# NL2BashCMD: State of The Art Evaluation of Natural Language to Bash Commands Translation

**Tommaso Sgroi**
Sapienza Università di Roma
sgroi.1852992
@studenti.uniroma1.it

**Damiano Gualandri**
Sapienza Università di Roma
gualandri.1892871
@studenti.uniroma1.it

**Luca Scutigliani**
Sapienza Università di Roma
scutigliani.1914401
@studenti.uniroma1.it

## Abstract

The task of translating natural language (NL) to Bash commands is a growing area of interest, particularly with the advent of large language models (LLMs). Currently, two major datasets are available: NL2Bash [20] and the more recent NL2CMD [14]. Much of the research so far has focused on developing models with higher accuracy, but even state-of-the-art transformer-based models have only managed to achieve around 50% accuracy.

However, a significant breakthrough was made in 2023, when a study applied ChatGPT [10] to the NL2Bash dataset, achieving a new state-of-the-art (SOA) with an accuracy score of 80.6% in zero-shot conditions.

In this paper, we contribute to the research on NL to Bash command translation by evaluating recent state-of-the-art LLMs on the NL2Bash and NL2CMD datasets.

## 1 Task description/Problem statement

The task addressed in this paper is the automatic translation of natural language (NL) instructions into executable Bash commands. This task belongs to the broader field of natural language processing (NLP) and specifically to the area of semantic parsing, where the goal is to map NL utterances into formal representations that can be executed by machines[16].

In our case, the formal representations are Bash commands, commonly used to perform tasks such as file manipulation, text processing, and system administration in Unix-based operating systems. Translating NL into Bash commands can significantly lower the barrier for users unfamiliar with command-line interfaces, enabling them to interact with systems more efficiently and intuitively.

This task is formally defined as follows: given an input sequence $x = (x_1, x_2, \ldots, x_n)$ consisting of a natural language instruction, the goal is to generate an output sequence $y = (y_1, y_2, \ldots, y_m)$ consisting of a valid Bash command that accurately reflects the intention of the NL input. The challenge lies in understanding the underlying semantics of the natural language input and correctly translating it into the syntactically rigid and concise Bash command format.

Despite advancements in model architecture, achieving high accuracy in this task remains difficult due to the ambiguous nature of natural language and the strict syntactic requirements of Bash commands. Most existing models struggle to surpass 50% accuracy, indicating that there is still significant room for improvement.[10] This paper seeks to address these challenges by exploring the application of large language models (LLMs) like ChatGPT.

### 1.1 Examples

To illustrate the task of translating natural language (NL) instructions into Bash commands, consider the following examples from Tellina NL2Bash[20] and NL2CMD[14] datasets.
These examples demonstrate how various NL instructions can be converted into corresponding Bash commands. The goal of the translation model is to accurately produce these commands based on the given natural language inputs, ensuring that the generated commands perform the desired operations as intended.

#### 1.1.1 Tellina NL2Bash - Examples

**NL**: *Calculate and output the amount of CPU load taken by all processes belonging to user 'abc'.*
**Bash**:

```
top -b -n 1 -u abc | awk 'NR>7 ↙
    { sum += $9; } END { print ↙
    sum; }'
```

**NL**: *Append *.java files from the current directory tree to tar archive 'myfile.tar'*
**Bash**:

```
find . -type f -name "*.java"   ↙
   | xargs tar rvf myfile.tar
```

**NL**: *Removes 'folderName', and removes all content within if 'folderName' is folder.*
**Bash**:

```
rm -rf folderName
```

#### 1.1.2 MagnumResearchGroup NL2CMD - Examples

**NL**: *Find all files with the extension .swpand delete them permanently.*
**Bash**:

```
find / -name '*.swp' -exec rm   ↙
   -rf {} \\;
```

**NL**: *Print "Random number chosen: [a random number between 1 and 1000]".*
**Bash**:

```
shuf -i 1-1000 -n 1 | xargs   ↙
   echo "Random number chosen:  ↙
    {}"
```

### 1.2 Real-world Applications

The ability to translate natural language into Bash commands has several impacts in real-world applications:

