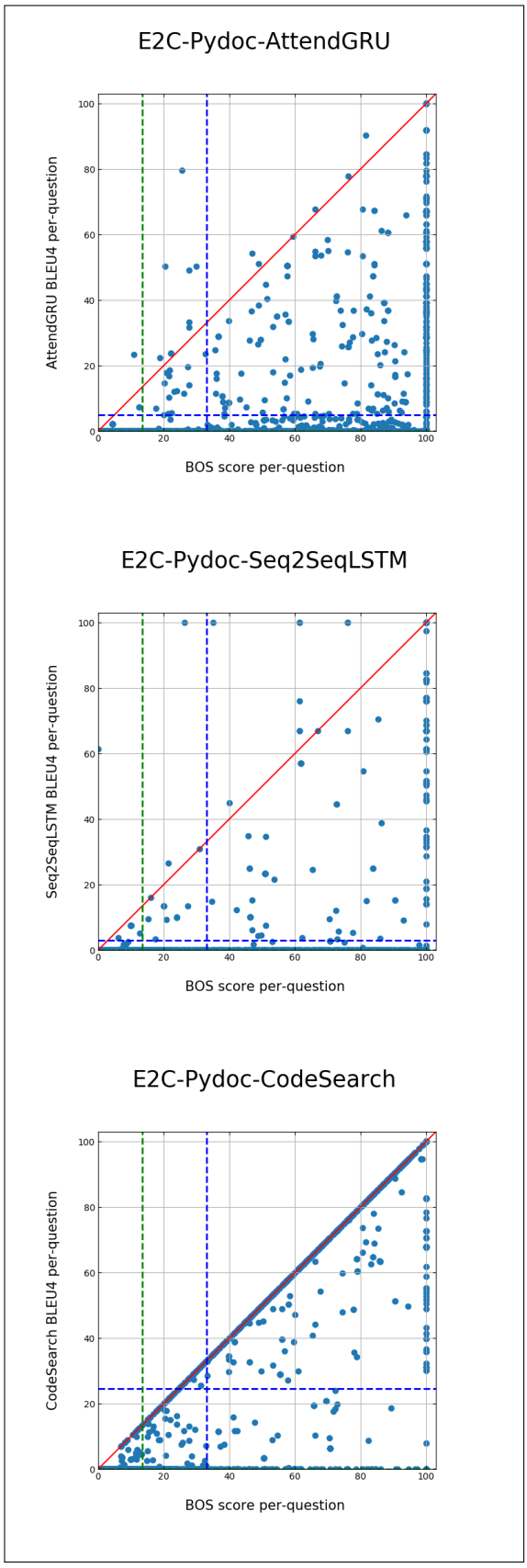

Figure 13: BOS versus Model score, per-question BLEU4, E2C-Pydoc.

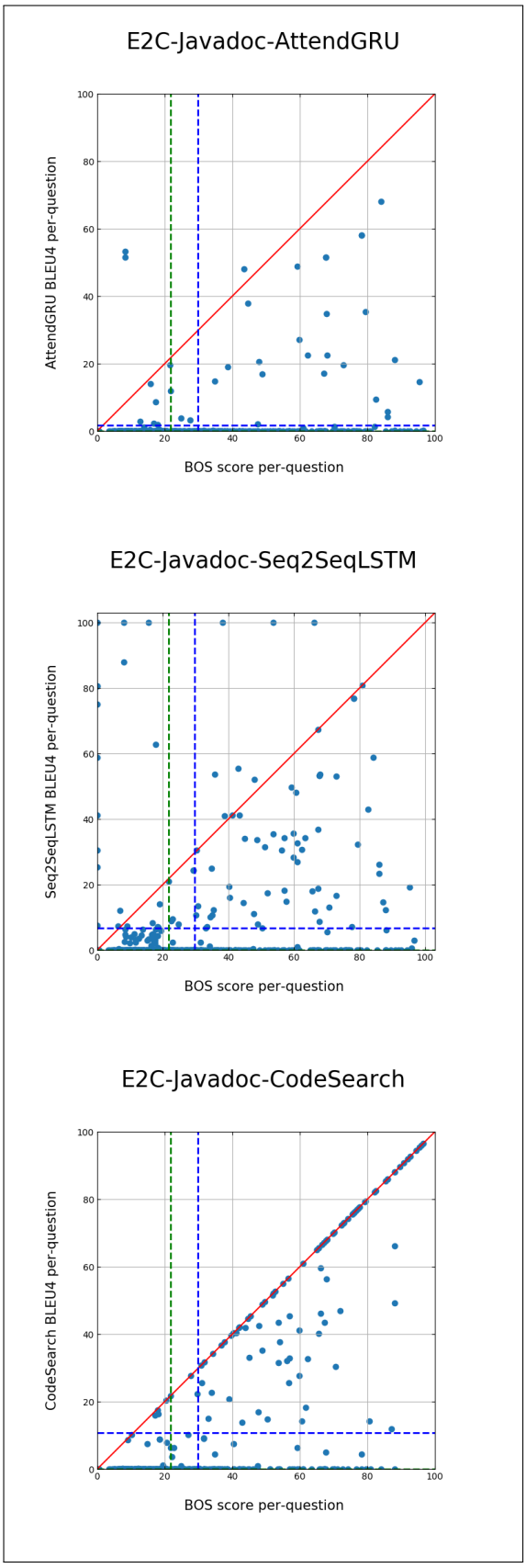

Figure 14: BOS versus Model score, per-question BLEU4, E2C-Javadoc.

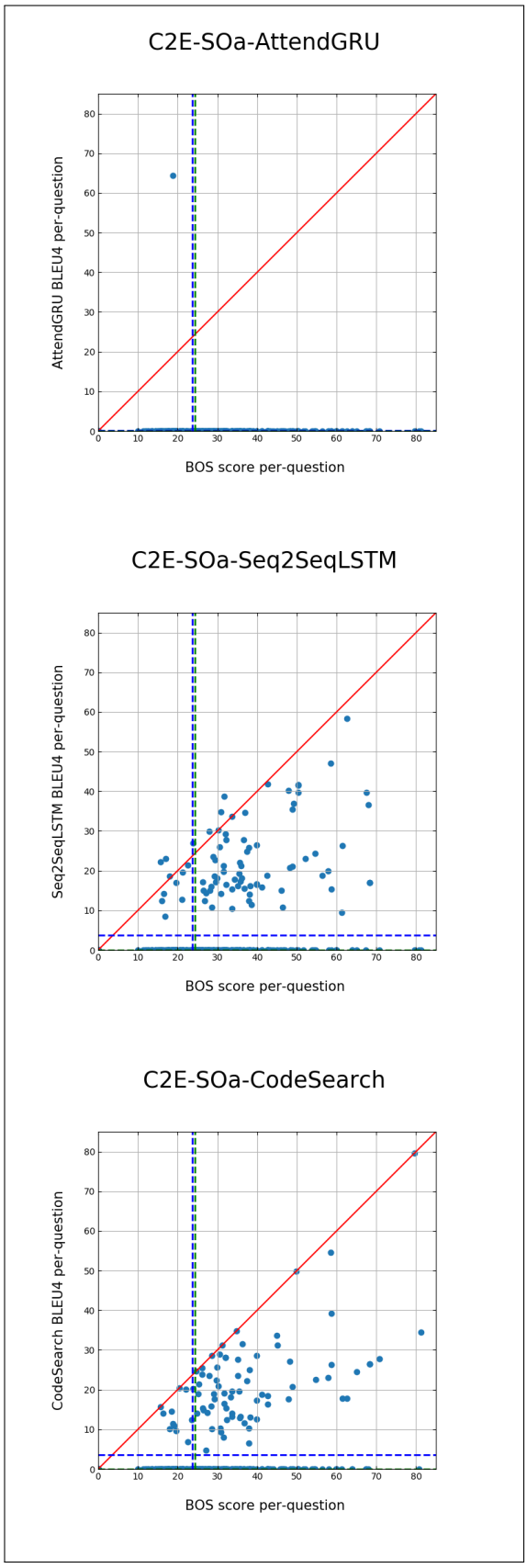

Figure 15: BOS versus Model score, per-question BLEU4, C2E-SOa.

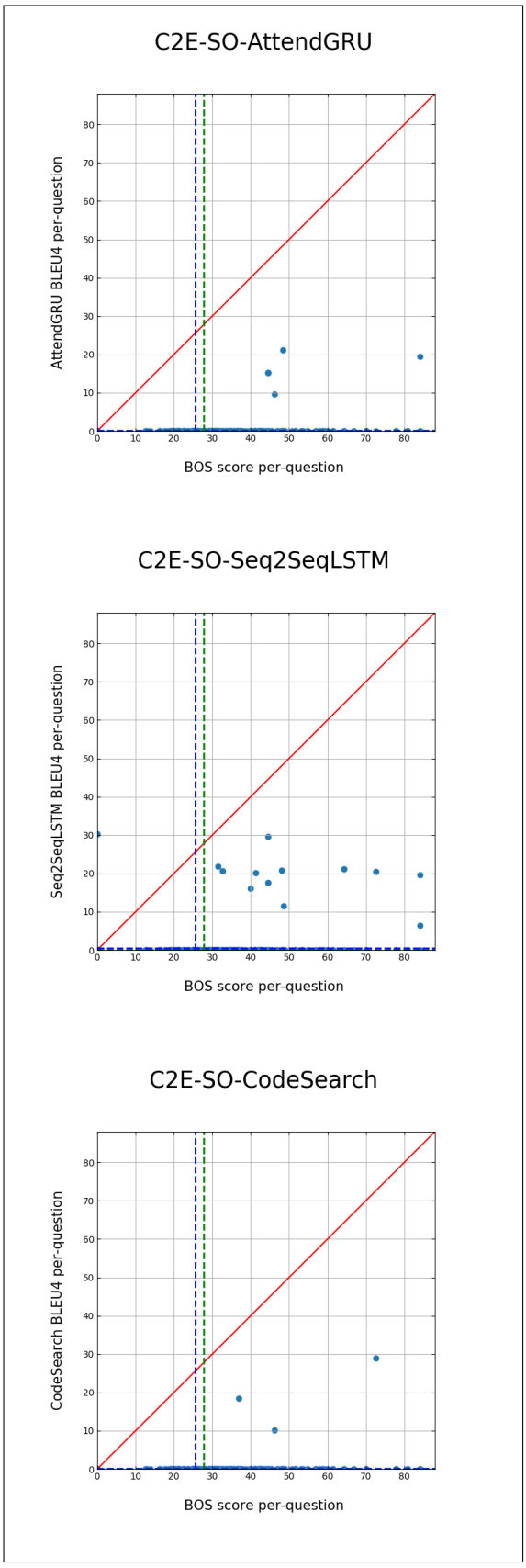

Figure 16: BOS versus Model score, per-question BLEU4, C2E-SO.

