Hasso Plattner Institut
Digital Engineering · Universität Potsdam

Syntax-error-free text generation

Machine Intelligence with Deep Learning
Winter Semester 2023/2024

Jan Vincent Hoffbauer

Supervisors:
Weixing Wang,
Dr. Haojin Yang

June 17, 2024

Abstract

Large Language Models (LLMs) have become integral in natural language processing tasks, yet they encounter challenges when incorporating external tools for tasks requiring symbolic reasoning or real-world interactions. Based on previous research, this study proposes a method to improve LLMs' tool-calling abilities by implementing syntax constraints during decoding. We present a dual-phase approach, integrating finite-state automata (FSA) to enforce syntactic correctness while preserving generative flexibility. Through experiments on mathematical reasoning tasks, we demonstrate significant accuracy gains over existing methods. Our findings underscore the efficacy of syntax-constrained decoding in enhancing LLMs' utility in complex computation and API interactions, paving the way for broader real-world applications.

Contents

1 Introduction

Large Language Models (LLMs) have developed into one of the major topics of research within the NLP community. Applications such as ChatGPT are used as highly capable, general-purpose assistants by many knowledge workers in day-to-day tasks. The first widespread adoption was found by GPT-3 [BMR+20] and sparked an interesting line of research as well as commercial product development. The release of publicly available models such as LLAMA-2 [TMS+23] and Mistral [JSM+23] democratized research efforts and led to a wide variety of research teams training LLMs as general assistants or question-answering systems. The recent Zephyr 7B β model [TBL+23] was trained with Direct Preference Optimization (DPO) [RSM+23] and counts as one of the best performing open-source dialogue-tuned LLMs [BFH+23].

However, despite these models being trained on trillions of tokens during pre-training and incorporating a vast set of capabilities, they still struggle with certain tasks that involve incorporating up-to-date world knowledge, symbolic reasoning, or calculations with large numbers. To alleviate these issues, a recent line of work allows these models to integrate with external APIs during inference, so-called tools. Such tools span from access to external APIs or knowledge bases to running Python code or using a calculator.

Accessing external tools in itself poses a vast variety of challenges, for example, models need to be able to decide which tools are relevant and potentially even plan the sequence of tool calls if multiple are required. Another challenge is that a tool is called using a highly structured language, such as programming languages. LLMs oftentimes fail to generate syntactically correct tool calls where simple constraints are not adhered to or non-existent tools are hallucinated. In the following, an exemplary question with a failed and syntax-constrained response is presented. Assume a question

```
Question: What is the area of a square with a sidelength
   of 5 meters?
```

While an LLM without syntax constraints could freely generate a toolcall such as

```
Answer: The area is <T>multiply(5*5)=25 metres ####
```

the syntax-constrained model would always generate the correct response

```
Answer: The area is <T>multiply(5, 5)=25 metres ####
```

Based on [ZCLW23], we evaluate the effectiveness of using syntactic constraints during inference to improve the tool-calling-ability of LLMs. To achieve that, we evaluate LLMs on the FuncQA [SDYD+23] dataset with and without syntactic constraints. We also further explore fine-tuning these models on the dataest to further improve the tool-calling ability.

In section 2, an overview of the existing body of work is provided. Then, in section 3 the methodology for our experiments is presented, and their results in section 4. Finally, in section 4, the work is summarized and critically discussed.

2 Related Work

[KHM+23] provides an overview of the limitations of LLMs which include things such as the dynamic retrieval of up-to-date knowledge, the access to external APIs to interact with the real world, or the ability to execute complex calculations or mathematical reasoning. [XCG+23] thus introduces the idea of agents, which are systems with an LLM at their core that have access to various external tools such as APIs, calculators, etc. In the following, we will give a brief overview of the developing landscape of works that improve the ability of LLMs to interact with such tools.

Prompting ReAct [YZY+23] is a prompting framework that presents a prompting template to be used with PaLM [CND+22] that allows the LLM to interact with a simple Wikipedia API. Their method outperforms raw LLMs without re-training the LLM.

