INSTITUT FÜR INFORMATIK
Dialog Systems and Machine Learning

Universitätsstr. 1 D–40225 Düsseldorf

Heinrich Heine Universität Düsseldorf

Context-Aware CLI autocompletion

Roger Sellin

Bachelorarbeit

Date of issue:	7. june 2023
Date of submission:	7. September 2023
Reviewers:	Dr. Konrad Völkel
	Dr. Marcus Brenneis

Erklärung

Hiermit versichere ich, dass ich diese Bachelorarbeit selbstständig verfasst habe. Ich habe dazu keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

Düsseldorf, den 7. September 2023

____

Roger Sellin

Abstract

We try to create an LLM based autocompletion system that uses context to create more accurate predictions. Our approach is to add context-information to the prompt. Therefore, we create a prompt-template that tells the model that it is an autocompletion function for a Linux terminal. Several variations of this are tested, and the best one is chosen. The prompt-template is populated during use with the path, the command is executed in and the names of the files in the current directory.

In search of a model to use, we have the criteria that it has to be transformer based and its tokenizer has to be of at least subword granularity. We also consider the tradeoff between accuracy and resource demand. We reject BERT for its architecture. GPT-3 and GPT-4 are considered but rejected because they are not free to use. So we decide on their predecessor, GPT-2 and the LLaMA based Alpaca that is fine-tuned to behave like GPT-3

Our tests show that with just the command, the models do not generate useful results. The usage of our prompt-template show an improvement. But the directory and files have little to no benefit, even tough they clearly influence the predictions. Which leads us to the conclusion that the base models are not good enough for this task.

Finally, we discuss methods to improve this, ranging from further prompt modifications over the use of different models to fine-tuning them. Also, the possible usage of methods to limit the requirements of computational resources are discussed.

Contents

1 Introduction **1**

2 Technical Background **1**

2.1 Transformer Architecture 1

2.2 Computational Resources 2

2.3 Tokenization 2

3 Setup **2**

4 The Program **3**

5 The Model **4**

5.1 Rejected Models 4

5.2 Accepted Models 6

6 Prompt Template **8**

7 Tests **9**

7.1 Tests with GPT-2 10

7.2 Tests with Alpaca-7b 11

8 Conclusion **12**

9 Possible Extensions **12**

9.1 Other Alpaca Versions 12

9.2 LLaMA2 13

9.3 Finetuning 13

9.4 Out-Of-Vocabulary Words 13

9.5 LoRa 13

9.6 Context Extension to Subdirectories 13

References **15**

1 Introduction

Stochastic Autocompletion systems for the terminal are nothing new, but the recent development of large-language-models(LLMs) allows for new approaches. These LLMs integrate world knowledge, leading to a potential spillover effect that can enhance autocompletion systems.

While the completion of CLI (Command Line Interface) commands is possible, a more intriguing question is whether using the path of command execution and the files contained in the current directory as additional context can improve the accuracy of completion predictions.

For example, it is far more likely to use a git command in a Git repository than in a downloads folder. Furthermore, a command beginning with 'git commit' is more likely to be succeeded by 'git push' than 'git pull'.

The specific prompt can hugely influence the accuracy of the model's predictions. For this reason, we try to find a prompt-template that improves the accuracy. Which consists of telling the model what it is, which is an autocompletion and an order what we want the model to do, which is to predict the CLI command. Additionally, we add the variable context of the current directory and names of the files in the current directory.

Next, there is the issue of computational resources. With access to a sufficient cloud infrastructure, even high-end LLMs can be deployed. But this only applies if network access is available. Also, to a lesser extent, network speed can be an issue.

Therefore, we should consider a local solution. This again would limit our available resources. Therefore, we have to make a trade-off between accuracy and size of the model.

2 Technical Background

The training of high-end Large-Language-Models (LLMs) can be very expensive [4]. Therefore, training one from scratch is not feasible. The solution to this issue is to resort to pre-trainded LLMs.

2.1 Transformer Architecture

The paper "Attention is all you need" [8] introduced the transformer architecture in 2017, which has since become a foundational building block for many state-of-the-art models. They are designed to handle sequential data, making them particularly effective for NLP tasks. Unlike earlier architectures like recurrent neural networks (RNNs) or long short-term memory (LSTM) networks, transformers can process data in parallel rather than sequentially and can outperform those. Its novel idea was the introduction of the attention mechanism. A mechanism to weight how significant a token is to another token, in other words, focusing on its relevance. This allows a model to learn long-range dependencies between words in a sentence. The transformer model consists of an encoder and a decoder. The encoder converts the input into a set of vectors representing the semantic information. Positional encoding vectors are added to the embedding vectors to give the model the ability to deduce the distance between two tokens so that the model has information about the position of each token. This is needed because transformers don't process data sequentially. Decoding is the process of generating output from the encoded input. In the transformer architecture, this process is autoregressive, which means that the model generates one token at a time. Each new token is used together with the previous tokens to generate another token. This is repeated until the end of the sequence is reached. The order of the tokens is encoded with the addition of positional encodings similar to the encoder.

Since then, it has been used to develop LLMs that set a new standard for NLP tasks. For this reason, we limit our choice of models to transformer-based models.

This, however, is not the only thing we have to consider. Since our goal is to create an autocompletion system, we need models that excel at autocompletion tasks.

2.2 Computational Resources

There are two cases for application, on a local machine and in the cloud.

We have to consider the model's memory footprint. While usage on a cloud system has no practical limits on memory size. But local systems present different challenges. Here we want a sufficiently small model.

In terms of inference speed, we aim for real-time or at least near-real-time responses. For our use case, the worst acceptable response time would be the variable time the user would need to just type the command. The additional response time in case of cloud-based systems is not considered in this thesis.

2.3 Tokenization

Tokenization is the process of splitting a sequence of text into subsequences. The granularity of those subsequences can change from entire phrases to single characters. For the task of autocompletion, we need a model with at least subword-based tokenization. A word-based model would not suffice because these models are not able to autocomplete words as well.

3 Setup

In this chapter we list the devices, languages and librarys used for our testsetup. The laptop described in Table 1 was used simply because it was available. It was used for local tests while larger models were done on the HILBERT, the HPC supercomputer of the HHU. The enviroment had 40GB of RAM available.

the Alpaca test were performed on HPC.

We used a Miniconda 3.1 enviroment

Item	Specification
Laptop Model	ThinkPad L590
Operating System	Ubuntu 22.04.3 LTS x86_64
CPU	Intel i7-8565U (8) @ 4.600GHz
RAM	16Gb

Table 1: local machine specs

- `transformers-4.30.2-pyhd8ed1ab_1.conda`
- `tokenizers-0.13.3-py38h7d131c9_0.conda`

Pip

- `numpy             1.25.0`
- `typing_extensions 4.7.0`

python version 3.11.3

4 The Program

For a practical application, we need a program that actually completes the commands.

Ideally, the autocompletion integrates seamlessly with the normal usage of the command-line interface. For the sake of this thesis, we have to limit our program to its base functionality. Which means that advanced functions like the coloring of the prompt are not implemented. Our approach here is to build a wrapper around the terminal interface, taking the user input, handing it to the terminal interpreter, and adding a completion if instructed to do so. The output of the command is displayed as usual.

We chose to implement it on Linux as a bash shell. The reason for this is that, aside from a pick-one solution, Linux among the major operating systems has the biggest focus on CLI usage and is even usable on Windows through the use of WSL.

Shell script is chosen as the language because it is able to run on most UNIX-like systems.

Our script, upon running, presents us with a terminal prompt containing the current directory, which is also given to the autocompletion along with the typed prompt by pressing the TAB key. The complete command can be executed like in a normal terminal by pressing the Enter key.

