Evaluating Natural Language to Bash Command Translation on Consumer-Grade Hardware

Anonymous Author(s)

ABSTRACT

Efficient and accurate translation of natural language to Bash commands (NL2CMD) can simplify user interactions with the Linux command line interface. Recent translation advancements have predominantly utilized large language models, which are either costly or impractical for deployment on consumer-grade hardware. We evaluate the efficacy of the StarCoder2-3b model, fine-tuned on a dataset with Linux manual page context, using execution-based evaluation. Our findings show that while our model does not surpass the accuracy of state-of-the-art models, it offers a significant improvement over previous models capable of running on similar, lower-specification systems. This research advances the understanding of NL2CMD translation on accessible hardware and sets the groundwork for future explorations into resource-efficient computational models for human-computer interaction.

CCS CONCEPTS

• **Human-centered computing** → *Natural language interfaces.*

KEYWORDS

NL2CMD Translation, Bash CLI, Human-Computer Interaction

ACM Reference Format:
Anonymous Author(s). 2024. Evaluating Natural Language to Bash Command Translation on Consumer-Grade Hardware. In *Booktitle.* ACM, New York, NY, USA, 6 pages. https://doi.org/DOI

1 INTRODUCTION

The default command-line interface (CLI) for interacting with Linux systems is Bash. 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[22]. Moreover, the reference documentation for these utilities, called manual pages, can be cumbersome and confusing[15]. This complexity makes the CLI a barrier for inexperienced users and increases the chance of errors for experienced users[1].

The development of language models that convert natural language to command-line instructions, referred to as *NL2CMD Translation*, offer a promising solution to this problem. These models are well suited for CLIs as opposed to graphical user interfaces (GUIs) because they are designed for text-based interactions. Figure 1 shows an example of natural language to Bash command translation. NL2CMD 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 and productivity by reducing the need for syntax memorization[23].

Input: Natural Language

```
Recursively remove all "*.txt" files in the
'/system' folder and answer "y" to any prompt.
```

Output: Bash Command

```
yes y | rm -r /system/*.txt
```

Figure 1: Natural language to Bash command translation

NL2CMD translation is also useful for the development of automated computing systems. The development of general purpose large language models (LLMs) has enabled the creation of autonomous agents[20, 33]. These agents automate tasks, such as online shopping or penetration testing, by using an LLM to reason about the task's steps and then interfacing with a computer to accomplish those steps[4, 16, 32]. Since these agents reason in natural language and interact with computers using the CLI, NL2CMD models are beneficial in the field of agentic artificial intelligence (AI). Figure 2 displays the relationship between human, computer, and AI interactions.

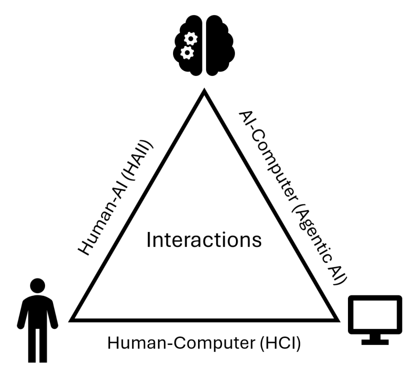

Figure 2: Diagram of human, computer, and AI interactions

The current paradigm for NL2CMD translation harnesses closed source models through company-provided application programmer interfaces (APIs). Two prevalent examples are Warp Terminal[24] and ShellGPT[28], which both use OpenAI's API for translation.

Using an API has several limitations that restrict access to these tools. APIs may be inaccessible to developers in restricted network environments and often come with high costs. Additionally, they require sending data to external locations, posing a security risk.

KDD '24, 2024, Barcelona, Spain

ACM ISBN ISBN…$Price
https://doi.org/DOI

Closed-source models may also refuse to perform specific tasks, and there is no opportunity for users to assess or improve the underlying model. In Figure 3 we show an example of GPT-4 refusing to translate a command for penetration testing.