- **Enhanced User Interaction**: By enabling users to interact with Unix-based systems through natural language, non-technical users can perform system administration tasks without needing to learn Bash syntax. This lowers the barrier to the effective use of powerful command-line tools.
- **Automated System Management**: In enterprise environments, translating NL instructions to Bash commands can automate routine system management tasks, such as file backups, system monitoring, and log analysis. This reduces manual intervention and minimizes the risk of human error.
- **Educational Tools**: Tools that convert NL to Bash commands can serve as educational aids, helping students and new users learn command-line operations by providing instant feedback on their natural language queries.
- **Software Development and Debugging**: Developers can use NL-to-Bash translation for scripting and automating repetitive tasks during the development and testing phases. It can also assist in debugging by allowing developers to quickly issue commands based on descriptive error messages or logs.
- **Integration with Cloud Services**: In cloud computing environments, where command-line interaction is common, translating NL to Bash can simplify the management of cloud resources and services, allowing users to manage their cloud infrastructure using natural language commands.

These applications highlight the impact of such technology, it can enhance the efficiency, accessibility, and user experience in Unix-like systems.

## 2 Related work

Early efforts to tackle the challenge of translating natural language (NL) into executable commands made use of Recurrent Neural Networks (RNNs) [20]. These RNN-based approaches achieved notable success in generating shell scripts, which set a foundational benchmark for subsequent models in this domain.

Another significant advancement was the development of the Tellina system [20], which employed Gated Recurrent Units (GRUs) [13], a variant of RNNs, to perform NL-to-Bash command translation. Tellina achieved an accuracy of 13.8% according to the NLC2CMD evaluation metrics [9] and contributed the NL2Bash dataset [20], which became a benchmark for further research in this field. This GRU-based model held state-of-the-art performance for a considerable period until newer models were introduced.

The focus of subsequent research shifted towards Transformer-based architectures, which generally offer superior accuracy and training efficiency compared to traditional RNNs [11]. These Transformer models have been effectively applied to NL-to-Bash translation tasks, demonstrating substantial improvements over previous RNN and GRU-based approaches [15].

The NLC2CMD competition, held at NeurIPS 2020 [9], marked a significant milestone by setting a new state-of-the-art in NL-to-Bash command translation. The competition's winning team, Magnum, utilized Transformer models for both encoding and decoding, achieving a remarkable accuracy increase from 13.8% with GRUs to 53.2% [15][14]. This development underscored the effectiveness of Transformer models in this area. The final leaderboard, shown in Table 1, presents the top-performing teams based on accuracy, energy consumption, and latency. Another interesting result was to test ChatGPT on the Tellina dataset, achieving 80.6% accuracy.

Transformer models have thus emerged as the leading approach for NL-to-command translation, significantly outperforming RNNs and GRUs [11]. This paper builds on the advancements made during the NLC2CMD competition by evaluating recent state-of-the-art large language models (LLMs) on the NL2Bash and NL2CMD datasets, as well as fine-tuning smaller models to achieve optimal performance.

**Table 1:** Final leaderboard for the NLC2CMD competition, showing the accuracy score, energy consumption, and latency for every invocation[9].

| Team Name | Accuracy Score | Energy (Joules) | Latency (sec) |
|---|---|---|---|
| **Magnum** | **0.532** | **0.484** | **0.709** |
| Hubris | 0.513 | 12.037 | 14.870 |
| Jb | 0.499 | 2.604 | 3.142 |
| AICore | 0.489 | 0.252 | 0.423 |
| AINixCLAISimple | 0.429 | N.A. | 0.010 |
| coinse-team | 0.163 | 343.577 | 0.452 |
| Tellina | 0.138 | 2.969 | 3.242 |

## 3 Datasets and benchmarks

We have 2 major datasets, one is the Tellina dataset[20] and the other is the Magnum dataset[14].

### 3.1 Tellina Dataset (NL2Bash)

The Tellina dataset was compiled by collecting Bash commands from various online sources, including question-answering forums, tutorials, tech blogs, and educational materials. Each command in the dataset is accompanied by a single-sentence description provided by an expert. The dataset comprises approximately 10,000 one-liner Bash commands, which collectively cover 102 distinct utilities and 206 flags, thus offering a diverse and comprehensive range of functional capabilities.

### 3.2 Magnum Dataset

The Magnum Research Group[14] developed two datasets for this task. The first dataset, containing 71,705 descriptions paired with valid Bash commands, was automatically generated by a transformer-based model using the following steps (however, this dataset was not accessible at the time of this paper):

- **Manual Page Scraping**: Data was scraped from manual pages for 38 utilities to determine their syntax, flags, and parameters.
- **Command Generation and Validation**: Using the syntactic rules, flags, and arguments, the commands were generated through different combinations of flags and piped commands. Then, placeholder arguments were replaced with actual arguments and executed on a virtual machine discarding the incorrect ones.
- **Back-translation**: A transformer-based model generated natural language descriptions from the validated Bash commands, forming NL-Bash pairs.

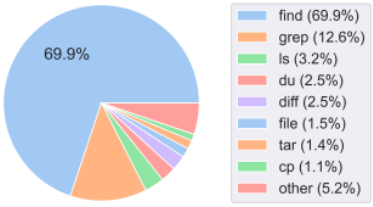


**Figure 1:** Utility Distribution in the Transformer Generated Dataset.

But this approach leads to some problems as they highlighted in section 6.7 "*Comparison to Existing Dataset*"[14]:

> This strategy resulted in a heavily biased dataset with little to no coverage of some powerful—yet uncommon—flags, as shown in Figure 8* depicting the utility distribution in the original NL2Bash dataset. Our generated dataset did not prioritize popularity and treated all flags equally.

*Figure 1 in this paper.

Despite these issues, the dataset remains valuable, as its distribution closely mirrors that of the original NL2Bash dataset. From now we will refer to this dataset as "NLC2CMD".
Their second dataset was generated via ChatGPT (section 8 "*Rethinking NL2CMD in the age of ChatGPT*"[14]), using the prompt "*Generate bash command and do not include example*" and temperature 1 to have the maximum variability. After they validated the scripts, they made ChatGPT also generate the descriptions of the scripts, with 0 temperature, yielding a dataset of 17'050 entries. They registered no performance drop on this dataset and realized that was of higher quality than the previously generated dataset.

### 3.3 Benchmarks

The NeurIPS2020 competition gave us a complete overview of the benchmarks of the state-of-the-art models on the Tellina dataset. As they report (in sec, the data were partitioned as follow:

- **Training**: participants were provided of the NL2Bash dataset and the man pages of Ubuntu 18.04 LTS[2].
- **Validation and Test**: the test set was initially created by randomly sampling 1,000 data samples from a filtered version of the NL2Bash dataset. In subsequent phases of the competition, approximately 800 new data samples were progressively added to the test set, bringing the total to over 1,800 samples.

The addition of new samples helped ensure the inclusion of Bash utilities not seen in the training data, thus testing the generalization ability of competing algorithms (see section 4.4 "*Data partitions and pipeline*"[9]).

#### 3.3.1 Metric

From all the metrics they propose to evaluate the scripts, they adopted a scoring mechanism that specifically checks for structural and syntactic correctness. As they described in the NeurIPS2020[14] paper: "*incentives precision and recall of the correct utility and its flags, weighted by the reported confidence*".
As the Magnum group describes the metric[14]:

> It defines two terms: **Flag score** $S_F^i$ and **Utility score** $S_U^i$. As shown in Equation 1[9], the flag score is defined as twice the union of reference flags and predicted flags number minus the intersection, scaled by the max number of either reference flags or predicted flags.
>
> The range of flag scores is between -1 and 1.
>
> As shown in Equation 2[9], the utility score is defined as the number of correct reference utilities scaled by capping flag score between 0 and 1, minus the number of wrong utilities, scaled by the max number of either reference utilities or predicted utilities.

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

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

This means that by summing all the utility scores within a predicted command, the final normalized function will be $f : S_U \rightarrow [-1, 1]$.
The metric is also expected to get $K$ predictions (they set $K = 5$ during the competition) and calculate the final score applying the following function, as shown in [9]:

$$Score(A(nlc)) =$$

$$= \begin{cases} \max\limits_{p\epsilon A(nlc)} S(p), \text{if } \exists_{p\epsilon A(nlc)} \text{ such that } S(p) > 0, \\ \frac{1}{|A(nlc)|} \sum\limits_{p\epsilon A(nlc)} S(p), \text{otherwise.} \end{cases}$$

#### 3.3.2 Results

As shown in table 1 and table 2 of the NeurIPS competition's final leaderboard, the top performing state-of-the-art models barely overcome the 50% of accuracy (see section 3.3.1). The Magnum Research Group also tried other ways, they tested their model on their first generated dataset (NLC2CMD), also in combination with the original NL2Bash dataset (table 3). The results highlighted some problems; as we can see, training the model on the Magnum NLC2CMD dataset and testing on the NL2Bash dataset, or vice versa, we

| Team | Model | Accuracy |
|---|---|---|
| Magnum | Transformer | 0.532 |
| Hubris | GPT-2 | 0.513 |
| Jb | Classifier+Transformer | 0.499 |
| AICore | Two-stage Transformer | 0.489 |
| Tellina | BRNN (GRU) | 0.138 |

