INSTITUT FÜR INFORMATIK
Datenbanken und Informationssysteme

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

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 way 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 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 locale solution. This again would limit our available

resources. Therefore, we have to make a trade-of between accuracy and size of the model.

Contents

1 Technical Background

Since the training of LLMs would take too much time and be very expensive, it's therefore not feasible. We just have to use pretrained models.

The paper "Attention is all you need" [**NIPS2017_3f5ee243**] introduced the transformer architecture in 2017. Its novelty idea was the introduction of the attention mechanism. A mechanism to weight how significant a token is to another token. This allows a model to learn long-range dependencies between words in a sentence. Since then, it has been used to develop LLMs that surpass models based on earlier architectures.

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 in autocompletion tasks.

We have to consider the model's memory footprint. While usage on a cloud system has no practical limits on memory size, locale systems are a different thing. Here we want a sufficiently small model.

In terms of inference speed, we aim for real-time or 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.

The final limitation given the task of autocompletion is that we need a model with sub-word based tokenization or similar. A word based model would not suffice because these models are not able to autocomplete words as well.

2 Setup

For reproducibility we have to mention what kind of system and what kind of software was used.

i used a Miniconda 3.1 enviroment

- `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

system i testet it on

3 The Programm

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 the 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 ordered to do so. The output of the command is displayed as usual.

We chose to implement it on Linux for a bash shell. The reason for this is that aside from a pick one solution is that 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 if 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, this is separate from the normal history. The idea behind it is that these commands along with the path they were executed in can be used to fine-tune the used model 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 is given to the model Then the model uses it as context to predict a completion for the command. The predicted command is then returned.

4 The Model

The Training of an LLM from scratch would be too cost/time intensive for this thesis, so the use of a pretrained LLM is far more practical.

Now we come to the question: Which model to choose and why?

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, it should be free to use, which limits the use of non-free APIs.

Since we want to use our autocompletion system on a <fullfill>

<deprecated> We choose the specific models according to the categories of size speed and accuracy of output. While these factors can be influenced by fine-tuning and methods like

Lora. We choose the strategie to pick the models with the best baseline.

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.

4.1 Rejected Models

4.1.1 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.

4.1.2 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 the impractibel for consumergrade computers. This can be cirkumvented by the use of the opeaiAPI 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 capsueld systems.

4.1.3 LLaMA?

LLaMA is a LLM first introduced 2023 in "LLaMA: Open and Efficient Foundation Language Models"[3] . It is trained on a diverse set of 100% openly available text data that is compatible with open sourcing[3] .

Table 1: Pre-training Data[3]

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

LLaMAs architektur is based on the transformer architektur but levarages several improvments. Pre-normalization where the input is normalized before each sublayer instead of just normalizing the output.It uses the Root Mean Square Layer Normalization[**zhang2019root**].

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

The ReLU activation functions have been replaced with SwiGLU activation functions. The replacement of ReLU activation functions 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 dosen't add significant computational overhead. This is benficial for our usecas 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 importent to note that the limit of quadratic growth of needed computational resources relative to the sequrnce length remains.

Its tokenizer uses byte-pair-encoding. The authors didn't use their own implementation instead they used the implementation from Sentence-Piece [**kudo2018sentencepiece**]

4.2 Accepted Models

4.2.1 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"[2] . Unlike BERT, it is unidirectional.

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.

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.

4.2.2 Alpaka

Stanfords Alpacka Model based on Metas LLama Model semes to be a good candidate. We will use the Alpaca-7B version because it is the smalest.

5 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.

Table 2: 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

6 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

they are run

tabelle fur die tests mit context

6.1 tests with gpt2

The unfinetuned gpt2 was chosen for its small size of round 550mb. Which makes it runnable on a low resource machine without any further processing.

We used it with the Huggingface API since it is easy to use and the most popular API for such tasks. The "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>" prompt as decided on in the previous section was used.

The model produced no valid commands while some of the git completions resembled some git commands, none of them were valid.

While the speed was sufficient, its failure to produce valid commands suggests that

it is probably better to use a different model.

I tested two scenarios with the gpt2 model first the apt based commands that worked insofar that it predicted the most commonly used apt commands update and install also with a lot of followup text, however most apt based commands are independent from their context insofar that they don't rely on the path they are executed in. It also has to be noted that the model produced more text appended to the command

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 even then not without a lot of unwanted text appended. The prepending of "Autocomplete the following terminal command:" seems to produce less unwanted text but still no usable output.

6.2 test with alpaca-7b

While alpaca gives far better results, there are limits.

A test without any prefix showed only nonsensical results.

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.

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, with the notable outlier "py" who gets completed to the misspelled "pyhton" following a text about python.

The git based commands get completed in to nonfunctional git commands and text about

these commands.

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

Noteworthy here is that sometimes the python commands dont get any completion at all.

This suggests that the base model is not able to do it in a sufficient way. Which makes methods like Lora not viable to counter its slow speed and big size. Since it would lower its accuracy.

Therefore fine-tuning the model is the next logical step.

7 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 therefore would be higher and the token limit of the model could be exceeded. Therefore, the commands need to be filtered.

A different model could also be tested. While this was written, LLama2 was released by Meta. It is a successor of LLama which alpaca is based on. It was not used solely for the reason that it came out late in the writing of this thesis.

fientuning the tested models

References

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

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

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

[4] Ashish Vaswani et al. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL].