Generating Bash Code from Prompt Using Codex

Eyal Finkelshtein and **Yuval Aidan**
Industrial Engineering and Management Department
The Technion - Israel Institute of Technology
Haifa 32000, Israel
{eyal.f, yuvalaidan}@campus.technion.ac.il

Abstract

Generating and translating natural language to programming language is one of the research fronts of the natural language processing (NLP) field. Bash code generation from prompt can help developers to accomplish command-line tasks in a terminal, in a more natural and effortless fashion, and allow an execution of complex, multi-line tasks without the necessity of meticulous syntax. This can, among other things, allow inexperienced developers to achieve results with a steeper learning curve.

Approaches using transformer based translation and approaches based on T5 and Seq2Seq models have been widely studied, partially in the scope of the NLC2CMD challenge at the NeurIPS 2020 conference. The approach presented in this paper suggests to use very large models, such as GPT3 and Codex, which can handle many types of problems, to generate Bash code from a structured prompt. This approach is based on the understanding that prompting makes it possible for downstream tasks to take the same format as the pre-training objectives. The intention of this paper is not to compete with the SOTA, but rather to examine the effects of various prompt structures on the accuracy of the translation.

The dataset used for this task is NL2Bash, which is given in the NLC2CMD challenge. The dataset contains over 10K pairs of a set of instructions written in natural language (invocation) and a Bash command which reaches the desired outcome (cmd).

The results of this paper emphasize the importance of a prompt which has a well defined and organized structure, and the benefits of using examples of the desired task as part of the prompt.

1 Introduction

Neural machine translation (NMT) is a learning approach to automated translation from one language to another. Unlike the traditional statistical machine translation methods, NMT attempts to learn the mapping from an input text to its associated output text directly. The architecture suggested recently for this task is usually belongs to the encoder-decoder architecture family, where the encoder learns to encode the input sentence (usually of a fixed length), and the decoder generates the translation (Sutskever et al., 2014; Cho et al., 2014).

Translation from natural language (NL) to Bash commands can be very useful. It can help developers use Bash command-line instructions without knowing the full syntax. This way is more natural, easy, and understandable. Also, it can help developers accomplish tasks with instructions they are not familiar with, with no need to search in forums like Stack Overflow. Generating Bash code from prompt is only one example of this problem. After solving this problem, given the relevant dataset, one can think of translation from natural language to any other programming language.

Previous works in this area have explored transformer-based architecture to solve this problem (Fu et al., 2021). In this paper, the approach presented is to use very large models, such as Codex or GPT3, and to find the proper prompt for this task. The code used to generate the results presented in this paper can be found on GitHub[1]. The reason for working with large generative models is that they have been trained on a large corpus and can tackle many different problems, including translation. Prompting (or prompt-based learning) allows the adaption of pre-trained models to a downstream task. The translation success of the model given a prompt is measured by the NLC2CMD accuracy metric[2] using the NL2Bash dataset, that was presented as part of

[1] https://github.com/EyalFinkel/NLP_project_NL2Bash

[2] https://github.com/IBM/clai/tree/2ad172acbed0c1ec870cc39f47635adea39f19c0/utils

the NeurIPS 2020 NLC2CMD competition (Agarwal et al., 2021). This dataset contains about 10K pairs of English instructions and their associated Bash command. The natural language invocation is included in the prompt, and the model output is the translation to bash command.

2 Related Work

In this paper, the translation of natural language (NL) to Bash commands is tackled with the help of a large model, and the effect of the prompt structure and content on the quality of the translation is inspected. The topics of NL to Bash translation and prompt learning are widely researched, and different methods were suggested for both of them.

NL to Bash Commands Translation

(Lin et al., 2017) introduced Tellina, which uses recurrent neural networks (RNNs) for translation of natural language instructions to a bash command. This paper also introduced the NL2Bash dataset (Lin et al., 2018), which became the baseline for English natural language to bash commands translation. Tellina achieved 13.8% accuracy with the metric proposed by IBM (Agarwal et al., 2021), which ignores arguments in the commands, considers the order of the utilities in case of piped commands and penalizes redundant flags with respect to the ground truth. This metric evaluates in the range of $[-1, 1]$.

