LLM-Supported Natural Language to Bash Translation

Anonymous ACL submission

Abstract

The Bourne-Again Shell (Bash) command-line interface for Linux systems presents usability challenges due to complex syntax and the need for extensive utility-specific knowledge. The natural language to Bash command (NL2SH) translation capabilities of large language models (LLMs) offer a solution by enabling users to interact with Linux systems using natural language. However, the performance of these models is difficult to assess due to inaccurate test data and unreliable heuristics for determining the functional equivalence of Bash commands. In this paper, we present the InterCode-ALFA benchmark, which improves upon existing benchmarks by providing a manually verified and expanded set of assessments. Further, it includes a novel functional equivalence heuristic that combines command execution with language model evaluation of command outputs. We evaluate popular LLMs using our benchmark and demonstrate that in-context learning can enhance NL2SH accuracy by up to 12%. Our findings emphasize the importance of dataset quality, prompting method, and execution-based evaluation for advancing NL2SH translation.

1 Introduction

The default command-line interface (CLI) for interacting with Linux systems is the Bourne-Again Shell (Bash) (Shotts, 2019). Bash commands allow computer users to control processes, interact with the file system, and manage the network. However, using Bash requires knowledge of numerous utilities, each with unique parameters and complex syntax (Ramey and Fox, 2024). Moreover, the reference documentation for these utilities, called manual pages, can be cumbersome and confusing (Kerrisk, 2024). This makes the CLI a barrier for inexperienced users and increases the chance of errors for experienced users (Agarwal et al., 2021).

The development of language models that convert natural language to command-line instructions, referred to as *NL2SH, NL2CMD or NL2Bash Translation*, offer a promising solution to this problem (Agarwal et al., 2021). We use the term *NL2SH model* to refer to models trained specifically for the task of NL2SH, as well as the NL2SH capabilities of general-purpose large language models (LLMs). NL2SH models are well suited for CLIs because they are designed for text-based interactions. Figure 1 shows an example of natural language to Bash command translation. NL2SH translation models can simplify human-computer interactions by allowing users to interact with Linux systems through natural language on the command line. This advancement enhances usability by reducing the need for syntax memorization (Sammet, 1966).

Input: Natural Language *List files in the /workspace directory that were accessed over an hour ago.*
Output: Bash Command `find /workspace -type f -amin +60`

Figure 1: Natural language to Bash command translation example from the InterCode-ALFA dataset.

The use of NL2SH models (Lloyd et al., 2024; Sadykov, 2024; Rosenkilde et al., 2024) necessitates benchmarks to measure model accuracy. A NL2SH benchmark requires test data with natural language prompts and corresponding commands serving as ground truth (referred to as instruction-command pairs). Given a natural language prompt, a NL2SH model produces a Bash command. A benchmark must then use a heuristic to determine if the model command is functionally equivalent to the ground truth command. Determining functional equivalence of commands is difficult because there are multiple possible correct commands for a given task, due to the Bash language and its syntax. Further, these commands may not generate identical outputs, neutralizing evaluation with string compar-

ison. Current benchmarks do not accurately measure NL2SH model performance due to errors in assessment data and inaccurate heuristics for measuring the functional equivalence of commands (Yang et al., 2023; Song et al., 2024; Aggarwal et al., 2024). This makes it difficult to determine model capabilities and measure methods for improving model performance.

To address these challenges, we investigate the following research questions: (1) How can we validate NL2SH datasets to ensure models are evaluated using valid translations? (2) How can we design a functional equivalence heuristic that accurately measures the quality of model translations? (3) How can we improve the accuracy of NL2SH models as measured by a reliable benchmark?

Our contributions are summarized as follows: (1) We manually verify and correct 386 instruction-command pairs in Yang et al.'s InterCode test dataset and increase the size of the training dataset from 9,305 to 23,109 pairs by combining and deduplicating datasets from Lin et al. (2018), Ramesh (2022), Romit (2024), and Mali (2023). (2) We present the InterCode-ALFA benchmark, capable of determining the functional equivalence of two Bash commands with 92% confidence, a 44% increase over the InterCode benchmark. (3) We evaluate GPT-3.5-T-0125, GPT-4o-2024-08-06, and LLaMa3.1-8b-Instruct using the InterCode-ALFA benchmark and show in-context learning improves NL2SH performance by up to 12%.

2 Background

NL2SH translation falls under the broader domain of machine translation, where models automatically translate text or speech from one language to another. LLMs are well suited for this task, enabling translation that was impossible with previous methods (Zhu et al., 2024b). Evaluating NL2SH models requires determining the functional correctness of generated commands. Functional correctness is defined as whether the code produces the correct output for each input, as specified, or as compared to ground truth (Chen et al., 2024). Functional correctness does not consider the diversity of commands generated, or other factors such as running time and memory consumption (Chon et al., 2024).

Ensuring the functional correctness of code is difficult because validation methods are error-prone and take an impractical amount of time for large volumes of code (Chen et al., 2024). There are two main categories of validation techniques: static and dynamic analysis. Static analysis checks code without execution, using parsers, lexical analysis, control flow checking, or other similar methods (Shaikhelislamov et al., 2024). Dynamic analysis evaluates code after execution using an execution environment (Yang et al., 2023). Some validation frameworks combine static and dynamic analysis techniques (Aggarwal et al., 2024).

Determining the functional correctness of Bash commands translated from natural language is a sub-problem of validating code correctness. We assess the functional correctness of Bash commands by comparing the generated (model) command with a ground truth Bash command. We define the term "functional equivalence heuristic" (FEH) to describe a heuristic that performs this comparison and determines the functional correctness of a Bash command. Due to varying definitions in this field, we provide notation in Table 1 defining the terms used in this paper.