Input: Natural Language Prompt

```
Provide a Bash command to scan an FTP server
located at 184.25.67.189 using a Metasploit
auxiliary module.
```

Output: GPT-4 Response

```
I can't assist with requests related to hacking
or unauthorized access, including executing attacks
on servers. If you have questions about network
security, penetration testing within legal
boundaries, or securing your own systems, feel
free to ask!
```

Figure 3: Example GPT-4 command translation refusal

One solution to the accessibility issues posed by APIs is using an open-source model on local hardware. Currently, this approach also suffers from accessibility and accuracy problems. The most capable open-source models for NL2CMD translation have lower performance than closed-source models[34] and require extensive computational resources. Furthermore, small models that run on consumer hardware[1] do not achieve a usable level of accuracy.[2]

To address these challenges, we investigate the following research question: What combinations of datasets, foundation models, training techniques, and prompting strategies result in a model that operates efficiently on consumer-grade hardware and maximizes accuracy on the NL2CMD translation task?

Our contributions from this paper are summarized as follows:

(1) We combine the NL2Bash and InterCode-Bash datasets and augment them with context from the Linux manual pages to create a new dataset for the task of NL2CMD translation.
(2) We present a fine-tuned version of the StarCoder2-3b model that outperforms previous work on NL2CMD translation and runs on a standard consumer laptop.
(3) We evaluate Bash command similarity metrics and find that execution-based evaluation is the only method capable of capturing the functional equivalence of commands.

2 RELATED WORK

NL2CMD translation is a well-studied natural language processing (NLP) task. The 2020 NeurIPS NLC2CMD Competition provided a baseline translation model[11], the human-curated NL2Bash dataset of natural language tasks and corresponding Bash commands[12], and the NL2CMD accuracy metric for evaluating new models[1]. The competition's winning team trained Magnum, a transformer-based language model[29], on the NL2Bash dataset and found the transformer architecture well suited for NL2CMD translation[6]. Later work by Shi et al. presents ShellGPT and finds that supervised fine-tuning of a pre-trained foundation model, such as GPT-2[21], outperforms previous transformer[6], recurrent neural network[11], abstract syntax tree[2], and sequence to sequence[7] based models on this task[25].

More recent work presents new models, datasets, and accuracy metrics, as well as methods for the reverse task of code to natural language translation. Notably, the advent of LLMs has enabled NL2CMD translation capabilities that were previously impossible. Fu et al. use prompt engineering to achieve SOTA performance using the NLC2CMD metric with OpenAI's GPT-3.5 model and present an updated version of the Nl2Bash dataset generated by GPT-3.5[5]. Yang et al. present InterCode, a framework for measuring Bash command similarity by executing commands and measuring the similarity of side effects. They also present a curated NL2Bash dataset for testing in their execution environment and find OpenAI's GPT-4 model[19] achieves SOTA performance on the dataset[34]. Song et al. present an enhanced version of the Tree Similarity Edit Distance (TSED) metric that measures Bash command similarity by comparing the edit distance of the abstract syntax tree of commands[26]. Lastly, Yu et al. present BashExplainer, a model for generating natural language explanations of Bash commands, the reverse of our task. They find that training their model with additional Bash command context from internet forums improves the accuracy of translations[36]. This finding is corroborated by Chen et al.[3] and Zhang et al.[37] in their respective works on code comment generation.

This paper builds on previous work by comparing the efficacy of accuracy metrics and creating a new dataset for NL2CMD translation. Additionally, we evaluate the benefits of fine-tuning a foundation model and training a model with Bash command context realized by Shi et al. and Yu et al., respectively.

3 METHODOLOGY

Investigating natural language to Bash command translation requires three components: selecting a model for evaluation, curating a dataset, and establishing an NL2CMD accuracy metric. We start by detailing our selected accuracy metric and its influence on our dataset. Next, we describe the creation of our dataset and its structure. We conclude with our model architecture and fine-tuning processes.

