

Oisin Doherty (oisind),
Abhinav Gottumukkala (anak4569),
Hans Jorgensen (thehans),
Lauren Martini (lmartini)

NL2Bash: Improving Bash Translation Accuracy Through Manual Verification

<https://github.com/oisindoherty/nl2bash>

1. Objective

We propose to increase the accuracy of the English to Bash translator, Tellina, by gathering new training data and designing a validation system to improve the quality of both new and existing data. Because Tellina is built upon machine learning, gathering more data and improving data quality should improve overall accuracy.

2. Background and Motivation

Bash is the de facto scripting language for interacting with any UNIX-based operating system, but because of its obscure commands, can be a difficult language to learn. Both new and veteran developers have experienced the confusion that comes from searching through numerous resources for Bash commands online. Even tasks that can be described simply in English, such as, “Count all the lines of php code in a directory” may require the use of several different Bash commands where each individual command may not be intuitive [1]. For such a query, one command that would work is:

```
find . -name '*.php' | xargs wc -l
```

For users unfamiliar with Bash, this is a difficult command to understand. The user would have to understand how to use the `find`, `wc`, and `xargs` commands, and the appropriate flags to count only php files and count lines. They would also have to know how to redirect output with the pipe operator. Especially for a beginner, the amount of knowledge required to use each command may make even simple operations difficult.

There is a fair amount of research on users understand Bash more intuitively, but few papers go on to create a working tool to address the issue. A well implemented translator, able to translate a valid natural language command into a Bash command, is one potential such tool. Instead of obtaining questionably correct translations from an online search, a natural language to Bash translator would provide users a quick solution, allowing them to simply describe the command they want. Individuals learning Bash would thus have an easy way to look up the commands required to complete a given task, greatly improving the rate at which they encounter and learn new commands.

Unfortunately, current efforts to develop natural language-to-Bash translators have not yet lived up to their full potential [2]. The translator examined for this project is Tellina, a natural language programming model available as an inbrowser tool [3]. Tellina provides mixed results, producing correct or nearly correct Bash commands for some English phrases, while producing completely incorrect commands for others.

Sometimes, Tellina produces output that indeed matches the intention of the natural language command:

Input: "find all pdf files in this directory and its subdirectories"
Output: `find . -name "*.pdf" -print`

Other times, however, Tellina produces only tangentially related commands with different functionality:

Input: "remove the first line from each text file in this directory"
Output: `find . -type f -exec grep California {} \; -exec rm {} \;`
Actual Function: "print all lines containing 'California' in all files in this directory, then remove the files"

Input: "view file.txt"
Output: `rev file.txt`
Actual Function: "view the contents of file.txt, but with each line reversed character-wise"

Tellina uses a neural network to learn from a dataset consisting of pairs of matching natural language and Bash commands. Based on our prior knowledge of machine learning, we expect that augmenting and verifying the current data set will increase Tellina's accuracy. A translation is defined to be accurate if the Bash command performs the task described in the English description.

Most of the current dataset used by Tellina was originally collected by freelancers who were hired to write down multiple permutations of various Bash coreutils, and write corresponding English descriptions [4 : Footnote 7]. While the data collected from this method is valuable, it is limited in usefulness, since these permutations are artificial and may not reflect real-world usage of the commands. Websites such as Stack Overflow [5], on the other hand, provide examples of real world usage of Bash commands that can be used to improve Tellina's dataset. The Tellina repository does have a data dump from Stack Overflow [6] but does not use it, nor does it include an automated way to gather more data. In addition, while the training data also has cleaning and filtering scripts in place to remove English-Bash command pairs with syntactically incorrect commands, it still has significant inaccuracies [4 : Appendix B]. Cleaning the existing data set gathered by the Tellina team is another way to increase accuracy.

3. Approach

3.1 Existing System Architecture

The existing Tellina system contains a dataset (the 12,606 Bash one-liners collected from Stack Overflow and through manual entry), TensorFlow neural-networks for translating English into Bash, and some extra utilities such as a Bash parser (which creates an AST) and a regex-based sentence tokenizer and an entity recognizer for Bash [7]. Aside from small bug fixes in the evaluation and table producing scripts, the existing code relating to training and displaying NL2Bash models remained unchanged.

3.2 Architecture and Implementation

There are two main facets to our approach in creating a better data set for Tellina: automated data collection and data verification. First, we have implemented a web-scraping tool to search Stack Overflow for Bash-related questions and answers to augment the existing data set. Second, we provide a method to present English-Bash command pairs from the original dataset to testers for verification.

The Tellina team initially verified their data by paying freelancers with experience in Bash to manually verify commands for syntax and accuracy. A large set of Bash commands and their natural language descriptions were provided by the freelancers from sites such as Stack Overflow and man pages for individual commands [8]. The original Tellina paper discussed how the training data from this approach was limited both in breadth of arguments and quantity of commands, and so they attempted to automate its collection. The original Tellina group attempted to parse natural language command pairs without manual verification by downloading a large quantity of Stack Overflow posts, parsing through the site's history to find relevant threads, and finding relevant pairs of Bash commands and their natural language descriptions [4 : Section 7, Collecting Methodology]. While we initially considered utilizing the Stack Overflow database archives, we found that the HTML contains specific classes for div elements that specify the languages that code segments are written in, whereas the Stack Overflow dump does not. Although both approaches were similar in complexity, the additional context given by the HTML div classes was enough of a benefit for us to decide to scrape Stack Overflow manually, since the data scraped this way would be more likely to be relevant in English to Bash translation.

In addition, we have created an interface used for manual verification to increase the quality of our data, both the data scraped from Stack Overflow and the existing English-Bash command pairs in the Tellina dataset. As described in Section 2, our ideal outcome would be to increase both the quality and quantity of our training data over the original Tellina dataset.

3.2.1 Architecture Diagram

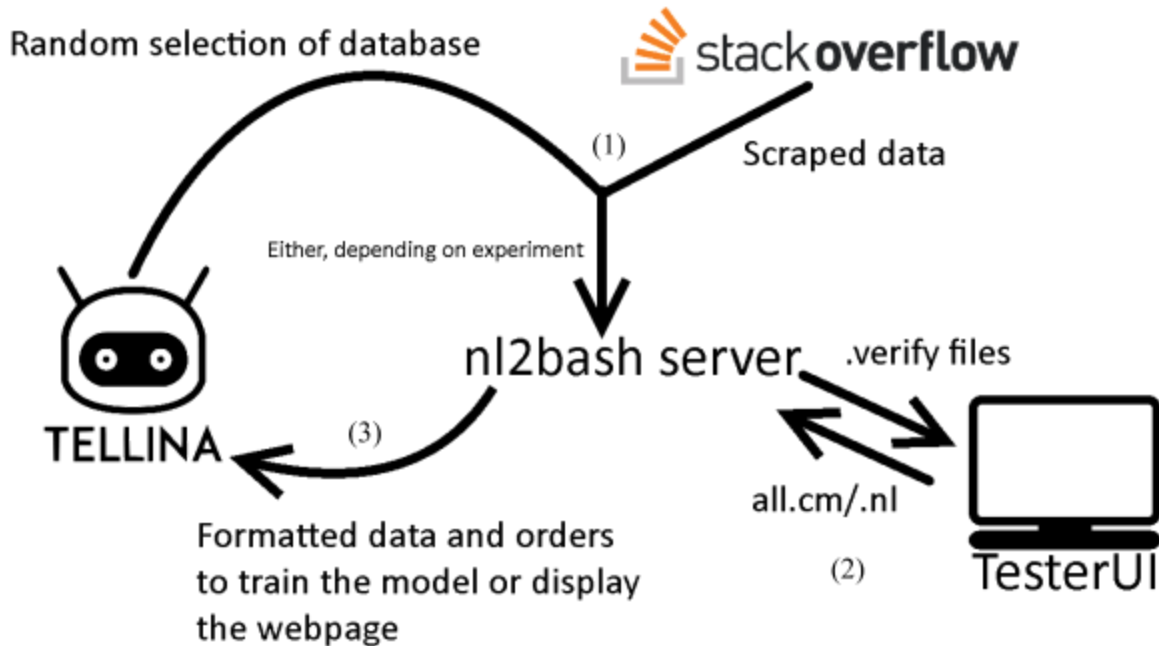


Figure 1. Architecture Diagram

Seen in Figure 1, the major components of our architecture are responsible for scraping new commands, verifying them with testers, and sending them to Tellina, respectively labeled (1), (2), and (3) in the above diagram.