Since then, (Fu et al., 2021) achieved SOTA results on the NL2Bash dataset, with 53.2% accuracy, as part of the NLC2CMD competition at the NeurIPS 2020 conference (Agarwal et al., 2021). This paper introduced a Transformer-based approach for the translation task and tested different combinations of encoder and decoder types (RNN, BRNN and Transformer).

Other approaches used abstract syntax tree (Bharadwaj and Shevade, 2021) and fine-tuned GPT-2 model (Agarwal et al., 2021).

Prompt Learning

Prompt-based learning attempts to utilize the knowledge of a pre-trained model to perform a variety of tasks such as sentiment classification, question answering, summarization, translation, etc. (Liu et al., 2021a) surveys the research works in prompt-based learning. Manual searching for a best-performing prompt is very difficult, and one is not guaranteed to get a fully optimized prompt. Thus, several works have focused on automatically searching prompts. One method to do so is by prompt learning, where there are prompt-relevant parameters that can be tuned together with the pre-trained model parameters in order to find the most efficient prompt for a specific task.

In (Ben-David et al., 2021), the authors suggest an autoregressive algorithm (PADA) for prompt learning based on the T5 language model. PADA is trained to generate a prompt for a task example. The prompt is a token sequence consisting of domain related features.

In (Liu et al., 2021b) the authors suggest the p-tuning method that automatically searches prompts, while optimizing continuous prompt embeddings. Another approach to prompt learning is automatic search for discrete prompts (Jiang et al., 2020; Reynolds and McDonell, 2021; Shin et al., 2020; Gao et al., 2021).

3 Model

The GPT3 language model was considered in this paper for the purpose of translating natural language to bash commands by inspection of different prompts. This model performs a wide variety of natural language tasks. Experiments of several prompts for this model were made in a small scale and it seems to produce good results. Due to budget limitations associated with the OpenAI API, it wasn't chosen eventually. OpenAI's beta version Codex model was proposed as an alternative.

The Codex model is a Generative Pre-trained Transformer (GPT) language model which is a descendant of the base GPT3 model (Chen et al., 2021). It is accessible through the OpenAI API in private beta. Codex was finetuned on a publicly available code corpus, containing more than a dozen programming languages and billions of code lines, and is especially suited for code related tasks. Thus, it can interpret NL commands and translate them to code, specifically Bash commands. Despite the advantages listed above, the OpenAI Codex model has its own limitation - due to the limited capacity of this model in the beta version, it has a requests rate limit of approximately 10-20 requests per minute. This limitation makes the prompt testing process very slow, but not impossible. This limitation enforces a different approach for translating invocations. To enable inspection of different prompts in a relatively reasonable running time, the prompt contains some number of invoca-

tions and a good translation output of the model is the translation of all the invocations.

The Codex model was chosen as the language model to apply the prompt containing invocations in English, desiring a Bash command translation, due to its suitability to the purpose of this paper.

4 Data

As mentioned before, the dataset used in this paper is NL2Bash, which is given in the NLC2CMD competition. The dataset contains 10347 pairs of natural language instructions (invocation) and a Bash one-liner command (cmd), which performs the given instructions and achieves the desired state. A sample example is displayed in Figure 1.

The NL2Bash dataset includes single commands, logical connectives (&& and ||), and nested commands (pipeline |, command substitution $(), and process substitution <()).

The data was split into an evaluation set containing 1000 samples, and an examples set containing the remaining 9347 samples. The imbalance of the split intended to allow for a reasonable runtime, since, as stated before, there is a request rate limit. At each inference to the model, different number of evaluated samples and example samples are sampled from both sets, to build the prompt.

```
"189": {
    "invocation": "Calculate the md5 sum of the contents of \"$FILES\"",
    "cmd": "cat $FILES | md5sum"
},
```

Figure 1: A sample example from the NL2Bash dataset.