Another function is that the program records the executed command in a history database, which is separate from the normal history. The idea behind it is that these commands, along with the path they were executed on, can be used to fine-tune the model in use if so desired.

The autocompletion itself happens in a separate Python file. The reason for this is that Python offers better support for machine learning tasks compared to Bash. This also offers the opportunity to use code from the model test with slight modifications.

In the Python file, the specific model is chosen in a config file, which can be changed by the user.

The prompt template is modified by the directory, and filenames are given to the model.

Then the model uses it as context to predict a completion of the command. After that the predicted command is then returned.

5 The Model

Since we want to complete CLI commands, we need a model that performs well on autocompletion tasks. So we prefer one whose architecture benefits such a task.

From a financial perspective, models that are not free to use,are not apropiate which limits the use of non-free APIs.

For local usage we want a sufficiently small model. This however does not apply for cloud usage.

For practical reasons, we limit ourselves to models that were publicized at the beginning of writing this thesis. So models like LLaMA2 which was publicized during the writing process of this thesis were not considered.

5.1 Rejected Models

5.1.1 BERT

Bidirectional Encoder Representations from Transformers or short BERT was introduced 2018 in BERT: Pre-training of Deep Bidirectional Transformers for

Language Understanding [1]. It was the first major Bidirectional model, which means it can use the context of right sided input in addition to left sided input. Unlike most previous models, which could only use left sided input.

BERT's training data consists of two sources, the Bookscorpus and the English Wikipedia. The Bookscorpus is a dataset of textbooks containing approximately 800 million words, providing roughly one third of BERT's training data. The English Wikipedia dataset as its name implies contains texts from English Wikipedia articles limited to only the text passages. The lists, tables and headers have been removed.

Its size consists of approximately 2500 million word which makes it the majority of BERT's training data. [1]

Given the nature of an autocompletion task, this ability is not helpful. Since the model has to predict the entire right side of its input, there is no right side input it could use. So it would behave like a unidirectional model, losing its biggest architectural benefit.

However, it is to note that BERT's bidirectional encoding during training and the so encoded knowledge can benefit the predictions for situations where only one directional encoding is possible.

The next point is that BERT is only able to use 512 tokens as context, which limits the size of its prompt length. While the template itself has a fixed number of tokens, the Path is of variable length. As well as the number of files.

The last issue with this model is the way its encoder handles out-of-vocabulary(OOV) tokens. Since Bert's encoder works with a fixed vocabulary, it will not be able to process words that are not in its vocabulary. This poses a huge issue for named entities. This would include the names of programs, with whom most terminal commands start. As well as the names of files and directories, which are given as context to the model.

In our specific use case, this would mean that BERT could not use the given context and would be unable to complete some commands reasonably.

These arguments lead to the conclusion that BERT is not suitable for this task. Therefore, we reject its use.

5.1.2 GPT-3/GPT-4

GPT-3 was first published in "Language Models are Few-Shot Learners" on May 28, 2020 by a group of researchers at OpenAI

While GPT based models as of now are considered to be the best LLMs. Their size and needed computational power to operate them make them impractical for consumer grade computers. This can be circumvented by the use of the OpenAI API, but this is not free of charge and needs to sent data over the internet, which prohibits its use with sensitive data and cannot be used on capsuled systems.

5.1.3 LLaMA

LLaMA is a LLM first introduced 2023 in "LLaMA: Open and Efficient Foundation Language Models"[7]. It is trained on a diverse set of 100% openly available text data that is compatible with open sourcing[7].

