Reading StackOverflow Encourages Cheating: Adding Question Text Improves Extractive Code Generation

Gabriel Orlanski
Rensselaer Polytechnic Institute
orlang2@rpi.edu

Alex Gittens
Rensselaer Polytechnic Institute
gittea@rpi.edu

Abstract

Answering a programming question using only its title is difficult as salient contextual information is omitted. Based on this observation, we present a corpus of over 40,000 StackOverflow question texts to be used in conjunction with their corresponding intents from the CoNaLa dataset (Yin et al., 2018). Using both the intent and question body, we use BART to establish a baseline BLEU score of 34.35 for this new task. We find further improvements of 2.8% by combining the mined CoNaLa data with the labeled data to achieve a 35.32 BLEU score. We evaluate prior state-of-the-art CoNaLa models with this additional data and find that our proposed method of using the body and mined data beats the BLEU score of the prior state-of-the-art by 71.96%. Finally, we perform ablations to demonstrate that BART is an unsupervised multimodal learner and examine its extractive behavior.[1]

1 Introduction

The goal of semantic parsing is to translate a Natural Language(NL) utterance to its logical components. There is a large body of research on applying semantic parsing for source code generation in a multitude of domain specific languages such as lambda calculus and SQL (Dahl et al., 1994; Zelle and Mooney, 1996; Zettlemoyer and Collins, 2005; Ling et al., 2016; Xiao et al., 2016; Rabinovich et al., 2017; Dong and Lapata, 2018; Guo et al., 2019; Hwang et al., 2019; Tabassum et al., 2020). However, the task of translating an NL utterance to a general-purpose programming language has proven to be more challenging. A significant issue contributing to this is the difficulty in acquiring quality data due to the necessary domain knowledge needed in the annotation process.

Despite this, the past few years have seen a large number of datasets released for different text-to-code related tasks (Ling et al., 2016; Iyer et al., 2018; Yao et al., 2018; Yu et al., 2018; Lu et al., 2021). Some datasets such as CodeSearchNet (Husain et al., 2019) contain snippets from a multitude of different languages. Others focus on distinct tasks within a specific language, such as JuICe (Agashe et al., 2019), which contains executable Python programming assignments. Utilizing these corpora, prior works (Suhr et al., 2018; Yin and Neubig, 2017, 2018; Sun et al., 2019; Hayati et al., 2018; Yin and Neubig, 2019; Xu et al., 2020; Drain et al., 2021) have found success with a large variety of model architectures. These methods, however, struggle with domain agnostic open-ended code generation in general-purpose languages. One idea to combat this is to utilize large pretrained language models.

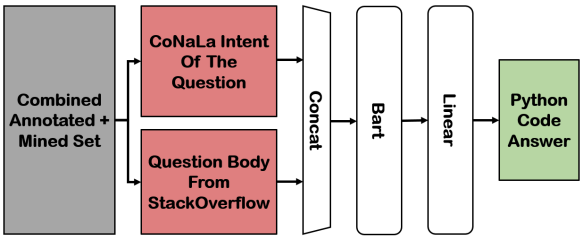


Figure 1: Overview of our approach. From the combined annotated + mined set, we concatenate the intent and question body for inputs to BART(Lewis et al., 2020) and use beam search for generation.

Transformers (Vaswani et al., 2017) have demonstrated that they can both be few-shot (Brown et al., 2020) and unsupervised multitask (Radford et al., 2019) learners. They have been successfully applied to programming language tasks. CodeBERT achieved strong performance on the CodeSearchNet task through pretraining on bimodal NL comment and code pairs(Feng et al., 2020), while Sun et al. (2019) used abstract syntax trees(AST) and transformers to achieve state of the art performance on the HearthStone benchmark(Ling et al., 2016). Roziere et al. (2021) proposed the deobfuscation

[1] https://github.com/gabeorlanski/stackoverflow-encourages-cheating

Proceedings of the 1st Workshop on Natural Language Processing for Programming (NLP4Prog 2021), pages 65–76

pretraining task to incorporate structural features of code into transformer models without the use of ASTs. More recently, Shin et al. (2021) explored the capabilities of large pretrained language models to be few-shot semantic parsers.

Yet open-domain programming question answering on sites such as StackOverflow(SO)[2] has remained an elusive goal. Yin et al. (2018) created an annotated dataset with the site in which the intent and answer snippet pairs were automatically mined from the question. They then had crowd workers rewrite the intents to reflect the corresponding code better. Currently, state-of-the-art was achieved by pretraining an LSTM model on resampled API and mined data (Xu et al., 2020). Subsequent work conducted an empirical study on the effectiveness of using a code generation model in an IDE plugin and find that developers largely had favorable opinions of their experience(Xu et al., 2021). An inherent issue with the approach of Xu et al. (2020), more fundamentally the dataset and parameters of the task, is that the intent can only contain a limited amount of information.

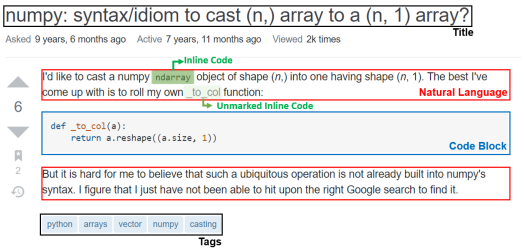


Figure 2: Example StackOverflow question with labeled elements. The corresponding rewritten intent for this question is "add a new axis to array `a`."