5 Experiments and Results

In this paper, three experiments were conducted and the effect on the accuracy score was considered. The first experiment tested the prompt structure's effect; the second tested the effect of the number of example samples (i.e. invocations and their ground truth translation to bash command) and the number of evaluated samples (i.e. invocations to be translated) in the prompt; the third tested the omitting rate of words in the prompt. Weights & Biases platform was used in this paper in order to conduct a grid search over all the options in the experiments. This platform enables quickly tracking experiments and visualizing the results.

5.1 Prompt Structure

In this experiment, the effect of the prompt structure on the accuracy score was tested. The prompt structure is the way of writing the prompt, which later will be sent to the model for inference, in terms of the prefix, the number of evaluated and example samples, and the spacial arrangement of the different parts in the prompt. The prompt contains example samples and evaluated samples in a specific order, determined by the prompt structure. For this experiment, 3 example samples and 5 evaluated samples were used in each prompt. Note that when using Codex it is advisable to use comments (# in the case of bash) in the prompt when describing what the code should do. The four prompt structures that were tested are listed below and demonstrated in Appendix A:

1. Numbered examples intertwined with evaluated examples.
2. Numbered examples separated from evaluated examples.
3. Prompted examples intertwined with evaluated examples.
4. Prompted examples separated from evaluated examples.

Table 1: Scores for the 4 prompt structures tested.

Prompt Structure	Score
1 - Numbered, Intertwined	-0.1099
2 - Numbered, Separated	-0.08279
3 - Prompted, Intertwined	-0.2597
4 - Prompted, Separated	-0.1136

As demonstrated in Table 1, the numbered structures were superior to the prompted ones; the structures in which the evaluated samples are separated from the example samples achieved better results than those in which the example samples and the evaluated samples are listed in the same list. Prompt structure 2 (numbered, separated) has achieved the best score and was used in the rest of the paper.

5.2 Number of Example Samples and Evaluated Samples

In this experiment, the effect of the number of example samples and the number of evaluated samples in the prompt on the accuracy score was tested. A grid search was performed over the number of example samples $m \in \{0, 1, ..., 9\}$ and the number of evaluated samples $n \in \{1, 2, ..., 10\}$.

