# A Transformer-based Approach for Translating Natural Language to Bash Commands

Quchen Fu, Zhongwei Teng, Jules White, Douglas C. Schmidt
*Dept. of Computer Science*
*Vanderbilt University*
Nashville, TN, USA
{quchen.fu, zhongwei.teng, jules.white, d.schmidt}@vanderbilt.edu

***Abstract*—This paper explores the translation of natural language into Bash Commands, which developers commonly use to accomplish command-line tasks in a terminal. In our approach a terminal takes a command as a sentence in plain English and translates it into the corresponding string of Bash Commands. The paper analyzes the performance of several architectures on this translation problem using the data from the NLC2CMD competition at the NeurIPS 2020 conference. The approach presented in this paper is the best performing architecture on this problem to date and improves the current state-of-the-art accuracy on this translation task from 13.8% to 53.2%.**

***Index Terms*—Bash Commands Generation, Transformer, Natural Language Processing**

## I. Introduction

Translating natural language into source code for software or scripts can help developers find ways of accomplishing tasks in programming languages they are not familiar with, similar to how help forums like Stack Overflow are used today. As early as 1966, Sammet [1] envisioned a future of automated code generation where people program in their native language. While generating software templates from configuration files is common practice today, the research in translating natural language into programming language code is still relatively nascent.

This paper focuses on the task of translating natural language into the Bash scripting language. Translating natural language into Bash Commands is an example of semantic parsing, which means natural language is translated into logical forms that can be executed [2]. For example, the phrase "how do I compress a directory into a bz2 file" can be translated to the Bash Command: `tar -cjf FILE_NAME PATH`.

In the near term, natural language to Bash Commands translation is unlikely to replace discussion groups or help forums. These translations, however, can provide a quick reference mechanism that may improve on-demand code suggestions and popups generated by integrated development environments (IDEs). This type of AI-based approach complements other prior work, such as SOFix [3], which can fix bugs in code by mining postings in Stack Overflow.

This paper provides the following contributions to the study of translating natural language into Bash Commands:

1) It describes an architecture that improves the state-of-the-art performance on translating national language to Bash Commands from 13.8% to 53.2%,
2) It presents results demonstrating that the Transformer model [4] is the current best-performing architecture,
3) It shows how parameter values can be modified to shrink vocabulary size by 90% and improve accuracy, and
4) It explores the use of Beam Search to provide multiple potential translations and adapt the beam score to create heuristic weights that improved the accuracy by 1.2%.

The remainder of this paper is organized as follows: Section II summarizes recent development in semantic parsing; Section III describes the overall structure of our model pipeline; Section IV analyzes the performance of different models and training techniques, as well as metric and error analysis for our SOTA model; and Section V presents concluding remarks.

## II. Background on Translation Models

The Natural Language to Command (NLC2CMD) competition [5] was launched at the NeurIPS 2020 conference to foster improvements in translating natural language to Bash Commands. The competition challenged teams to build models that could transform descriptions of command-line tasks in English to their Bash syntax. NLC2CMD is an updated challenge based on the NL2Bash [6] dataset, in which the NL2Bash is used as the public training dataset and hidden validation/test data are provided by IBM [5].

The best results in prior work on the problem of translating natural language to Bash Commands were produced by Tellina [7]. Tellina used the Gated Recurrent Unit Network (GRU) [8], which is a recurrent neural network (RNN) that achieved 13.8% accuracy on the NLC2CMD metrics proposed by IBM [5]. The Tellina [7] paper produced the NL2Bash [6] dataset and new semantic parsing methods that set the baseline for mapping English sentences to Bash Commands.

Transformer models generally have better accuracy and faster training times [4] than RNNs [9] on machine translation tasks. Prior research on machine translation has primarily investigated a single architecture, Gated Recurrent Unit Network (GRU), for translating natural language to Bash Commands. This paper enhances prior research by exploring the performance of several architectures on the NLC2CMD dataset.

## III. Application Architecture

We tested several data processing, architecture, and post-processing strategies on the problem of translating natural language to Bash commands. Although this model will likely be improved by subsequent work, it provides a starting point for researchers focusing on natural language to Bash Command translation. In particular, our results show that the Transformer model is a robust foundation for future research in this area.

Bash Commands can be complex and nested, as shown in Figure 1. This structure helps explain why programmers may find it hard to create—or even comprehend—Bash Commands, thereby motivating the need for a customizable parser. We built our parser atop the Tellina [7] parser that was developed based on Bashlex [10] in prior work. This parser can parse a Bash Command into an abstract syntax tree (AST) that consists of utility nodes, each of which may contain multiple corresponding flags and parameters. During the tokenization stage, utilities, and flags are kept "as is" and parameters are categorized and replaced with `_NUMBER`, `_PATH`, `_FILE`, `_DIRECTORY`, `_DATETIME`, `_PERMISSION`, `_TIMESPAN`, `_SIZE`, with the default option of `_REGEX`. Natural language sentences are pre-processed by filtering out the stop words. The remaining words are then decapitalized and lemmatized to create a relatively smaller word vocab.

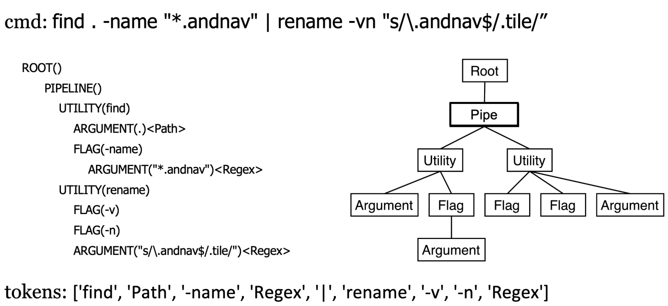


Fig. 1. Visualization of the Tokenization Process

Our application approach uses Transformers for both the encoder and the decoder. The encoder and decoder each consist of six layers. The model is trained for 2,500 steps and uses an ensemble of the four top-performing single models. Since the focus is on training an efficient and robust model that can be deployed easily, the need to modify the network structure was relatively low. We therefore chose to implement our Transformer model with OpenNMT [11], which is an open-source neural sequence learning framework.

We found the Transformer model is (1) sensitive to learning rate and that (2) larger batch sizes will produce better results. The detailed training hyperparameters are available on our GitHub repository[1]. We trained our model on 2 Nvidia 2080 Ti Graphic cards with 64GB memory. Our model achieved 53.2% accuracy on the hidden test dataset for the NLC2CMD competition and had the top performance in both inference time and energy consumption.

## IV. Improving Natural Language to Bash Translation

This section investigates specific research questions and provides empirically grounded answers that helped guide the design of our architecture. Building a natural language to Bash translation model involves three phases: (1) pre-processing of the training data, (2) selecting the best model architecture for the task, and (3) devising an effective approach for determining which of many possible translations should be presented to the user. Here we discuss key research findings in each of these areas and provide information that should aid other researchers in developing more accurate methods of translating natural language to Bash Commands.

For our investigation we utilized the NLC2CMD dataset, which contains 10,347 pairs of English sentences and their corresponding Bash Commands. Of the 10,347 pairs of data, 29 were not syntactically correct Bash Commands and were therefore excluded. The size of this public dataset was relatively small in the natural language processing research field and the goal for data processing was to create a small word vocabulary and utilize as much data as possible [12].

The ideal metric for an evaluation would check if the predicted Bash Command produces the same result as the reference answer. That metric is not practical, however, since establishing a simulated environment for 10K variant situations is beyond the scope of this paper. Instead, our scoring mechanism specifically checks for structural and syntactic correctness that "incentivizes precision and recall of the correct utility and its flags, weighted by the reported confidence" [5]. The metric first define two terms: Flag score $S_F^i$ and Utility score $S_U^i$.

The flag score is defined as twice the intersection of reference flags and predicted flags number minus the union, scaled by the max number of either reference flags or predicted flags (Equation 1 [5]). The range of flag scores is between -1 and 1.

$$S_F^i(F_{\text{pred}}, F_{\text{ref}}) = \frac{1}{N}\Big(2 \times |F_{\text{pred}} \cap F_{\text{ref}}| - |F_{\text{pred}} \cup F_{\text{ref}}|\Big) \quad (1)$$

Utility score is defined as the number of correct reference utilities scaled by capping the flag score between 0 and 1, minus the number of wrong utilities, scaled by the max number of either reference utilities or predicted utilities (Equation 2 [5]). By summing all the utility scores within a predicted command, the range of normalized utility scores is between -1 and 1.

$$S_U = \sum_{i \in [1,T]} \frac{1}{T} \times \left( |U_{\text{pred}} = U_{\text{ref}}| \times \frac{1}{2}\Big(1 + S_F^i\Big) - |U_{\text{pred}} \neq U_{\text{ref}}| \right) \quad (2)$$

**Comparison of Transformer, RNN/BRNN, and their combination architectures in terms of accuracy and training, inference time.** Since the literature published on translating natural language to Bash Commands is sparse, an important concern is identifying which architectures perform best. In particular, Sequence-to-Sequence [13] models have been studied extensively in the context of translations, so we explored their performance on this particular task. These models consist of two main components: an encoder and a decoder. The encoder turns the inputs into vectors and the decoder reverses the process. We compared different combinations of encoder-decoder layers, including RNN, Bidirectional RNN

[1]github.com/magnumresearchgroup/Magnum-NLC2CMD

(BRNN), and Transformer, to translate natural language to Bash Commands.

Chen *et al.* [14] discovered that (1) Transformer quality gains stemmed mostly from the Transformer encoder and (2) RNN decoders often have faster inference times. We therefore mixed and tested different combinations of encoder and decoder types. Table I summarizes the performance comparison (measured in seconds) between different model structures.

TABLE I
MODEL PERFORMANCE COMPARISON

| Encoder | Decoder | Accu. (Para) | Accu. | Train | Inference |
|---|---|---|---|---|---|
| Trans. | Trans. | 0.509 | **0.522*** | 1625 | 0.126 |
| Trans. | RNN | - | - | - | - |
| RNN | Trans. | 0.448 | 0.486 | 1490 | 0.116 |
| RNN | RNN | 0.151 | 0.336 | **1151*** | 0.069 |
| BRNN | Trans. | 0.483 | 0.495 | 1411 | 0.120 |
| BRNN | RNN | 0.301 | 0.476 | 1218 | **0.065*** |

The results shown in Table I indicate that using the Transformer as both an encoder and decoder has the best accuracy in this particular case.[2] Likewise, the model with an RNN as the decoder can reduce inference time by 50%.

The NLC2CMD competition brought together a widely diverse set of initial approaches. We entered our architecture into the NLC2CMD competition and achieved the highest accuracy with one of the lowest energy and latency scores of all techniques. The complete results for the competition are shown in Table II.

TABLE II
THE NLC2CMD COMPETITION RESULTS

| Team | Model | Accuracy | Power | Latency |
|---|---|---|---|---|
| Magnum | (this paper) | **0.532*** | 682.3 | 0.709 |
| Hubris | GPT-2 | 0.513 | 809.6 | 14.87 |
| Jb | Clas.+Trans. | 0.499 | 828.9 | 3.142 |
| AICore | Two-stage Trans. | 0.489 | **596.9*** | 0.423 |
| Tellina [7] | BRNN (GRU) | 0.138 | 916.1 | 3.242 |

**Effect of parameters masking on vocabulary size and translation accuracy.** Since obtaining training data of paired English and Bash Commands is hard, the model may not be able to learn the entire vocabulary that it must translate to or from. Finding ways of reducing vocabulary size is thus essential to develop more accurate models.

Bash Commands typically consist of three terms: (1) utilities that specify the main goals of the command (*e.g.*, `ls`), (2) flags that provide metadata regarding command execution (*e.g.*, `-verbose`), and (3) parameters that specify directories, strings, or other values that the command should operate on. Each utility has a bounded number of flags that can be passed to it. In contrast, parameters have a much larger range of values. Training examples for translating natural language to Bash Commands provide values for the parameters, which can vary significantly between translated examples of the same command.

[2]The OpenNMT [11] framework currently does not support a Transformer encoder + RNN decoder.

We hypothesized that including the actual parameter values (such as `ls /var/www` and `ls /etc`) from the training examples would greatly increase the overall vocabulary size and decrease model accuracy. Our rationale for this hypothesis was that there were few limited paired examples and translation models typically perform worse with large vocabulary sizes and limited training data.

$$\underbrace{\text{split}}_{\text{utility}}\quad \underbrace{\text{-n}}_{\text{flag}}\quad \underbrace{\text{1000}}_{\text{parameter}}\quad \underbrace{\text{/usr/bin/gcc}}_{\text{parameter}}$$

split -n _NUMBER _FILE

Fig. 2. Example of a Bash Command

To test this hypothesis, we used the English and Bash tokenizers from the Tellina model [7] with our modification. As shown in Figure 2, Bash tokens can be categorized as utilities, flags, and parameters (*i.e.*, arguments, such as a specific path). The English tokenizer decapitalized all the letters and replaced parameters with generic forms. The Bash tokenizer parsed Commands into syntax trees with each element labeled as utility, flag, or parameter.

The accuracy metric focused mainly on the structure and syntactic correctness of the Bash command. We therefore replaced all the parameters in Bash with their corresponding generic representations. For example, a folder path like `tmp/bin` is replaced with `PATH`. By applying this transformation, the Bash vocabulary size was reduced from 8,184 to 776 tokens, and the accuracy of the Transformer models we tested increased by 1.3%. As shown in Table I, we achieved accuracy and performance increases across all architectures, especially for the ones with lower accuracy.

**Utilizing Beam scores for balancing between exploration gains and minimizing scoring penalties.** To better utilize the Bash suggestion ability of the translation model, users were provided with five Bash translations for each English sentence, ranked by confidence from high to low. One way to produce multiple translations is to enable Beam Search [15], which creates a tree structure exploration space and evaluates the probability of words at each step. This scoring mechanism thus allowed for five predictions for each invocation (English sentence) and expected confidence weights at each position.

We explored the impact of Beam Search on translation accuracy. When Beam Search is enabled, the Transformer model produced a negative beam score for each corresponding prediction. We observed a strong correlation between a high score and a correct prediction by mapping the exponential of the beam score to the range 0-1. Figure 3 shows the confidence score distribution contrasted with correct and incorrect predictions. We also found the translation quality of the same sentence between Beam Searches tended to have a strong correlation, which means Beam Search alone is an insufficient solution for five Bash translations.

When the first prediction in our tests got a negative score, only 9.2% of the time did the other four predictions get any positive score. Based on our observations, we hypothesized

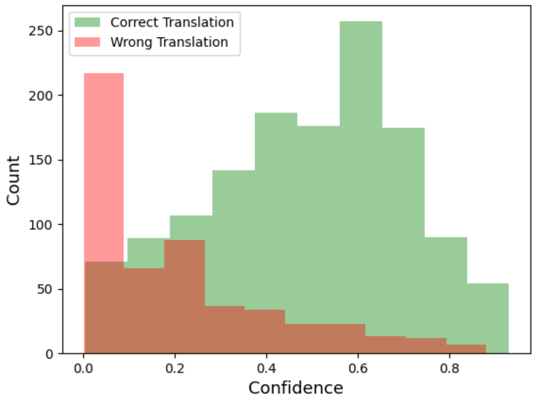


Fig. 3. Histogram of Confidence Score Distribution

that when the first result from Beam Search led to a wrong prediction, the rest of the predictions would also likely be incorrect. A better solution would thus involve ensembling different models to decrease the bias for the five predictions.

The metric used to measure the "accuracy" of predictions is critical in any deep learning model. A key question is therefore how to communicate the expected "quality" or "accuracy" of a translation to a user. The goal is to ensure they see a wide range of possible translations, but also understand which translations the model are more confident about.

For each English sentence, five translations to Bash Commands are produced to better assist users by providing multiple choices. The metric for evaluating the final (*i.e.*, all five) accuracy can be summarized as the following [5]: If any of the predictions have a positive score, take the max among the five, otherwise take the average:

if $\exists_{p \in A(nlc)}$ such that $S(p) > 0$,

$$\text{Score}(A(nlc)) = \max_{p \in A(nlc)} S(p) \tag{3}$$

otherwise,

$$\text{Score}(A(nlc)) = \frac{1}{|A(nlc)|} \sum_{p \in A(nlc)} S(p) \tag{4}$$

Considering the scoring mechanism that every predicted result contributes to model performance evenly, we hypothesized that using 1.0 as the first confidence weight and using Beam Search to produce the other four prediction weights would improve model performance. The evaluation score increased accordingly. In contrast, the improvement vanishes if completely neglecting other predictions when the first prediction is wrong. To achieve the balance between exploration and minimize punishment, we capped our confidence score empirically with the following equation:

$$Confidence_i = \left(e_i^{\text{BeamScore}}\right)/2, 2 \leq i \leq 5 \tag{5}$$

### A. Error Analysis

Previous research [6] listed sparse training data, utility errors, and flag errors as the top three causes of wrong predictions. Since sparse training data is a subjective metric, we only analyze the incorrect utility and flag predictions. We used a separate, independently created testing dataset of 1,867 samples (previous work manually analyzed 100 samples from the dev dataset collected the same way as the training dataset), and evaluated the accuracy results in more detail.

For all the test Bash translation items, 52.2% were translated correctly, 33.9% were translated to the wrong utility, and 14.0% had flag errors in the translation. More than two-thirds of all errors are utility errors, meaning the variety of flags is less significant than having enough data for each utility.

## V. Concluding Remarks

This paper evaluated various deep learning approaches to translating natural language into Commands in the Bash scripting language and presented the highest performing model published to date on the NLC2CMD dataset. Our results showed that Transformer-based models considerably outperform the RNN-based models in the English-to-Bash translation task. Word vocabulary size and whether to post-process translations directly affected the results.

## References

[1] J. Sammet, "The use of english as a programming language," *Commun. ACM*, vol. 9, pp. 228–230, 1966.

[2] J. Berant, A. Chou, R. Frostig, and P. Liang, "Semantic parsing on freebase from question-answer pairs," in *EMNLP*, 2013.

[3] X. Liu and H. Zhong, "Mining stackoverflow for program repair," *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pp. 118–129, 2018.

[4] I. Caswell and B. Liang, "Recent advances in google translate," 2020. [Online]. Available: https://ai.googleblog.com/2020/06/recent-advances-in-google-translate.html

[5] M. Agarwal, T. Chakraborti, Q. Fu, D. Gros, X. V. Lin, J. Maene, K. Talamadupula, Z. Teng, and J. White, "Neurips 2020 nlc2cmd competition: Translating natural language to bash commands," *ArXiv*, vol. abs/2103.02523, 2021.

[6] X. V. Lin, C. Wang, L. Zettlemoyer, and M. D. Ernst, "Nl2bash: A corpus and semantic parser for natural language interface to the linux operating system," *ArXiv*, vol. abs/1802.08979, 2018.

[7] X. V. Lin, "Program synthesis from natural language using recurrent neural networks," 2017.

[8] J. Chung, Çaglar Gülçehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *ArXiv*, vol. abs/1412.3555, 2014.

[9] T. Mikolov, M. Karafiat, L. Burget, J. Cernocky, and S. Khudanpur, "Recurrent neural network based language model," in *INTERSPEECH*, 2010.

[10] I. Kamara, "Bashlex," 2014. [Online]. Available: https://github.com/idank/bashlex

[11] G. Klein, Y. Kim, Y. Deng, J. Senellart, and A. M. Rush, "Opennmt: Open-source toolkit for neural machine translation," *ArXiv*, vol. abs/1701.02810, 2017.

[12] B. Ahmadnia, P. Kordjamshidi, and G. Haffari, "Neural machine translation advised by statistical machine translation: The case of farsi-spanish bilingually low-resource scenario," *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pp. 1209–1213, 2018.

[13] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *NIPS*, 2014.

[14] M. Chen, O. Firat, A. Bapna, M. Johnson, W. Macherey, G. Foster, L. Jones, N. Parmar, M. Schuster, Z.-F. Chen, Y. Wu, and M. Hughes, "The best of both worlds: Combining recent advances in neural machine translation," in *ACL*, 2018.

[15] S. Wiseman and A. M. Rush, "Sequence-to-sequence learning as beam-search optimization," in *EMNLP*, 2016.