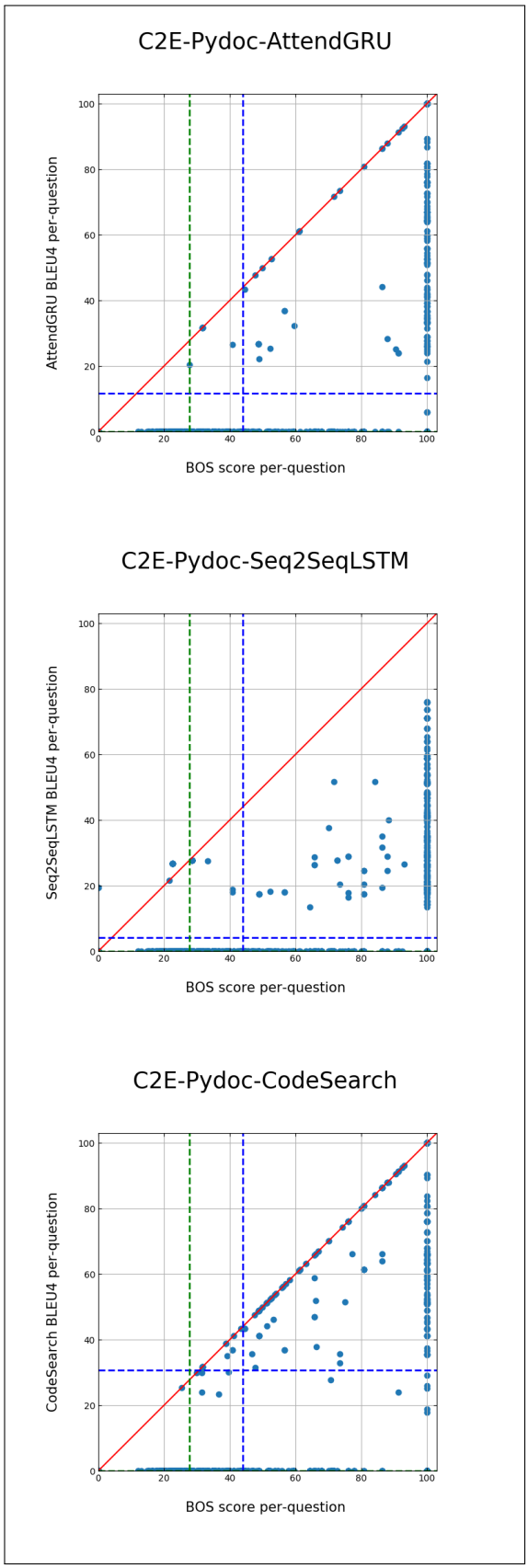

Figure 17: BOS versus Model score, per-question BLEU4, C2E-Pydoc.

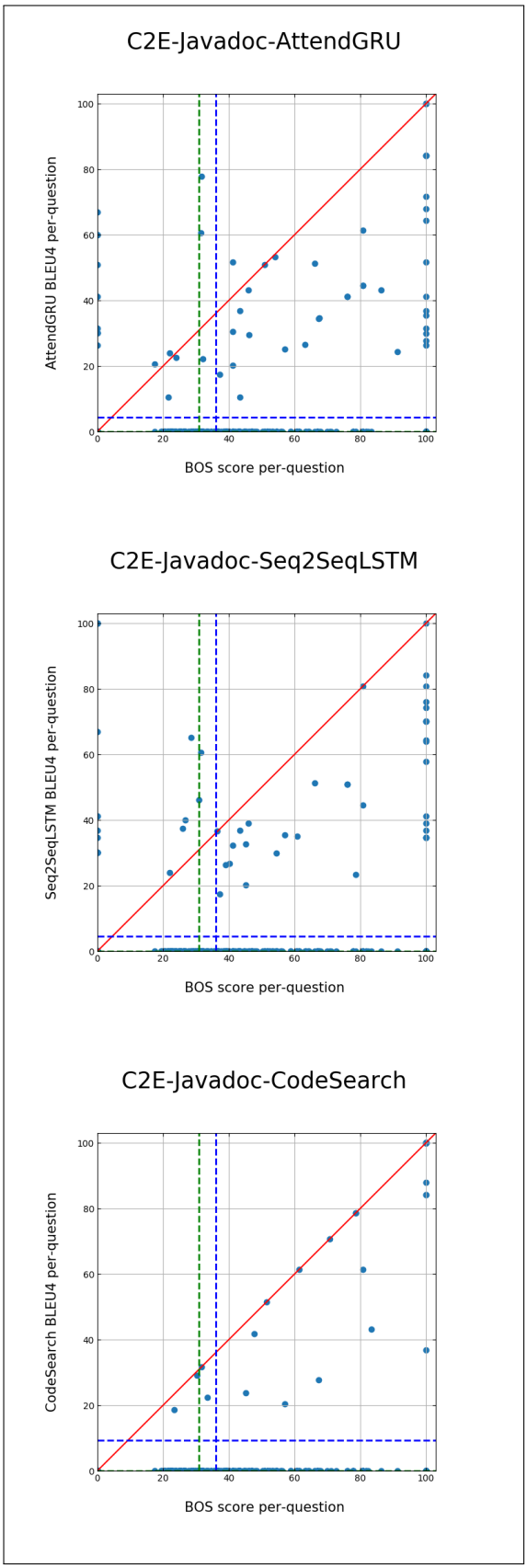

Figure 18: BOS versus Model score, per-question BLEU4, C2E-Javadoc.

model scores for each test-set question on the task-dataset combination. Any point that is above the <(0,0) (1,1)> diagonal (red) represents an answer that scored above the BOS. As we can see in Figures 11 to 18, model answers that are above the BOS for the test-set question do happen, but are rare.

Manually inspecting model answers that score above the BOS leads to discovery of dataset-artifacts. The datasets where the two neural sequence-to-sequence models score above the BOS most frequently are the docstring-derived datasets. For example, on C2E-Javadoc, the dataset from [54], both the AttendGRU and Seq2SeqLSTM are able to score above the BOS 15 and 22 times, respectively, out of the sampled 610 test-set questions.

These 12 instances constitute a very large portion (25.86%) of the 58 non-zero answers of the AttendGRU. In the case of the Seq2SeqLSTM, these 12 instances constitute 39.29% of its 56 non-zero answers. We inspect all outputs that scored above the BOS on this task-dataset combination. In every single one of these instances, we observe that the test-set question contains the answer, that is, the docstring is a duplicate or quasi-duplicate of the function name. In other words, the high score for that test-set question is caused by the question containing the answer; that is, the function identifier split by underscore by the dataset preprocessing from [54, 90] contains almost all the words of the docstring and in the same order or a significant subset of it. Unfortunately, in some code-docstring datasets, there may exists a large number of docstrings that are vertbatim copy of the function name, split by underscore, and since the preprocessing on these studies handle splitting by underscore, the model can simply copy the question as an answer. In Table 5.3 we show two examples of such artifacts on the C2E-Javadoc dataset from the original AttendGRU study.

On E2C-Javadoc, this time, the Seq2SeqLSTM scores 20 times above the BOS on the 610 question random sample, or 3.28% of the questions, which represent 11.68% of its non-zero answers. On E2C-Javadoc, the AttendGRU scores 3 times above the BOS, i.e. 0.49% of the questions, which account for 4.72% of its non-zero answers. It is interesting to note that on E2C-Javadoc, the task of copying tokens from the question as answer is slightly more difficult than in the C2E direction, as the sequence-to-sequence models need to *guess* the function modifier, the return type, as well as the body of the function. This explains why the sequence-to-sequence models do not seem to be able to take advantage of the dataset-artifacts as much in the E2C direction, and why we observe less above-BOS answers for the sequence-to-sequence models. In the C2E direction, sequence-to-sequence models need only *extract* information from the question in order to get a high BLEU score for the question, which is an easier task than to reconstruct the function. Some examples for the E2C direction (E2C-Javadoc-Seq2SeqLSTM) are shown in Table 5.3.

On E2C-Pydoc, the Seq2SeqLSTM scores above the BOS 9 times out of the 3582 test-set questions for which we computed the BOS score, or 0.25%, and this account for 3.58% of its non-zero answers.

Table 12: Examples of *C2E-Javadoc-AttendGRU* hypotheses scoring above the BOS.

	Test-set question	**test-set answer**	**AttendGRU hypothesis**
Example 1	private void write out lat long bounding box lat long bounding box llbb int count throws ioexception	write out a lat long bounding box to the stream	write out a lat long bounding box
Example 2	private void set current user comments text string comment changed true user comments text set text comment	sets the current user comments text area to comment	sets the current user comments text

Table 13: Examples of *E2C-Javadoc-Seq2SeqLSTM* hypotheses scoring above the BOS.

	Test-set question	**Test-set answer**	**Seq2SeqLSTM hypothesis**
Example 2	register a bug code	public void register bug code bug code bug code bug code map put bug code get abbrev bug code	public void register bug code bug code bug code bug code bug code
Example 1	gets controlling attribute name	public string get controlling attribute name return controlling attribute	public string get controlling attribute name return controlling attribute name

On E2C-Pydoc again, the AttendGRU scores 13 times out of the 3582 test-set questions for which we computed the BOS score, or 0.36%, which account for 2.20% of its non-zero answers. The reason why dataset-artifacts are so much rarer on the docstring-derived Pydoc dataset, is that in this corpus, the code snippets only contain the function body, and exclude the function name. Docstrings tend to contain tokens more similar to the function name than to the function body itself.