Training [HLWH24] presents a novel fine-tuning method that creates new tokens for each tool which are then fine-tuned. This method is significantly more resource-efficient than full-parameter tuning. They also propose a method that involves backtracking during the generation to deal with syntax errors that can occur during tool calls. Their method outperforms previous state-of-the-art approaches.

Decoding As one of the main issues when calling tools is that the model generates syntactically incorrect tool calls, [ZCLW23] proposes a mechanism that constrains model generation to only generate valid toolcalls. This approach outperforms previous state-of-the-art and is an important foundation for our work.

3 Methodology

3.1 Syntax constrainted decoding with finite-state-automata

To allow syntactically constrained toolcalls while keeping the model's ability to generate an internal monologue, i.e. CoT [WWS+23], we break down the generation process into multiple phases:

- **Thinking** during which the model generates free text that represents its internal monologue
- **Toolcalling** during which the generation is syntax constrained and the generation is evaluated as a toolcall.

The transition from the *thinking* phase into the *toolcalling* is induced by the model generating a special token $\langle \texttt{T} \rangle$.

In the following, the implementation of the syntactic constraints during the *toolcalling* mode is explained

3.1.1 Syntax of toolcalls

The syntax of toolcalls must be well-defined. Formally, it can be defined for all programming languages using a context-free grammar (CFG) [Cho56]. In special cases, when the syntax is non-recursive, it can be defined using a regular expression (RegEx) [Cho56]. While context-free grammar can be parsed effectively on the CPU, fast parsing is challenging on the GPU. On the other hand, a RegEx can be directly converted to a finite-state automata (FSA), which can be represented as a set of Tensors and effectively evaluated.

The syntax of our toolcalls can be provided as a CFG as follows:

$$
\begin{aligned}
S &\rightarrow N \text{ '(' } A \text{ ')'} \\
N &\rightarrow \texttt{string} \\
A &\rightarrow A' \mid A' \text{ ', ' } A \\
A' &\rightarrow N \mid N \text{ '.' } N \\
N &\rightarrow \texttt{digit}\, N \mid \texttt{digit}
\end{aligned}
$$

It is worth mentioning that this specific CFG is in fact non-recursive, i.e. a regular expression, We can represent it as a FSA, which is depicted in Figure 1.

A CFG or an FSA accepts a string $w_1^T = w_1, \ldots\, w_T$ of length T if that string can be generated by the CFG or ends in a final state when applying the FSA.

3.1.2 Number of hops

Given that certain tasks can only be accomplished by combining multiple tools, ambiguate **multi-hop** tasks, which require more than one toolcall, and **single-hop** calls which require exactly one toolcall to be accomplished. During generation, the model needs to decide when a given answer is considered done, i.e. when the last toolcall yielded the correct result. To implement this, a sequence `####` is defined upon which the generation terminates.

3.1.3 Decoding Algorithm

The general decoding algorithm is depicted in Algorithm 1. It uses the logit processor described below to implement the syntax constraint. For simplicity, the algorithm is presented in a single-entry format while the actual implementation supports batching.