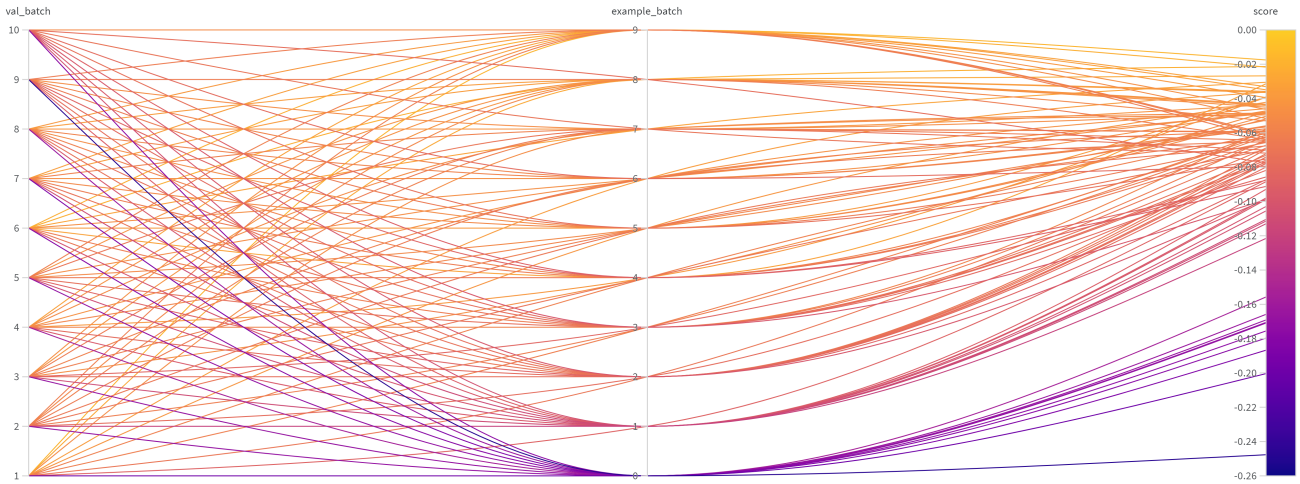

Figure 2: Each line in the diagram representing one run, and the intersection points of that line with the left, middle and right columns representing the number of evaluated samples and the number of example samples that were used in that run, and the score the run has achieved, respectively. The lowest scored run is colored in dark blue and the highest scored run is colored in light orange.

The sweep diagram is presented in Figure 2. It is clear from the diagram that runs with no example samples achieve a significantly lower accuracy score than other runs, and that the accuracy score gets higher with the increase of the example samples in the prompt. Although more example samples cause higher accuracy scores, for large numbers of example samples, there is a large overlap in achieved scores, as can be seen in the diagram.

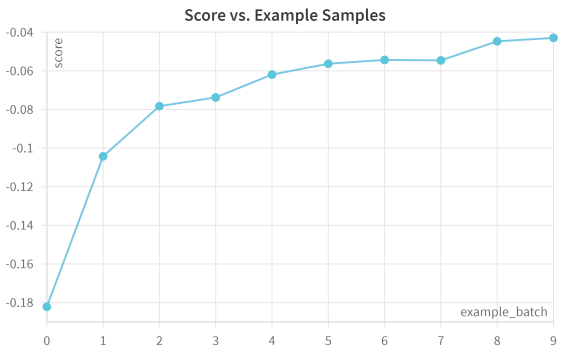

Figure 3: Accuracy score as a function of the **number of example samples** used in the prompt, where each data point is an average over the scores of different numbers (1 to 10) of evaluated samples in each request to the model.

The relation between the accuracy score and the number of example samples used in the prompt is presented in Figure 3. It can be concluded from the graph that the accuracy score is increasing with a growing number of examples in the prompt. No example samples at all results in the worst accuracy score, and there is a significant increase of the accuracy score for a prompt with small number of example samples. After the initial significant score increase with the number of examples, the slope becomes more moderate, and the score reaches its maximum value for the maximum number of example samples tested. With consideration of resources (compute time, number of tokens, etc.), one might choose the number of example samples to be less than 9, due to the moderate slope in this region.

The relation between the accuracy score and the number of evaluated samples used in the prompt is presented in Figure 4. It can be concluded from the graph that the accuracy score is negatively affected by a large number of evaluated samples (8 to 10). In the range of 0 to 7 evaluated samples, the difference in the accuracy score is not significant, and it is difficult to deduce with certainty what the optimal number of samples is. Nevertheless, it can be seen from the graph that a single evaluated sample achieves relatively good results, following the intuition that a more focused task will achieve better results.

For graphs demonstrating the score as a function of the number of example samples for all numbers of evaluated samples, and the score as a function of the number of evaluated samples for all numbers of example samples, see Appendix B.

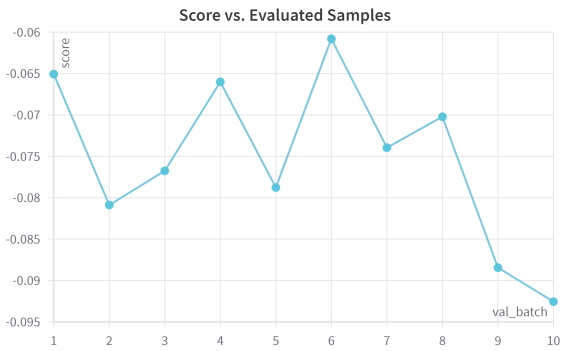

Figure 4: Accuracy score as a function of the **number of evaluated samples** used in the prompt, where each data point is an average over the scores of different numbers (0 to 9) of example samples in each request to the model.