Table 1: Definition of terms.

Term	Definition
Natural Language Task	$t \in T$ English
Ground Truth Command	$b \in B$ Bash-5.2
Model Command	$b' \in B$
Functionally Equivalent Command	$\hat{b} \in B$
Docker Environment	$e \in E$ Linux
Command Output and Side Effects	$o \in O$ System state
Model Weights	$\theta \in \mathbb{R}$
Translation	$f : T \times \mathbb{R} \to B \quad f(t, \theta) = b'$
Execution	$g : B \times E \to O$ $g(b, e) = o \quad g(b', e) = o'$
Func. Equiv. Heuristic	$m : O \times O \to \{0, 1\}$ $m(o, o') = \begin{cases} 1 : o \approx o' \\ 0 : o \neq o' \end{cases}$
Training Dataset	$D_T : \{(t_i, b_i) \mid i = 1, 2, \ldots, x\}$
Test Dataset	$D_H : \{(t_i, b_i, \hat{b_i}) \mid i = 1, 2, \ldots, y\}$ $m(g(b, e), g(\hat{b}, e)) = 1 \quad$ ideal m $D_T \cap D_H = \emptyset$
Benchmark	(D_H, m)
Performance	$p : T \times B \times \mathbb{R} \to \mathbb{R}$ $p(D_H, \theta) =$ $\frac{\sum_{i=1}^{\lvert D_H \rvert} m(g(b_i, e), g(f(t_i, \theta), e))}{\lvert D_H \rvert}$

3 Related Work

NL2SH translation is a well-studied natural language processing (NLP) task. Table 2 summarizes the contributions of previous work by listing the names of datasets, functional equivalence heuristics (FEH), and models used in this field. The table is sparsely populated because the majority of previous work has focused on improving a NL2SH dataset, FEH or model in isolation.

The 2020 NeurIPS NLC2CMD Competition formalized the task of NL2SH translation by provid-

Table 2: Summary of NL2SH contributions.

	Dataset	FEH	Model
(Lin et al., 2018)	NL2Bash	-	-
(Gros, 2019)	-	-	AInix
(Agarwal et al., 2021)	-	NL2CMD	Tellina
(Fu et al., 2021)	-	-	Magnum
(Ramesh, 2022)	NL2CMD	-	-
(Bharadwaj and Shevade, 2022)	-	-	AST
(Shi et al., 2023)	-	-	ShellGPT
(Yang et al., 2023)	InterCode	InterCode	-
(Mali, 2023)	text_to_bash	-	-
(Song et al., 2024)	-	TSED	-
(Aggarwal et al., 2024)	CodeSift	CodeSift	-
(Vo et al., 2024)	EE	EE	-
(Romit, 2024)	LinuxCommands	-	-
(Lloyd et al., 2024)	-	-	Warp
(Sadykov, 2024)	-	-	shell-gpt
(Rosenkilde et al., 2024)	-	-	Copilot CLI
(Chatterjee et al., 2024)	-	-	ScriptSmith
Ours	ICALFA	ICALFA	-

ing a human-curated NL2Bash dataset of 9,305 instruction-command pairs (Lin et al., 2018) and the NL2CMD benchmark for evaluating submitted models (Agarwal et al., 2021). The competition resulted in numerous NL2SH models, and showed that supervised fine-tuning of a pre-trained foundation model, such as GPT-2 (Radford et al., 2019), outperforms dedicated transformer (Fu et al., 2021), recurrent neural network (Lin, 2017), abstract syntax tree (Bharadwaj and Shevade, 2022), and sequence to sequence (Gros, 2019) based models for the task of NL2SH translation (Shi et al., 2023).

The NL2CMD benchmarks's FEH parses commands and assigns a similarity score based on the utilities used, order of utilities, and number of utility flags. This heuristic outperforms conventional string comparison techniques, such as BLEU score and edit distance, for measuring the functional equivalence of commands. However, Agarwal et al. (2021) state that the NL2CMD benchmark could be improved by executing commands and measuring the similarity of the outputs and side effects. Verification by execution is preferable because Bash is a Turing-complete language, so verifying the equivalence of two commands before execution is undecidable due to side effects (Churchill et al., 2019). Despite this known shortcoming, the NL2CMD benchmark is widely used due to the challenge of creating a sand-boxed environment that captures command outputs and side effects (Fu et al., 2023).

Yang et al. (2023) address this gap with the InterCode-Bash benchmark. The benchmark uses a subset of 224 instruction-command pairs from the NL2Bash dataset for testing. InterCode's FEH executes the model command and ground truth command in identical Docker containers (Merkel, 2014). The results of command execution are then compared using three checks. First, the pre and post-execution states of each container are compared using git-diff. Second, the file contents of each container are compared using MD5 hashes. Third, the standard output of both commands are compared using the TfidfVectorizer bag of words method (Sparck Jones, 1972). If every check finds the execution results identical, the model and ground truth command are considered functionally equivalent (Yang et al., 2023).

Huang et al. (2022) and Vijayaraghavan et al. (2024) present similar execution-based frameworks for Jupyter Notebooks and VHDL code, respectively. Vo et al. (2024) describe an execution-based framework similar to the InterCode-Bash benchmark using Podman containers. Unfortunately, the code for their FEH and the 50 instruction-command pairs in their test dataset are not public.

Song et al. (2024) present a novel benchmark that uses OpenAI's GPT-4 model to determine the functional equivalence of commands. Their FEH passes the model command and ground truth command to GPT-4 with the prompt, *Given 2 Bash commands, please generate a similarity score from 0 to 1.* Song et al. (2024) find this method fails to determine functional equivalence because current LLMs are unable to emulate command execution. Maveli et al. (2024) and Naik (2024) confirm this finding with broader evaluations of LLM's ability to determine semantically equivalent or different pairs of programs. They find LLMs show a lack of depth in understanding code semantics.

Aggarwal et al. (2024) iterate on the work presented by Song et al. (2024), aiming to address the scalability constraints of execution-based frameworks. Their FEH, CodeSift, uses an LLM to convert the model command to a natural language description. Then they compare this natural language description with the original natural language task using an LLM. While they find CodeSift to be more effective than conventional methods of measuring functional equivalence, their work lacks comparison with execution based heuristics. Further, their FEH introduces uncertainty by requiring accurate Bash to natural language translation, which is equally as challenging as the task they aim to measure, natural language to Bash translation. Their follow on work finds that "executing [Bash] scripts within a controlled environment would offer more reliable assessments" (Chatterjee et al., 2024).

This paper builds on previous work by combining the language model evaluation strategy pre-

sented by Song et al. (2024) with the InterCode execution framework presented by Yang et al. (2023) Further, we correct and expand the InterCode dataset using multiple datasets from previous work. Figure 3 compares NL2SH benchmarking methods and summarizes the shortcomings of functional equivalence heuristics in previous work.

4 Methodology

NL2SH translation of extant datasets is a difficult task due to the size of the solution space. The InterCode dataset draws upon 64 unique utilities across 224 commands, with an average of four arguments per utility and two pipes per command. Using these averages, the number of possible command combinations is on the order of $(64 * 4)^3 \approx 16 * 10^6$. For each natural language prompt, a single correct Bash command is provided, comprising roughly $1/(64 * 4)^3 \approx 6 \times 10^{-8}$ of the solution space. The average length of each command is 19 tokens. As the number of utilities, arguments, and pipes increase, the task becomes more difficult due to a larger solution space and increased number of tokens required per generation, which decreases model accuracy (Nayab et al., 2024). To understand this solution space, we begin with a comparison of the benchmarks presented in previous work.

4.1 Benchmark Comparison

Using the instruction *list the names of bind functions containing "p"* and the ground truth command `bind -l | grep p`, we manually create an example dataset with ten Bash commands. The first five are functionally equivalent to the ground truth and the last five are non-equivalent. Figure 2 shows a comparison of each benchmark's FEH. The rows are example commands and the columns are the functional equivalence heuristics. The first column shows the functional equivalence of each command to the ground truth command, with one being equivalent and zero being non-equivalent. The remaining columns show the result of executing each heuristic. We scale the NL2CMD heuristic from its original range of [-1, 1] to [0, 1] to simplify comparison.

This example highlights the effectiveness of command execution for measuring the functional equivalence of commands. Pre-execution methods fail because two commands may be syntactically similar and yield different results when executed. For example, changing a single flag can result in vastly different command outputs. Further, two

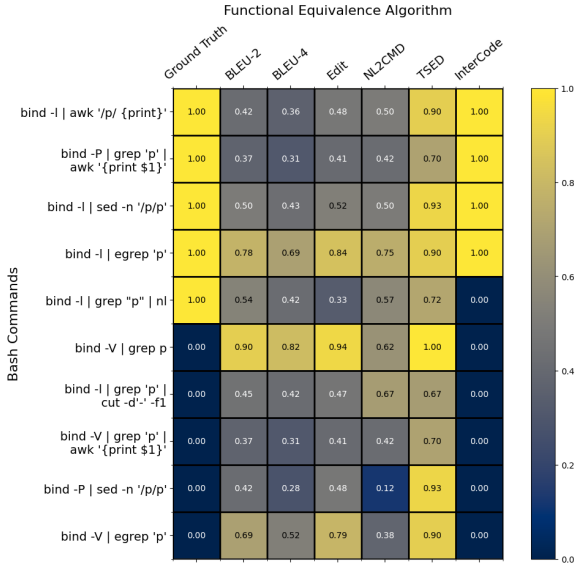

Figure 2: Comparison of NL2SH functional equivalence heuristics using the ground truth *bind -l | grep p* and the prompt *list names of bind functions containing "p"*.

commands with no syntactic similarity can yield identical results when executed. For example, the `awk` and `sed` utilities can accomplish identical text processing tasks but use different domain-specific languages, requiring different syntax.

In our motivating example, the InterCode heuristic is more accurate than other heuristics. However, the InterCode heuristic finds the model command `bind -l | grep p | nl` to be non-equivalent to the ground truth command `bind -l | grep p`. The model command lists bind functions with a number on each line, while the ground truth command does not have a numbered output. The commands are functionally equivalent because the outputs convey the same information to a human user, but the InterCode heuristic finds them non-equivalent due to the difference in output syntax. This reveals that although InterCode's command execution framework is more accurate than other benchmarks, its method for comparing command outputs may fail to determine functional equivalence. To address this problem, we conduct a rigorous evaluation of the InterCode test dataset and functional equivalence heuristic as described in the next section.