Labeled (1), the process begins with the scraping and parsing of data from Stack Overflow (detailed in 3.2.2). This data is intermittently scraped and formatted by our JSoup Web scraper and stored in a sqlite3 database. Potential English-Bash command pairs are distributed to testers (personnel assigned to verify English-Bash command pairs) who interface with the database through a system we will refer to as TesterUI (detailed in 3.3). Each of these client applications pulls unverified questions individually, so that each tester verifies different questions and all testers can do meaningful work simultaneously. As the tester verifies or rejects each potential pair, the verified commands and questions are sent back to the main server. This process of the main server sending unverified data and receiving verified data is shown in the diagram as label (2). Finally, once the verification has been agreed upon by a sufficient number of testers, this verified and formatted data is ultimately integrated into the Tellina database, with this data set transfer shown in the diagram as label (3). As the database is updated, Tellina is periodically retrained, and its accuracy is re-evaluated. We then directly pull the evaluation results from the Tellina server.

These components are implemented across three main machines. One machine is tasked with periodically scraping Stack Overflow data and preparing it for transfer. Another machine was tasked with storing verified and unverified command pairs, and serving these commands to testers to be verified. The third

server runs Tellina itself, hosting the final data and retraining the translation models. Our current implementation for our web scraper checks for highly voted Stack Overflow questions with certain tags, as detailed in the section 3.2.2. Our scraper updates the page and parses any questions that have been asked since the last time the tool checked. Focusing primarily on Stack Overflow allows us to both obtain data from multiple contexts (tutorials, question threads, documentation, commented shell scripts, etc) and expand our training data beyond the current Tellina database. At this point, the data is stored in a database on our central server, where it can be served to testers. Ultimately, this means that our central server scrapes data, supplies it to testers, and places their responses into a database so that Tellina can interact with it.

3.2.2 Data Collection

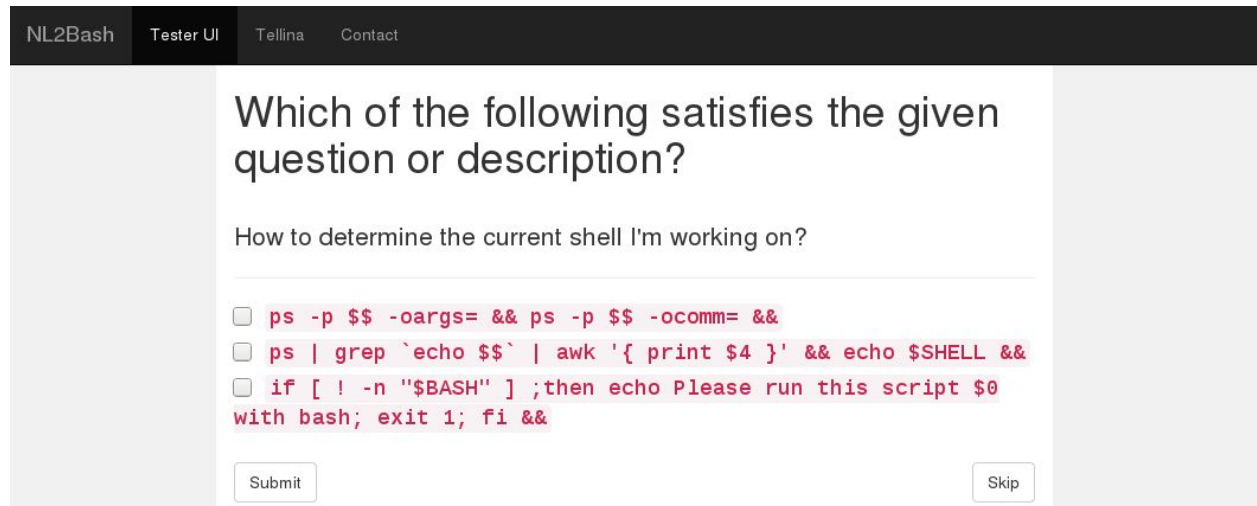
Our web scraper scrapes Stack Overflow by submitting a query for all questions marked with either the ‘shell’ or ‘bash’ tags [9]. From the resulting page, the scraper scrapes all URLs that link to valid Stack Overflow question pages. For each of these question pages, the scraper generates a JSON file that contains the question asked by the original poster and an array of strings that hold the commands listed in the top five answers. We refer to these generated files as .verify files, as they are output into the NL2BashWebScraper directory, which our verification server pulls from to serve to verifiers. When a verifier requests a new verification, the verification server chooses a .verify file and serves it to the verifier. These .verify files are named with the question ID cached by the web server to ensure that we serve each .verify file exactly once.