5.3 Words Omitting Rate

In this experiment, the effect of the words omitting rate from the prompt on the accuracy score was tested. For this experiment, 5 example samples and 5 evaluated samples were used in each prompt. The evaluation set was evaluated with omit rates in the range $[0.0, 0.5]$ with a step of 0.05. The omitting rate determines the relative portion of the words that will be omitted. Every word in the prompt can be sampled to be omitted with a discrete uniform distribution. For this matter, words can be numbers and signs, such as #, which is used to imply a comment in code. The sampling of the words is performed in a way that preserves the structure of the prompt, meaning line breaks will be preserved in their original location in the text.

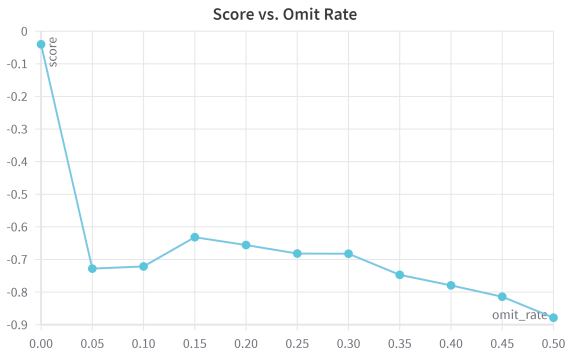

Figure 5: Accuracy score as a function of the **omitting rate** of words in the prompt.

As demonstrated in Figure 5, the model is very sensitive even to the smallest omitting rate, which causes a significant decrease in the score. The score then decreases moderately with the omit rate, and achieves its minimum for the maximum omit rate that was tested. This significant drop can be due to the omission of keywords that define the utility of the bash command, which may result in an ambiguity as to the desired command. Also, omitting words that are important to the prompt structure, such as #, numbers or bash> prefixes, can disrupt the structure of the response from the model, thus disrupting the parsing of the response to separate commands.

6 Conclusion

In order to examine the effects of different prompts on the translation of NL invocation to Bash command, using the generative language model Codex, three experiments were conducted. The first experiment tested the effect of the prompt structure. Prompt structure 2 has achieved the highest accuracy score. The second experiment tested the effect of the numbers of example and evaluated samples. The runs that achieved the highest accuracy score had a large number of example samples and a small number of evaluated samples. The third experiment tested the effect of the words omitting rate. The results from this experiment suggest that the model has a high sensitivity to word omission, and the best accuracy score is obtained when no words are omitted from the prompt.

Future work on this subject may include implementation of a generative model for learning the best prompt, using the accuracy score as the loss function, and investigation of the correspondence of the generated prompts with the conclusions from this paper.

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. In *Proceedings of the NeurIPS 2020 Competition and Demonstration Track*, volume 133 of *Proceedings of Machine Learning Research*, pages 302–324. PMLR.

Eyal Ben-David, Nadav Oved, and Roi Reichart. 2021. Pada: Example-based prompt learning for on-the-fly adaptation to unseen domains. *CoRR*, abs/2102.12206.

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*, pages 258–267, Online. Association for Computational Linguistics.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harrison Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating large language models trained on code. *CoRR*, abs/2107.03374.

Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078.

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.

Tianyu Gao, Adam Fisch, and Danqi Chen. 2021. Making pre-trained language models better few-shot learners.

Zhengbao Jiang, Antonios Anastasopoulos, Jun Araki, Haibo Ding, and Graham Neubig. 2020. X-factr: Multilingual factual knowledge retrieval from pretrained language models.

Xi V. Lin, Chenglong Wang, Deric Pang, Kevin Vu, Luke Zettlemoyer, and Michael D. Ernst. 2017. Program synthesis from natural language using recurrent neural networks. (UW-CSE-17-03-01).

Xi V. 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. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan. European Language Resources Association (ELRA).

Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2021a. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing.

Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. 2021b. Gpt understands, too. *arXiv:2103.10385*.

Laria Reynolds and Kyle McDonell. 2021. Prompt programming for large language models: Beyond the few-shot paradigm.