Consider the question from Figure 2 in which a valid python snippet could be `a[:, (np.newaxis)]`. Arriving at this answer from the intent "add a new axis to array `a`" requires not only the disambiguation of data types for variable `a`, but also the use of multiple distinct library-specific concepts. Further, this must be accomplished while maintaining syntactically correct code and proper order of arguments. However, neither the original title nor the rewritten intent contains the necessary information to accomplish this task. Although the previous state-of-the-art-model by Xu et al. (2020) uses abstract syntax trees (AST) to guarantee syntactically valid python code, it incorrectly generates `a[(-1),:]=a`. One potential remedy would be to increase the amount of training data, but as discussed previously, getting high-quality annotated code generation data is especially difficult.

Motivated by the limitations to the amount of information a given intent can contain and the substantial difficulty involved with gathering more labeled data, we utilize the multimodal text from the question bodies provided by the StackExchange API[3]. We take advantage of the strong performances of transformer models to beat the previous state-of-the-art by 3.06 BLEU. We ensure a fair comparison by training the models from prior works with the extra data to adequately evaluate our proposed method. When all models are trained with the extra data, using BART beats the previous state of the art by 15.12 BLEU.

Our main contributions are the following:

- Expanding upon the original CoNaLa dataset (Yin et al., 2018) to include the multimodal textual question bodies and thus the pertinent contextual information they contain such as inputs, outputs, and required libraries.
- Demonstrating that BART does not rely on a single modality, but rather achieves its best performance on our dataset when all modalities are included. This indicates at least a shallow understanding of both natural and programming language as well as how they are related in the context of SO questions.
- Conducting experiments revealing that BART's struggle to generate syntacically correct code is likely a result of its tendency to be extractive rather than generative in the task of text-to-code generation.

2 Methodology

As detailed in Figure 1, our overarching approach is to: *(1)* gather textual bodies from SO for both the annotated and mined examples in the CoNaLa corpus, *(2)* use the concatenated intents and question bodies as inputs for a large pretrained language model, and *(3)* use beam search to generate the answer code snippet.

2.1 StackOverflow Data

Every example $e_i \in E$ from the CoNaLa dataset (Yin et al., 2018) is comprised of an intent $x_i \in X$ that concisely summarizes what the poster wants

[2]https://stackoverflow.com/

[3]https://api.stackexchange.com/

and a snippet of Python code $y_i \in Y$ that represents an implementation of x_i. Crowd sourcing was used to rewrite a selection of the mined intents to reflect the snippet better and to ensure that the snippet was indeed a correct answer. As discussed, these intents are limited in the amount of information they can contain. The intent "add a new axis to array `a`" from Figure 2 could refer to a wide variety of different Python objects. It could range from the default `list` to the `Tensor` object from PyTorch[4]. The full question, or either its tags or title, is typically enough for a human to disambiguate the correct library to use. But the annotated intent lacks this crucial information as it is rather difficult to design an annotation task for SO data.[5]

We address this problem directly by using the additional data found in the SO question. In Figure 2 there are four direct mentions of the NumPy library: two in the question body and one each in both the tags and the title. Further, there is a direct mention of the `ndarray` data type from NumPy. It is, therefore, rather intuitive to include this additional data as input with the hope that it improves the answer generation performance. Although we did mention that both the tags and title provide salient information, the focus of this paper is only on using the noisy textual question bodies. Therefore, for every example e_i the inputs now become the concatenation of x_i and the body $q_{x_i} \in Q$ from the original SO question. It is important to note that $|Q| \neq |E|$ as a single question can have many examples while every question is, by definition, unique.

2.2 Unsupervised Modality Learning

Multiple modalities are present in the textual body of a given question. These can range from embedded images to messages from administrators (or upset users) stating that the question is a duplicate of some tangentially related post that does not have an answer. While these are useful to readers, we limit our focus to three modalities: code blocks, inline code, and NL. These modalities are marked in Figure 2 with blue, green, and red, respectively. Ideally, we would prefer to leave in the HTML tags to serve as sentinel tokens, but, looking at Figure 2, one immediately finds that the poster forgot to mark `_to_col` as inline code. Therefore, we remove all HTML tags from the inputs, creating an unsupervised learning environment. Therefore, we propose that a transformer model will learn each of the three modalities and learn to use the relationships between each. We use BART(Lewis et al., 2020) because its pretraining focuses on denoising textual data and, to the best of our knowledge, has minimal exposure to code examples. We used HuggingFace's (Wolf et al., 2020) `BartForConditionalGeneration` model which has a default BART encoder-decoder model with a linear layer and bias for outputs.

2.3 Unlabeled Data

We followed Xu et al. (2020) by using large amounts of the mined but not annotated data. Unlike Xu et al. (2020), however, we do not use this data for pretraining. Instead, we combine this data with the annotated data in our main training and validation sets. By adding more questions to the training set, we directly increase the probability that the model encounters a larger and more representative distribution of libraries. Intuitively, this will reduce the variances between experiments as we have reduced the dependency on the specific examples used in the training and validation sets. This variance reduction is especially useful when working with a small dataset such as CoNaLa.