For each answer on Stack Overflow question pages, all commands found in the answer are wrapped in a HTML div denoting that it contains Bash commands. This tagging is based on heuristics implemented by Stack Overflow from the Markdown content of the original posts and thus contains some false positives (such as command output), but because these .verify files are subsequently subjected to manual verification, this does not matter. The choice to use commands from the top five answers was arbitrary, but our initial analysis of Stack Overflow questions found that answers other than the first often provided alternative accurate commands, though past the fifth answer, there are rarely unique or accurate commands. Earlier iterations of the scraper included each individual line of code from the top five answers as an individual command to verify, but many questions required the user to verify an unreasonable number of lines (more than 20, especially for longer answers.) Considering this, we found that even though some commands that are technically correct but contain console output or comments will be marked as inaccurate, keeping our interface both simple and usable was our main concern.

Because there are so many questions that fit our scraper’s parameters (25,000 at the time of writing), we implemented a heuristic into the scraper that determines whether or not a question should be scraped in the future. It is unnecessary to scrape questions that have been thoroughly vetted by the Stack Overflow community more than once. The two metrics we found most effective for identifying such questions were whether or not the question had an accepted answer and the time since last activity on the question. Questions that have a verified answer tend to indicate that the original poster found the commands present in the answer were accurate (for their purposes). Newer questions that have verified answers can still be scraped in the future as users find the question and add additional answers to the question; consequently,

scraping such questions can lead to us finding more commands to present for verification. These features are both factored into our heuristic, and has allowed us to drastically reduce the amount of redundant scraping done on each execution of the web scraper.

3.3 Tester Interface



The screenshot shows a web interface for 'NL2Bash'. The top navigation bar includes links for 'NL2Bash', 'Tester UI', 'Tellina', and 'Contact'. The main content area has a heading 'Which of the following satisfies the given question or description?' followed by the question 'How to determine the current shell I'm working on?'. Below the question are three checkboxes, each followed by a Bash command snippet. The first command is 'ps -p \$\$ -oargs= && ps -p \$\$ -ocomm= &&'. The second command is 'ps | grep `echo \$\$` | awk '{ print \$4 }' && echo \$SHELL &&'. The third command is 'if [! -n "\$BASH"] ;then echo Please run this script \$0 with bash; exit 1; fi &&'. At the bottom of the form are two buttons: 'Submit' and 'Skip'.

NL2Bash Tester UI Tellina Contact

Which of the following satisfies the given question or description?

How to determine the current shell I'm working on?