Taylor Shin, Yasaman Razeghi, Robert L. Logan IV au2, Eric Wallace, and Sameer Singh. 2020. Autoprompt: Eliciting knowledge from language models with automatically generated prompts.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. 27.

A Prompt Structures

Four prompt structures were tested in this paper, in order to find the best preforming prompt structure. The structure of each prompt is given below. $m \in \{0, 1, ..., 9\}$ is the number of example samples and $n \in \{1, 2, ..., 10\}$ is the number of evaluated samples. Notice the # used to imply a comment in code.

1. Numbered examples intertwined with evaluated samples

Translate the following set of instructions to
Bash command:
Invocations:
1. example invocation 1
⋮
m. example invocation m
$m + 1$. evaluated invocation 1
⋮
$m + n$. evaluated invocation n

Bash commands:
1. example bash command ground truth 1
⋮
m. example bash command ground truth m

2. Numbered examples separated from evaluated samples

Translate the following set of instructions to
Bash command:
Invocations:
1. example invocation 1
⋮
m. example invocation m

Bash commands:
1. example bash command ground truth 1
⋮
m. example bash command ground truth m

Invocations:
1. evaluated invocation 1
⋮
n. evaluated invocation n

Bash commands:

3. Prompted examples intertwined with evaluated samples

Translate the following set of instructions to
Bash command:
Invocations:
1. example invocation 1
⋮
m. example invocation m
$m + 1$. evaluated invocation 1
⋮
$m + n$. evaluated invocation n

Bash commands:
$bash >$ example bash command ground truth 1
⋮
$bash >$ example bash command ground truth m

4. Prompted examples separated from evaluated samples

Translate the following set of instructions to
Bash command:
Invocations:
1. example invocation 1
⋮
m. example invocation m

Bash commands:
$bash >$ example bash command ground truth 1
⋮
$bash >$ example bash command ground truth m

Invocations:
1. evaluated invocation 1
⋮
n. evaluated invocation n

Bash commands:

B Number of Example Samples and Evaluated Samples

Detailed graphs presenting the score as a function of the number of example samples for all numbers of evaluated samples (Figure 6), and the score as a function of the number of evaluated samples for all numbers of example samples (Figure 7).