3 Experiments

3.1 Datasets

CoNaLa (Yin et al., 2018)[6] is an open domain text to code generation task constructed from SO questions. It has 2,879[7] annotated NL-code pairs with more than 590K mined pairs from over 40,000 unique SO questions in the dataset.

StackOverflow Data For every unique question in both the annotated and mined sets, we gather additional data from the StackExchange API. As discussed in subsection 2.1, we only use the question body as input. Therefore the task is to generate a valid answer snippet from both the intent and the textual body. Detailed statistics for this dataset are given in Table 1 and Table 2.

3.2 Methods

We removed 238 (10%) examples from the training set to form the validation set. We then followed Xu et al. (2020) in taking the top mined samples based on their given probability that the NL-Code pair

[4]https://pytorch.org/

[5]We direct readers to Yin et al. (2018) for a full discussion of these challenges.

[6]https://conala-corpus.github.io/

[7]Actual Number is lower due to errors in the dataset preventing the usage of some examples.

Split	$\|E\|^*$	$\|Q\|^*$	$\|E\|/\|Q\|^{①}$	Intent Tokens[②]	Snippet Tokens[②]	Body Tokens[③]
Train	2376	1708	1.39±1.02	16.45±7.51	17.23±8.58	221.90±202.65
Test	498	364	1.37±0.88	15.98±6.62	18.47±12.90	208.04±164.74
Mined-10K	9988[†]	7181	1.39±0.80	11.29±3.94	16.58±9.27	297.53±367.09
Mined	593837	40522	14.65±7.01	11.41±4.22	28.70±42.81	371.24±483.67

Table 1: Statistics for the CoNaLa dataset with data from the StackOverflow API. $|E|$ is # of examples. $|Q|$ number of questions. Values are reported as $\mu \pm \sigma$ unless the column header has *. [①]Mean # of examples for a Question. [②]Per example. [③] Number of tokens in the body regardless of modalitiy. [†]12 of the 10K questions were removed because there was an issue with them.

Split	Have Answer*	Has Code	Inline[①]	Blocks[①]	Code Tokens[①]	NL Tokens[①]
Train	87.88%	85.95%	1.21±2.09	1.42±1.26	95.54±157.52	124.60±92.02
Test	87.09%	87.91%	1.08±1.87	1.50±1.26	88.21±116.01	118.52±79.51
Mined-10K	86.16%	84.00%	1.30±2.36	1.46±1.34	133.20±278.20	164.54±207.08
Mined	81.92%	81.83%	1.50±2.86	1.47±1.44	172.57±372.32	197.98±257.71

Table 2: Detailed statistics for the StackOverflow questions. Mined-10K represents the top 10,000 samples selected from the Mined data based on their probability that they are a valid NL-Code pair. *Percent of questions that have an accepted answer. [①]Per question body.

is valid. However, we only used 10,000 samples rather than the 100,000 Xu et al. (2020) used. From this, we remove 1000 for validation.[8] For all tests of our model with the mined data, we combine the two training and validation sets into one.

Every experiment and test conducted in this work was conducting using Google's Colab Pro service. It afforded us the ability to use 512 input tokens with a batch size of 16. More importantly, we were able to use P100 and V100 graphics cards. Following that, we perform an ablation study using BART and the different components of our approach. Every ablation is run five separate times with different seeds and validation splits. For each test, the model with the lowest validation loss is used in the evaluation. Each test is run for ten epochs as we consistently observed overfitting after five to eight epochs.

Because we introduce new data at inference, we needed to ensure we fairly compare our methods with previous work. To this end, we run the prior works with the question bodies as inputs. However, for testing Xu et al. (2020) with the question bodies, we limited the amount of mined data in pretraining to 10,000 instead of 100,000. This was done due to Google Colab's execution time limits, as it took upwards of four hours for each run of Xu et al. (2020) with only 10,000 samples.

[8]Some questions were deleted from StackOverflow in both the annotated and mined sets, so we could not use those.

3.3 Metrics

We measure the corpus level BLEU score of the generated code snippets with the same postprocessing methods and smoothing as Xu et al. (2020). We evaluate our ablations by comparing the corpus BLEU score and unigram, bigram, and trigram precision. Finally, we calculate the percentage of test examples for which our model generated a syntactically valid Python snippet.

For the previous state-of-the-art, we also report the Oracle BLEU proposed by Yin and Neubig (2019). This is calculated by choosing the candidate snippet s_i with the highest sentence level BLEU score out of n generated snippets. Formally, given the candidate list $C = [c_1, \ldots, c_n]$ and ground truth y_i,

$$z = \underset{c_j \in C}{\text{argmax}}\ \text{BLEU}(c_j, y_i) \quad (1)$$

Furthermore, we want to quantify how much our model relies on the body of the question or "cheats." To do this, we calculate the cheating for the generated snippet $s_i \in [s_1, \ldots, s_N] = S$ and ground truth $y_i \in [y_1, \ldots, y_N] = Y$ with respect to the input text $b_i \in [b_1, \ldots, b_N] = B$. Given the function $m(a, b)$ that calculates a textual similarity metric m, we define the cheating w.r.t. m as

$$C_m(S) = \frac{\sum_{i \in [1;N]} (m(s_i, b_i) - m(y_i, b_i))}{N} \quad (2)$$

If the model is heavily "cheating" from the input, then $m(s_i, b_i) \gg m(y_i, b_i)$, which leads to a large

C_m. The quantity C_m is, by design, similar to a standard mean squared error. The largest difference is that the difference is not squared, to facilitate distinguishing between less and more similar.