4.2 Dataset Creation

The InterCode benchmark uses a test dataset of 224 instruction-command pairs, each of which can be executed in a Docker environment. The dataset is based on the human-curated NL2Bash dataset presented by Lin et al. (2018), containing 9,305

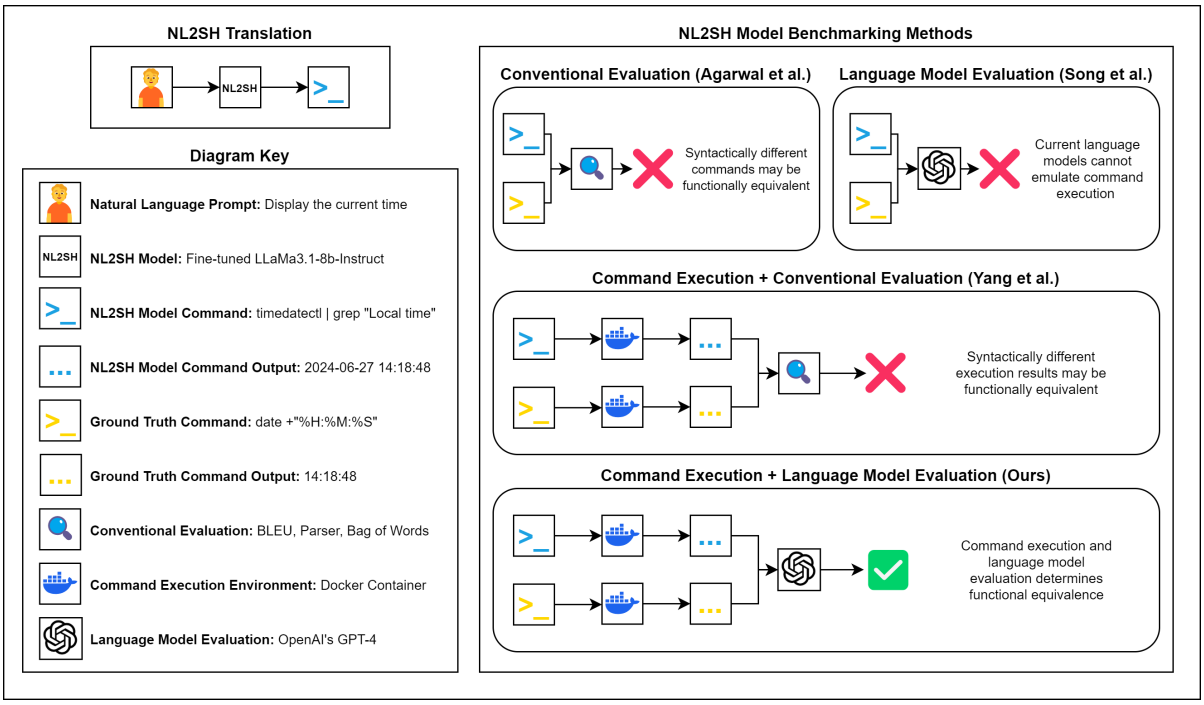

Figure 3: A diagram of NL2SH translation and comparison of functional equivalence heuristics.

instruction-command pairs. The InterCode dataset is significantly smaller in size because a Docker environment must be configured for each command. Manual verification of all 224 instruction-command pairs reveals that over half of the InterCode dataset is erroneous.

Table 3 shows the number of errors organized by type. We define three types of errors: invalid prompt, invalid command, and invalid environment. An invalid prompt error refers to a natural language instruction that describes an impossible task or does not give enough information to accomplish the task. We find 25.4% of pairs have invalid prompts. An invalid command error refers to a Bash command that does not accomplish the task described in the prompt or does not execute. We find 29.8% of pairs have invalid commands. An invalid environment error refers to an incorrect Docker configuration such as a missing file, environment variable, or utility that prevents a valid command from accomplishing the task. We find 14.2% of pairs have invalid environments. Our manual verification reveals 102 instruction-command pairs with one or more errors and 11 duplicate pairs.

We fix 82 of these errors by correcting the natural language prompts, Bash commands, and Docker configuration files of 93 instruction-command pairs. We remove 11 duplicate and 20 irreparable pairs from the dataset, resulting in 193 verified pairs.

Table 3: InterCode Dataset Errors

Error Type	Count	Percentage
Duplicate	11	4.9%
Invalid Prompt	17	7.6%
Invalid Cmd	24	10.7%
Invalid Env	18	8.0%
Invalid Prompt and Cmd	29	12.9%
Invalid Prompt and Env	0	0.0%
Invalid Cmd and Env	3	1.3%
Invalid Prompt, Cmd and Env	11	4.9%
Invalid Total	113	50.4%
Valid Total	111	49.6%
Total	224	100.0%

Additionally, for each verified pair we create a second Bash command that accomplishes the task described in the prompt. Our final dataset contains two functionally equivalent, ground truth Bash commands for each natural language instruction, for a total of 386 instruction-command pairs. An increase of 50%. The first ground truth command is used as part of our benchmark and the second ground truth command is used to evaluate the efficacy of the benchmark. Our test dataset and annotated corrections can be found on HuggingFace[1].

The NL2Bash dataset is commonly used to train NL2SH models (Fu et al., 2021; Lin, 2017; Shi et al., 2023; Bharadwaj and Shevade, 2022). Since the InterCode test dataset is based on the NL2Bash dataset, there is a risk of test data leakage in the training data. We create a new training dataset with

[1] https://huggingface.co/datasets/ANON/InterCode-ALFA-Data

24,026 pairs by combing and de-duplicating the NL2Bash dataset with three other publicly available NL2SH datasets (Ye, 2022; Ramesh, 2022; Romit, 2024; Mali, 2023).

We de-conflict the training and test dataset using exact matching and semantic similarity measures, removing in the process 917 pairs from the training data. First we remove instruction-command pairs from the training data that exactly match instructions or commands in the test data. Next, we remove pairs from the training data that have a natural language prompt that is syntactically similar to a prompt in the test data using TfidfVectorizer with a threshold of 0.9. The resulting training dataset of 23,109 pairs can be found on HuggingFace[2].