These dataset-artifacts inflate scores only for the neural sequence-to-sequence models, Seq2SeqLSTM and AttendGRU, and only on the docstring-derived dataset. This dataset-artifact, present mainly in the two docstring corpora, and at higher rates on the Javadoc dataset, lead to inflated scores for sequence-to-sequence models, as they learn to copy the first tokens of the "question" as "answer". CodeSearch on the other hand cannot take advantage of these artifacts, since it can only return an existing training-set instance, intact. Despite this handicap, CodeSearch scores higher than the sequence-to-sequence models on 3 out of 4 of the docstring-derived task-datasets combinations (E2C-Pydoc, E2C-Javadoc, C2E-Pydoc). As we can see, the datasets which produced the most artifact-related above-BOS model answers are, in order:

1. **C2E-Javadoc**, which is one of the only 2 task-dataset combinations in our 8 experiments on which a sequence-to-sequence model could outperform CodeSearch; here the Seq2SeqLSTM and AttendGRU obtain "for free" *at least* 39.29% and 25.86% of their non-zero scoring answers from dataset-artifacts.

2. **E2C-Javadoc**; here the Seq2SeqLSTM and AttendGRU obtain *at least* 11.68% and 4.72% of their non-zero scoring answers from dataset-artifacts.

3. **C2E-Pydoc**; here the Seq2SeqLSTM and AttendGRU obtain *at least* 3.58% and 2.20% of their non-zero scoring answers from dataset-artifacts.

Note that these percentages of non-zero answers for which a dataset-artifact was identified is only a lower bound, since we only inspected answers that scored above the BOS. It is possible that some answers that scored below the BOS still benefited in BLEU score from the dataset-artifacts by copying parts of the question in the answer. By looking at these numbers, it becomes more apparent why the C2E-Javadoc task-dataset from the original AttendGRU study is the only task-dataset on which neural sequence-to-sequence models were able to outperform CodeSearch: *it is the dataset on which sequence-to-sequence models benefit the most from artifacts*, with more than a third of their answers contained in the question, in our random sample. We conclude that the relatively low BLEU scores for sequence-to-sequence models on novel task-dataset combinations are *not* an anomaly, and that, contrarily to what it appears when only looking at the task-dataset combinations from previous works, CodeSearch can be expected to surpass several variants of neural sequence-to-sequence models on the code generation and summarization tasks, as long as the datasets are free, or sufficiently free

of dataset-artifacts (such as test-set questions containing all, or a subset of the answer tokens).

Cases where a model scores above the BOS without the presence of a dataset-artifact in the question are extremely rare. For example, on the E2C-SOa task-dataset combination (CoNaLa challenge), the Seq2SeqLSTM surpasses the BOS on 1/500 test-set questions, the AttendGRU 0/500. These numbers are similar on other E2C-SO and C2E-SO/SOa: above-BOS answers constitute less than 0.012% of model answers if we exclude cases of dataset-artifacts. We conclude that the remaining cases where sequence-to-sequence models score above the BOS are only due to chance, and the stochasticity of token-to-token predictions. In conclusion, we find that the sequence-to-sequence models cannot outperform an optimal search.

BOS helps to show which datasets could have good results. C2E-SO has the lowest BOS score at the test-set level, of all task-datasets combinations examined, at 4.41 BLEU4. It yields extremely low scores from *all models*, below 2.01 BLEU4, with more than 97% of zero-scoring answers. Since the same models perform otherwise extremely well on other datasets, obtaining scores higher than several previous studies [54, 53, 38, 36, 111], we conjecture that the dataset is particularly difficult to learn and predict, as the low BOS score of 4.41 BLEU4 indicates.

On some task-dataset combinations, we observe that some models score low even when the BOS score is high. For example, the E2C-Pydoc but has a BOS score of 31.70 BLEU4, but yields extremely low scores from both the AttendGRU and the Seq2SeqLSTM, below 2 BLEU4, with 83.56% and 93.77% of zero-scoring answers, respectively. It is interesting to note that with a similar number of zero-scoring answers, thus a similar number of answers scoring above zero, the same models are able to obtain very high scores of 19.58 BLEU4 (AttendGRU) and 21.28 (Seq2SeqLSTM) on the C2E-Javadoc task-dataset combination which contained a large number of dataset-artifacts. This could be explained by the fact that that a smaller number of very high scoring answers (scoring above the BOS due to the dataset-artifacts), are able to skew the overall BLEU4 score on the Leclair et al. dataset [55] (C2E-Javadoc). The brevity penalty (a step in the BLEU score computation procedure as described in 3.2.1) can also bring down the scores if model answers are too short on average over the whole test-set. This can be another reason for lower model score, compared to what the BOS score could allow in theory.

The Seq2SeqLSTM and AttendGRU do not surpass the BLEU4 optimal search score, i.e. the retrieval ceiling score, except when the answer is contained in the question (presence of dataset-artifact, or "trivial question"). The BOS score should be used in future work as a reference point to evaluate sequence-to-sequence model performance.

5.4 Threats To Validity

Hyperparameter tuning could be explored further. A threat to validity of our conclusions pertains to the sequence-to-sequence models' hyperparameters, which, if explored more thoroughly, might increase their scores further. As described in Chapter 4, we did not perform grid search, random search, or any hyperparameter tuning for any of the models used in this study. We did adjust the batch size when necessary to run the models with longer sentences in the available GPU memory, but made no other modification to the original models to get all BLEU scores recorded. Therefore, there might exist sets of hyperparameters that could lead to increased scores, on any of those datasets. There are two problematic scores in particular that stand out as outliers: E2C-Javadoc-AttendGRU and C2E-SOa-AttendGRU. It would be interesting to explore the hyperparameter space for the AttendGRU on those task-dataset combinations, since the Seq2SeqLSTM is able to score substantially higher on them, and indication that there is space for improvement for the AttendGRU on those task-dataset combinations. However, we suspect that variation in dataset difficulty is at least one important factor in the trends we observe. We believe it is likely that even if we explored the hyperparameter space and increased model scores to some extend, our conclusions would remain the same. In future work, to explore hyperparameter space further, a good approach would be to use Bayesian or evolutionary hyperparameter search [27, 114], especially on the batch-size and learning rate parameters which have been shown to affect scores significantly in some recent informal hyperparameter optimization experiments on neural language models [57].

Random sample of test-set questions might not be representative. It is possible that our random sampling of the test-set for the BOS analysis does not represent the true distribution. For 3 datasets out of 8, we computed the BOS only for a random sample subset of the test-set questions. On these 3 datasets (C2E-Javadoc, E2C-Javadoc, E2C-Pydoc), it is possible that this random sample does not represent the true distribution on the overall test-set. The Javadoc-derived corpus has more than 90K test-set entries, and we computed the BOS score for only 610 of them, which represents a sample of 0.67% of the test-set. Among those 610, we examined all the model's generations that scored above the BOS, (as reported in Chapter 5). There is a possibility that this

sample of 610 questions is not representative, because it is relatively small.

Both these threats are considered to be low, and we believe it is unlikely that our conclusions (on CodeSearch generally outperforming neural sequence-to-sequence models, and neural sequence-to-sequence model being incapable of generalizing above the BOS) are reversed by a change in hyperparameters, or by taking a new random sample from the test-set for the BOS evaluation.

Addendum, confirming results with a larger random sample on C2E-Javadoc. Because the 610 sample of test-set questions represents only 0.67% of the very large test-set from Leclair et al. (90,908 code-English pairs), we wanted to confirm our results further on a larger sample of the test-set. We thus re-computed the BOS scores over a larger sample of 10,399 test-set questions for that dataset.

On this larger random sample of the test-set (sampled 10,399 test-set questions), we recorded 532 model answers scoring above the BOS for the AttendGRU model. In a manual inspection, we observed again that every single model answer scoring above the BOS corresponds to a test-set question which contained the answer, i.e. whose first tokens simply had to be copied over in order to get a high BLEU score for the question. These questions represented 34.34% of the AttendGRU's 1549 non-zero answers on this larger sample. This result is higher than the ratio of 25.86% that we reported in section 5.3 and concurs with the initial observations made on the smaller sample of 610 test-set questions: model scores are greatly inflated on the C2E-Javadoc task-dataset combination, due to the dataset-artifacts.

In the case of the Seq2SeqLSTM, again, all of its 659 answers scoring above the BOS were observed to correspond to test-set questions which contained the answer tokens, and they represented 39.16% of the Seq2SeqLSTM's 1683 non-zero scoring answers on the 10,399 larger sample of the 90,908 Leclair et al. test-set. This percentage on the larger sample of the test-set is similar to the 39.29% result that we initially reported in section 5.3 for the Seq2SeqLSTM, confirming our previous results.

Chapter 6

Discussion

In this Chapter, we discuss the main takeaways from our results, their causes, or potential causes, and formulate hypotheses to be tested in future work. In particular, we discuss the five following topics:

1. Sequence-to-sequence models performed better on originally reported datasets than on novel task and datasets. Why?
2. Distribution of Individual BLEU Scores and Zero BLEU Scores
3. BOS, Dataset Size, and Resource Requirements
4. CodeSearch Outperforms Neural Seq2Seq Models
5. Seq2SeqLSTM Outperforms the AttendGRU

6.1 Sequence-to-sequence models performed better on originally reported datasets than on novel task and datasets. Why?

One important finding of the present study is that, although we could obtain high scores for the neural sequence-to-sequence models on the original study datasets (the Seq2SeqLSTM even setting a new high score on the AttendGRU's original study dataset), the same neural models did not perform as well on the novel task-dataset combinations, both in absolute BLEU scores and in relation to CodeSearch and the BOS. In Chapter 5, we identified reasons why some models are favoured on some datasets, which we further discuss in this section.

Dataset filtering with Seq2SeqLSTM on E2C-SO. We suspect that the use of the Seq2SeqLSTM model to filter the CoNaLa dataset (SO dataset) gives this model an unfair advantage on that dataset (E2C-SO). Using a model to filter a dataset in the way described in [111] could lead to favouring question-answer pairs for which the Seq2SeqLSTM sees a strong link (strong prediction certainty from question to answer), even if those links are the result of model priors or bias. This pre-filtering of the CoNaLa dataset can help explain the surprisingly large gap (of approximately 10 BLEU4 points) between the Seq2SeqLSTM and the two other models on this dataset. This task-dataset combination *is the only one on which the Seq2SeqLSTM leads and surpasses the two other models by such a large margin.* The next largest gap between the Seq2SeqLSTM and the second best-performing model is approximately 2 BLEU4 points on the C2E-Javadoc task-dataset combination. To provide additional evidence on the effect of model filtering of the corpus, it would be interesting to test whether filtering the CoNaLa corpus with another model also boosts its score on the leaderboard.