For the metric function m, we use BLEU and ROUGE (Lin, 2004). For the former, we take the bigram (C_{BB}) and trigram (C_{BT}) precision from BLEU. For ROUGE, we use bigram ROUGE (ROUGE-2/C_{R2}) and the longest common subsequence (ROUGE-L/C_{RL}). The intuition behind using these metrics is that there is a high probability that unigram precision is large. The answers to a question must address the contents of the said question, leading to shared tokens between inputs and outputs. However, the probability should massively drop when considering multiple grams. Therefore, the similarity between n-grams when $n > 1$ should indicate the reliance on the inputs.

3.4 Implementation

We implemented our model with Python and HuggingFace's transformer library (Wolf et al., 2020)[9]. We used a BART model with a linear layer and a separate bias for text generation. We utilized the smallest available BART model from FAIR, which was the Facebook/BART-base[10]. For training, we again rely on HuggingFace's trainer and their implementation of the learning rate scheduler. We used Adam (Loshchilov and Hutter, 2017) as our optimizer with a learning rate of $5e{-}5$ and a linear learning rate scheduler. We also used a warmup ratio of 0.05. Finally, for generation, we used beam search with four beams, early stopping, and a length penalty of 0.9.

4 Results

We list the previous state-of-the-art BLEU scores for the CoNaLa dataset as well as the performance of our models in Table 3. Using the intent and question bodies achieved a BLEU score of $34.35{\pm}1.01$. This was further increased to $35.32{\pm}0.42$ by including the mined data in the training and validation set. To better understand our model, we perform ablation tests and report their results in Table 4. When comparing our top performance with the previous top performance, regardless of the data used, our model beats the previous state of the art by 3.40 BLEU, a 10.54% increase. Notably, our model outperforms the previous SoTA by 14.78 BLEU, a 71.96% increase when only comparing the experiments with the question body. Furthermore, BART with the mined data and question bodies beats their Oracle BLEU by 1.61 BLEU, translating to a 4.78% increase. However, it is important to note that Xu et al. (2020) outperforms our model by 1.71(5.30%) when we do not use the textual body. But they still both beat the baseline TranX by 25.72% and 7.98%, respectively. The use of the mined data further beat the reranker by 1.46%.

The 71.96% increase is likely because TranX models were never intended to perform well with very noisy data, as evidenced by the 36% dropoff in corpus BLEU when adding the body to Xu et al. (2020). In choosing BART, we intentionally picked a transformer model designed for denoising (Lewis et al., 2020). Further testing is likely needed to determine if our approach is heavily dependent on the underlying transformer, but that is beyond the scope of this paper.

4.1 Impact of adding the Question Body

Adding the body of the question objectively improved the performance of the model. The BLEU score increased 30.92% to 34.35 and, per Table 4, there was an increase across unigram, bigram, and trigram precision. While they all do increase, the amount is far from constant. The unigram precision only saw a 3.61% increase, whereas bigram and trigram precision increased by 12.77% and 22.90%, respectively. This indicates that while the model selected slightly more correct tokens, it greatly improved its ordering of said tokens.

Similar improvements, albeit smaller in value, also occurred when including the mined data without the question bodies. However, there was a sharp drop in the standard deviations for the three precision metrics. In contrast, adding the question body resulted in a steep increase in variance. This is most probably a result of the "shrinking" of the dataset that occurred when we added the bodies. In Table 1 we report that *every* split of the dataset has fewer unique questions than it does examples. Also reported is that the number of tokens in the body is, on average, significantly greater than that of the intents. The effective dataset size is now much smaller, while the number of unique answer snippets stayed the same. The result is that the model now performs better on the difficult test set, at the cost of being more reliant on the training and validation split. Using both the bodies and mined

[9]https://github.com/huggingface/transformers
[10]https://huggingface.co/facebook/bart-base

Model	No Body Corpus BLEU	With Body Corpus BLEU	With Body Oracle BLEU
TranX (Yin and Neubig, 2018)	24.30	18.85±1.26	31.21±0.30
RR (Yin and Neubig, 2019)	30.11	19.85±1.21	31.21±0.30
EK (Xu et al., 2020)	30.69	20.37±1.44	33.71±0.83
EK+RR(Xu et al., 2020)	**32.26**	20.54±0.85	33.71±0.83
BART	26.24±0.31[①]	34.35±1.01	$\geq$ 34.35
BART W/ Mined	30.55±0.38[①]	**35.32**±0.42	$\geq$ 35.32

Table 3: Results compared to previous papers both with and without the use of the question body at inference. We do not calculate the Oracle BLEU for either of our models as our corpus BLEU already surpasses their Oracle BLEU. EK=Using External Knowledge. RR=Using Reranking. [①]Using only the rewritten intent, if available else normal intent, as input.

data does mitigate this "shrinking" effect, as shown by the lower standard deviations than those when only using the body.