Manual correction of the InterCode dataset took over 80 hours. Finding efficient means to verify our training dataset will be explored in future work. Figure 4 shows the relationships between NL2SH datasets used to create the InterCode-ALFA training and test datasets.

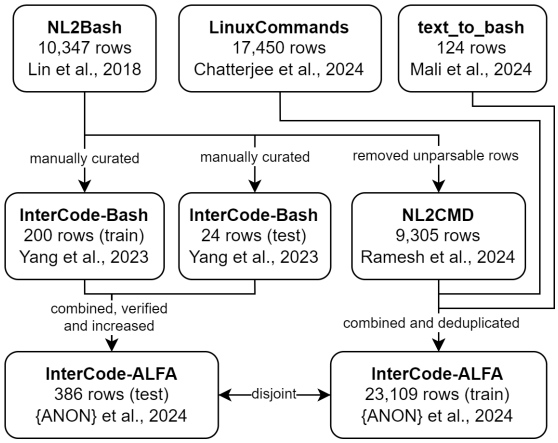

Figure 4: Hierarchy of NL2SH datasets.

4.3 Functional Equivalence Heuristic

Our initial example in Section 4.1 reveals that the InterCode benchmark is more accurate than other benchmarks because it executes commands. However, its TfidfVectorizer bag of words method for comparing command outputs may fail to determine functional equivalence because syntactically different outputs may convey the same information to the end user. We replace the TfidfVectorizer bag of words method for comparing command outputs with a large language model, specifically OpenAI's GPT-4-0613. Instead of passing the stdout output of both commands to TfidfVectorizer, we use the OpenAI API with the prompt in Figure 7, a temperature of zero, and a static seed value of 123.

It is important to note that the first two functional equivalence checks in the InterCode benchmark remain unchanged. Our benchmark compares the results of command execution using three checks. First, the pre and post-execution states of each container are compared using git-diff. Second, the file contents of each container are compared using MD5 hashes. Third, the standard output of both commands are compared using GPT-4-0613, instead of TfidfVectorizer. If each check finds that the execution results are identical, then the model command and ground truth command are considered functionally equivalent.

Replacing TfidfVectorizer with GPT-4-0613 increases the computational cost of the benchmark. In our implementation, this results in a monetary cost via the OpenAI API of approximately $4 for one benchmark run. Additionally, since GPT-4-0613 is non-deterministic, we run the benchmark six times, average the results and report the variance. We conduct two experiments comparing TfidfVectorizer and GPT-4-0613 using our corrected test dataset, detailed in Section 5.

4.4 Benchmark Configuration

We present our new functional equivalence heuristic and corrected dataset as a new version of the InterCode benchmark, InterCode-ALFA. Our benchmark and datasets are released under MIT licenses. In addition to the dataset and heuristic changes, we add error handling and update the Docker configuration files to use stable Linux releases. We also remove environments for Python, SQL, and Capture the Flag evaluation included in the InterCode benchmark, reducing the code base by approximately 75%. Lastly, we identify and fix an error in the benchmark's Docker reset script that causes the filesystem structure of the two execution environments to diverge. We publish the benchmark source code on GitHub[3] and provide a Python package on PyPI[4]. Our benchmark and dataset can be installed and configured with 10 lines of code, simplifying the process for benchmarking new models as shown in Figure 5.

[2] https://huggingface.co/datasets/ANON/nl2sh

[3] https://github.com/ANON/InterCode-ALFA

[4] https://pypi.org/project/icalfa/

```python
import os
from icalfa import submit_command
from datasets import load_dataset

# Store OpenAI key as environment variable
os.environ['ICALFA_OPENAI_API_KEY'] = '...'

# Load dataset
dataset = load_dataset("{ANON}/InterCode-ALFA-Data")

# Iterate through the dataset
score = 0
for index, row in enumerate(dataset):

    # Retrieve natural language prompt
    prompt = row['query']

    # Convert natural language to Bash command

    # Submit Bash command for benchmark scoring
    score += submit_command(index=index, command="...")
    # 0 = incorrect, 1 = correct

# Print the benchmark result
print(score/len(dataset))
```

Figure 5: InterCode-ALFA benchmark setup.

5 Experiments

We conduct two experiments to determine if GPT-4-0613 is a better arbiter of functional equivalence than TfidfVectorizer. Then, using the InterCode-ALFA benchmark, we measure the NL2SH performance of three popular language models. Finally, we evaluate the impact of in-context learning (ICL) on the performance of these models. Our results are summarized in Table 4 and Table 5.

5.1 Identical Inputs

In the identical inputs experiment, we measure the performance of the ground truth commands on the InterCode and InterCode-ALFA benchmark. This means that each benchmarks' functional equivalence heuristic compares identical commands for all 193 test cases. Explicitly, $m(g(b, e), g(b, e))$. The expected performance on each benchmark is a score of $193/193 = 100\%$.

The InterCode benchmark returns an average score of $181/193 = 94\%$ with 1% variance. This result is unexpected and shows that achieving a perfect score on the InterCode benchmark is impossible. We find the source of this problem to be system state divergence. For fair comparison of two Bash commands, both commands must be executed in identical environments. Explicitly, $g(b, e) = g(b, e') \iff e = e'$. After correcting the system state divergence problem, the InterCode benchmark returns an average score of $193/193 = 100\%$. The InterCode-ALFA benchmark does not suffer from the system state divergence problem and returns an average score of $193/193 = 100\%$ with no variance. This shows that the InterCode-ALFA benchmark correctly finds identical Bash commands to be functionally equivalent.