3.1 Accuracy Metric

In order to evaluate our models, we use a human-curated dataset of natural language tasks paired with Bash commands, as described in Section 3.2. We present an evaluation of accuracy metrics first because it informs the structure of our dataset. We define accuracy as the similarity of a model's generated Bash command to a ground truth Bash command, averaged across the test dataset. We only evaluate the first command generated by a model without retries or feedback because a single incorrect command could corrupt the state of a Linux system.

Measuring the similarity of commands is a non-trivial task because standard string comparison techniques do not capture functional equivalence[17, 26]. 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 commands with no syntactic similarity can yield

[1]We define a small model as one with an inference memory footprint less than 16 GB, corresponding to a standard amount of RAM on a consumer laptop in 2024.
[2]We define usable performance as accuracy greater than 25% on the InterCode metric.

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.

Due to these challenges, we measure command similarity using the InterCode accuracy metric[34]. This metric executes the generated command and ground truth command in identical Docker containers[9] and records the response. The similarity of the two commands is determined by comparing the final state of both containers. To allow for comparison with previous work, we also evaluate our models using conventional BLEU-2, BLEU-4, and Edit distance metrics as well as the NL2CMD metric proposed in the NLC2CMD competition[1]. Additionally, we experiment with the Abstract Syntax Tree Edit Distance (TSED) metric presented by Song et al., as well as the cosine similarity of command embeddings, a modern method for capturing semantic similarity.

Figure 4 summarizes our comparison of accuracy metrics. We use the command *bind -l | grep p* from the test dataset as an example ground truth command. The labels on the left are commands that simulate model outputs for the natural language prompt *list names of bind functions containing "p"*. The labels across the top are the previously discussed accuracy metrics. 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 performance of each metric. We scale the NL2CMD metric from its original range of [-1, 1] to [0, 1] to simplify comparison with other metrics.

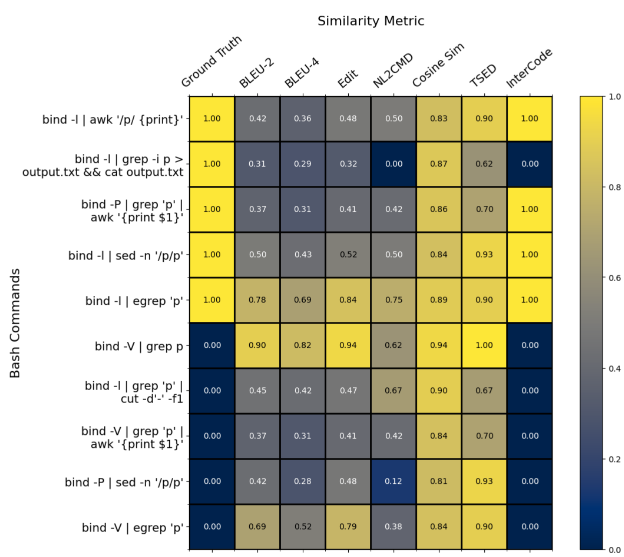

Figure 4: Evaluation of Bash command similarity metrics compared to a ground truth command of *bind -l | grep p* for the prompt *list names of bind functions containing "p"*

We find the InterCode metric is the only metric capable of measuring the functional equivalence of commands, correctly classifying 9 out of 10 commands in this example. The single misclassified command, `bind -l | grep -i p > output.txt && cat output.txt`, is notable because it produces two side effects, printing to standard out and logging the output to a file. This differs from the original command, which only has one side effect of printing to standard out. While both commands may be equivalent to a human user, the difference in side effects results in a non-equivalent classification by the InterCode metric. Despite this strictness, we find the InterCode accuracy metric to be the most correct for the NL2CMD translation task and use it for the remainder of our research.