4.2 Is BART Reliant on a Single Modality

As discussed in subsection 2.2, we focus on three modalities in the textual bodies: code blocks, inline code, and natural language. We put forth the idea that a large pretrained language model such as BART learns each modality in an unsupervised manner. We designed four distinct ablations to test if this was the case. Each was run both with and without the mined data totaling eight ablations. We report the full BLEU scores from these in Table 4. Further, we calculate the performance with respect to baselines in Table 5. Notably, there was no modality whose removal resulted in a BLEU score worse than when the question body was not used in the input. There was also not a modality whose removal improved performance. From our ablations, it is clear that the most important modality in the question bodies is the code regardless of if it is inline or in a block. But, using only code is still 2.25% worse than when all three modalities are included with mined. This indicates that the NL surrounding acts not only as additional context, but likely further both direct and indirect indicators of salient code for the model.

4.3 Removing Code Improves Syntax

In Table 4 we report the percent of generated snippets that are syntactically valid—adding only the mined data results in a 9% increase. When using the question bodies, the addition of the mined data also increases the percent of valid snippets generated by 7.88%. While it is an improvement, it is still a 3.76% drop from when the body was excluded. Further, removing the code from the bodies resulted in the highest percentages of 92.00% and 84.92% with and without the mined data. We then performed a finer analysis using a single seed and the same training and validation data across all ablations and reported the results in Appendix A. Across all ablations, the majority of errors are caused by mismatches of parentheses. In reality, a large percentage of general syntax errors are likely caused by this. However, syntax errors prevent the extraction of the AST for further investigation of these errors.

We also report in Table 9 the percentage of valid snippets generated when the `print` function is present. One of the more commonly occurring incompatibilities between Python 2 and 3 is that `print` now requires parentheses. Considering that the questions in the CoNaLa dataset are from March 2017 or earlier (Yin et al., 2018) and that support for Python 2.x only ended in January 2020[11], we hypothesize that these deprecated calls are a large cause of the errors. When both the body and snippet have `print`, the inclusion of the question body led to the percent of valid snippets dropping by 21.06 with and 21.05 without the mined data with respect to their baselines. While there are only 19 such questions in the test set, this is a significant drop. The likely cause is that the autoregressive decoder of BART struggles to remember to close the parentheses when wrapping the snippet with a `print` statement. One solution would be to run the `2to3`[12] translator on all of the code. However, the possibilities for code blocks to contain code and other modalities such as error messages and console executions present significant hurdles

[11]https://www.python.org/doc/sunset-python-2/
[12]https://docs.python.org/3/library/2to3.html

Input	BLEU	Unigram*	Bigram*	Trigram*	Valid①
Baseline	26.24±0.31	67.53±0.46	44.10±0.60	29.80±0.69	84.08±1.27
+Mined	30.55±0.38	67.81±0.23	45.55±0.27	31.69±0.37	**93.08**±1.28
Body	34.35±1.01	69.97±0.89	**49.74**±0.99	**36.62**±0.97	81.44±2.25
-NL	34.06±0.48	68.29±0.48	47.91±0.45	35.33±0.40	81.92±0.75
-Code	27.67±0.40	68.29±0.53	44.93±0.57	30.12±0.69	84.92±1.00
-Blocks	29.53±0.47	68.14±0.26	45.69±0.10	31.36±0.15	80.84±1.37
-Inline	33.57±0.94	**70.50**±0.27	49.56±0.40	36.54±0.46	82.16±1.53
Body+Mined	**35.32**±0.42	67.62±0.76	47.69±0.82	35.00±0.87	89.32±1.49
-NL	34.53±0.88	66.24±0.90	46.11±1.15	33.54±1.02	90.08±0.48
-Code	31.39±0.75	67.00±0.75	45.65±0.97	31.60±0.88	92.00±1.31
-Blocks	32.14±0.14	66.96±1.03	45.32±0.97	31.49±0.74	89.24±1.30
-Inline	35.06±0.49	67.04±1.54	46.99±1.29	34.31±1.04	89.20±0.42

Table 4: Ablation Experiments all with BART ran on 5 different random initializations. All tests have rewritten intent as input in addition to the input described in the **Input** column. The **bolded ablation** indicates our best performance while red text represents the worst performance. *Precisions. ①Percent of generated snippets that are valid python.

	Body	Body+Mined
-NL	-0.29	-0.80
-Code	-6.68	-3.93
-Blocks	-4.83	-3.18
-Inline	-0.79	-0.26

Table 5: Change in BLEU score for each ablation versus their respective baseline.

as `2to3` does not support these. Therefore we leave that to future work.

4.4 Cheating

In subsection 3.3 we define the "cheating" equation to measure if the generated snippet is more similar to the question body than the ground truth is. The ideal model would maximize the BLEU score while minimizing the $|C_m|$. We run multiple ablations on a single seed and calculate the "cheating" as defined by Equation 2 and present these results in Table 6.