☐ `ps -p $$ -oargs= && ps -p $$ -ocomm= &&`

☐ `ps | grep `echo $$` | awk '{ print $4 }' && echo $SHELL &&`

☐ `if [! -n "$BASH"] ;then echo Please run this script $0 with bash; exit 1; fi &&`

Submit Skip

Figure 2. Verification Website Interface

The only part of our project that will be facing the testers is the verification website. When a tester visits the website, a new set of questions will be downloaded to present to the tester for verification. In the case of a Stack Overflow verification, the tester is presented two major displays; one with the question asked by the Stack Overflow user and one with several of the answers provided. There are checkboxes next to each of the possible valid Bash commands that the tester may check to indicate that the command is accurate. When the tester has selected the answers they feel best satisfy the question (if any), they click the “Submit” button at the bottom of the screen to pull up a new question to be verified. The tester does not have a set number of questions to answer and is free to close the program at any time. We believe that this webpage interface is simple enough to allow a wide variety of testers to verify these commands while still producing meaningful results.

3.4 Tellina Website

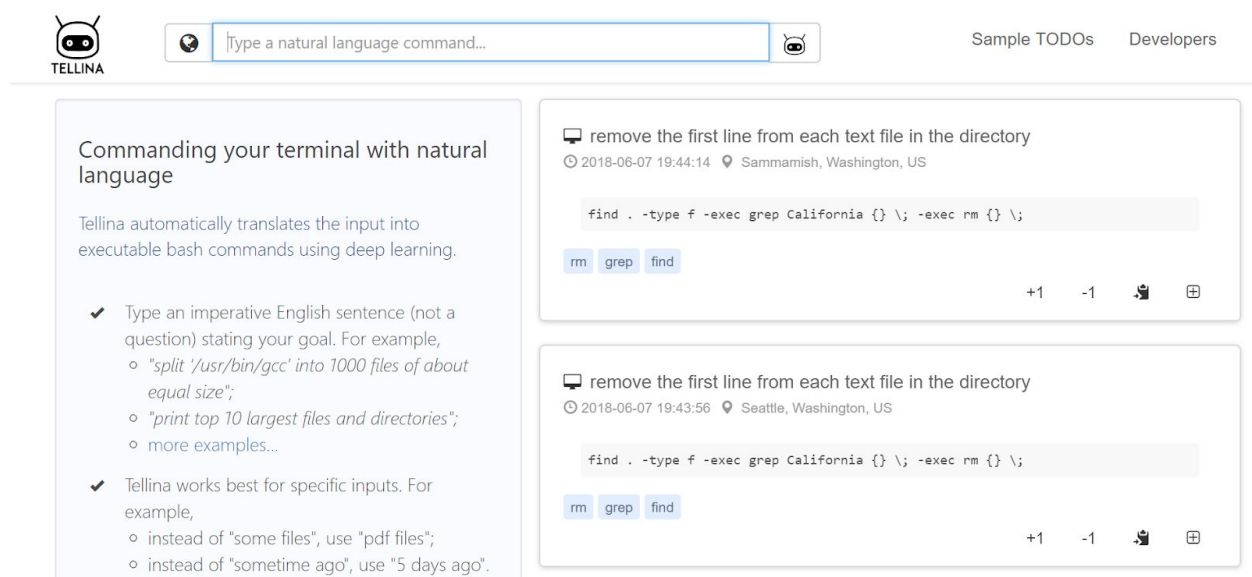


Figure 3: The web interface for Tellina.

The other major user interface, which appears for all end users attempting to translate English into Bash, is the Tellina web interface itself, which provides a search-engine-like GUI with a search box for inputting natural language and a sorted results list for reading candidate Bash commands. Because our project's scope involves giving the Tellina translation tool better data, rather than changing how the algorithm works, we have no plans to change this interface.

3.5 Current Technologies

Regarding web scraping, we use the JSoup library to scrape through search query results and question pages on Stack Overflow. This application is run on the same machine as our server, scheduled to run intermittently as a cron job. As detailed in our architecture diagram (3.2.1), and the data generated from the scraper is uploaded to the database on our web server and distributed to verification client applications. These client applications would then send back the verified pairs to the server, where their entries in the database are updated and ready to be output for training.

Many of our choices in both language and frameworks come from the current Tellina project, such as our use of a Django server; we choose these technologies primarily so that we could utilize existing code in order to interface with Tellina, such as the format of its own raw data dump files.

Because training the models with a CPU or laptop-tier GPU would have taken too long, we chose to train Tellina on AWS Elastic Compute Cloud, specifically using a p2.xlarge instance. This instance contains a NVIDIA Tesla K80 GPU, 4 CPU cores, and 61 GB of RAM, with storage provided by the network-based

Elastic Block Storage service. The entire training model and dataset fit into the memory of the GPU during training.

4. Experiment and Evaluation

To test that the two approaches to improving Tellina’s dataset, cleaning part of the original dataset and augmenting the original dataset with new English-Bash command pairs, we designed four versions of the dataset to find metrics of: the original dataset, a partially cleaned dataset, an augmented data set and a partially cleaned and augmented dataset. To partially clean the dataset, we removed a random sample of the original data, verified that sample in TesterUI to keep only the English-Bash command pairs that were correct and added them back to the original dataset. To augment the dataset, we scraped pages from Stack Overflow, verified this data in TesterUI and added to the original dataset. To partially clean and augment the dataset, we performed both of these operations. We explain with more detail the creation of these datasets in section 5.1.1 and how successful these experimental datasets were in 5.2.

The Tellina learning module repository contains a Makefile target to automatically generate accuracy metrics for the current trained model, which means that, after training a new model from one of the datasets for our experiment, we can generate tables for it and compare its metrics against our other dataset results. The tables generated give us four metrics: BLEU1, BLEU3, TM1 and TM3. BLEU stands for Bilingual Evaluation Understudy and is an approximate measurement of full command accuracy [4: Appendix C]. TM is defined in the original Tellina paper and measures the overlap between an expected result command and a model’s predicted command. The “1” and “3” refer to how many of the top predictions of the trained model are compared against the expected result. While it was noted that these automatic evaluation metrics do not agree with manual evaluation in the original paper, they did correctly guess the system with the highest full command accuracy and highest command structure accuracy [4 : Appendix C], making us believe higher values of these metrics are a good indicator of more accurate translation. Our best measure of success is if a model generated from either the cleaned, augmented or cleaned and augmented datasets has higher values for these metrics.

5. Results

5.1 Data

5.1.1 Cleaning and Scraping

Our task to improve the dataset was twofold: cleaning the original dataset to remove erroneous or unclear English-Bash command pairs and adding additional correct pairs to the dataset. In our cleaning test, we selected 423 existing command pairs at random from Tellina’s original dataset (from the subset that was not already cleaned by the existing filter scripts) and personally verified each using the TesterUI. This process identified 92 erroneous command pairs, or 22% of our sample. The cleaned dataset is the 331 remaining pairs added to the part of the dataset left unverified.

For our augmentation test, we first scraped two hundred threads, which yielded 793 potential command pairs. We then personally verified these using our TesterUI, resulting in 74 verified pairs. The augmented dataset is these additional pairs added to the original Tellina dataset.

For our combined cleaning and augmentation set, we added the 74 verified pairs from the augmented dataset to the cleaned dataset, thus performing both the cleaning and the augmentation operations to the original dataset.

These results are summarized in the tables below:

Dataset	Commands Added	Commands Removed	Total
Original (control)	0	0	12606
Cleaned	0	92	12514
Augmented	74	0	12680
Cleaned & Augmented	74	92	12588

Table 1: The number of commands added and removed for each dataset used in our experiments.

5.1.2 Model Accuracy

On May 10th, 2018, we collected initial results from our data scraping and retraining, which are presented against Tellina’s initial results below. To reproduce these results, follow the README instructions in our root repository. Note that we can train and evaluate our model from scratch or from the output of the TesterUI, the all.nl and all.cm files; instructions for both are in the main repository README. As it is hard to reproduce the exact results of scraping Stack Overflow or verifying English-Bash command pairs in TesterUI, training and evaluating the model from the output of TesterUI is the suggested way to reproduce our results.

Table 2 shows the results of reproduce the baseline results in the original paper with our new execution environment by obtaining the automatic evaluation results.