Answer contained in the question (Seq2SeqLSTM and AttendGRU on C2E-Javadoc). Dataset-artifacts, as previously discussed in 2.4, are spurious correlations between question and answer, which allow machine learning models to answer correctly through shortcuts [65]. They create "trivial" test-set questions that are easily answered by models, and can substantially inflate their score if they are frequent in the data. On the original AttendGRU study dataset [54], as we showed in Chapter 5, every neural model answers that scored above the BOS in the random sample can be attributed to a dataset-artifact: the first tokens of the test-set question (code snippet) contain the answer (docstring, reference translation). The dataset-artifact is inherent to the Github-scraped English-docstring datasets, and the only way to avoid it would be to filter the dataset-artifact during preprocessing by matching tokens in question and answer in the pair. After manual inspection described in Chapter 5, we observe that these dataset-artifacts affected more than 25% and 39% of the non-zero answers of the AttendGRU and Seq2SeqLSTM models respectively in our random sample. This means that, if the random sample is representative, the artifacts account for a *very large portion* of the BLEU points earned by the neural sequence-to-sequence models on this dataset. As mentioned in 5.4, we observed very similar results on a larger random sample of 10K model answers (11% of the Leclair et al. C2E-Javadoc test-set), to further confirm our conclusions. We observed this time 39% and 34% of non-zero answers scoring above the BOS score, for the Seq2SeqLSTM and AttendGRU, respectively, and observed again through manual inspection that every single one of those answers corresponded to the dataset-artifact (the first tokens from the question contained a significant number of tokens for the answer, which were simply copied by the models). As we already discussed in Section 5.3, the simpler CodeSearch model is not affected by this dataset-artifact, as it can only return answers from the training set, and thus cannot learn to copy the question as answer, word-by-word.

Evaluation on a single dataset. Our study highlights the need to evaluate models on a wider range of datasets, and the need to report both high and low results. When evaluating a model, researchers often try different datasets before deciding to use a dataset on which their model performs relatively well and obtains a score worth publishing. Moreover, it is possible that papers that do not use the datasets most favourable for a particular model are less likely to be published, either due to the review process or self-selection from the authors. But such scores might not be representative. McCoy et al. [65] observe discrepancies in model score on different datasets, due to the reliance of sequence-to-sequence models on spurious patterns and dataset-artifacts on certain datasets to obtain high scores. Feng et al. [28] also note the problem of dataset-artifacts and highlight the need to test on several datasets, in the domain of language understanding. Even when there are no identified dataset-artifacts, different datasets and their test-train splits could still vary in their degree of novelty, and difficulty, and ranking models on only one dataset might not be statistically significant and representative.

The need to evaluate competing models on several datasets to obtain a more representative sample is rarely mentioned in the literature. We found one important mention for this idea in a completely different application domains of machine learning: Coleman et al. [22] observe that rankings of reinforcement learning models that are done on only one environment often are not ultimately representative of model rankings on a wider ensemble of testing environments. This is for them a motivation for automatically generating a large number of new testing environments for reinforcement learning models. In the NLP domain, an initiative like Robustness Gym [32] (developed by Salesforce Research and Standford Hazy Research), can help speed up the evaluation of models on more diverse tasks and datasets, and could make such evaluations more complete, and robust, on a variety of dimensions.

6.2 Distribution of Individual BLEU Scores and Zero BLEU Scores

Another important finding of this study is that the majority of model answers have a BLEU score of zero, even for the current state-of-the-art models. Even on the two original study datasets (E2C-SO, C2E-Javadoc), the number of zeros is extremely high. On the original CoNaLa dataset (E2C-SO), the best-scoring Seq2SeqLSTM has 67.40% of answers with a zero BLEU score. On the original Javadoc dataset [54], the AttendGRU from the Leclair et al. obtains 90.49% of answers with a zero BLEU score. This means that on most datasets, a small minority of model answers on the test-set account for all the BLEU points gained by the model. As we showed in section 5.3, this situation can result in a disproportionate influence of dataset-artifacts on the BLEU score, even if they represent a

minority of examples in the dataset.

The BLEU evaluation procedure can lead to a large number of zero-scoring answers even for a well-performing model if the programming language allows for multiple correct solutions to the same software engineering task. Other evaluation metrics such as precision and recall, the F1 score or METEOR would also suffer from this problem, when alternate answers are not available in the test-set to cover the different acceptable solutions. This, in our opinion, highlights the need to add alternate answers to a same test-set questions, especially for tasks such as English-to-code and code-to-English, where the number of possible, valid mappings between intents and code could be greater and more diverse than in natural language.

6.3 BOS, Dataset Size, and Resource Requirements

Another important finding from this study is that, on our particular tasks and datasets, neural sequence-to-sequence models almost never surpass the BOS score, except due to the dataset-artifacts noted. We observe that the neural sequence-to-sequence models are incapable of scoring higher than answers already contained in the training set.

Do we simply need more data and bigger models? Would neural sequence-to-sequence models eventually be able to generalize above the BOS ceiling if we increased the number of trainable parameters of the models, or if we increased the dataset size? This might be possible, and would be an interesting question to investigate in future work. Strictly looking at the results that we collected, however, nothing allows us to conclude that increasing the dataset size will allow the models to surpass the BOS ceiling. In our experiments, we observe that the BOS score ceiling is still in effect, as the size of the dataset increases, from 3K to 2M pairs. The lack of answers above the BOS appears more like a constant in our experiments, and neural sequence-to-sequence models consistently behave like retrieval models in terms of BLEU score: they are capped by the same BOS ceiling score as the retrieval model. One possibility with the very large sequence-to-sequence models trained on big text corpora, such as BERT and GPT-3, is that as the training-set size increases, the BOS score increases along with it for most test-sets that remain small, and constant in size. The BOS procedure really measures the novelty, and difficulty of the test-train split. A large, high-quality dataset could still have a very hard test-train split and thus a very low BOS (e.g. if duplicates across test and train sets are avoided, which is frequent in published datasets). Vice-versa, a small, low-quality dataset could have a very easy test-set, and a high BOS.

This question of whether a sequence-to-sequence model's occasionally impressive text-generations are already present in the training-set, is briefly addressed in the study on GPT-3 [13], but not examined in detail due to "a technical difficulty" encountered by the researchers ("a bug in the filtering"

of the test-set). In other previous studies, we do not observe a clear capacity for generalization in sequence-to-sequence models. Jastrzebski et al. [49] developed a novelty metric for test-set questions (based on word embeddings), and observe that a neural model is unable to answer test-set questions when they are too novel. Studies using the SCAN dataset [51, 89] also observe limited generalization capacity from neural sequence-to-sequence models, except when using a novel "compositional" architecture (which has not been widely adopted yet), and only to a limited extend. Nogueira et al. [77] observe the limited ability of neural sequence-to-sequence models to generalize to do arithmetic on numbers of length longer than those contained in the training-set.

The conclusion from these observation, is that in order to increase model performance for practical applications, increasing the BOS score is, in most cases, a pre-requisite. For end-users, whether the model generalized or memorized the answer, the result is the same. Low-scoring queries could be targeted for additional data collection, or data augmentation, in order to increase model scores and user experience. Interestingly, if some particular neural sequence-to-sequence architectures do not in practice score above the BOS for a task, then they should be used as, in essence, compressed databases with fast query-times. Their downside is that they provide no warning when the query is too novel, or "out-of-distribution", and are likely to produce a non-sensical output. Another downside is that they tend to favour high frequency tokens and sequences in their generations, and might answer incorrectly questions that require the use of rare tokens, as noted by Gu et al. [7].

Comparison of the BOS analysis to the SCAN generalization dataset. To measure the degree of generalization of a model, some recent studies have used the SCAN dataset [51], as well as other similar datasets for generalization. SCAN and other similar datasets contain specifically tailored generalization questions that are estimated to be "within reach", possible and reasonable to answer for a machine learning model, with different levels of difficulty, when given a particular training-set. For example, an answer for a given SCAN test-set question could contain a token that is completely absent from the training-set, while at the same time the training-set contains plenty of helpful examples of similar cases, composed of other tokens, for models to learn and generalize from. We believe that the BOS, although not *as precise* an assessment of the generalization capacity as the short tailored questions from SCAN, can be a very good complement to such a study. In particular, the BOS can help examine the generalization capacity of a model on any given dataset, in a "real setting", i.e. on actual questions on which the model will be used, and with the actual training-data available. The BOS analysis as done in Section 5.3, uses the *existing* diversity of test-set questions to assess capacity of the model to generalize, by sorting them by degree of novelty / difficulty and correlating model scores with BOS scores, on each of these more, or less, novel questions. Test-set questions with a non-perfect but high BOS score *could* in several cases be good candidate questions for some of the easier generalization tests as done with SCAN, since those test-set questions will

have only one or a few tokens differing from an existing training example. The BOS score could in fact be used to collect good candidate generalization questions faster, from existing datasets.

Comparison of BOS to the novelty metric from Jastrzebski et al. [49]. Another approach reminiscent of the BOS procedure is the "question novelty metric" used by Jastrzebski et al. [49] to create subsets of test-set questions, or bins, which are measured to have different degrees of novelty. In their work, the novelty, or "difficulty" of a test-set question is measured using word embedding distance between word-tokens in relation triples, as discussed earlier in Section 2.4.

We believe that the BOS presents substantial advantages to both a specialized generalization dataset and the novelty metric of Jastrzebski, as it is a simpler approach, that is less subjective. The BOS also allows to carry out a more granular examination of generalization on any test-set. The BOS allows to inspect model performance on test-set questions ranging from *train-set duplicates* (BOS = 100) to *quasi duplicates* (test-set questions with very high BOS), and all the way to test-set questions with (BOS = 0).

A low CodeSearch score indicates intents that are not descriptive, complementing the BOS retrieval ceiling analysis. While the BOS can help identify the best-scoring training example for a question, CodeSearch can give the score of the code snippet associated with the most semantically similar intent to the test-set question, i.e. give an idea of the scores obtained by training examples with intents that are reasonably descriptive, and related to the question. The BOS does not guarantee that the best-scoring example for a question has a relevant, sufficiently descriptive intent, and it is possible that the highest answer is unreachable, or "unlearnable" by the models, due to the inconsistency of its descriptions. CodeSearch scores can be used to discover which datasets have intents that are not descriptive of their code snippets.

Comparison of our results with previous Leclair et al. studies using the Javadoc dataset [55], noting the similarity of scores of a wide variety of models. Thanks to the recent efforts from Leclair et al. [54, 53, 38] and the additional study from Gupta [36], we can finally compare several different sequence-to-sequence architectures on the same dataset (C2E-Javadoc), for the task of code summarization. As we previously mentioned in 2.3, what is striking from this sequence of studies is that a wide variety of models (attend-GRU, with and without AST as encoder input, sequence-to-sequence with attention to file context, graph neural networks, Transformer, CodeNN LSTM-based model) all obtain very similar BLEU scores, (e.g. 19.4, 19.6 BLEU4), that are within the range one could obtain by re-training a sequence-to-sequence model with identical hyperparameter, or by choosing a different epoch for the final model once the validation accuracy has stabilized during training. The 21.28 BLEU4 score that we obtained with the Seq2SeqLSTM on that same dataset, although higher than previous attempts, is also still very close to the other models. Its slightly higher score could be due to the use of beam search in particular, which is well-known to

improve BLEU scores [83] for any sequence-to-sequence model, but is not used in the Leclair et al.'s ablation studies [54].

One possibility is that sequence-to-sequence models, if they are indeed capped by the BOS score on those tasks in practice, as we hypothesized in this section, will continue to stagnate. It is possible that new models with an improved capacity to generalize will be necessary in order to increase scores further, on the code summarization task, and other similar tasks. It would be interesting to systematically compare machine learning models scores with BOS scores, on a wide variety of tasks and architectures. Either we will be able to clearly identify whether the deep sequence-to-sequence paradigm is fundamentally limited in the same way retrieval models are on these types of tasks, or, we will discover certain architectures, training regimen or learning algorithms which are able to generalize above it, on certain tasks, and that will be very interesting. Although the use of an AST-aware model seems promising intuitively, it seems from the Leclair et al. studies that the improvements are not as significant as expected. Maybe some approaches outside of the sequence-to-sequence paradigm are better able to generalize, such as for example evolutionary or genetic program synthesis [31], sketch discovery [78] and hybrid systems [107], or modular, "compositional" models such as the one developed by Russin et al. [89]. A combination of approaches, i.e. hybrid approaches, seems also a reasonable avenue for exploration since they can combine the strengths of different models. One particularly interesting study from Ellis et al. [26] used a recombination-oriented approach for program synthesis, where the model develops a library of code functions, over time, which can be recombined by trial-and-error to solve new problems. Maybe this work can be adapted for English-to-code. What is interesting is that the BOS analysis allows to discover which of these models or combination of models is able to generalize to new questions, *even if each study uses a different training dataset*. So obtaining those answers about model generalization becomes a more realistic project.

Can some models score above the BOS score on certain tasks but not on others? It is possible that when the task involves copying tokens from input sequence to output sequence, with a relatively limited number of transformations such as is the case in natural language translation (i.e. "well-aligned" input and output sentences), sequence-to-sequence models (and other models) are able to generalize above the BOS score. It would be interesting to observe those cases. The capacity of sequence-to-sequence models to take advantage of the dataset-artifact we observed is a strong clue that this could be the case, especially because we anecdotally observe that sequence-to-sequence models can often make "relevant changes" during copying, such as changing *"get edge appearance calculator"* to *"returns the edge appearance calculator"*, which is essentially performing a translation.

It is possible that when the task is more difficult, beyond simple mappings between inputs and outputs, models are not able to generalize above the BOS anymore, as we observe in the present study.

If the input-output mapping rules necessary to succeed on the task are too complex, it is possible that the sequence-to-sequence model is only incentivized to overfit, to memorize the training-examples during training with gradient-descent, and has no way of reaching the correct internal representation necessary to succeed on new inputs. If the training-examples do not fully define, or "teach" the correct internal representation, then maybe it is too unlikely to appear during training.

One way to estimate the complexity of a task, and the complexity of the underlying internal representation necessary for a model to answer a set of questions correctly, would be to ask several human experts to report the steps in the decision-making that leads to the correct answers. The depth of the tree in such an expert system could be used as a proxy for task complexity. Each decision in the (experimental) expert system can be translated into a set of tensor operations (matrix multiplications) to reach the correct output at each step, and this way a model size (and problem complexity) can be estimated. Even if incomplete, such an expert system might give us an idea of how unlikely it is that its structure (i.e. internal representation) will be discovered during training with gradient-descent. Such a study could also provide us with a set of relevant sub-steps, for which independent, modular machine learning models could be trained. Sequence-to-sequence models already are composed of two sub-steps: the encoding of the meaning of the input sentence in a thought-vector, and the next-token prediction by the decoder based on the thought-vector and the previously generated tokens. These two sub-steps might not be sufficient on tasks beyond natural language translation. After an empirical examination of the decision-making for the tasks when carried out by humans, more sub-steps might be added, especially for a task like code generation. Such an empirical study on expert systems could be carried out with software developers, to measure the complexity of the task of writing code from requirements formulated either as tests (program synthesis, English-to-code) or as an English intent (English-to-code).

6.4 CodeSearch Outperforms Neural Seq2Seq Models

Maybe the most surprising conclusion of this comparative study is that CodeSearch, a relatively simple retrieval model, can on average surpass the two neural sequence-to-sequence models. When there are no dataset-artifacts and when the dataset is not filtered by one of the sequence-to-sequence models, the CodeSearch model performs above sequence-to-sequence models 5 out of 6 times, and when it is surpassed, it is only by a very small margin, on a task-dataset combination where all three model scores are near chance (E2C-SO). Even if we do not exclude the two problematic datasets CodeSearch is still the best performing model overall, surpassing the two neural sequence-to-sequence models on 5 out of the 8 data-set task combinations.

On some datasets, CodeSearch outperforms the neural sequence-to-sequence models by very large

margins in BLEU4 score. On C2E-Pydoc and E2C-Pydoc, CodeSearch surpasses the neural models by a large margin also, of 20.12 and 19.46 BLEU4 points. One reason could be that the Pydoc dataset does not include the function name in the code snippet, only the function body. This reduces the number of dataset-artifacts of the type identified in section 5.3, and might explain the low BLEU scores of the neural models on that dataset, compared to the Javadoc dataset.

Some previous studies have also reported retrieval models outperforming standard off-the-shelf neural sequence-to-sequence models, in similar experiments [61, 112, 116, 4]. Liu et al. [61] found that the high-scoring answers from sequence-to-sequence NMT approaches tend to always contain token sequences that are very similar to those in existing training-examples. We reached a similar conclusion in Section 5.3: neural sequence-to-sequence models can only score high for a given test-set question, if the answer is already in the training-set. They also find 16% of “trivial", or easy code-English pairs, which they find are responsible for a large portion of the high-scoring NMT models' answers. They find that NMT models' scores are more than halved (31.92 to 14.19 BLEU) after removing those trivial training-examples from the dataset. When they compare the NMT models to a kNN-based retrieval model on a version of the dataset cleaned of the trivial examples that boosted NMT model scores, they observe that the retrieval model outperforms NMT by a substantial margin. In another study, Yin et al. [112], report that a retrieval model surpasses an off-the-shelf neural sequence-to-sequence model on a code generation task, but the retrieval model was ultimately surpassed by one of the study's more sophisticated generative models, a seq2tree model, on that particular dataset. Allamanis et al. [4] report that a TF-IDF based retrieval model surpassed an off-the-shelf NMT sequence-to-sequence model in their experiments. although again here, their more sophisticated sequence-to-sequence model (copy-attention) goes above the retrieval model. In a similar study for the code-to-English task, Zhang et al. [116] report that, out of their three hybrid models, only one scored above pure retrieval models, and by a very small margin (0.5 BLEU), a change in score that can sometimes be obtained by simple re-training of the same model. These results confirm that, despite being more recent, sequence-to-sequence models do not consistently outperforms retrieval models on the code generation and code summarizaton task, and that the outcome can be affected by issues with dataset quality, such as the ones we noted in section 5.3, and the ones noted by Liu et al. [61].

Another important aspect in the comparison between neural sequence-to-sequence and retrieval models is that CodeSearch has the advantage of always returning a complete, working code snippet (in so far as the dataset is composed of relatively coherent code snippets). This can be important for developer end users. CodeSearch returns code snippets intact from the dataset, and for that reason, it might be more useful for developers in production environments. When performance is measured by users instead of with the BLEU score, CodeSearch might be markedly preferred for that reason.

It is important to note that BLEU favours the neural generative models, in that it does not test for the syntactic validity of the code. Neural sequence-to-sequence models can generate code that does not compile, uses API elements incorrectly nested for example, yet still score high in BLEU.

6.5 Seq2SeqLSTM Outperforms the AttendGRU

We also observe that the Seq2SeqLSTM outperforms the AttendGRU on all 8 out of 8 task-dataset combinations. GRUs were explicitly designed to minimize training time complexity, but in terms of BLEU4 score, we observe that they are unable to match the Seq2SeqLSTM and CodeSearch models in our experiments. Several previous studies have also noted a small but significant decrease in performance with the use of GRUs instead of LSTMs in sequence-to-sequence models [12, 108]. Britz et al. [12] in a comparative study of LSTMs and GRUs in sequence-to-sequence architectures observe that LSTM consistently surpass GRU cells in BLEU score. They also do not observe large differences in training speed between the two variants. Yang et al. [108] find that the GRU is 29.29% faster to train on an identical dataset, and also outperforms the LSTM in some cases, in the case of long texts and a small dataset. They find that the LSTM outperforms the GRU in other configurations, however. Chung et al. [21] report that a GRU-based sequence-to-sequence model can train faster than its LSTM counterpart, and also can generalize better. In our experiments, the Seq2SeqLSTM always outperforms the AttendGRU model, as discussed in section 5.

Chapter 7

Contributions and Concluding Remarks

In this study, we have assessed the performance of three state-of-the-art models on the code generation task (code-to-English) and the code summarization task (English-to-code). We replicated two seq2seq models, Seq2SeqLSTM and AttendGRU and a semantic CodeSearch model (document retrieval model) for comparison. We first ran these models on their original study's dataset, to ensure that the models are working correctly and that the scores are similar to the original published scores. We then ran these models on the datasets from other studies.

We observed that CodeSearch scores higher than the neural seq2seq models (Seq2SeqLSTM, AttendGRU) on 5 out of 8 task-dataset combinations examined. On 2 of the 3 task-dataset combinations where a sequence-to-sequence model scores above CodeSearch, problems with the dataset were discovered: a dataset-filtering issue on the CoNaLa corpus [111] (E2C-SO), and the presence of dataset-artifacts on the code-Javadoc corpus from Leclair et al. [55] (C2E-Javadoc). We showed that these problems give an unfair advantage to Seq2SeqLSTM and AttendGRU, further confirming CodeSearch's superiority on the tasks. We also observed that the Seq2SeqLSTM always outperforms the AttendGRU model, including on the relatively large (2M pairs) dataset originally used in the AttendGRU study, which sets a new high score on that dataset (higher than four previous studies on that dataset). We observed very large differences in scores from dataset to dataset, and sometimes very low BLEU scores compared to scores seen in natural language translation.