Suffice to say that *serious* violations of academic integrity have occurred. As expected, the baseline is less similar to the question bodies than the ground truth is. When the body was used as input, C_{BT} increased by 20.28 points, while C_{RL} rose by 3.16 points, representing a 293.49% and 159.60% increase over their respective baselines. Including the mined data resulted in increases of 18.59 (308.13%) and 0.77(265.52%) when compared to using only the intents. Both indicate that the model's generated output has significantly more

	C_{BB}	C_{BT}	C_{R2}	C_{RL}
Baseline	-6.82	-6.91	-1.62	-1.98
+Mined	-6.03	-6.03	-1.41	-0.29
Body	11.65	13.37	1.75	1.18
-Code	-4.18	-5.05	-1.06	-1.40
-NL	10.55	12.27	1.24	0.47
-Blocks	-3.32	-3.48	-0.89	-1.09
-Inline	9.19	10.39	0.90	0.44
Body+Mined	10.19	12.55	1.39	0.48
-Code	-4.37	-5.05	-1.08	-1.47
-NL	9.40	11.32	1.19	0.17
-Blocks	-3.48	-4.16	-0.84	-1.19
-Inline	7.93	9.73	1.11	0.25

Table 6: Cheating Measurements calculated by Equation 2 using a single run but same seed and environment. C_{BB} and C_{BT} are the cheating w.r.t. BLEU Bigram and Trigram Precision. C_{R2} and C_{RL} are the cheating w.r.t. ROUGE-2 and ROUGE-L.

shared multigram subsequences with the question body than the ground truth does. In the ablations where code was removed from the body, C_{BT} increased by only 0.98 and 1.86 with and without the mined data. This represents a percent of increase of only 16.25% and 26.92% over their respective baselines. However, in the case where all NL was removed, C_{BT} increased by 17.35(287.73%) and 19.18(277.57%) points with respect to their baselines. The fact that these increases are lower than that when all modalities are included provides further evidence that BART is an unsupervised mul-

timodal learner and understands the relationships between each modality. The NL likely provides both explicit and implicit hints about the importance of certain code spans.

4.5 Examples

Intent: multiply a matrix ‘p‘ with a 3d tensor ‘t‘ in scipy
✓ `scipy.tensordot(P, T, axes=[1, 1]).swapaxes(0, 1)`
① `np.einsum('...j,...j->...', P, T)`
② `np.einsum('ij->ij->ik->j->ik', p)`
① `P.dot(T).transpose(1, 0, 2)`
Intent: concatenate items of list ‘l‘ with a space ’ ’
✓ `print(' '.join(map(str, l)))`
① `list(map(tuple,[]))`
② `[item for item in L if '' in item]`
③ `print(' '.join(str(x) for x in L))`
Intent: concatenate elements of list ‘b‘ by a colon ":"
✓ `""":""".join(str(x) for x in b)`
① `[' '.join(x) for x in b]`
② `b = [int(i) for i in b]`
③ `print(':'.join(map(str, b))`

Table 7: Example intents and generated snippets. Screenshots of the questions are located in Appendix B and each intent links to the question. Red text indicates that it is incorrect while blue text marks correct tokens in the wrong place. ✓ground truth. ①EK+RR no body (Xu et al., 2020). ②Mined. ③Body+Mined.

We select three examples that demonstrate the benefits of our approach while also highlighting the issues in both the use of the question body and SO corpora in general and report them in Table 7. In the first example, we can see that both ① and ② have learned how to use einsums, but neither is correct. ③ in this case produces an answer that returns the correct value. It is highly probable that BART understood from the poster's explicit mention that `P.dot(T).transpose(1, 0, 2)` gives the desired result and thus extracts it. However, this example has two critical issues: the poster's intent is to find a "cleaner" way to multiply a matrix with a tensor, and `scipy.tensordot` is deprecated. The latter is to be expected, considering the answer is from 2010. But it does indicate that a better evaluation based on inputs and outputs is likely needed.

The next two examples are quite similar but are from two separate questions. ① likely mistakes the core intent to be type conversion due to the inclusion of the words "items" and "with." ② also suffers from the inclusion of these tokens but believes the problem involves filtering. In the final example, ① recognizes that it must convert the items in `b` to `str`, but does not return a joined string. ② recognizes that, again, the answer involves type conversion but predicts the incorrect type.

Similar to the first example, ③ produces answers for both the second and third examples that functionally return the correct results. However, running ③'s solution for the third example would result in a syntax error due to the missing "`)`." On further inspection of the question bodies, it becomes apparent that the probable reason why one snippet is syntactically valid while the other is not is the presence of a Python 2 `print`. The model recognizes that a suitable answer can be found in the question but must be converted to python 3. As discussed in subsection 4.3, these print statements are prone to cause syntactical issues.

5 Conclusion

We expand the CoNaLa dataset by adding the textual question bodies from the StackExchange API and achieve state-of-the-art performance with a simple BART model. Further, we demonstrate that, for this task, BART performs best when code blocks, inline code, and NL are all present. We then examine the impact of the question body on syntax errors and BART's cheating through multimodal understanding. Finally, we examine examples that highlight the issues with both StackOverflow data and code evaluation in general. Future work should focus on extracting desired inputs and outputs for a given intent. Further, additional efforts put into creating corpora of executable code are likely to improve not only generation but evaluation. Both will also protect datasets from deprecated functions and abandoned libraries.

References

Rajas Agashe, Srinivasan Iyer, and Luke Zettlemoyer. 2019. JuICe: A large scale distantly supervised dataset for open domain context-based code generation. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5436–5446, Hong Kong, China. Association for Computational Linguistics.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda

Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners.

Deborah A. Dahl, Madeleine Bates, Michael Brown, William Fisher, Kate Hunicke-Smith, David Pallett, Christine Pao, Alexander Rudnicky, and Elizabeth Shriberg. 1994. Expanding the scope of the ATIS task: The ATIS-3 corpus. In *Human Language Technology: Proceedings of a Workshop held at Plainsboro, New Jersey, March 8-11, 1994*.

Li Dong and Mirella Lapata. 2018. Coarse-to-fine decoding for neural semantic parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 731–742, Melbourne, Australia. Association for Computational Linguistics.

Dawn Drain, Changran Hu, Chen Wu, Mikhail Breslav, and Neel Sundaresan. 2021. Generating code with the help of retrieved template functions and stack overflow answers.

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. CodeBERT: A pre-trained model for programming and natural languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1536–1547, Online. Association for Computational Linguistics.

Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. 2019. Towards complex text-to-SQL in cross-domain database with intermediate representation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4524–4535, Florence, Italy. Association for Computational Linguistics.

Shirley Anugrah Hayati, Raphael Olivier, Pravalika Avvaru, Pengcheng Yin, Anthony Tomasic, and Graham Neubig. 2018. Retrieval-based neural code generation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 925–930, Brussels, Belgium. Association for Computational Linguistics.

Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. 2019. Codesearchnet challenge: Evaluating the state of semantic code search. *arXiv preprint arXiv:1909.09436*.

Wonseok Hwang, Jinyeong Yim, Seunghyun Park, and Minjoon Seo. 2019. A comprehensive exploration on wikisql with table-aware word contextualization.

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. 2018. Mapping language to code in programmatic context. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1643–1652, Brussels, Belgium. Association for Computational Linguistics.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online. Association for Computational Linguistics.

Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain. Association for Computational Linguistics.

Wang Ling, Phil Blunsom, Edward Grefenstette, Karl Moritz Hermann, Tomáš Kočiský, Fumin Wang, and Andrew Senior. 2016. 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. Association for Computational Linguistics.

Ilya Loshchilov and Frank Hutter. 2017. Fixing weight decay regularization in adam. *CoRR*, abs/1711.05101.

Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin B. Clement, Dawn Drain, Daxin Jiang, Duyu Tang, Ge Li, Lidong Zhou, Linjun Shou, Long Zhou, Michele Tufano, Ming Gong, Ming Zhou, Nan Duan, Neel Sundaresan, Shao Kun Deng, Shengyu Fu, and Shujie Liu. 2021. Codexglue: A machine learning benchmark dataset for code understanding and generation. *CoRR*, abs/2102.04664.

Maxim Rabinovich, Mitchell Stern, and Dan Klein. 2017. Abstract syntax networks for code generation and semantic parsing. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1139–1149, Vancouver, Canada. Association for Computational Linguistics.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Baptiste Roziere, Marie-Anne Lachaux, Marc Szafraniec, and Guillaume Lample. 2021. Dobf: A deobfuscation pre-training objective for programming languages. *arXiv preprint arXiv:2102.07492*.

Richard Shin, Christopher H. Lin, Sam Thomson, Charles Chen, Subhro Roy, Emmanouil Antonios Platanios, Adam Pauls, Dan Klein, Jason Eisner, and Benjamin Van Durme. 2021. Constrained language models yield few-shot semantic parsers.

Alane Suhr, Srinivasan Iyer, and Yoav Artzi. 2018. Learning to map context-dependent sentences to executable formal queries. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2238–2249, New Orleans, Louisiana. Association for Computational Linguistics.

Zeyu Sun, Qihao Zhu, Yingfei Xiong, Yican Sun, Lili Mou, and Lu Zhang. 2019. Treegen: A tree-based transformer architecture for code generation. *CoRR*, abs/1911.09983.

Jeniya Tabassum, Mounica Maddela, Wei Xu, and Alan Ritter. 2020. Code and named entity recognition in StackOverflow. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4913–4926, Online. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *arXiv preprint arXiv:1706.03762*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Chunyang Xiao, Marc Dymetman, and Claire Gardent. 2016. Sequence-based structured prediction for semantic parsing. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1341–1350, Berlin, Germany. Association for Computational Linguistics.

Frank F. Xu, Zhengbao Jiang, Pengcheng Yin, Bogdan Vasilescu, and Graham Neubig. 2020. Incorporating external knowledge through pre-training for natural language to code generation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6045–6052, Online. Association for Computational Linguistics.

Frank F Xu, Bogdan Vasilescu, and Graham Neubig. 2021. In-ide code generation from natural language: Promise and challenges. *arXiv preprint arXiv:2101.11149*.

Ziyu Yao, Daniel S. Weld, Wei-Peng Chen, and Huan Sun. 2018. Staqc: A systematically mined question-code dataset from stack overflow. In *Proceedings of the 2018 World Wide Web Conference*, WWW '18, page 1693–1703, Republic and Canton of Geneva, CHE. International World Wide Web Conferences Steering Committee.

Pengcheng Yin, Bowen Deng, Edgar Chen, Bogdan Vasilescu, and Graham Neubig. 2018. Learning to mine aligned code and natural language pairs from stack overflow. In *2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR)*, pages 476–486. IEEE.

Pengcheng Yin and Graham Neubig. 2017. A syntactic neural model for general-purpose code generation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 440–450, Vancouver, Canada. Association for Computational Linguistics.