Table 2: Pre-training Data[7]

Dataset	**Sampling prop.**	**Epochs**	**Disk size**
CommonCrawl	67.0%	1.10	3.3 TB
C4	15.0%	1.06	783 GB
Github	4.5%	0.64	328 GB
Wikipedia	4.5%	2.45	83 GB
Books	4.5%	2.23	85 GB
ArXiv	2.5%	1.06	92 GB
StackExchange	2.0%	1.03	78 GB

The datasets have been preprocessed. The English CommonCrawl was deduplicated at the line level, and non-English pages and pages not reverenced on Wikipedia have been removed. C4 has been deduplicated and filtered with heuristics like the presence of punctuation and number of words on a webpage. GitHub projects that are distributed under Apache BSD and MIT licenses were used Low quality code and boilerplate code were removed. Wikipedia dumps from the June-August 2022 period, covering the languages: bg, ca, cs, da, de, en, es, fr, hr, hu, it, nl, pl, pt, ro, ru, sl, sr, sv, uk. Hyperlinks comments and formatting boilerplate have been removed. The Gutenberg Project corpus and Books3 corpus have been deduplicated and books with an 90% overlap have been removed. ArXiv Latex files have been filtered by removing anything before the first section, as well as the bibliography. Stack Exchange data includes the high quality questions and answers.

LLaMAs architecture is based on the transformer architecture, but leverages several improvements. Pre-normalization, where the input is normalized before each sublayer instead of just normalizing the output. It uses the Root Mean Square Layer Normalization[10].

$$\|\mathbf{x}\|_{\text{RMS}} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}|\mathbf{x}_i|^2}$$

The ReLU activation functions have been replaced with SwiGLU activation functions.

$$\text{SwiGLU}(x) = x \cdot \sigma(\beta x) + (1 - \sigma(\beta x)) \cdot x \tag{1}$$

where $\sigma(z) = \frac{1}{1+e^{-z}}$ is the sigmoid function.

Last, the absolute positional embeddings have been replaced with rotary positional embeddings. It is faster than absolute positional embeddings and doesn't add significant computational overhead. This is beneficial for our use-case of generating completions in real-time. The reason for this is that it does not need to learn a separate embedding for each position in the sequence. Since it does not assume that the positions of tokens in a sequence are independent of each other, it is more capable of learning long-range dependencies between tokens. This also removes the maximum limit of tokens a model can use as input. In other words, our prompt can be as long as we like, so there is no limit on the length of our prompt template and the information we can give it. But it is important to note that the limit of quadratic growth of needed computational resources relative to the sequence length remains.

Its tokenizer uses byte-pair-encoding[7]. The authors didn't use their own implementation, instead they used the implementation from Sentence-Piece [2]

While this model is acceptable for our task, we reject it in favor of Alpaca, which is a fine-tuned version of LLaMA.

5.2 Accepted Models

5.2.1 GPT-2

Generative Pre-trained Transformer 2 or short GPT-2 was introduced in 2019 in the paper titled "Language Models are Unsupervised Multitask Learners"[3]. Unlike BERT, it is unidirectional. Its architecture is transformer based. Similar to the GPT model with the modification, the Layer normalization was moved to the input of each sub-block and an additional layer normalization was added after the final self-attention block[3].

Its Training data consists of Bookcorpus and Webtext thus containing the same training data as BERT. Additionally, it contains code from GitHub and text from Reddit.

The size of its vocabulary is 50,257[3].

It utilizes byte pair encoding (BPE) as an encoding method, which is a form of subword tokenization. This process works by repeatedly merging the most common tokens.

This way the most common words will be represented as single tokens while less common words will be split into subwords. This way words that are OOV can be split into repetitively till known tokens are reached. This way, unknown words can be handled based on their subwords.

5.2.2 Alpaca

Stanford's Alpaca Model is a fine-tuned version of LLaMA. The underlying architecture is the same LLaMA's, therefore, we refrain from describing them in detail again. It was fine-tuned with 52K instruction-following demonstrations, which were generated in the style of self-instruct [9] with text-davinci-003. It is important to note that this was only done for LLaMA-7B and LLaMA-13B, not for the larger versions.