We introduced the BOS score (retrieval ceiling score), to understand model performance in relation to available data. For each test-set question, we search the training-set for the most relevant, highest-scoring training-example, and reported its BLEU score. We observed that the BOS retrieval ceiling is rarely surpassed by the AttendGRU and Seq2SeqLSTM models in our experiments. In

a manual inspection, we observed that the rare model answers scoring above the BOS are caused every time by dataset-artifacts, and concentrated mostly on one dataset: the Javadoc dataset from Leclair et al. [54] (C2E-Javadoc). In two random samples of test-set questions, we measured that these dataset-artifacts are responsible for at least 25% and 39% of the non-zero answers from the two sequence-to-sequence models, Seq2SeqLSTM and AttendGRU respectivey, greatly inflating their BLEU score. We observed that on the rest of the task-dataset combinations examined in our experiments, without the presence of dataset-artifacts, neural sequence-to-sequence models are unable to generalize on the task, and produce an answer that systematically scores below the BLEU score of the best-scoring training-set answer for the given test-set question.

Potential future work includes examining neural seq2seq models such as LSTMs and GRUs and Transformers (BERT family, GPT-C), to see whether they are able to surpass the BOS as the dataset grows in size, and as the number of trainable parameters is increased. In our relatively modest experiments, we see no evidence that increasing the dataset size enables neural seq2seq models to score above the BOS. We formulate the possibility that, as the training-set size is increased, the BOS score is simply increased along with it, and is still not surpassed by the sequence-to-sequence models. An interesting question is whether a seq2seq model's occasionally impressive test-set answers are already present in the training-set. It would be interesting to discover which models and types of architectures are able to generalize, and score above the BOS for a given dataset. Of course the BOS can be adapted to any scoring metric, not just the BLEU.

We provide a final summary of our contributions as follows:

1. We compare sequence-to-sequence and retrieval models for the code generation and code summarization tasks, on four different datasets.

2. We produce four new state-of-the-art high scores on three datasets (C2E-Javadoc-Seq2SeqLSTM, E2C-SO-Seq2SeqLSTM, E2C-SOa-Seq2SeqLSTM, E2C-SOa-CodeSearch), using replicated and reproduced models. These results surpassed six previous published BLEU scores on the same datasets.

3. We observe that CodeSearch outperforms Seq2SeqLSTM and AttendGRU, which is also occasionally observed in previous works on other datasets, but is nonetheless a surprising result.

4. We introduce a new question-novelty metric (BOS score), which allows to inspect whether a model generalizes to unseen questions; this metric also helps to accelerate the identification of dataset-artifacts (by looking at the rare model answers that score above their BOS score).

5. We observe that original study datasets [111, 55] unfairly favour sequence-to-sequence models,

which helps to explain why 2 out of the 3 times CodeSearch is surpassed correspond the two datasets from previous studies [111, 54].

6. We observe that, in the absence of dataset artifacts, neural seq2seq models never score above the BOS, and do not generalize at all to unseen questions on the particular tasks and datasets examined. In other words, we observe that, on the particular tasks and datasets from previous studies that we examine, neural models cannot answer a question correctly unless it is already in the training-set.

Bibliography

[1] A. Agarwal and A. Lavie. METEOR, M-BLEU and M-TER: Evaluation metrics for high-correlation with human rankings of machine translation output. In *Proceedings of the Third Workshop on Statistical Machine Translation*, StatMT '08, page 115–118, USA, 2008. Association for Computational Linguistics.

[2] M. Allamanis, E. T. Barr, P. T. Devanbu, and C. Sutton. A survey of machine learning for big code and naturalness. *ACM Computing Surveys (CSUR)*, 51(4):81:1–81:37, 2018.

[3] M. Allamanis, M. Brockschmidt, and M. Khademi. Learning to represent programs with graphs. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

[4] M. Allamanis, H. Peng, and C. Sutton. A convolutional attention network for extreme summarization of source code. In *International conference on machine learning*, pages 2091–2100. PMLR, 2016.

[5] M. Allamanis, D. Tarlow, A. Gordon, and Y. Wei. Bimodal modelling of source code and natural language. In *International conference on machine learning*, pages 2123–2132. PMLR, 2015.

[6] U. Alon, S. Brody, O. Levy, and E. Yahav. code2seq: Generating sequences from structured representations of code. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[7] B. Athiwaratkun, C. dos Santos, J. Krone, and B. Xiang. Augmented natural language for generative sequence labeling. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 375–385, 2020.

[8] J. Bader, A. Scott, M. Pradel, and S. Chandra. Getafix: learning to fix bugs automatically. *Proceedings of the ACM on Programming Languages*, 3 (OOPSLA):159:1–159:27, 2019.

[9] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. In Y. Bengio and Y. LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[10] S. Banerjee and A. Lavie. METEOR: an automatic metric for MT evaluation with improved correlation with human judgments. In J. Goldstein, A. Lavie, C. Lin, and C. R. Voss, editors, *Proceedings of the Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization@ACL 2005, Ann Arbor, Michigan, USA, June 29, 2005*, pages 65–72. Association for Computational Linguistics, 2005.

[11] L. Bentivogli, A. Bisazza, M. Cettolo, and M. Federico. Neural versus phrase-based machine translation quality: a case study. In J. Su, X. Carreras, and K. Duh, editors, *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*, pages 257–267. The Association for Computational Linguistics, 2016.

[12] D. Britz, A. Goldie, M.-T. Luong, and Q. Le. Massive exploration of neural machine translation architectures. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1442–1451, 2017.

[13] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[14] E. Bugliarello, S. J. Mielke, A. Anastasopoulos, R. Cotterell, and N. Okazaki. It's easier to translate out of english than into it: Measuring neural translation difficulty by cross-mutual information. In D. Jurafsky, J. Chai, N. Schluter, and J. R. Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 1640–1649. Association for Computational Linguistics, 2020.

[15] W. R. Caid, S. T. Dumais, and S. I. Gallant. Learned vector-space models for document retrieval. *Information Processing & Management*, 31(3):419–429, 1995.

[16] C. Callison-Burch, M. Osborne, and P. Koehn. Re-evaluating the role of Bleu in machine translation research. In D. McCarthy and S. Wintner, editors, *EACL 2006, 11st Conference of the European Chapter of the Association for Computational Linguistics, Proceedings of the Conference, April 3-7, 2006, Trento, Italy*. The Association for Computer Linguistics, 2006.

[17] J. Cambronero, H. Li, S. Kim, K. Sen, and S. Chandra. When deep learning met code search. In M. Dumas, D. Pfahl, S. Apel, and A. Russo, editors, *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2019, Tallinn, Estonia, August 26-30, 2019*, pages 964–974. ACM, 2019.

[18] Q. Chen, H. Hu, and Z. Liu. Code summarization with abstract syntax tree. In T. Gedeon, K. W. Wong, and M. Lee, editors, *Neural Information Processing - 26th International Conference, ICONIP 2019, Sydney, NSW, Australia, December 12-15, 2019, Proceedings, Part V*, volume 1143 of *Communications in Computer and Information Science*, pages 652–660. Springer, 2019.

[19] Q. Chen and M. Zhou. A neural framework for retrieval and summarization of source code. In M. Huchard, C. Kästner, and G. Fraser, editors, *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018, Montpellier, France, September 3-7, 2018*, pages 826–831. ACM, 2018.

[20] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, 2014.

[21] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *NIPS 2014 Workshop on Deep Learning, December 2014*, 2014.

[22] O. J. Coleman, A. D. Blair, and J. Clune. Automated generation of environments to test the general learning capabilities of AI agents. In D. V. Arnold, editor, *Genetic and Evolutionary Computation Conference, GECCO '14, Vancouver, BC, Canada, July 12-16, 2014*, pages 161–168. ACM, 2014.

[23] D. Cubranic and G. C. Murphy. Hipikat: Recommending pertinent software development artifacts. In L. A. Clarke, L. Dillon, and W. F. Tichy, editors, *Proceedings of the 25th International Conference on Software Engineering, May 3-10, 2003, Portland, Oregon, USA*, pages 408–418. IEEE Computer Society, 2003.

[24] D. Cubranic, G. C. Murphy, J. Singer, and K. S. Booth. Hipikat: A project memory for software development. *IEEE Transactions on Software Engineering*, 31(6):446–465, 2005.

[25] D. Dua, S. Singh, and M. Gardner. Benefits of intermediate annotations in reading comprehension. In D. Jurafsky, J. Chai, N. Schluter, and J. R. Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 5627–5634. Association for Computational Linguistics, 2020.

[26] K. Ellis, C. Wong, M. I. Nye, M. Sablé-Meyer, L. Cary, L. Morales, L. B. Hewitt, A. Solar-Lezama, and J. B. Tenenbaum. Dreamcoder: Growing generalizable, interpretable knowledge with wake-sleep bayesian program learning. *CoRR*, abs/2006.08381, 2020.

[27] S. Falkner, A. Klein, and F. Hutter. BOHB: robust and efficient hyperparameter optimization at scale. In J. G. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 1436–1445. PMLR, 2018.

[28] S. Feng, E. Wallace, and J. L. Boyd-Graber. Misleading failures of partial-input baselines. In A. Korhonen, D. R. Traum, and L. Màrquez, editors, *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 5533–5538. Association for Computational Linguistics, 2019.

[29] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, and M. Zhou. Codebert: A pre-trained model for programming and natural languages. In T. Cohn, Y. He, and Y. Liu, editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings, EMNLP 2020, Online Event, 16-20 November 2020*, pages 1536–1547. Association for Computational Linguistics, 2020.

[30] P. Fernandes, M. Allamanis, and M. Brockschmidt. Structured neural summarization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[31] S. Forstenlechner, D. Fagan, M. Nicolau, and M. O'Neill. Towards understanding and refining the general program synthesis benchmark suite with genetic programming. In *2018 IEEE Congress on Evolutionary Computation, CEC 2018, Rio de Janeiro, Brazil, July 8-13, 2018*, pages 1–6. IEEE, 2018.