Model		Testing Accuracy w/ Given Metric on Original Dataset			
		BLEU1	BLEU3	TM1	TM3
Seq2Seq	Char	0.44	0.53	0.48	0.56
	Token	0.36	0.44	0.66	0.73
	Sub-Token	0.48	0.54	0.64	0.71
CopyNet	Char	0.39	0.47	0.41	0.46
	Token	0.44	0.54	0.66	0.73
	Sub-Token	0.55	0.63	0.63	0.70
Tellina		0.49	0.55	0.62	0.70

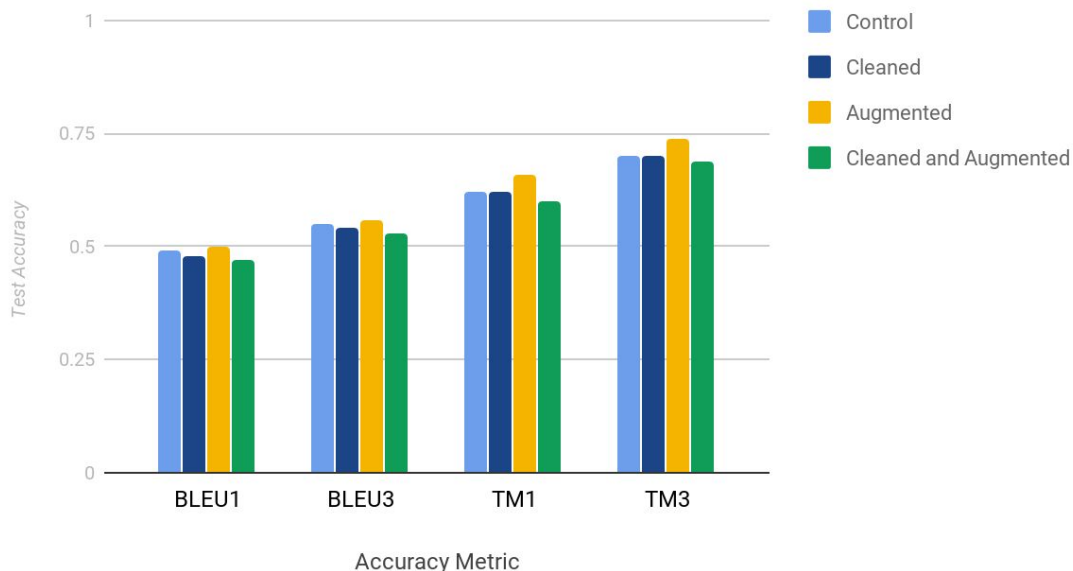
Table 2: The table of baseline results. Each row refers to a different combination of natural language model and input type (either Seq2Seq or CopyNet, taking in either individual characters, entire Bash tokens, or partial Bash tokens). Tellina is a normalized version of Seq2Seq taking full tokens as input.

Tables 3 and Graph 1 show the results of our experiments.

Dataset / Experiment	Tellina Testing Accuracy w/ Given Metric			
	BLEU1	BLEU3	TM1	TM3
Control	0.49	0.55	0.62	0.70
Tellina - Cleaned	0.48	0.54	0.62	0.70
Tellina - Augmented	0.50	0.56	0.66	0.74
Tellina - Cleaned and Augmented	0.47	0.53	0.60	0.69

Table 3: A table of Tellina’s accuracy with the manually cleaned dataset.

Translation Accuracy



Graph 1: Comparison of Tellina’s performance under each dataset.

5.2 Discussion

The results in the Tellina row of Table 2 differ by a few percentage points from the results presented in [4 : Appendix C]. Such deviation is expected, as a few variables have changed in the new execution environment, including the version of Tensorflow being used (previously 1.4, now 1.7), as well as some likely performance changes from using an Amazon AWS p2.xlarge instance instead of the original training machine. However, because the algorithms themselves have not changed, we did not expect the metrics to significantly change, and they did not. Note that we only include the results for all seven models in Table 2 for completeness - because we are only concerned with the accuracy of the Tellina model itself, only the Tellina model is tested for each dataset experiment.