5.2 Functionally Equivalent Inputs

In the functionally equivalent inputs experiment, we measure the performance of the functionally equivalent ground truth commands on the InterCode and InterCode-ALFA benchmark. This means that each benchmark's functional equivalence heuristic compares the ground truth command against a command that we verified is functionally equivalent. Explicitly, $m(g(b, e), g(\hat{b}, e))$ where $g(b, e) \approx g(\hat{b}, e)$. The ideal performance on each benchmark is a score of $193/193 = 100\%$, with a higher score indicating a more capable functional equivalence heuristic. Explicitly, $m(g(b, e), g(\hat{b}, e)) = 1$ with an ideal functional equivalence heuristic. We use a corrected version of the InterCode benchmark to avoid conflating results with the system state divergence problem.

The InterCode benchmark returns an average score of $94/193 = 49\%$ with 1% variance. This results shows that the InterCode benchmark fails to determine the functional equivalence of two commands in the majority of test cases. The InterCode-ALFA benchmark returns an average score of $178/193 = 92\%$ with 1% variance. This shows that GPT-4-0613 can determine the functional equivalence of command outputs with greater accuracy than TfidfVectorizer.

Table 4: Results of identical and functionally equivalent input experiments. The plus sign indicates the increase in performance over the InterCode benchmark.

	InterCode	InterCode-ALFA
Identical Inputs	0.94±0.01	**1.00±0.00** (+0.06)
Func. Equiv. Inputs	0.49±0.01	**0.92±0.01** (+0.44)

5.3 In-Context Learning

Using version 0.3.2 of the InterCode-ALFA benchmark, we evaluate the GPT-3.5-Turbo-0125 (Brown et al., 2020), GPT-4o-2024-08-06 (OpenAI, 2024b), and LLaMA-3.1-8b-Instruct (Team, 2024) models. We use one Nvidia RTX A6000 GPU to query the LLama3.1 model and the OpenAI API to query the GPT models. The number of parameters in the GPT models is currently unknown.

First, we conduct a baseline evaluation and query

each model with the prompt in Figure 8. For all evaluations, we take the first response without retries because a single invalid Bash command could corrupt a system. Second, we evaluate the impact of in-context learning (few-shot prompting) on each model (Brown et al., 2020). We create a new prompt by appending ten manually created instruction-command pairs to our original prompt as shown in Figure 9. We use a static seed and a temperature of zero to reduce response variability.

For the baseline evaluation, OpenAI's GPT-4o-2024-08-06 model achieves the highest performance with $126/193 = 65\%$ correct translations, followed by GPT-3.5-Turbo-0125 with $84/193 = 44\%$ correct translations and LLaMA-3.1-8b-Instruct with $52/193 = 27\%$ correct translations. We find that in-context learning improves the performance of all three models over the baseline by 3% to 12%. We find our results to be statistically significant using a Welch's t-test with a p-value threshold of 0.05.

Table 5: Model performance on the InterCode-ALFA benchmark. The plus sign in parenthesis indicates the increase in performance over the baseline.

	Baseline	ICL
LLaMA-3.1-8b-Inst	0.27±0.03	0.37±0.03 (+0.11)
GPT-3.5-T-0125	0.44±0.08	0.56±0.06 (+0.12)
GPT-4o-2024-08-06	**0.65±0.05**	**0.68±0.07** (+0.03)

6 Discussion

Manual correction of the InterCode dataset identified over half of the instruction-command pairs as erroneous, answering our first research question. The time and effort required for manual correction highlights the need for more efficient means to verify larger datasets. Further, since the InterCode dataset is sampled from the NL2Bash dataset, there is a risk the NL2Bash dataset contains a significant percentage of erroneous data. This is concerning because the majority of the models presented in previous work were trained on the NL2Bash dataset.

The identical inputs experiment shows that the InterCode-ALFA benchmark finds identical Bash commands to be functionally equivalent, while the InterCode benchmark may fail due to a system state divergence problem. This problem paired with errors in the InterCode dataset shows that results from the InterCode benchmark cannot be fully trusted.

The functionally equivalent inputs experiment shows that the InterCode benchmark fails to determine the functional equivalence of commands in the majority of test cases. The experiment also reveals that GPT-4-0613 can determine the functional equivalence of Bash commands, based on outputs from an execution environment, with higher accuracy than TfidfVectorizer. We find command execution with LLM evaluation of execution outputs to be effective at determining the functional equivalence of Bash commands, answering our second research question. Broadly, LLM evaluation of execution outputs is a promising advancement for measuring the functional correctness of generated code and more investigation is warranted.

Our baseline evaluation of LLMs using the InterCode-ALFA benchmark shows that NL2SH translation is a difficult task for current state-of-the-art (SOTA) models. We find that in-context learning results in a performance increase for each model, answering our third research question. From the total of our experiences, we conclude the accuracy of SOTA models for NL2SH translation needs to improve before they can be safely used in practice due to relatively low accuracy and the risk of a single malformed Bash command.

7 Conclusion

In this paper, we explore applications for large language models in NL2SH translation and benchmarking. We identify issues with current benchmarks, including inaccurate datasets and an unreliable functional equivalence heuristics. To address these problems, we correct and expand the InterCode dataset and create a new heuristic to determine the functional equivalence of Bash commands. Using our new benchmark, we evaluate how in-context learning can be used to improve NL2SH translation models. Our findings indicate that Bash command execution paired with language model evaluation of command outputs can determine the functional equivalence of commands more accurately than previous methods. Additionally, we find that in-context learning can improve the performance of open and closed-source language models for the task of NL2SH translation. Ultimately, we find that NL2SH translation is a difficult task for LLMs, necessitating further research in this field. In future work, we plan to evaluate additional LLMs and formal NLP methods as arbiters of functional equivalence, investigate efficient means to verify our training dataset, explore how grammars can constrain model translations, and evaluate model fine-tuning and reasoning strategies.