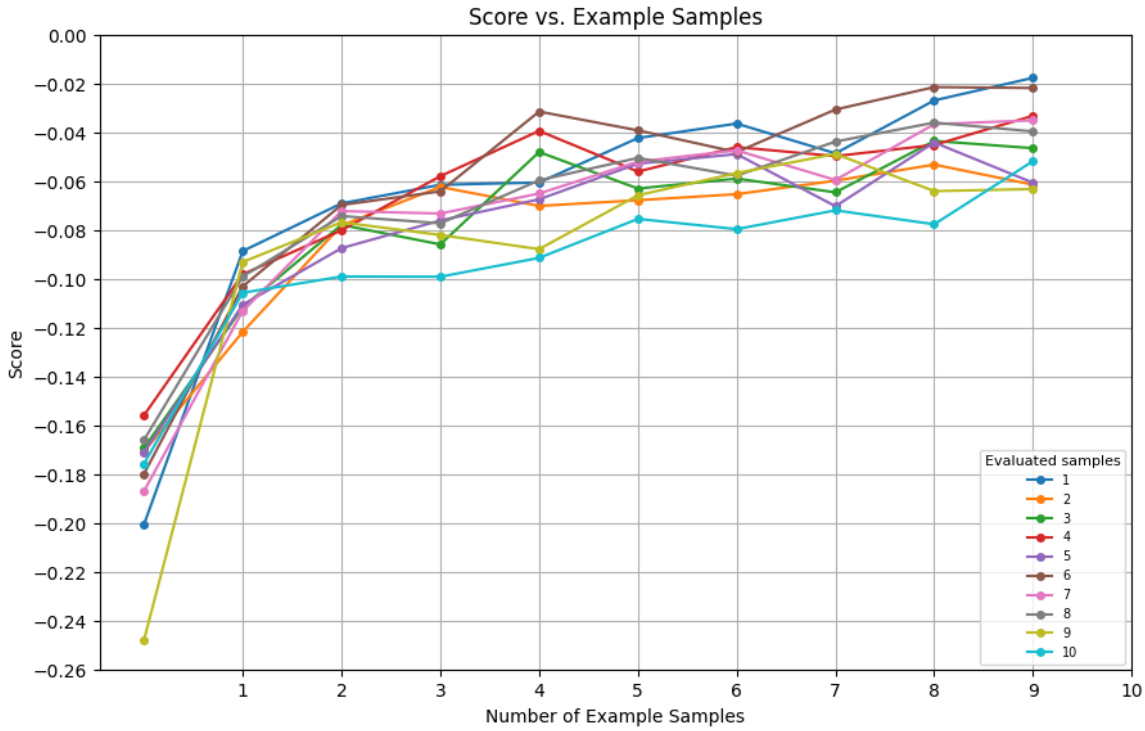

Figure 6: Accuracy score as a function of the **number of example samples** used in the prompt, for all numbers of evaluated samples.

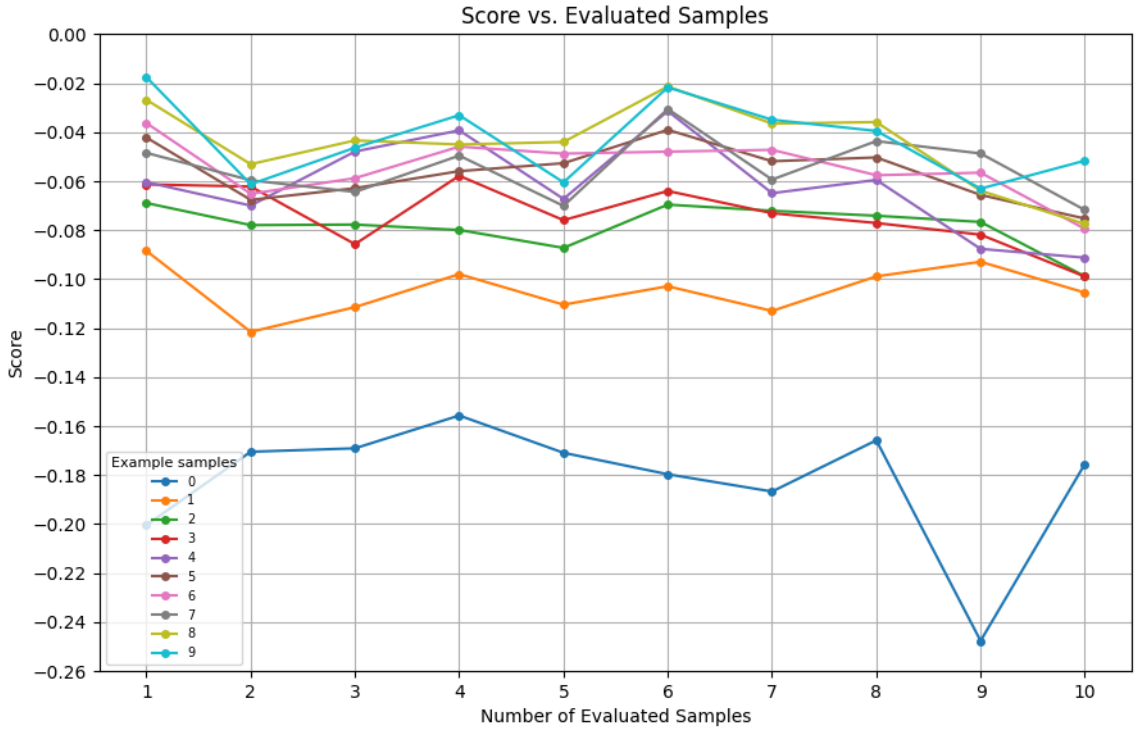

Figure 7: Accuracy score as a function of the **number of evaluated samples** used in the prompt, for all numbers of example samples.