**Table 2:** Comparison of team models and accuracy scores.

register a huge performance drop. A more accurate error analysis showed that the increased error rate in the new dataset was attributed to a larger variety of utilities, particularly less common ones, and the complexity of piped commands.
Another interesting result was the benchmark done on the ChatGPT-generated dataset (see section 3.2) augmenting the original NL2Bash dataset. They registered no significant performance drop, suggesting that the newly generated dataset has a higher quality than the previous generation method. Also, they tested ChatGPT on the original NL2Bash dataset achieving 80.6% of accuracy score (see section 3.3.1).

**Table 3:** Comparison of Magnum's Model Performance Across Datasets, Tellina is NL2Bash, NLC2CMD in the artificially generated dataset described in section 3.2

| Training Dataset | Test Dataset | Accuracy |
|---|---|---|
| NL2Bash | NL2Bash | 52.3% |
| NLC2CMD | NLC2CMD | 31.6% |
| NLC2CMD | NL2Bash | -13% |
| NL2Bash | NLC2CMD | -6.67% |
| NLC2CMD+NL2Bash | NL2Bash | 48% |
| NLC2CMD+NL2Bash | NLC2CMD | 31.7% |

## 4 Existing tools, libraries, papers with code

It is good to know that most of the previously discussed papers and works in section 3 have been implemented and their source code publicly released.
The vast majority of the projects discussed here based on Deep Learning techniques have been implemented with PyTorch or TensorFlow. Other libraries were used, like the classic "NTKL" and "sklearn". Moreover, they are available on GitHub and can be integrated into other experiments using the transformers library [4].
The Tellina NL2Bash GitHub repository [20], comes with the associated dataset and utility function to use it. It also provides the model provided in the paper.
**Evaluation Tools.** We plan to evaluate the models using the metric on the official IBM GitHub repository of the competition[19], which will allow us to avoid inconsistencies and evaluate the models fairly with the challenge participants. To make this project run properly, we noted all Python's packaged to use in a virtual environment, including all the necessary dependencies that are not accessible via packet managers.
**OpenAI API.** To access the new generation of GPTs, we plan to use the OpenAI's[6] API endpoints to run our evaluations.
**Together AI.** To access larger Llama models we plan to use the Together AI's API endpoints.
**Unsloth AI.** To run the benchmarks with smaller LLMs we plan to use the Unsloth library [1].
**Datasets.** There are 3 possible datasets, but only 2 are available, the NL2Bash[20], the Magnum's generated datasets[14]. Unfortunately, the first generated dataset of the Magnum Research Group is not available at the time of this paper, so we can access only the GPT-generated one.
**Models.** We plan to use models from the online available libraries Hugging Face [5].

## 5 State-of-the-art evaluation

- **Magnum Transformer:** The Magnum Transformer is a state-of-the-art model developed for translating natural language (NL) instructions into Bash commands. It was used in the NLC2CMD competition, where it achieved significant accuracy improvements over previous models.

- **ChatGPTs:** It has been trained on a large textual dataset, which includes books, articles, websites, and other online resources, to understand and generate natural language in different languages and contexts. Its training is based on Generative Pretrained Transformer (GPT) series models, which use deep neural networks to predict the next word in a sequence of text. We used both GPT4o and GPT4o-mini: the difference is in the size of the model: GPT4o-mini is a smaller and lighter version of GPT4o, with fewer parameters, which can be used in contexts with limited computational resources, sacrificing some performance and accuracy.
  In particular, as OpenAI(2023) mention in Its paper[21]:

  > GPT-4 is a Transformer-style model [23] pre-trained to predict the next token in a document, using both publicly available data (such as internet data) and data licensed from third-party providers. The model was then fine-tuned using Reinforcement Learning from Human Feedback (RLHF)[12].

This means that GPT4 may have already seen the dataset.

- **LLaMAs:** Family of pre-trained models on a diverse set of data, including English CommonCrawl, GitHub repositories, and academic sources like Arxiv. LLaMA is strongly exposed to code (GitHub), therefore it improves its ability to handle command line syntax, making it a strong candidate for our task.