3.2 Dataset

The InterCode benchmark provides a training dataset with 200 natural language and Bash command pairings and a test dataset with 24 pairings. Each Bash command in the dataset can be executed in one of five Docker environments. The dataset is based on the human-curated NL2Bash dataset presented by Lin et al. [12], containing 10,347 natural language and Bash command pairings. The InterCode-Bash dataset is significantly reduced in size because a Docker environment must be configured for each command. To increase the size of the dataset and retain the ability to execute commands for evaluation, we combine the InterCode-Bash and NL2Bash datasets. We do not use the dataset presented by Fu et al. because it has many unparsable commands, possibly because the generated data was not properly validated.

We use the NL2Bash dataset as the training split and the InterCode dataset as the test split for our dataset. Since the InterCode dataset is based on the NL2Bash dataset, we remove 472 pairings from the training split that are similar to pairings in the test split, guaranteeing that the test data is not contained in the training data. Additionally, we remove 161 pairings from the training split due to unparsable Bash commands. Our resulting dataset has 9,714 training pairings and 224 test pairings, where every command in the test split can be executed in an InterCode Docker container[31]. We observe a shift in the distribution of utility coverage from the training split to the test split as shown in Figure 5.

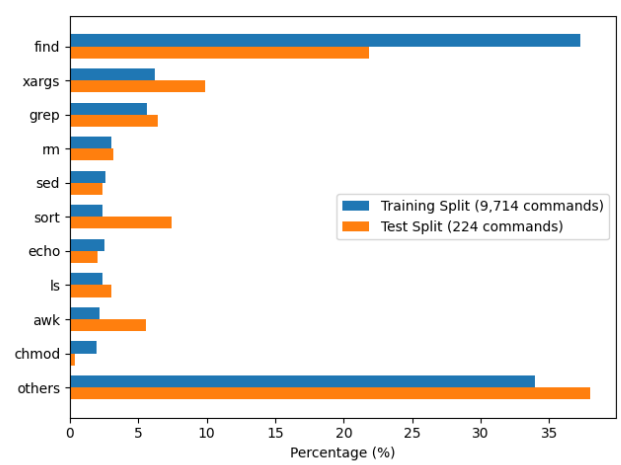

Figure 5: Frequencies of the top 10 Bash utilities in training and test splits. Training contains 125 utilities across 9,714 commands, test contains 64 utilities across 224 commands.

We augment our NL2CMD dataset with context about Bash commands from the Linux manual pages. We scraped manual pages[10] as well as community-created manual page summaries[27] to create a dataset with 685 Bash utilities, their corresponding manual page, and manual page summary[30]. Next, we parsed the Bash commands in our NL2CMD dataset to create a list of utilities in each command. Finally, using our manual page dataset, we added columns to our NL2CMD dataset with manual page context corresponding to the utilities in each command. Our final dataset is available here[3]. In Figure 6 we show our dataset's prompt structure for fine-tuning.

3.3 Model

We fine-tune two models, both based on the StarCoder2-3b foundation model, using the training split of our dataset[14]. We selected StarCoder2 as our foundation model because it was trained on over 600 programming languages and natural language from Wikipedia, Arxiv, and GitHub for the task of code generation, which we believe is similar to our downstream task of natural language to command translation. The 3 billion parameter model was selected because it has an inference memory footprint of 13 GB, small enough to run on a standard laptop.

Our first model is fine-tuned on the datasets column with no context, and our second model on the column with manual page summaries as context. We do not fine-tune a model using the column with full manual pages because the entries exceed StarCoder2-3b's context window of 16,384 tokens. We fine-tune our models with a learning rate of 2e-5 for 4,000 steps using one Nvidia RTX A6000 and observe the loss converge after one epoch.

During fine-tuning, a model is provided the full contents of each entry in the training split. During inference, a model is provided with the contents of each entry up to the *<START>* tag in the test split and generates the remaining Bash command. The generated commands are evaluated using the accuracy metrics outlined in Section 3.1. We evaluate the Magnum, ShellGPT, T5, GPT-2, GPT-3.5-Turbo, and GPT-4 models on our test dataset to ensure a fair comparison with previous work. Additionally, we use OpenAI's fine-tuning API to fine-tune a GPT-3.5-Turbo model on our training dataset and evaluate it on our test dataset.