The results in Table 3 indicate that cleaning the dataset, with or without the augmented data, reduced the BLEU and TM metrics. On the other hand, the augmented dataset provided significant improvement over the control (original) dataset, with the results showing an increase in every metric by at least a single percentage point. These results indicate that aggregating new results from a variety of sources is the most viable method, of the methods we explored, of improving Tellina’s accuracy. Hence, we have accomplished our original goal, as laid out in Section 1, of improving the dataset in order to have Tellina perform better than it does under the control dataset.

We note that in our cleaning that we removed 22% of the 423 command pairs we sampled from the dataset. In comparison, the original Tellina paper notes that two freelancers evaluated a sample of 100 English-Bash command pairs and found that 15% of them were erroneous. While this variation is not statistically significant (a two-valued Z-test on both samples yields a p-value of 0.12, meaning that the

difference in proportions could simply be due to random chance), we also observe that our cleaning might have been too strict, because several of the commands we removed were syntactically correct and mostly accurate, but differed in a single component, such as a filename where none was provided in the English description. Because the neural network can still get value from a syntactically legal Bash command, removing some of these samples might have brought down the accuracy of our neural network. However, more worryingly, both the 15% and 22% error rates for commands are quite high, and suggest that we might have needed to perform the cleaning operation on a census of the dataset instead of just a sample in order to get an increase in positive results.

Additionally, while the scraping and augmenting process did allow us to accomplish our goal, we were only able to use 9.3% of the 793 potential command pairs we scraped, largely due to output being included with the code or commands being listed multiple times. While we were able to separate actual commands from commentary by the answerer, the low acceptance of English-Bash pairs means that the improvement that came from scraping took came at the cost of human verification time. Future work could potentially improve the efficiency of scraping by applying multiple automatic filters to the scraped data, such as attempting to generate an AST for each command.

6. Next Steps

While we did manage to accomplish our goal, improvements to both cleaning and augmenting are possible ways to further improve the quality of the dataset and thus improve Tellina's ability to correctly translate English descriptions. As mentioned in section 5.2, a complete cleaning of the dataset, rather than just a cleaning of the sample, could well improve the dataset by making it more consistent, and better filtering techniques during scraping might provide better augmentations. Improving the human verification process itself is also a possibility, such as giving testers the ability to slightly edit the commands that are being cleaned or scraped in order to adjust them to quality standards. Yet another possibility is to provide a protected virtual environment to actually run each command and verify the results, as one might for a unit test.

While our project focused on verifying and augmenting the existing Tellina repository, another interesting continuation of the project could look into utilizing new machine learning architectures that are more resilient to noise. It is intrinsically hard to separate signals from noise when dealing with natural language; as much as we can manually format incoming commands and natural language descriptions to a specific template, it is much harder to automate given our current implementation. Changing the architecture that the original NL2Bash system is built upon to a more modern architecture more suitable for NLP could be a good next step. A system that implements word embeddings may be a good first step, as might a system with good support for multi-line inputs and outputs in order to support more complex descriptions and commands.

7. References

1. user77413, “How to count all the lines of code in a directory recursively? ”, <https://stackoverflow.com/questions/1358540/>.
2. Lin, X. V., Wang, C., Pang, D., Vu, K., & Ernst, M. D. (2017). *Program synthesis from natural language using recurrent neural networks* (Vol. 2). Technical Report UW-CSE-17-03-01, University of Washington Department of Computer Science and Engineering, Seattle, WA, USA.
3. <http://kirin.cs.washington.edu:8000/>
4. Lin, X. V., Wang, C., Zettlemoyer, L., & Ernst, M. D. (2018). NL2Bash: A Corpus and Semantic Parser for Natural Language Interface to the Linux Operating System. arXiv preprint arXiv:1802.08979.
5. <https://stackoverflow.com/>
6. Stack Exchange, “Stack Exchange Data Dump”, <https://archive.org/details/stackexchange> .
7. <https://github.com/TellinaTool/nl2bash>
8. <https://tiswww.case.edu/php/chet/bash/bashref.html>
9. <https://stackoverflow.com/questions/tagged/shell+bash?page=1&sort=votes&pagesize=50>