- **Mistral:** Mistral 7B is a 7.3 billion parameter transformer-based language model, released on September 27, 2023, under the Apache 2.0 license. It reportedly outperforms LLaMA 2 13B and matches LLaMA 34B on various benchmarks. The model employs grouped-query attention (GQA) for efficient attention computation.
  Their training set is not publicly available, but as they mentioned [3] it is trained on the web.

- **Gemma:** Gemma[17] is a family of lightweight, state-of-the-art open models built from the same research and technology used to create the Gemini models. It has been developed by Google DeepMind and other teams across Google.

## 6 Comparative evaluation

### 6.0.1 Datasets

To run our comparative results we decided to test our models in the Tellina and Magnum datasets.
**Tellina**. Expert-based dataset of natural language to bash commands.
**Magnum ChatGPT**. ChatGPT automated augmentation of the Tellina dataset.

### 6.0.2 Models evaluated

We evaluated the following models on the previous datasets:

- **GPT-4o.** OpenAI latest state-of-the-art chat LLM.

- **GPT-4o-mini.** OpenAI lightweight version of ChatGPT.

- **Llama-3.1-8B.** The base not quantized Meta's model with 8 billion parameters with full precision.

- **Llama-3-70B.** The base not quantized Meta's model with 70 billion parameters with full precision.

- **Llama-3-8B.** The base not quantized Meta's model with 8 billion parameters with full precision.

- **CodeLlama-7B.** The fine-tuned version of Llama 2 for coding, with 7 billion parameters.

- **Mistral-7B-v0.3.** Mistral base model 7 billions of parameters, full precision.

- **Mistral-7B-instruct-v0.3.** Mistral fine-tuned instruct model 7 billions of parameters, full precision.

- **Gemma-2b.** The base Google's Gemma model has 2 billion parameters with full precision.

- **CodeGemma-2b.** Google's Gemma model fine-tuned for code generation has 2 billion parameters with full precision.

We get the models from the famous platform HuggingFace, we make sure that we get the models from the original authors, or in another case, we have assured to avoid fine-tuned models (if not otherwise specified). So, we preserved the consistency and power of the original models.

### 6.0.3 Metrics and evaluation protocol

To fairly evaluate the models with the current state of the art, we used the same metric from the official IBM NL2Bash competition [19]; we extracted the metric utility from the archived (read-only) repository to avoid potential errors in case we rewrote it ourselves.
Due to the high computational resources of the models we chose to use models both from libraries that allow to use of CUDA devices, and both API to send requests. The devices we used were a "NVIDIA RTX 3060" and a "NVIDIA RTX 3090", the service we used to access the API was "TogetherAI"[8] and "OpenAI"[6]. About the libraries we used particularly, we used Unsloth[1], because by using it we were able to get the model's confidence since the metric requires it.
The evaluation protocol was simple, all the LLMs were tested under zero shots conditions (if not specified otherwise), using a base prompt that allows the model to complete the bash command:

*Task: Convert the following descriptions into bash commands.*
*Description: {}.*
*Bash command:*

Substituting the "{}" with the corresponding description and command.
For **chat** LLMs we modified the prompt to make the models able to achieve the desired results.

> *You are a professional bash command writer. Your sole task is to write a bash command based on the description provided by the user.*
> *Your output must be strictly the command itself, no explanations or additional information are allowed—only the command.*

To extract the model's confidence, we used two methods in two different scenarios. The first scenario was using the OpenAI compatible API endpoints, which returned the log probabilities for each token; so we extracted the command confidence by doing the average of probability over all tokens.

$$Confidence(S) = \frac{1}{N} \sum_{tk \in S} \exp \log(p_{tk})$$

The second scenario was local inference with Unsloth, where we were able to get the logits and the associated log probabilities; then we calculated the average confidence with the above function.
**Benchmarks.** For all benchmarks we stored the prediction, confidence, and ground truth for each entry and the computed score.
**Hyperparameters.** During the benchmarks we used the temperature of 0.2 (if not specified otherwise) to achieve outputs with higher confidence without sampling every time from only the most probable tokens.

#### 6.0.4 Dataset description

When we started the benchmarks we noticed two things:

- In the original IBM repository[19] there was an update of the Tellina dataset, where were fixed syntax issues and incompatibility commands.
- The Magnum dataset, presented ≈ 1000 entries that contained hallucinations[18] or the command in the description.