[32] K. Goel, N. F. Rajani, J. Vig, S. Tan, J. Wu, S. Zheng, C. Xiong, M. Bansal, and C. Ré. Robustness gym: Unifying the NLP evaluation landscape. *CoRR*, abs/2101.04840, 2021.

[33] X. Gu, H. Zhang, and S. Kim. Deep code search. In M. Chaudron, I. Crnkovic, M. Chechik, and M. Harman, editors, *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, pages 933–944. ACM, 2018.

[34] X. Gu, H. Zhang, D. Zhang, and S. Kim. Deep API learning. In T. Zimmermann, J. Cleland-Huang, and Z. Su, editors, *Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2016, Seattle, WA, USA, November 13-18, 2016*, pages 631–642. ACM, 2016.

[35] L. Guerrouj, D. Bourque, and P. C. Rigby. Leveraging informal documentation to summarize classes and methods in context. In A. Bertolino, G. Canfora, and S. G. Elbaum, editors, *37th IEEE/ACM International Conference on Software Engineering, ICSE 2015, Florence, Italy, May 16-24, 2015, Volume 2*, pages 639–642. IEEE Computer Society, 2015.

[36] V. Gupta. Deepsumm - deep code summaries using neural transformer architecture. *CoRR*, abs/2004.00998, 2020.

[37] S. Gururangan, S. Swayamdipta, O. Levy, R. Schwartz, S. R. Bowman, and N. A. Smith. Annotation artifacts in natural language inference data. In M. A. Walker, H. Ji, and A. Stent, editors, *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 2 (Short Papers)*, pages 107–112. Association for Computational Linguistics, 2018.

[38] S. Haque, A. LeClair, L. Wu, and C. McMillan. Improved automatic summarization of subroutines via attention to file context. In S. Kim, G. Gousios, S. Nadi, and J. Hejderup, editors, *MSR '20: 17th International Conference on Mining Software Repositories, Seoul, Republic of Korea, 29-30 June, 2020*, pages 300–310. ACM, 2020.

[39] H. Hassan, A. Aue, C. Chen, V. Chowdhary, J. Clark, C. Federmann, X. Huang, M. Junczys-Dowmunt, W. Lewis, M. Li, S. Liu, T.-Y. Liu, R. Luo, A. Menezes, T. Qin, F. Seide, X. Tan, F. Tian, L. Wu, S. Wu, Y. Xia, D. Zhang, Z. Zhang, and M. Zhou. Achieving human parity on automatic Chinese to English news translation. *ArXiv*, abs/1803.05567, 2018.

[40] A. Hindle, E. T. Barr, Z. Su, M. Gabel, and P. Devanbu. On the naturalness of software. In *Proceedings of the 34th International Conference on Software Engineering*, ICSE '12, page 837–847. IEEE Press, 2012.

[41] S. Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *Int. J. Uncertain. Fuzziness Knowl. Based Syst.*, 6(2):107–116, 1998.

[42] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, 1997.

[43] R. Holmes and G. C. Murphy. Using structural context to recommend source code examples. In *Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005.*, pages 117–125, 2005.

[44] H. Hu, Q. Chen, and Z. Liu. Code generation from supervised code embeddings. In T. Gedeon, K. W. Wong, and M. Lee, editors, *Neural Information Processing - 26th International Conference, ICONIP 2019, Sydney, NSW, Australia, December 12-15, 2019, Proceedings, Part IV*, volume 1142 of *Communications in Computer and Information Science*, pages 388–396. Springer, 2019.

[45] X. Hu, G. Li, X. Xia, D. Lo, and Z. Jin. Deep code comment generation. *2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC)*, pages 200–20010, 2018.

[46] L. Huang, K. Zhao, and M. Ma. When to finish? Optimal beam search for neural text generation (modulo beam size). In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2134–2139, 2017.

[47] H. Husain, H.-H. Wu, T. Gazit, M. Allamanis, and M. Brockschmidt. CodeSearchNet Challenge: Evaluating the state of semantic code search. *ArXiv*, abs/1909.09436, 2019.

[48] S. Iyer, I. Konstas, A. Cheung, and L. Zettlemoyer. Summarizing source code using a neural attention model. 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, 2016.

[49] S. Jastrzebski, D. Bahdanau, S. Hosseini, M. Noukhovitch, Y. Bengio, and J. C. K. Cheung. Commonsense mining as knowledge base completion? A study on the impact of novelty. *NAACL HLT 2018*, page 8, 2018.

[50] S. Karaivanov, V. Raychev, and M. T. Vechev. Phrase-based statistical translation of programming languages. In A. P. Black, S. Krishnamurthi, B. Bruegge, and J. N. Ruskiewicz, editors, *Onward! 2014, Proceedings of the 2014 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software, part of SPLASH '14, Portland, OR, USA, October 20-24, 2014*, pages 173–184. ACM, 2014.

[51] B. M. Lake and M. Baroni. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In J. G. Dy and A. Krause, editors, *Proceedings*

of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018, volume 80 of *Proceedings of Machine Learning Research*, pages 2879–2888. PMLR, 2018.

[52] A. Lavie, K. Sagae, and S. Jayaraman. The significance of recall in automatic metrics for MT evaluation. In R. E. Frederking and K. Taylor, editors, *Machine Translation: From Real Users to Research, 6th Conference of the Association for Machine Translation in the Americas, AMTA 2004, Washington, DC, USA, September 28-October 2, 2004, Proceedings*, volume 3265 of *Lecture Notes in Computer Science*, pages 134–143. Springer, 2004.

[53] A. LeClair, S. Haque, L. Wu, and C. McMillan. Improved code summarization via a graph neural network. In *Proceedings of the 28th International Conference on Program Comprehension*, pages 184–195, 2020.

[54] A. LeClair, S. Jiang, and C. McMillan. A neural model for generating natural language summaries of program subroutines. In J. M. Atlee, T. Bultan, and J. Whittle, editors, *Proceedings of the 41st International Conference on Software Engineering, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*, pages 795–806. IEEE / ACM, 2019.

[55] A. LeClair and C. McMillan. Recommendations for datasets for source code summarization. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 3931–3937, 2019.

[56] H. Li, S. Kim, and S. Chandra. Neural code search evaluation dataset. *ArXiv*, abs/1908.09804, 2019.

[57] R. Liaw. Hyperparameter search with transformers and Ray Tune. Available at `https://huggingface.co/blog/ray-tune`.

[58] W. Ling, P. Blunsom, E. Grefenstette, K. M. Hermann, T. Kočiský, F. Wang, and A. Senior. Latent predictor networks for code generation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 599–609, Berlin, Germany, Aug. 2016. Association for Computational Linguistics.

[59] C. Liu, X. Wang, R. Shin, J. E. Gonzalez, and D. Song. Neural code completion. 2016. Retrieved from `https://openreview.net/pdf?id=rJbPBt9lg`.

[60] J. Liu, S. Kim, V. Murali, S. Chaudhuri, and S. Chandra. Neural query expansion for code search. *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, 2019.

[61] Z. Liu, X. Xia, A. Hassan, D. Lo, Z. Xing, and X. Wang. Neural-machine-translation-based commit message generation: How far are we? *2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 373–384, 2018.

[62] K. E. Lochbaum and L. A. Streeter. Comparing and combining the effectiveness of latent semantic indexing and the ordinary vector space model for information retrieval. *Information Processing & Management*, 25(6):665–676, 1989.

[63] S. Luan, D. Yang, C. Barnaby, K. Sen, and S. Chandra. Aroma: code recommendation via structural code search. *Proceedings of the ACM on Programming Languages*, 3 (OOPSLA):152:1–152:28, 2019.

[64] C. Maddison and D. Tarlow. Structured generative models of natural source code. In E. P. Xing and T. Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 649–657, Bejing, China, 22–24 Jun 2014. PMLR.

[65] T. McCoy, E. Pavlick, and T. Linzen. Right for the wrong reasons: Diagnosing syntactic heuristics in natural language inference. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3428–3448, 2019.

[66] I. D. Melamed, R. Green, and J. P. Turian. Precision and recall of machine translation. In M. A. Hearst and M. Ostendorf, editors, *Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics, HLT-NAACL 2003, Edmonton, Canada, May 27 - June 1, 2003*. The Association for Computational Linguistics, 2003.

[67] A. V. Miceli-Barone and R. Sennrich. A parallel corpus of python functions and documentation strings for automated code documentation and code generation. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 314–319, 2017.

[68] A. Michail. Codeweb: Data mining library reuse patterns. In H. A. Müller, M. J. Harrold, and W. Schäfer, editors, *Proceedings of the 23rd International Conference on Software Engineering, ICSE 2001, 12-19 May 2001, Toronto, Ontario, Canada*, pages 827–828. IEEE Computer Society, 2001.

[69] J. Moore, B. Gelman, and D. Slater. A convolutional neural network for language-agnostic source code summarization. In E. Damiani, G. Spanoudakis, and L. A. Maciaszek, editors, *Proceedings of the 14th International Conference on Evaluation of Novel Approaches to Software*

Engineering, ENASE 2019, Heraklion, Crete, Greece, May 4-5, 2019, pages 15–26. SciTePress, 2019.

[70] V. Murali, L. Qi, S. Chaudhuri, and C. Jermaine. Neural sketch learning for conditional program generation. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

[71] M. Naseriparsa, M. S. Islam, C. Liu, and L. Chen. XSnippets: Exploring semi-structured data via snippets. *Data & Knowledge Engineering*, 124:101758, 2019.

[72] A. Nguyen and T. Nguyen. Graph-based statistical language model for code. *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, 1:858–868, 2015.

[73] A. T. Nguyen, P. C. Rigby, T. Van Nguyen, M. Karanfil, and T. N. Nguyen. Statistical translation of english texts to API code templates. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, pages 331–333, 2017.

[74] T. Nguyen, N. Pham, S. Stüker, and A. Waibel. High performance sequence-to-sequence model for streaming speech recognition. In H. Meng, B. Xu, and T. F. Zheng, editors, *Interspeech 2020, 21st Annual Conference of the International Speech Communication Association, Virtual Event, Shanghai, China, 25-29 October 2020*, pages 2147–2151. ISCA, 2020.

[75] T. Nguyen, P. C. Rigby, A. T. Nguyen, M. Karanfil, and T. N. Nguyen. T2API: Synthesizing API code usage templates from english texts with statistical translation. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2016, page 1013–1017, New York, NY, USA, 2016. Association for Computing Machinery.