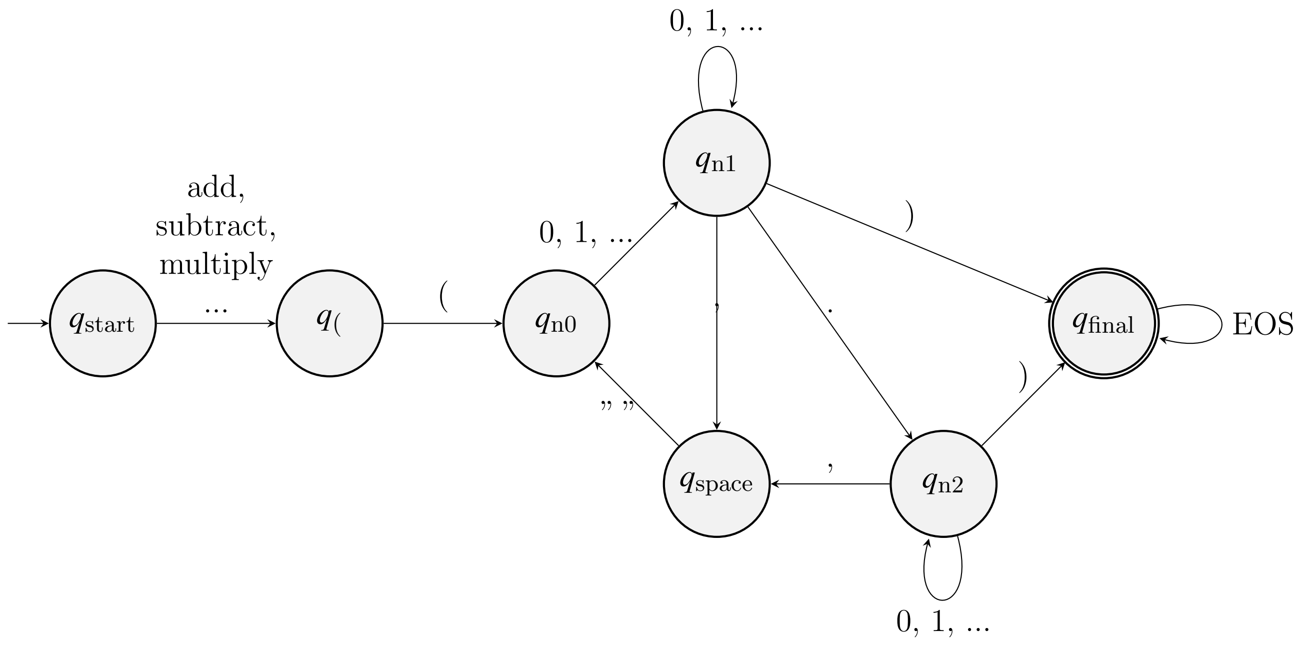

Figure 1: FSA for toolcalls in the format that we use for evaluating on FuncQA.

Logits Processor Constrained generation uses a concept called *Logits Processor* that changes the logits (i.e. the un-normalized scores in log space) according to the syntax constraints. Our logit processor will mask the logits during the generation. I.e., let $q(w_1^T, t) \in \mathbb{R}$ be the LLM that calculates the logit of a token $t \in V$, given $w_1^T \in V^T$ as the concatenation of the prompt and the previously generated tokens. The constraint is applied by redefining q as follows

$$q'(w_1^T, t, i) = \begin{cases} q(w_1^T, t), & \text{if } w_i^T \text{ is accepted by the CFG or FSA} \\ -\texttt{inf}, & \text{otherwise} \end{cases}$$

with the additional parameter i that denotes the starting index of the toolcall.

It's worth mentioning that the FSA can be represented as a set of tensors and thus be fully evaluated on the GPU, yielding significantly faster inference speeds.

3.2 Datasets

To train and evaluate our models, we use two different datasets for mathematical reasoning tasks. An overview of their size and structure is provided in Table 1.

3.2.1 FuncQA

The original FuncQA dataset [HLWH24] consists of 650 single-hop questions in the train set and 65 multi-hop questions in the test set, which are synthetically generated using

Algorithm 1 Generate Interleaving

```
1: procedure GENERATE(prompt, model)    ▷ Interleave thinking and toolcalling mode
2:     $i \leftarrow 0$
3:     $r \leftarrow$ None
4:     while $r \neq$ error and #### not generated and $i <$ max_iterations do
5:         $o \leftarrow$ model.generate(prompt, stop_on='<T>')    ▷ Thinking
6:         $t \leftarrow$ model.generate(prompt + o, logits_processor)    ▷ Toolcalling
7:         $r \leftarrow$ eval(t)    ▷ Evaluate toolcall or error
8:         $i \leftarrow i + 1$
9:     end while
10:    return $r$    ▷ The final answer is the result of the last toolcall ####
11: end procedure
```

Dataset	Train	Evaluation	Test	#tools
FuncQA	585	65	65	13
GSM8k	4128	791	459	4

Table 1: Overview of the datasets used during these experiments

ChatGPT [Ope24]. The dataset contains questions for 13 different math operators.

To make sure that the dataset allows both comparability with the original metrics, as well as features a proper train, validation, and test split, we adapt the split. We also want to ensure that we have an equal distribution between our training and test data. Thus, we choose the following split:

- test: 65 multi-hop questions from the original test set, and 65 single-hop questions from the original train set
- validation: 65 single-hop questions from the original train set
- train: the remaining 585 single-hop questions from the original train set

In addition to changing the split, we reformat the toolcalls to feature the syntax outlined in subsection 3.1 by applying suitable regex-based preprocessing.

3.2.2 GSM8K

The Grade School Math 8k (GSM8K) dataset is a dataset of 8.5K linguistically diverse grade school math questions designed for multi-hop reasoning. Solutions involve basic arithmetic operations and can be solved by a middle school student without advanced concepts, using only addition, multiplication, subtraction, and division.

The train set contains 7.47k questions while the test set contains 1.32k questions. Since we only use the four basic operators, we have four toolcalls.

To preprocess the dataset, all specially marked equations, e.g. $\langle\langle 4/2\rangle\rangle$, are converted to appropriate toolcalls by following the below sequence

1. Remove all answers with nested equations, e.g. $\langle\langle 4/2+1\rangle\rangle$ as we do not allow any nested toolcals
2. Convert all marked equations into toolcalls with our pre-defined syntax.

In addition to that, the original train set is split into a train and validation set.
It is important to note, that the dataset does not mark every single equation in the answer text, so that our preprocessing pipeline might miss them.

3.3 Prompt

The model is prompted using the following prompt that explains the available toolcalls and the used syntax in a few-shot format.

```
You are a helpful chatbot. You answer the following
   question using toolcalls to the tools: add, subtract,
   multiply, divide, power, sqrt, log, lcm, gcd, ln,
   choose, remainder, and permutate. You initiate the
   toolmode with the token <T>. You answer only the
   question the user provides you. End every answer with
   ####

Question: A coin is tossed 8 times, what is the probability
   of getting exactly 7 heads ?
Answer: The total number of possible outcomes to toss a
   coin 8 times is 2^8=<T>power(2,8)=256. The number of
   ways of getting exactly 7 heads is 8C7=<T>choose(8,7)=8.
   The probability of getting exactly 7 heads is
   8/256=<T>divide(8,256)=0.03125. ####

Question: If paint costs $3.2 per quart, and a quart covers
   12 square feet, how much will it cost to paint the
   outside of a cube 10 feet on each edge?
Answer: The total surface area of the 10 ft cube is
   6*10^2=6*<T>power(10,2)=100=<T>multiply(6,100)=600
   square feet. The number of quarts needed is
   600/12=<T>divide(600,12)=50. The cost is
   50*3.2=<T>multiply(50,3.2)=160. ####

Question: log(x)=2, log(y)=0.1, what is the value of
   log(x-y) ?
```

```
Answer: log(x)=2, so x=10^2=<T>power(10,2)=100; log(y)=0.1,
   so y=10^0.1=<T>power(10,0.1)=1.26;
   x-y=100-1.26=<T>subtract(10,1.26)=98.74, so
   log(x-y)=log(98.74)=<T>log(98.74)=1.99. ####

Question: How many degrees does the hour hand travel when
   the clock goes 246 minutes?
Answer: The hour hand travels 360 degrees in 12 hours, so
   every hour it travels 360/12=<T>divide(360,12)=30
   degrees. 246 minutes is 246/60=<T>divide(246,60)=4.1
   hours. The hour hand travels
   4.1*30=<T>multiply(4.1,30)=123 degrees. ####

Question: {question}
Answer:
```

3.4 Model Selection

All experiments were conducted with the Zephyr 7B beta model [TBL+23]. It is based on the Mistral model [JSM+23] which was pre-trained on a large dataset of text. And, it is dialogue-tuned using supervised fine-tuning and Direct Preference Optimization (DPO) on the UltraChat and UltraFeedback datasets [CYD+23]. This model is one of the high-quality dialogue-tuned LLMs with a size of 7B according to the leaderboard presented by [ZCS+23].