We removed the bad Magnum entries from the dataset manually.

### 6.1 Results

The results of the benchmarks were extremely *disappointing*.

We used "zero-shot conditions"; in this approach, the model makes predictions without having seen any specific examples of the tasks in the context. Instead of generating using examples, the model relies on semantic relationships or other auxiliary information to predict the correct bash command.

Before giving a look at the results (table 4), here are some important notes:

- We didn't test Llama-3-8b on the Magnum dataset; since we ran it only with Tellina with a really low score and we've been using a pay-to-use service. We got a low score with Tellina, therefore we cannot confirm its value but we estimate is not so good, like the others.
- We didn't test Llama-3-70b on the Magnum dataset; since the high price of the model on pay-to-use service. The results were decent, but still not so valuable to test it on an even larger dataset.
- The Magnum dataset was not subjected to testing with ChatGPT4o and ChatGPT4o-mini, as it was created by an earlier version of the same model, rendering the benchmarks invalid.

| MODEL | TELLINA | MAGNUM |
|---|---|---|
| `Llama3-70B-base` | 0.170 | - |
| `GPT4o`* | 0.117 | - |
| `GPT4o-mini`* | 0.074 | - |
| `GPT4o-mini`† | 0.024 | - |
| `Llama 3-8b`§ | 0.044 | - |
| `CodeLlama 7b-instruct` | 0.023 | 0.026 |
| `Mistral 7b-v0.3` | 0.022 | 0.022 |
| `MetaLlama 3.1-8B` | 0.014 | 0.014 |
| `Mistral 7b-instruct-v0.3` | -0.515 | -0.518 |
| `Gemma 2B`** | - | - |
| `CodeGemma 2B`** | - | - |

**Table 4:** LLM's benchmarks results.

*We set the temperature to 1.25 of 2.0 to achieve more variety of consistent generations, according to the OpenAI API documentation (august-2024) [7].
†We set temperature 0.8 of 2.0 see footnote*.
§We set temperature 0.8 of 1.0 .
**The generation were completely inconsistent, no command was correct.

From the results, it is evident that different models show varying degrees of performance on the nl2bash task. For instance, the `Llama3-70B-base` model achieves a relatively high score of **17%**, indicating poor performance but overall it is the highest. Similarly, models such as `GPT4o` and `GPT4o-mini` exhibit positive but modest scores, respectively 0.117 to 0.074. These results suggest that the models are generally capable of handling the task, albeit with varying success, and the best results are obtained with the Tellina dataset. LLama3-70b is a winner in this context, and the second-best result was obtained by using generic GPT, despite the existence of CodeLlama, fine-tuned for code.

On the other hand, some models, such as `Mistral-7b-instruct-v0.3`, show negative performance across multiple datasets, with scores as low as -0.518.

### 6.1.1 Considerations

Several considerations arise from the analysis provided in *Appendix B* of the NeurIPS paper [9]. The authors identified some key limitations in their models:

- **Limited understanding of utility flags:** The models often lack a comprehensive understanding of utility flags. For instance, when given the prompt "copy a folder", Team Magnum's model incorrectly generated *cp File File*. While it correctly identified the "cp" command, it failed to include the necessary '-R' flag, which is essential for recursively copying a directory and its contents.
- **Contextual knowledge in Bash commands:** Effective operation in the Bash terminal requires more than just knowledge of utilities and flags. The models often fail to encode the necessary contextual knowledge, leading to incorrect responses. For example, understanding the significance of files like 'etc/passwd', which contains the list of system users, is critical for accurately fulfilling certain prompts.

However, it is important to note that large language models (LLMs) that scored positively, such as *Llama3 8B*, were less affected by these limitations. The dataset was highly biased towards the *find* command, covering more than 60% of the ground truths. Despite this, LLMs demonstrated a better understanding of flags and utilities. For example:

**Command description:** *Copy all .png files from the home directory tree to imagesdir/*

**Ground Truth:**

```
find . -type f -name "*.png" - ↙
   exec cp {} imagesdir/ \\
```

**Llama3 8B predictions:**

```
cp -r \$HOME/*.png imagesdir/
```

```
find ~ -name '*.png' -exec cp   ↙
   {} imagesdir/ \\;
```

Other models similarly demonstrated improved usage of flags and utility functions, indicating a better grasp of these elements.

In conclusion, while most LLMs produced high-quality outputs, they were unfortunately penalized by certain limitations in the evaluation process. Compared to the models proposed in the NeurIPS paper, the LLMs showed significantly better context awareness.

### 6.1.2 Training Considerations

There are additional considerations to keep in mind, with the most significant being the use of different evaluation metrics for LLMs compared to the models used by NeurIPS participants. The competition models were trained specifically with the chosen evaluation metric[†], which led to a specialization in utility and relative flags, ignoring completely context and correctness. In contrast, LLMs are trained with a variety of metrics, which can affect their performance and generalization.

## 6.2 Discussion

Despite the advancements in natural language to Bash command translation, several limitations and errors persist in the evaluated systems.

### 6.2.1 Limitations of Compared Systems

1. **The Metric:** As was discussed in section 6.1.1, a key difference between LLMs and NeurIPS competition models is their use of evaluation metrics. NeurIPS models were optimized for specific metrics, focusing on utility flags but neglecting context and correctness. LLMs, trained with a range of metrics,

[†]See section 3.3.1

exhibited better generalization and context understanding, although this varied based on the metrics used.

2. **Contextual Understanding:** Smaller models have been seen that have less contextual understanding, for example, 8 billion parameters models could not properly understand the task (due to the zero-shot condition), while the Gemmas 2 billion parameters models couldn't even understand the task, generating inconsistent outputs.

3. **Dataset Bias:** The datasets used for training and evaluation, including NL2Bash and NL2CMD, have inherent biases. For example, the NL2CMD dataset may not fully represent all possible commands or flags due to its automated generation process, leading to reduced generalization capabilities of the models.

4. **Command Complexity:** Smaller models often struggle with complex Bash commands involving multiple utilities and flags. They are more affected by hallucinations and biases. Bigger models (as GPTs) don't have the same issue but may prefer other utilities to reach their goal.

#### 6.2.2 Committed Errors

1. **Syntax and Structural Errors:** In particular smaller models, as Gemma and CodeGemma 2B or 8B models, may generate commands with incorrect syntax or structure due to hallucinations or inability to capture the nuanced syntax rules of Bash commands in particular descriptions. Additionally, we noted that many times one command was correctly predicted but incorrectly flagged as wrong, due to the metric limitations.

2. **Flag and Argument Mismatches:** Errors in predicting the correct flags or arguments for a given utility are common. This occurs because models might prefer the usage of other flags combined with other utilities despite the ground truth.

3. **Ambiguity in Natural Language:** The ambiguity and variability in natural language expressions often lead to errors in translation. For example, different phrasing of the same command might result in varied outputs that may not match the intended command precisely.

## 7 Conclusions

In this paper, we evaluated recent state-of-the-art large language models (LLMs) for the task of translating natural language instructions into Bash commands, focusing on the NL2Bash and NL2CMD datasets. Our analysis included evaluating pre-trained models to explore their effectiveness and efficiency.
The application of LLama-3-70B-base to the NL2Bash dataset achieved accuracy of 17% in zero-shot conditions, demonstrating the presence of a good concurrent against the dominance of ChatGPT.

Despite improvements, the NeurIPS competition models still face challenges with command complexity and context understanding[9]. Performance varies significantly based on the dataset used for training and evaluation, highlighting the need for more comprehensive and diverse datasets.

In contrast to the Magnum research group [14], which reported successful results of 80.6% accuracy with the use of ChatGPT, and 52.3% with two Transformer models as encoder-decoder; we encountered significant challenges even when using more recent and advanced versions of ChatGPT (ChatGPT4o and ChatGPT4o-mini). These models struggled to outperform the older Tellina Gated Recurrent Unit model [20], which achieved a performance of 13.8%. Assuming there were no errors in our evaluation process, it is important to note that we utilized the official NeurIPS repository [9][19]. This discrepancy could suggest that the earlier version of ChatGPT may have overfitted to the Tellina dataset. Additionally, while we attempted to replicate the inference steps used by the Magnum research group, their documentation did not specify key hyper-parameters, such as the temperature, which may have influenced the results.

All code and benchmarks results can be found in our Github repository[22].

#### 7.0.1 Future Work

**More Benchmarks:** By doing more benchmarks we can have a wider overview of the state-of-the-art LLMs on the natural language to Bash

translation.

**Other Accuracy Scores:** To assess the real performance of various LLMs and Magnum Transformer Model, we can evaluate their performance using different accuracy metrics that can provide a more comprehensive understanding of their capabilities. While a specific metric might highlight the strengths of a model in certain areas, it could also obscure weaknesses in others.

**Evaluation Metrics** Refining evaluation metrics to better understand command correctness and utility will provide more accurate assessments of model performance. This includes developing metrics that account for the practical execution and effectiveness of generated commands.

**Finetuning** By training the model on a smaller, task-specific dataset after its general training, fine-tuning allows models to better understand user requests, improving accuracy and performance in generating complex or domain-specific commands.

By addressing these areas, future research can further advance the field of natural language to Bash command translation, leading to more robust and reliable systems for anyone who needs to write simple or complex Bash scripts.

### Contributions.

- **Tommaso Sgroi**: Python developer, benchmarks and models evaluation, information gathering and analysis, paper review, draft, and supervision.
- **Damiano Gualandri**: Information gathering and analysis, paper review, draft, and supervision.
- **Luca Scutigliani**: Benchmarks.

## References


[1] Unsloth ai - github repository. https://github.com/unslothai/unsloth. 5, 6

[2] Ubuntu 18 man pages - canonical, 2018. 4

[3] Mistralai - training data? https://huggingface.co/mistralai/Mistral-7B-v0.1/discussions/8, Sep 2023. Community Discussions. 6

[4] Hugging face transformers. https://huggingface.co/docs/transformers/index#-transformers, august 2024. 5

[5] Llama library. https://ollama.com/library, 2024. 5

[6] Openai. https://openai.com/about/, 2024. 5, 6

[7] Openai api reference. https://platform.openai.com/docs/api-reference, 2024. 7

[8] Together.ai. https://www.together.ai/, 2024. 6

[9] Mayank Agarwal, Tathagata Chakraborti, Quchen Fu, David Gros, Xi Victoria Lin, Jaron Maene, Kartik Talamadupula, Zhongwei Teng, and Jules White. Neurips 2020 nlc2cmd competition: Translating natural language to bash commands. In Hugo Jair Escalante and Katja Hofmann, editors, *Proceedings of the NeurIPS 2020 Competition and Demonstration Track*, volume 133 of *Proceedings of Machine Learning Research*, pages 302–324. PMLR, 06–12 Dec 2021. 2, 3, 4, 8, 9

[10] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, 2020. 1

[11] Isaac Caswell and Bowen Liang. Recent advances in google translate, 2020. Accessed: 2024-08-16. 2, 3

[12] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, 2017. 5

[13] Junyoung Chung, Çağlar Gülçehre, Kyunghyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *ArXiv*, abs/1412.3555, 2014. 2

[14] Quchen Fu, Zhongwei Teng, Marco Georgaklis, Jules White, and Douglas C Schmidt. Nl2cmd: An updated workflow for natural language to bash commands translation. *Journal of Machine Learning Theory, Applications and Practice*, pages 45–82, 2023. 1, 3, 4, 5, 9

[15] Quchen Fu, Zhongwei Teng, Jules White, and Douglas C. Schmidt. 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, 2021. 2, 3

[16] P GNU. Free software foundation. bash (3.2. 48)[unix shell program], 2007. 1

[17] Google. Google gemma. https://blog.google/technology/developers/gemma-open-models/, 2024. Accessed: 2024-09-02. 6

[18] Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, and Ting Liu. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions, 2023. 7

[19] IBM. Nlc2cmd challenge, command line artificial intelligence clai. https://github.com/IBM/clai/tree/2ad172acbed0c1ec870cc39f47635adea39f19c0, 2020. Accessed: 2024-08-16. 5, 6, 7, 9

[20] Xi Victoria Lin, Chenglong Wang, Luke Zettlemoyer, and Michael D. Ernst. Nl2bash: A corpus and semantic parser for natural language interface to the linux operating system. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation LREC 2018, Miyazaki (Japan), 7-12 May, 2018.*, 2018. 1, 2, 3, 5, 9

[21] OpenAI(2023). Gpt-4 technical report, 2024. Full authorship contribution statements appear at the end of the "GPT-4 Technical Report" document. 5

[22] Damiano Gualandri Tommaso Sgroi. Nl2bashcmd-llm. https://github.com/Tommaso-Sgroi/NL2BashCMD-LLM, 2024. 9

[23] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017. 5