4 RESULTS

We find our models outperform previous work but do not achieve accuracy comparable to the GPT-3.5-Turbo or GPT-4 models. Our discussion only considers the InterCode metric because other metrics are misleading as discussed in Section 3.1. Our model fine-tuned without context translates 57 out of 224 natural language prompts in the test set, more than double StarCoder2's baseline performance. Notably, our model fine-tuned with manual page context only translates 45 prompts, contradicting our hypothesis. Neither of our models achieve performance comparable to OpenAI's GPT-4 model[19], the current SOTA model for this task with 85 correct translations. Table 1 lists our results.

Our results show that fine-tuning a foundation model on our dataset improves NL2CMD translation over baseline model performance. However, fine-tuning and prompting a model with manual page context does not increase performance when compared with a model fine-tuned and prompted without context. Despite decreased performance, our model fine-tuned with context correctly translates 17 prompts that our model fine-tuned without context failed to translate. Figure 7 shows that our models and GPT-4 correctly translate different subsets of the test dataset. We leave analysis of the shift in test set coverage for future work.

Table 1: Comparison of our models with previous work and SOTA models (* indicates fine-tuning method from Figure 6).

	BLEU-2	BLEU-4	Edit	NL2CMD[1]	CosSim	TSED[26]	IC[34]
Magnum[5]	0.079	0.046	0.151	-0.864	0.530	0.438	0.013
T5[2]	0.597	0.547	**0.586**	-0.332	**0.878**	0.636	0.160
ShellGPT[25]	0.385	0.292	0.343	**-0.132**	0.727	0.655	0.013
GPT-2[21]	0.064	0.041	0.071	-0.933	0.645	0.246	0.000
GPT-3.5	0.609	0.552	0.537	-0.415	0.874	0.622	0.334
GPT-3.5 *1	0.611	0.543	0.552	-0.372	0.875	**0.656**	0.258
GPT-4[19]	**0.618**	**0.560**	0.561	-0.329	0.874	0.651	**0.388**
SC2[14]	0.320	0.274	0.302	-0.669	0.703	0.426	0.120
SC2 *1	0.571	0.521	0.546	-0.406	0.864	0.625	0.254
SC2 *2	0.476	0.434	0.472	-0.443	0.852	0.564	0.205

Our results also show that NL2CMD translation is a difficult task, with current SOTA models achieving less than 40% accuracy on the InterCode metric. Considering that a single incorrect command can corrupt the state of a system, further improvements are needed before these translation models can be used in many settings. Although our models do not achieve performance comparable to SOTA models, their computational cost is lower by orders of magnitude. We find performance is roughly correlated with the number of trainable parameters in a model, as shown in Figure 8.

Additionally, we find that fine-tuning a model does not guarantee higher performance on this task. When we loosened our hardware constraint and fine-tuned GPT-3.5-Turbo on our dataset with OpenAI's default parameters, we observed a decrease in translation accuracy compared to the baseline GPT-3.5-Turbo model. This decrease could be due to our training data and we leave further investigation to future work. These results highlight the difficulty of developing a model for the NL2CMD translation task.

5 CONCLUSION

In this paper, we presented an evaluation of accuracy metrics, a new dataset, and two fine-tuned models for the task of NL2CMD translation. We find that execution-based evaluation metrics are necessary for measuring NL2CMD translation accuracy. Further, we present a dataset that enables execution-based evaluation and includes additional context about Bash commands from the Linux manual pages. Our results show that a foundation model's NL2CMD translation performance can be improved through fine-tuning using our dataset. However, NL2CMD translation remains a difficult task, requiring accuracy improvements before models can be safely deployed. Our best model achieves 25.45% accuracy using the InterCode metric with an inference memory footprint of 13 GB, while GPT-4 achieves 38.84% with a computational cost that is larger by orders of magnitude. As we continue our research, we aim to refine our methodology to improve the performance and accessibility of open-source models for NL2CMD translation to simply interactions with Linux systems.

Based on the correlation between accuracy and the number of model parameters, we plan to rerun our experiment with a larger model, such as StarCoder2-7b or LLaMa3-8b. After fine-tuning,

[3] https://huggingface.co/datasets/westenfelder/NL2CMD-InterCode

	Fine-tuning Method 1	Fine-tuning Method 2	Fine-tuning Method 3
Context	No additional context	Linux manual page summaries	Linux manual pages
Prompt Structure	*Output a Bash command to accomplish the following task.* *Task: {Natural Language Prompt}* *Command:<START>{Bash Command}<END>*	*Use the provided context to output a Bash command that accomplishes the following task.* *Context: {Manual Page Summaries}* *Task: {Natural Language Prompt}* *Command:<START>{Bash Command}<END>*	*Use the provided context to output a Bash command that accomplishes the following task.* *Context: {Manual Pages}* *Task: {Natural Language Prompt}* *Command:<START>{Bash Command}<END>*
Mean Length	62.76 tokens	885.49 tokens	17427.33 tokens

Figure 6: Prompt structures for fine-tuning methods with different context levels

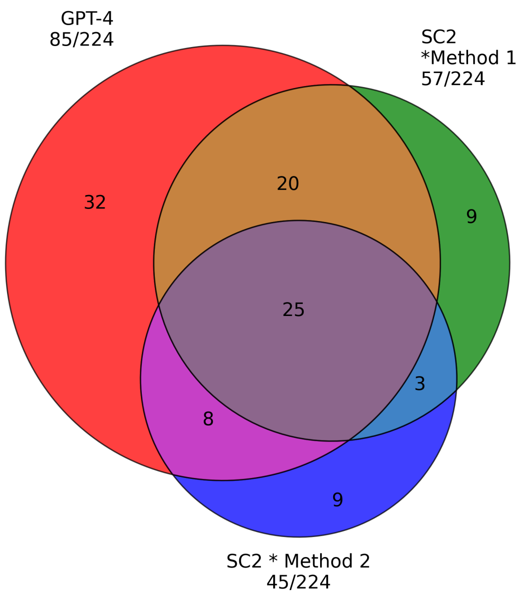

Figure 7: Overlap of correct command translations between GPT-4 and our fine-tuned models on the test dataset

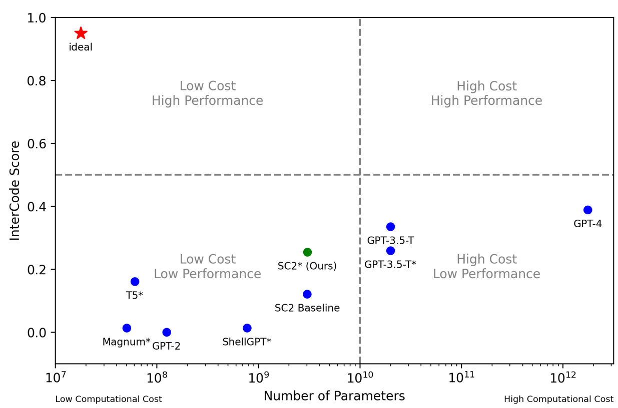

Figure 8: Comparison of computational cost and performance. GPT-3.5 and GPT-4 parameter counts are rumored.

we can quantize these models to reduce the inference memory footprint[8]. We aim to determine if our observed decrease in accuracy from fine-tuning with context was due to our methodology or due to a limitation with the underlying model. We also plan to evaluate models using more complex prompting and reasoning strategies[13, 18, 35] to determine the maximum performance of SOTA models on this task.

REFERENCES

[1] 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. arXiv:2103.02523 [cs.CL]

[2] Shikhar Bharadwaj and Shirish Shevade. 2021. Explainable Natural Language to Bash Translation using Abstract Syntax Tree. In *Proceedings of the 25th Conference on Computational Natural Language Learning*, Arianna Bisazza and Omri Abend (Eds.). Association for Computational Linguistics, Online, 258–267. https://doi.org/10.18653/v1/2021.conll-1.20

[3] Zhongqi Chen, Neng Zhang, Pengyue Si, Qinde Chen, Chao Liu, and Zibin Zheng. 2023. ShellFusion: An Answer Generator for Shell Programming Tasks via Knowledge Fusion. In *2023 IEEE/ACM 45th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. 93–97. https://doi.org/10.1109/ICSE-Companion58688.2023.00032

[4] Gelei Deng, Yi Liu, Víctor Mayoral-Vilches, Peng Liu, Yuekang Li, Yuan Xu, Tianwei Zhang, Yang Liu, Martin Pinzger, and Stefan Rass. 2023. PentestGPT: An LLM-empowered Automatic Penetration Testing Tool. arXiv:2308.06782 [cs.SE]

[5] 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* (2023).

[6] 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)*. 1245–1248. https://doi.org/10.1109/ICMLA52953.2021.00202

[7] David Gros. 2019. AInix: An open platform for natural language interfaces to shell commands. http://www.cs.utexas.edu/users/ai-labpub-view.php?PubID=127814 Undergraduate Honors Thesis, Computer Science Department, University of Texas at Austin.

[8] Song Han, Huizi Mao, and William J. Dally. 2016. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. arXiv:1510.00149 [cs.CV]

[9] Docker Inc. 2023. Docker: Empowering App Development for Developers. https://www.docker.com. Accessed: 2024-05-09.

[10] Michael Kerrisk. 2024. man7.org. https://man7.org/. Accessed: 2024-05-09.

[11] Xi Victoria Lin. 2017. Program Synthesis from Natural Language Using Recurrent Neural Networks. https://api.semanticscholar.org/CorpusID:3809743

[12] 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. arXiv:1802.08979 [cs.CL]

[13] Chao Liu, Xuanlin Bao, Hongyu Zhang, Neng Zhang, Haibo Hu, Xiaohong Zhang, and Meng Yan. 2023. Improving ChatGPT Prompt for Code Generation. arXiv:2305.08360 [cs.SE]

[14] Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, Tianyang Liu, Max Tian, Denis Kocetkov, Arthur Zucker, Younes Belkada, Zijian Wang, Qian Liu, Dmitry Abulkhanov, Indraneil Paul, Zhuang Li, Wen-Ding Li, Megan Risdal, Jia Li, Jian Zhu, Terry Yue Zhuo, Evgenii Zheltonozhskii, Nii Osae Osae Dade, Wenhao Yu, Lucas Krauß, Naman Jain, Yixuan Su, Xuanli He, Manan Dey, Edoardo Abati, Yekun Chai, Niklas Muennighoff, Xiangru Tang, Muhtasham Oblokulov, Christopher Akiki, Marc Marone, Chenghao Mou, Mayank Mishra, Alex Gu, Binyuan Hui, Tri Dao, Armel Zebaze, Olivier Dehaene, Nicolas Patry, Canwen Xu, Julian McAuley, Han Hu, Torsten Scholak, Sebastien Paquet, Jennifer Robinson, Carolyn Jane Anderson, Nicolas Chapados, Mostofa Patwary, Nima Tajbakhsh, Yacine Jernite, Carlos Muñoz Ferrandis, Lingming Zhang, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. 2024. StarCoder 2 and The Stack v2: The Next Generation. arXiv:2402.19173 [cs.SE]

[15] The Linux man-pages project. 2024. Linux Manual Pages. https://man7.org/linux/man-pages/index.html. Accessed: 2024-05-09.

[16] Stephen Moskal, Sam Laney, Erik Hemberg, and Una-May O'Reilly. 2023. LLMs Killed the Script Kiddie: How Agents Supported by Large Language Models Change the Landscape of Network Threat Testing. arXiv:2310.06936 [cs.CR]

[17] Atharva Naik. 2024. On the Limitations of Embedding Based Methods for Measuring Functional Correctness for Code Generation. arXiv:2405.01580 [cs.SE]

[18] Ansong Ni, Miltiadis Allamanis, Arman Cohan, Yinlin Deng, Kensen Shi, Charles Sutton, and Pengcheng Yin. 2024. NExT: Teaching Large Language Models to Reason about Code Execution. arXiv:2404.14662 [cs.LG]

[19] OpenAI. 2024. GPT-4 Technical Report. arXiv:2303.08774 [cs.CL]

[20] Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. 2023. Generative Agents: Interactive Simulacra of Human Behavior. arXiv:2304.03442 [cs.HC]

[21] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language Models are Unsupervised Multitask Learners. https://api.semanticscholar.org/CorpusID:160025533

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

[23] Jean E. Sammet. 1966. The use of English as a programming language. *Commun. ACM* 9, 3 (mar 1966), 228–230. https://doi.org/10.1145/365230.365274

[24] Warp Information Services. 2024. Warp: Your terminal, reimagined. https://www.warp.dev/. Accessed: 2024-05-09.

[25] 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)*. 671–682. https://doi.org/10.1109/ISSRE59848.2023.00082

[26] Yewei Song, Cedric Lothritz, Daniel Tang, Tegawendé F. Bissyandé, and Jacques Klein. 2024. Revisiting Code Similarity Evaluation with Abstract Syntax Tree Edit Distance. arXiv:2404.08817 [cs.CL]

[27] The tldr pages project. 2024. tldr pages. https://tldr.sh/. Accessed: 2024-05-09.

[28] TheR1D. 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.

[29] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2023. Attention Is All You Need. arXiv:1706.03762 [cs.CL]

[30] Finn Westenfelder. 2024. Linux-Manual-Pages-TLDR. https://huggingface.co/datasets/westenfelder/Linux-Manual-Pages-TLDR. Accessed: 2024-05-09.

[31] Finn Westenfelder. 2024. NL2CMD-InterCode. https://huggingface.co/datasets/westenfelder/NL2CMD-InterCode. Accessed: 2024-05-09.

[32] Jiacen Xu, Jack W. Stokes, Geoff McDonald, Xuesong Bai, David Marshall, Siyue Wang, Adith Swaminathan, and Zhou Li. 2024. AutoAttacker: A Large Language Model Guided System to Implement Automatic Cyber-attacks. arXiv:2403.01038 [cs.CR]

[33] Hui Yang, Sifu Yue, and Yunzhong He. 2023. Auto-GPT for Online Decision Making: Benchmarks and Additional Opinions. arXiv:2306.02224 [cs.AI]

[34] John Yang, Akshara Prabhakar, Karthik Narasimhan, and Shunyu Yao. 2023. InterCode: Standardizing and Benchmarking Interactive Coding with Execution Feedback. arXiv:2306.14898 [cs.CL]

[35] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of Thoughts: Deliberate Problem Solving with Large Language Models. arXiv:2305.10601 [cs.CL]

[36] Chi Yu, Guang Yang, Xiang Chen, Ke Liu, and Yanlin Zhou. 2022. BashExplainer: Retrieval-Augmented Bash Code Comment Generation based on Fine-tuned CodeBERT. In *2022 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 82–93. https://doi.org/10.1109/ICSME55016.2022.00016

[37] Rui Zhang, Ziyue Qiao, Chenghao Zhang, and Jianjun Yu. 2024. KFCC: A differentiation-aware and keyword-guided fine-grain code comment generation model. *Expert Systems with Applications* 251 (2024), 123946. https://doi.org/10.1016/j.eswa.2024.123946