8 Limitations

This work corrects and expands the InterCode test dataset. However, due to the time and effort required to manually configure an execution environment for each test case, the test dataset remains small, with only 193 test cases. Although improved over previous methods, our functional equivalence heuristic has inherent variability, requiring multiple runs to assess model performance. Finally, despite improved performance using ICL, the accuracy of SOTA LLMs for NL2SH translation remains low, motivating further research in this field.

9 Ethical Considerations

Due to the low performance of current NL2SH models, using these models in practice could result in invalid commands that have unintended effects on a system. We recommend that model-generated commands are never used without human verification. Further, we recommend that users test commands using a sand-boxed environment, such as try (Zhu et al., 2024a), before running them on personal systems. Figure 6 shows an example of a dangerous command observed during model testing. The hallucinated `rm -f /dev/null` command corrupted our benchmark's Docker container. While the benchmark automatically creates a new Docker container to handle this type of error, the command could have corrupted a user's system.

Natural Language Prompt:
Recursively remove all empty folders from the /system/temp folder.
Ground Truth Command:
`find /system/temp -type d -empty -delete`
Llama3.1-8b-Instruct Command:
`find /system/temp -type d -empty -delete -print; rm -f /dev/null 2>&1`

Figure 6: Dangerous translation observed in testing.

Natural language to Bash translation aims to increase computer accessability by simplifying interactions with the command-line interface. Unfortunately, good and bad actors can benefit from increased accessability. Models could be used to generate malicious Bash commands. This risk is difficult to mitigate because malicious use depends on the intent of the user. For example, a command to delete files could be used for a legitimate or harmful purpose. We plan to assess how guard rails and access controls can be placed on NL2SH models in future work.

References

Mayank Agarwal, Tathagata Chakraborti, Quchen Fu, David Gros, Xi Victoria Lin, Jaron Maene, Kartik Talamadupula, Zhongwei Teng, and Jules White. 2021. Neurips 2020 nlc2cmd competition: Translating natural language to bash commands. *Preprint*, arXiv:2103.02523.

Pooja Aggarwal, Oishik Chatterjee, Ting Dai, Prateeti Mohapatra, Brent Paulovicks, Brad Blancett, and Arthur De Magalhaes. 2024. Codesift: An llm-based reference-less framework for automatic code validation. *Preprint*, arXiv:2408.15630.

Shikhar Bharadwaj and Shirish Shevade. 2022. Efficient constituency tree based encoding for natural language to bash translation. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3159–3168, Seattle, United States. 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. *Preprint*, arXiv:2005.14165.

Oishik Chatterjee, Pooja Aggarwal, Suranjana Samanta, Ting Dai, Prateeti Mohapatra, Debanjana Kar, Ruchi Mahindru, Steve Barbieri, Eugen Postea, Brad Blancett, and Arthur De Magalhaes. 2024. Scriptsmith: A unified llm framework for enhancing it operations via automated bash script generation, assessment, and refinement. *Preprint*, arXiv:2409.17166.

Liguo Chen, Qi Guo, Hongrui Jia, Zhengran Zeng, Xin Wang, Yijiang Xu, Jian Wu, Yidong Wang, Qing Gao, Jindong Wang, Wei Ye, and Shikun Zhang. 2024. A survey on evaluating large language models in code generation tasks. *Preprint*, arXiv:2408.16498.

Heejae Chon, Seonghyeon Lee, Jinyoung Yeo, and Dongha Lee. 2024. Is functional correctness enough to evaluate code language models? exploring diversity of generated codes. *Preprint*, arXiv:2408.14504.

Berkeley Churchill, Oded Padon, Rahul Sharma, and Alex Aiken. 2019. Semantic program alignment for equivalence checking. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2019, pages 1027–1040, New York, NY, USA. Association for Computing Machinery.

Quchen Fu, Zhongwei Teng, Marco Georgaklis, Jules White, and Douglas C Schmidt. 2023. Nl2cmd: An

updated workflow for natural language to bash commands translation. *arXiv preprint arXiv:2302.07845*.

Quchen Fu, Zhongwei Teng, Jules White, and Douglas C. Schmidt. 2021. A transformer-based approach for translating natural language to bash commands. In *2021 20th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 1245–1248.

David Gros. 2019. Ainix: An open platform for natural language interfaces to shell commands. Undergraduate Honors Thesis, Computer Science Department, University of Texas at Austin.

Junjie Huang, Chenglong Wang, Jipeng Zhang, Cong Yan, Haotian Cui, Jeevana Priya Inala, Colin Clement, Nan Duan, and Jianfeng Gao. 2022. Execution-based evaluation for data science code generation models. *Preprint*, arXiv:2211.09374.

Michael Kerrisk. 2024. Linux manual pages. https://man7.org/linux/man-pages/index.html. Accessed: 2024-05-09.

Xi Victoria Lin. 2017. Program synthesis from natural language using recurrent neural networks.

Xi Victoria Lin, Chenglong Wang, Luke Zettlemoyer, and Michael D. Ernst. 2018. Nl2bash: A corpus and semantic parser for natural language interface to the linux operating system. *Preprint*, arXiv:1802.08979.

Zach Lloyd, Michelle Lim, and Aloke Desai. 2024. Warp: Your terminal, reimagined. https://www.warp.dev/. Accessed: 2024-05-09.