3.5 Training

We train all models using supervised fine-tuning (SFT). During SFT, the likelihood of the correct answer is maximized. Every question-answer pair is interpreted as a sequence w, where w is the result of inserting the question-answer pair into the format described in subsection 3.3. The model is trained to minimize the language model loss

$$\mathscr{L}_{\text{SFT}} = \sum_i q(w_i \mid w_1 ... w_{i-1}). \tag{1}$$

3.6 Performance Optimization

Low-Rank Adaptation (LoRA) LoRA adapters are a fine-tuning technique where updates to the query and value projection matrices in the transformer decoder are approximated using low-rank matrices [HSW+21].

Quantization To ensure efficient utilization of computational resources, the base weights are stored in a quantized format. We either use 8bit or 16bit quantization.

4 Evaluation

4.1 Metrics

To evaluate the correctness of our model we calculate the accuracy of the answers given the ground truth. Formally, the accuracy is calculated as

$$\text{accuracy}(Y_{\text{pred}}, Y_{\text{true}}) = \frac{1}{n}\sum_{i}^{n} \text{isclose}(x_{\text{pred},i}, x_{\text{true},i}),$$

where X_{pred} is the list of predicted results and X_{true} is the list of ground-truth results. The function `isclose` denotes equality with certain tolerance threshold. Two numbers a and b are considered equal, if any of the following conditions hold true:

- Relative Tolerance: $|a-b| < 0.1 * \min(|a|, |b|)$
- Absolute Tolerance: $|a-b| < 0.1$

4.2 Experimental Setup

To implement the provided methods, Huggingface Transformers [WDS+20] and PyTorch [PGC+17] were used. Specifically, to implement the logits processor, a custom `LogitsProcessor` was defined. To evaluate context-free-grammars, `lark` was used as a parser.

The models were trained for 5 epochs, with an effective batch size of 32, a learning rate of `5e-5`, and the AdamW optimizer [LH19]. LoRA adapters were created with $r = 16$, $\alpha = 32$, a dropout rate of 0.05 for the query and key projection layers.

All experiments were conducted on an NVIDIA V100 GPU with 32GB of VRAM.

4.3 Results

Table 2 shows the results of various configurations on the original FuncQA test set. It can be seen that the original ToolDec method is vastly outperformed by our results. However, a key difference is that we use Zephyr 7B as the base model which is known to vastly outperform the LLaMA 1 30B model [TLI+23]. Generally, it can be seen that applying syntax constraints significantly improves the performance of the model on the given task, while additional supervised fine-tuning improves the performance even further.

In Table 3, we show various hyperparameter configurations for the Zephyr 7B model. It can be seen that 16bit quantization is vastly superior to 4bit quantization. While we could not evaluate our models with `bfloat16`, this could further improve the model quality as that is the setup the models were trained with originally. In addition to that, it can be seen that training the model on the FuncQA training set improves the model quality, while however including the GSM8K training set into the training set

Model Name	Accuracy [%]
Zephyr 7B	10.29
Zephyr 7B + CFG	13.23
Zephyr 7B + CFG + SFT	**14.7**
ToolDec (Llama 30B)	*13.2
ChatGPT (0-shot)	*9.0

Table 2: Results on the original FuncQA test-set for comparison with previous work. Results marked with an asterisk are extracted from [ZCLW23] and [HLWH24] respectively. CFG marks runs that employ our presented method of syntax-error-free decoding. SFT marks runs that employ models fine-tuned on the FuncQA train set.

Generation Mode	Quantization	Trained	Includes GSM8K	Accuracy [%]
Unconstrained	16bit	no	-	28.57
Constrained	16bit			33.08
	4bit			27.07
	16bit	yes	no	**36.84**
	4bit		no	27.82
	16bit		yes	27.07

Table 3: Results on the FuncQA test set proposed by us. A fine-grained breakdown into multiple hyperparameter configurations is provided. All results are based on the Zephyr 7B model.

yields a degradation. This can be caused by the aforementioned data quality issues of the GSM8K dataset or by the fact that the dataset uses simpler questions and fewer toolcalls, i.e. having a different distribution than the FuncQA test set.