Hyperparameter	**LLaMA-7B**	**LLaMA-13B**
Batch size	128	128
Learning rate	2×10^{-5}	1×10^{-5}
Epochs	3	5
Max length	512	512
Weight decay	0	0

Table 3: Hyperparameters for LLaMA-7B and LLaMA-13B models.[5]

The instruction-following demonstrations followed the format: instruction, input and output[9]. The instructions describe the task the model should perform. In the training data, all of them are unique. The input provides optional context and only appears in roughly 40% of the data. This is similar to our approach. In our case the instruction is our prompt-template with added context for directory and files and the input is the command that is to be completed. The output, however, is to be generated by the model.

We are interested in an attempt to run it on a local machine, therefore, we will use the Alpaca-7B version because it is the smaller.

6 Prompt Template

The specific input to guide the behavior of a machine learning model is called a "prompt". The model uses it as context. The way the prompt is worded can heavily influence the behavior of the model. Therefore, we need a prompt that accurately describes the tasks at hand. Which is to complete CLI commands. Therefore, it makes sense to simply tell the model what it is and what we want the model to do.

Providing a Linux system as context has the advantage to give the model context which completion would be plausible.

But this limits its usage to system specific commands. To give an example of this, the "ls" command would closer relate to a Linux terminal as the Windows/DOS equivalent "dir". This can easily be adjusted by replacing the system's name in the prompt.

Another thing to note is that all sentences in the prompt-template, and all variations of it, are using the English language. The reason for this is that most LLMs including the ones chosen in the models section are trained with mostly English training data. Therefore, our models will likely have an easier time processing the English language compared to other languages.

Table 4: Prompt Variations

Premises	• You are an autocomplete function. • This is a Linux terminal. • This is a Linux terminal command. • This is an autocomplete function.
Order	• Autocomplete the following Linux terminal command and provide no further explanation for the command:
File Contexts	• There are the following files in the current directory: • Files: • These files are in this directory:

A number of prompt combinations have been tested and for simplicity we decided to choose the variation that provided the best outcome

The "You are an autocomplete function. " and "This is an autocomplete function. " tend to provide significantly worse results than other premises. There are less likely to produce a terminal command, but a text about said command.

"This is a Linux terminal command. " provides better output but tends to append an explanation of the command. "This is a Linux terminal. " tends to provide the best solutions.

Path and file in the current directory are not specified in the final prompt because these are defined by the context.

The file contexts show no significant differences, so "There are the following files in the current directory: "is chosen to pick one for simplicity.

So the final prompt is: "This is a Linux terminal. There are the following files in the current directory: <files>,Path: <path>, Autocomplete the following Linux terminal command and provide no further explanation for the command: <command>".

Now that we have the prompt template, we can tend to the tests.

7 Tests

List of commands used

We used the following commands to autocomplete:

"sudo apt","sudo apt up","sudo apt in","ls","py","pyt","pyth","pytho","git","git i","git in","git ini","git co","git comm"

The reasons we chose these are to test four scenarios. First, the relative context independent apt commands. Here we try to take a look how successful the autocompletion can be on a command that is not influenced by its context to have a comparison with commands that are more influenced by their context.

Second, the "ls" command. The goal here is to see what happens when the models have to handle very short commands or, in other words, have few tokens to work with.

Third, python based commands. The python command is supposed to run python code, this can be in an interactive shell or a python script. While the interactive shell isn't that reliant on context, the running of a python script is. A prediction in this case would only be viable if the predicted file is a python file present in the current directory, not accounting for possible parameters given to the script.

Last git based commands. The git commands are heavily influenced depending on where you run them and what files are in the current directory. This however varies from command to command.

For context data we had a simple a mock git repository to see how the predictions of the git based commnads are influenced. The ".deb" files are for the sam reason for the apt based commands.

Files
"tests.py, data.txt ,webserver.py "
"webserver.py, config.txt "
".git, .gitignore, README.md, webserver.py, config.txt, tests.py, data.txt "
"javasdk.deb"
"geany.deb"

Table 5: List of tested files

Paths
"/home "
"/home/user/project "

Table 6: List of tested paths

7.1 Tests with GPT-2

All tests with the GPT-2 model were conducted on the laptop mentioned in the setup section. Our first test with the model had just the incomplete command as input without any context added. The completions generated with the model were just nonsensical text.