Pengcheng Yin and Graham Neubig. 2018. TRANX: A transition-based neural abstract syntax parser for semantic parsing and code generation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 7–12, Brussels, Belgium. Association for Computational Linguistics.

Pengcheng Yin and Graham Neubig. 2019. Reranking for neural semantic parsing. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4553–4559, Florence, Italy. Association for Computational Linguistics.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921, Brussels, Belgium. Association for Computational Linguistics.

John M. Zelle and Raymond J. Mooney. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence - Volume 2*, AAAI'96, page 1050–1055. AAAI Press.

Luke S. Zettlemoyer and Michael Collins. 2005. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*, UAI'05, page 658–666, Arlington, Virginia, USA. AUAI Press.

A Error Categories

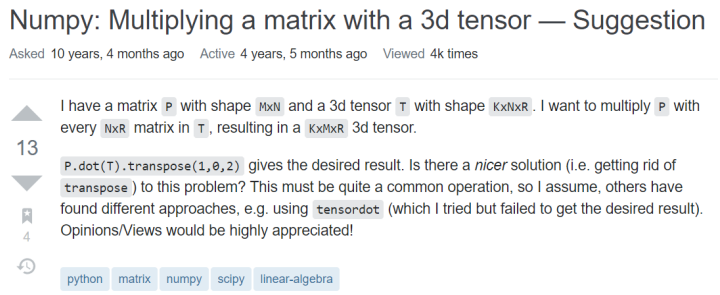
	Error Count	General Invalid Syntax	Paranthesis Matching	Other Matching
Baseline	61	39.34	45.90	14.75
+Mined	38	47.37	39.47	13.16
Body	104	39.42	46.15	14.42
-Code	82	30.49	60.98	8.54
-NL	75	25.33	53.33	21.33
-Blocks	94	36.17	53.19	10.64
-Inline	85	28.24	61.18	10.59
Body+Mined	59	38.98	49.15	11.86
-Code	52	26.92	67.31	5.77
-NL	48	33.33	47.92	18.75
-Blocks	51	23.53	66.67	9.80
-Inline	51	41.18	54.90	3.92

Table 8: Percentages of syntax errors for ablations in a single run.

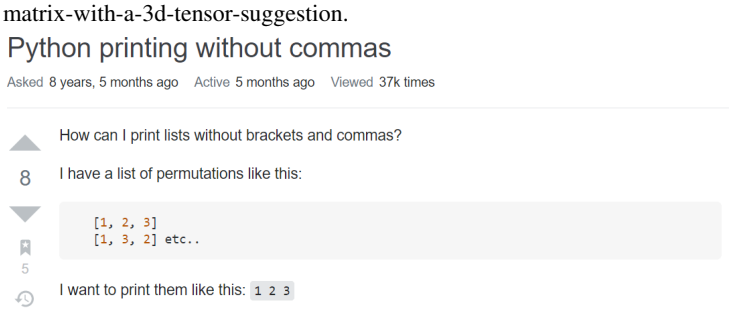
	No Print	Has Print in Snippet	Has Print in Body	Has Print in Both
Baseline	88.28	84.62	86.59	84.21
+Mined	92.97	100.00	91.46	78.95
Body	78.91	84.62	82.93	63.16
-Code	84.38	92.31	80.49	73.68
-NL	84.38	84.62	89.02	78.95
-Blocks	82.29	92.31	76.83	68.42
-Inline	83.07	76.92	86.59	68.42
Body+Mined	90.89	92.31	81.71	57.89
-Code	91.67	69.23	84.15	84.21
-NL	89.84	100.00	91.46	89.47
-Blocks	90.36	92.31	92.68	63.16
-Inline	90.89	92.31	87.80	73.68

Table 9: Percentage of valid snippets based on the presence of `print`.

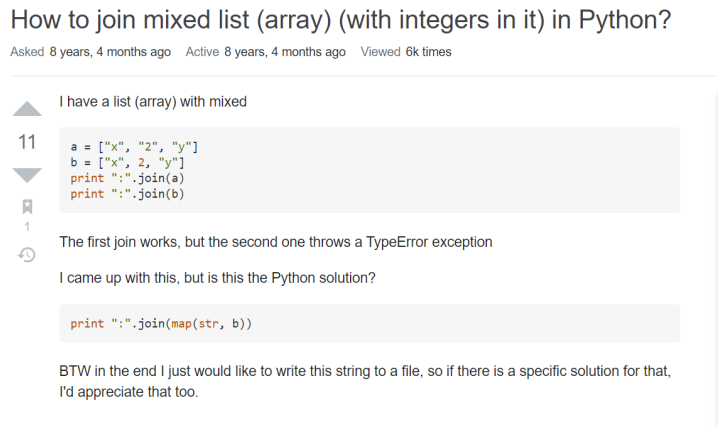
B Full Questions for Examples



(a) Full Stack Overflow Question for Example 1 in Table 7. Question can be found https://stackoverflow.com/questions/4490961/numpy-multiplying-a-matrix-with-a-3d-tensor-suggestion.



(b) Full Stack Overflow Question for Example 2 in Table 7. Question can be found https://stackoverflow.com/questions/13550423/python-printing-without-commas.



(c) Full Stack Overflow Question for Example 3 in Table 7. Question can be found https://stackoverflow.com/questions/13954222/how-to-join-mixed-list-array-with-integers-in-it-in-python.