[76] T. T. Nguyen, A. T. Nguyen, H. A. Nguyen, and T. N. Nguyen. A statistical semantic language model for source code. In B. Meyer, L. Baresi, and M. Mezini, editors, *Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, ESEC/FSE'13, Saint Petersburg, Russian Federation, August 18-26, 2013*, pages 532–542. ACM, 2013.

[77] R. Nogueira, Z. Jiang, and J. Li. Investigating the limitations of the transformers with simple arithmetic tasks. *ArXiv*, abs/2102.13019, 2021.

[78] M. I. Nye, L. B. Hewitt, J. B. Tenenbaum, and A. Solar-Lezama. Learning to infer program sketches. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 4861–4870. PMLR, 2019.

[79] K. Papineni, S. Roukos, T. Ward, and W. Zhu. 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*, pages 311–318. ACL, 2002.

[80] V. Premtoon, J. Koppel, and A. Solar-Lezama. Semantic code search via equational reasoning. In A. F. Donaldson and E. Torlak, editors, *Proceedings of the 41st ACM SIGPLAN International Conference on Programming Language Design and Implementation, PLDI 2020, London, UK, June 15-20, 2020*, pages 1066–1082. ACM, 2020.

[81] M. Raghothaman, Y. Wei, and Y. Hamadi. Swim: Synthesizing what i mean: Code search and idiomatic snippet synthesis. In *Proceedings of the 38th International Conference on Software Engineering*, ICSE '16, page 357–367, New York, NY, USA, 2016. Association for Computing Machinery.

[82] M. Rahman. *Analyzing the Predictability of Source Code and its Application in Creating Parallel Corpora for English-to-Code Statistical Machine Translation.* PhD thesis, Concordia University Montréal, Québec, Canada, 2018.

[83] M. Ranzato, S. Chopra, M. Auli, and W. Zaremba. Sequence level training with recurrent neural networks. In Y. Bengio and Y. LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

[84] B. Recht, R. Roelofs, L. Schmidt, and V. Shankar. Do CIFAR-10 classifiers generalize to CIFAR-10? *ArXiv*, abs/1806.00451, 2018.

[85] B. Recht, R. Roelofs, L. Schmidt, and V. Shankar. Do ImageNet classifiers generalize to ImageNet? In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 5389–5400. PMLR, 2019.

[86] A. Ross, A. Marasović, and M. E. Peters. Explaining NLP models via minimal contrastive editing (MiCE). *ArXiv*, abs/2012.13985, 2020.

[87] B. Rozière, M. Lachaux, L. Chanussot, and G. Lample. Unsupervised translation of programming languages. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[88] A. M. Rush, S. Chopra, and J. Weston. A neural attention model for abstractive sentence summarization. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 379–389, 2015.

[89] J. Russin, J. Jo, R. C. O'Reilly, and Y. Bengio. Compositional generalization in a deep seq2seq model by separating syntax and semantics. *ArXiv*, abs/1904.09708, 2019.

[90] S. Sachdev, H. Li, S. Luan, S. Kim, K. Sen, and S. Chandra. Retrieval on source code: A neural code search. In *Proceedings of the 2nd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, MAPL 2018, page 31–41, New York, NY, USA, 2018. Association for Computing Machinery.

[91] C. Sadowski, K. T. Stolee, and S. G. Elbaum. How developers search for code: a case study. In E. D. Nitto, M. Harman, and P. Heymans, editors, *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, Bergamo, Italy, August 30 - September 4, 2015*, pages 191–201. ACM, 2015.

[92] N. Sahavechaphan and K. T. Claypool. XSnippet: mining for sample code. In P. L. Tarr and W. R. Cook, editors, *Proceedings of the 21th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2006, October 22-26, 2006, Portland, Oregon, USA*, pages 413–430. ACM, 2006.

[93] G. Salton, A. Wong, and C. Yang. A vector space model for automatic indexing. *Commun. ACM*, 18(11):613–620, 1975.

[94] F. Scheidegger, R. Istrate, G. Mariani, L. Benini, C. Bekas, and C. Malossi. Efficient image dataset classification difficulty estimation for predicting deep-learning accuracy. *The Visual Computer*, pages 1–18, 2020.

[95] Z. Shi, J. Tang, H. Yu, Y. Xing, Z. Liu, W. Bai, and T. Li. A quantitative benefit evaluation of code search platform for enterprises. *Science China Information Sciences*, 63(9):1–3, 2020.

[96] C. Si, S. Wang, M.-Y. Kan, and J. Jiang. What does BERT learn from multiple-choice reading comprehension datasets? *ArXiv*, abs/1910.12391, 2019.

[97] S. Stapleton, Y. Gambhir, A. LeClair, Z. Eberhart, W. Weimer, K. Leach, and Y. Huang. A human study of comprehension and code summarization. In *ICPC '20: 28th International Conference on Program Comprehension, Seoul, Republic of Korea, July 13-15, 2020*, pages 2–13. ACM, 2020.

[98] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, page 3104–3112, Cambridge, MA, USA, 2014. MIT Press.

[99] A. Svyatkovskiy, S. K. Deng, S. Fu, and N. Sundaresan. Intellicode compose: Code generation using transformer. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1433–1443, 2020.

[100] H. H. Tan. ChordAL: A chord-based approach for music generation using Bi-LSTMs. In K. Grace, M. Cook, D. Ventura, and M. L. Maher, editors, *Proceedings of the Tenth International Conference on Computational Creativity, ICCC 2019, Charlotte, North Carolina, USA, June 17-21, 2019*, pages 364–365. Association for Computational Creativity (ACC), 2019.

[101] H. Trivedi, N. Balasubramanian, T. Khot, and A. Sabharwal. Is multihop QA in dire condition? measuring and reducing disconnected reasoning. In B. Webber, T. Cohn, Y. He, and Y. Liu, editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, pages 8846–8863. Association for Computational Linguistics, 2020.

[102] E. Wallace, S. Feng, N. Kandpal, M. Gardner, and S. Singh. Universal adversarial triggers for attacking and analyzing NLP. In K. Inui, J. Jiang, V. Ng, and X. Wan, editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pages 2153–2162. Association for Computational Linguistics, 2019.

[103] J. Wang and J. Han. BIDE: efficient mining of frequent closed sequences. In Z. M. Özsoyoglu and S. B. Zdonik, editors, *Proceedings of the 20th International Conference on Data Engineering, ICDE 2004, 30 March - 2 April 2004, Boston, MA, USA*, pages 79–90. IEEE Computer Society, 2004.

[104] X. Wang and G. Neubig. Target conditioned sampling: Optimizing data selection for multilingual neural machine translation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5823–5828, 2019.

[105] M. Xia, A. Anastasopoulos, R. Xu, Y. Yang, and G. Neubig. Predicting performance for natural language processing tasks. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8625–8646, 2020.

[106] X. Xia, L. Bao, D. Lo, P. S. Kochhar, A. E. Hassan, and Z. Xing. What do developers search for on the web? *Empirical Software Engineering*, 22(6):3149–3185, 2017.

[107] Q. Xin and S. P. Reiss. Better code search and reuse for better program repair. *2019 IEEE/ACM International Workshop on Genetic Improvement (GI)*, pages 10–17, 2019.

[108] S. Yang, X. Yu, and Y. Zhou. LSTM and GRU neural network performance comparison study: Taking Yelp review dataset as an example. *2020 International Workshop on Electronic Communication and Artificial Intelligence (IWECAI)*, pages 98–101, 2020.

[109] W. Ye, R. Xie, J. Zhang, T. Hu, X. Wang, and S. Zhang. Leveraging code generation to improve code retrieval and summarization via dual learning. In Y. Huang, I. King, T. Liu, and M. van Steen, editors, *WWW '20: The Web Conference 2020, Taipei, Taiwan, April 20-24, 2020*, pages 2309–2319. ACM / IW3C2, 2020.

[110] P. Yin, E. Chen, B. Vasilescu, and G. Neubig. The code/natural language challenge. Available at `https://conala-corpus.github.io/`.

[111] P. Yin, B. Deng, E. Chen, B. Vasilescu, and G. Neubig. Learning to mine aligned code and natural language pairs from stack overflow. *2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR)*, pages 476–486, 2018.

[112] P. Yin and G. Neubig. A syntactic neural model for general-purpose code generation. In R. Barzilay and M. Kan, editors, *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, pages 440–450. Association for Computational Linguistics, 2017.

[113] A. T. T. Ying, G. C. Murphy, R. T. Ng, and M. Chu-Carroll. Predicting source code changes by mining change history. *IEEE Transactions on Software Engineering*, 30(9):574–586, 2004.

[114] S. R. Young, D. C. Rose, T. P. Karnowski, S.-H. Lim, and R. M. Patton. Optimizing deep learning hyper-parameters through an evolutionary algorithm. In *Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments*, pages 1–5, 2015.

[115] L. Zeng, X. Zhang, T. Wang, X. Li, J. Yu, and H. Wang. Improving code summarization by combining deep learning and empirical knowledge (S). In Ó. M. Pereira, editor, *The 30th International Conference on Software Engineering and Knowledge Engineering, Hotel Pullman, Redwood City, California, USA, July 1-3, 2018*, pages 566–565. KSI Research Inc. and Knowledge Systems Institute Graduate School, 2018.

[116] J. Zhang, X. Wang, H. Zhang, H. Sun, and X. Liu. Retrieval-based neural source code summarization. *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*, pages 1385–1397, 2020.

[117] Y. Zhang, S. Vogel, and A. Waibel. Interpreting BLEU/NIST scores: How much improvement do we need to have a better system? In *Proceedings of the Fourth International Conference on Language Resources and Evaluation, LREC 2004, May 26-28, 2004, Lisbon, Portugal*. European Language Resources Association, 2004.

[118] S. Zhao, E. Deng, M. Liao, W. Liu, and W. Mao. Generating summary using sequence to sequence model. *2020 IEEE 5th Information Technology and Mechatronics Engineering Conference (ITOEC)*, pages 1102–1106, 2020.

[119] H. Zhong, T. Xie, L. Zhang, J. Pei, and H. Mei. MAPO: Mining and recommending api usage patterns. In *Proceedings of the 23rd European Conference on ECOOP 2009—Object-Oriented Programming*, pages 318–343, 2009.

[120] Y. Zhu and M. Pan. Automatic code summarization: A systematic literature review. *ArXiv*, abs/1909.04352, 2019.