7.1.1 Tests with the Prompt-Template

The apt based commands worked insofar that it predicted the most commonly used apt commands update and install. also tey contained a lot of followup text relating to the command, or a repetition of text that resembles apt based commands.

The ls command gets completed with random parameters 60% of the time.

The python based commands are completed into a nonsensical text about python. However, the command never gets completed into the word "python" even if the word appears in the appended text. A common pattern is that the python command gets repeated, for example "pyth" gets completed into "pythpyth".

The git commands had way worse results. While trying to use the "git init" command given the "git ini" was not able to predict the "t" of "init" correctly, and the "git c" wasn't able to predict anything near "git commit" not even the "git comm" could predict "commit". Most of the time, a lot of unwanted text was appended.

7.1.2 Tests with Context

The apt based commands had similar completions compared to the predictions without the prompt-template, it is however to note that sometime the path or the file names were mentioned.

The ls command sometimes gets the directory as an argument which is valid, but sometimes a file in the same manner which is not.

The biggest difference with the python based commands here was that the context information was referenced in the completions.

The completions of the git based commands showed references to the given context, but no improvements, save some hallucinated git repository.

7.2 Tests with Alpaca-7b

All tests with the Alpaca model were conducted on the HPC.

The first test with the model used just the incomplete command as input, without any additional context.

The apt commands often get completed into a text about apt based commands. While the ls command doesn't get interpreted as a command, presumably because it is short.

The python based commands mostly get completed into nonsensical text. Sometimes with python, commands with more letters of the word "python" gets interpreted as a type of snake, which is not surprising considering the training data probably contains more text about snakes than the programming language. The git based commands again get completed into nonsensical text.

7.2.1 Tests with the Prompt-Template

The added prefix "This is a Linux terminal. There are the following files in the current directory:,Path:, Autocomplete the following Linux terminal command and provide no further explanation for the command: <command>" shows an improvement, with apt and ls commands,

If the command "py" gets completed to the intended completion "python" it always gets completed with the spelling error "pyhton" switching "h" and "t". The command "pyt" gets completed into "pytorch" 80% of the time, but the other 20% it gets completed into other misspelled versions of "python".

The git based commands get completed in to nonfunctional git commands and text about these commands. One exception is "git" which 2/3 of the time gets completed into a valid command, except that the git repositories are not existing. Otherwise, the command was a valid command with added text.

7.2.2 Tests with Context

If we add a path and files in the current directory as context, we still cannot produce valid commands most of the time.

Noteworthy here is that 25% of the time the python commands don't get any completion at all.

A similar issue sometimes happens with the ls command, but this is still a valid command.

The ls command gets completed into a valid command 30% of the time. The most common invalid prediction was "ls." which was occurring round 30% of the time.

Also, the ls command sometimes gets completed with the Path. For example, "ls" with Path:"/home/user/project" gets completed into "ls -l /home/user/project"

With added context, the git based commands 1/5 of the time get completed into a text referencing the mentioned files and directory. Also, two things are noteworthy. The first is that one completion occurred, that clones an existing repository. The other is the only valid occurrence of a completion of "git commit" which was completed from "git" and not from "git co" nor from "git comm".

8 Conclusion

The use of a prompt template showed a clear improvement compared to a prompt containing just the uncompleted command insofar that the text produced by both models was always to some extent related to a Linux terminal command even tough mostly an explanation of a command, even tough the prompt -template was designed to avoid this.

The added information of path and filenames added little to no benefit to the completions. Often, the models hallucinated mixtures of both values. In our applied measures, we could not get either model to stop hallucinating things.

In terms of inference neither model had the speed to complete the commands in real-time while GPT-2 with an average time of 4 seconds was closer than Alpacawith an inference time ranging from 9 to 128 seconds. The more accurate Alpaca model is too large to be deployed on a low powered machine, but can be deployed on a cloud server. Alpacas inference time was way too slow for practical use.