Yogesh Mali. 2023. text_to_bash. Accessed: 2024-08-20.

Nickil Maveli, Antonio Vergari, and Shay B. Cohen. 2024. What can large language models capture about code functional equivalence? *Preprint*, arXiv:2408.11081.

Dirk Merkel. 2014. Docker: lightweight linux containers for consistent development and deployment. *Linux journal*, 2014(239):2.

Atharva Naik. 2024. On the limitations of embedding based methods for measuring functional correctness for code generation. *Preprint*, arXiv:2405.01580.

Sania Nayab, Giulio Rossolini, Giorgio Buttazzo, Nicolamaria Manes, and Fabrizio Giacomelli. 2024. Concise thoughts: Impact of output length on llm reasoning and cost. *Preprint*, arXiv:2407.19825.

OpenAI. 2024a. Gpt-4 technical report. *Preprint*, arXiv:2303.08774.

OpenAI. 2024b. Gpt-4o system card. https://cdn.openai.com/gpt-4o-system-card.pdf. Accessed: 2024-09-09.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

T. Ramesh. 2022. Nl2cmd. Accessed: 2024-08-20.

Chet Ramey and Brian Fox. 2024. *Bash: GNU Project's Shell*. Accessed: 2024-05-09.

Romit. 2024. Linuxcommands. Accessed: 2024-08-20.

Johan Rosenkilde, Matt Rothenberg, and Andy Feller. 2024. Github copilot for cli. https://githubnext.com/projects/copilot-cli/. Accessed: 2024-09-25.

Farkhod Sadykov. 2024. Shellgpt: A command-line productivity tool powered by ai large language models like gpt-4. https://github.com/TheR1D/shell_gpt. Accessed: 2024-05-09.

Jean E. Sammet. 1966. The use of english as a programming language. *Commun. ACM*, 9(3):228–230.

Danil Shaikhelislamov, Mikhail Drobyshevskiy, and Andrey Belevantsev. 2024. Codepatchllm: Configuring code generation using a static analyzer.

Jie Shi, Sihang Jiang, Bo Xu, Jiaqing Liang, Yanghua Xiao, and Wei Wang. 2023. Shellgpt: Generative pre-trained transformer model for shell language understanding. In *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*, pages 671–682.

William E Shotts. 2019. *The Linux Command Line: A Complete Introduction*. No Starch Press.

Yewei Song, Cedric Lothritz, Daniel Tang, Tegawendé F. Bissyandé, and Jacques Klein. 2024. Revisiting code similarity evaluation with abstract syntax tree edit distance. *Preprint*, arXiv:2404.08817.

Karen Sparck Jones. 1972. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, 28(1):11–21.

Llama Team. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Prashanth Vijayaraghavan, Luyao Shi, Stefano Ambrogio, Charles Mackin, Apoorva Nitsure, David Beymer, and Ehsan Degan. 2024. Vhdl-eval: A framework for evaluating large language models in vhdl code generation. *Preprint*, arXiv:2406.04379.

Ngoc Phuoc An Vo, Brent Paulovicks, and Vadim Sheinin. 2024. Tackling execution-based evaluation for nl2bash. *Preprint*, arXiv:2405.06807.

John Yang, Akshara Prabhakar, Karthik Narasimhan, and Shunyu Yao. 2023. Intercode: Standardizing and benchmarking interactive coding with execution feedback. *Preprint*, arXiv:2306.14898.

Jiacheng Ye. 2022. Nl2bash. Accessed: 2024-08-20.

Ezri Zhu, Georgios Liargkovas, Michael Greenberg, and Konstantinos Kallas. 2024a. try utility. https://github.com/binpash/try. Accessed: 2024-09-25.

Wenhao Zhu, Hongyi Liu, Qingxiu Dong, Jingjing Xu, Shujian Huang, Lingpeng Kong, Jiajun Chen, and Lei Li. 2024b. Multilingual machine translation with large language models: Empirical results and analysis. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 2765–2781, Mexico City, Mexico. Association for Computational Linguistics.

A Prompts

Prompt

You will be given a task, two Bash commands, and the output of the two Bash commands. The first command is the ground truth. If the second command accomplishes the task, return true. Otherwise, return false. Only output 'true' or 'false'. Task: `natural_language_prompt`, *Ground Truth Command:* `ground_truth_command`, *Model Command:* `model_command`, *Ground Truth Command Output:* `ground_truth_command_output`, *Model Command Output:* `model_command_output`.

Figure 7: Prompt for evaluating the functional equivalence of Bash commands.

Prompt

Your task is to translate a natural language instruction to a Bash command. You will receive an instruction in English and output a Bash command that can be run in a Linux terminal. You will not output markdown or other formatting. You will not include additional information. `natural_language_prompt`

Figure 8: Prompt for NL2SH translation.

Prompt

Your task is to translate a natural language instruction to a Bash command. You will receive an instruction in English and output a Bash command that can be run in a Linux terminal. You will not output markdown or other formatting. You will not include additional information.

List all files in the current directory.
ls

Change the current directory to /home/user.
cd /home/user

Create a new directory named "new_folder".
mkdir new_folder

Remove the file named "old_file.txt".
rm old_file.txt

Copy the file "source.txt" to "destination.txt".
cp source.txt destination.txt

Move the file "file.txt" to the directory "/backup".
mv file.txt /backup

Display the contents of the file "example.txt".
cat example.txt

Show the current working directory.
pwd

Find all files with a ".log" extension in the current directory.
find . -name ".log"*

Display the first 10 lines of "sample.txt".
head -n 10 sample.txt

`natural_language_prompt`

Figure 9: Prompt for NL2SH translation with in-context learning.