The base models are unfit for an autocompletion function, so further measures like fine´tuning and LoRa are reasonable.

9 Possible Extensions

It could be interesting to investigate if and how previous commands influence the outcome. Commands from the history could be added to the prompt. Although, since this could be a high amount of tokens, the computational power needed, would be higher and the token limit of the model could be exceeded. Therefore, the commands need to be filtered.

9.1 Other Alpaca Versions

Stanford's Alpaca model was created with the 7B version of LLaMA as a base, but LLaMA has bigger versions. Since the code for the training process and the used training data are publicized by the creators. The same process can be used to create larger versions of Alpaca. Which could possibly perform better than the 7B version. The process of fine-tuning the larger models is more expensive than the fine-tuning of the 7B version. The same or similar tests to those already conducted, could be performed with such models. It is noteworthy that, this could not be performed on most consumer devices, but a cloud application might be feasible.

9.2 LLaMA2

While this was written, LLaMA2 was released.[6]

LLaMA2 itself could be used as a model. But since the architecture of LLaMA2 remains largely unchanged from its predecessor, LLaMA. It could also be trained like Alpaca. It could be interesting to see if this new model produces a similar outcome or not.

9.3 Finetuning

In our tests, we concluded that the models used did not produce satisfying results. For this reason, we should consider fine-tuning them. If real-world data is not available. Training data can be generated by AI's such as Chat-GPT. But to mitigate the bias of the model that produces the training's data. Several models should be used to produce training data. However, this doesn't help if the models have shared biases.

9.4 Out-Of-Vocabulary Words

A general problem for all models is the handling of tokens that are unknown to the tokenizer. Although some models handle this problem better than others, all models tend to handle these tokens worse than their in-vocabulary counterparts. Therefore, it could be interesting to research how presence and absence of tokens in the tokenizer influence the predictions of our models.

9.5 LoRa

The larger the model, the larger the computational resources needed to train and deploy them.

In the case of LLMs this can be very costly as seen in with our tests with Alpaca.

Here, Low-rank adaptation (LoRA) can help us. The idea is to decompose the model's weight-matrix into a lower-dimensional representation, that captures most of the information of the original model. This would reduce the number of parameters greatly. Ideally this would allow us to run larger models like Alpaca on local machines.

9.6 Context Extension to Subdirectories

While our current approach is limited to the files in the current directory, we could extend this to files in the subdirectories of the current directory. This could give the model a better idea of its surroundings and therefore, provide more information for the model's predictions. However, depending on the model that is used, the size of the context might be an issue.

References

[1] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL].

[2] Taku Kudo and John Richardson. *SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing*. 2018. arXiv: 1808.06226 [cs.CL].

[3] Alec Radford et al. "Language Models are Unsupervised Multitask Learners". In: 2019. URL: https://api.semanticscholar.org/CorpusID:160025533.

[4] Or Sharir, Barak Peleg, and Yoav Shoham. *The Cost of Training NLP Models: A Concise Overview*. 2020. arXiv: 2004.08900 [cs.CL].

[5] Rohan Taori et al. *Stanford Alpaca: An Instruction-following LLaMA model*. https://github.com/tatsu-lab/stanford_alpaca. 2023.

[6] Hugo Touvron et al. *Llama 2: Open Foundation and Fine-Tuned Chat Models*. 2023. arXiv: 2307.09288 [cs.CL].

[7] Hugo Touvron et al. *LLaMA: Open and Efficient Foundation Language Models*. 2023. arXiv: 2302.13971 [cs.CL].

[8] Ashish Vaswani et al. "Attention is All you Need". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[9] Yizhong Wang et al. *Self-Instruct: Aligning Language Models with Self-Generated Instructions*. 2023. arXiv: 2212.10560 [cs.CL].

[10] Biao Zhang and Rico Sennrich. *Root Mean Square Layer Normalization*. 2019. arXiv: 1910.07467 [cs.LG].