To further assess this question, Table 4 shows a breakdown of the results on the test set by single- and multi-hop questions. It can be seen that the accuracy solely degrades on the single-hop questions, suggesting that the distribution shift is indeed a problem.

5 Conclusion

In this study, we explored the enhancement of Large Language Models' (LLMs) tool-calling capabilities through the implementation of syntactic constraints and supervised fine-tuning. The research focused on mitigating the prevalent issue of generating syntactically incorrect tool calls—a challenge that hinders the full utilization of LLMs in executing complex, multi-step computations and accessing external APIs effectively.

By integrating finite-state automata (FSA) for syntax constraint and deploying a strategic training regime that includes both the FuncQA and GSM8k datasets, we

Model	Accuracy [%]		
	Single-Hop	Multi-Hop	Average
Zephyr + SFT	60.00	14.70	36.84
+ GSM8K	40.00	14.70	27.07

Table 4: Breakdown of the accuracy of the model on the FuncQA test set by single- and multi-hop. It uses the Zephyr 7B model trained either solely on the FuncQA or also on the GSM8K dataset.

demonstrated significant improvements in the accuracy and reliability of tool calls made by the models. Our methodology, which distinguishes between ”thinking” and ”tool-calling” phases, allows the model to maintain its generative flexibility while ensuring syntactical accuracy during tool interactions. This dual-phase approach ensures that LLMs can leverage external computational resources efficiently, marking a substantial step forward in enhancing their applicability in real-world tasks.

Our findings, detailed through comprehensive experimentation, underscore the effectiveness of syntax-constrained decoding in boosting tool call accuracy. However, the study also highlights the challenges of data availability, due to the small size of the FuncQA dataset, as well as data quality, particularly when expanding training sets to include GSM8k. The nuanced impact on single-hop and multi-hop question accuracy points to the need for careful dataset selection and preprocessing to align with the models' training objectives.

In conclusion, this research not only advances our understanding of enhancing LLMs' interaction with external tools but also sets the groundwork for future investigations into optimizing these models for a broader range of applications. The successful integration of syntactic constraints opens new avenues for developing more intelligent, versatile LLMs capable of performing complex tasks with greater autonomy and precision. As we continue to explore this promising field, the potential for LLMs to revolutionize various sectors—from academia to industry—becomes increasingly tangible.

References

[BFH^{+}23] Beeching, Edward ; Fourrier, Clémentine ; Habib, Nathan ; Han, Sheon ; Lambert, Nathan ; Rajani, Nazneen ; Sanseviero, Omar ; Tunstall, Lewis ; Wolf, Thomas: *Open LLM Leaderboard*. `https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard`, 2023

[BMR^{+}20] Brown, Tom B. ; Mann, Benjamin ; Ryder, Nick ; Subbiah, Melanie ; Kaplan, Jared ; Dhariwal, Prafulla ; Neelakantan, Arvind ; Shyam, Pranav ; Sastry, Girish ; Askell, Amanda ; Agarwal, Sandhini ; Herbert-Voss, Ariel ; Krueger, Gretchen ; Henighan, Tom ; Child, Rewon ; Ramesh, Aditya ; Ziegler, Daniel M. ; Wu, Jeffrey ; Winter, Clemens ; Hesse, Christopher ; Chen, Mark ; Sigler, Eric ; Litwin, Mateusz ; Gray, Scott ; Chess, Benjamin ; Clark, Jack ; Berner, Christopher ; McCandlish, Sam ; Radford, Alec ; Sutskever, Ilya ; Amodei, Dario: *Language Models are Few-Shot Learners*. 2020

[Cho56] Chomsky, N.: Three models for the description of language. In: *IRE Transactions on Information Theory* 2 (1956), Nr. 3, S. 113–124. `http://dx.doi.org/10.1109/TIT.1956.1056813`. – DOI 10.1109/TIT.1956.1056813

[CND^{+}22] Chowdhery, Aakanksha ; Narang, Sharan ; Devlin, Jacob ; Bosma, Maarten ; Mishra, Gaurav ; Roberts, Adam ; Barham, Paul ; Chung, Hyung W. ; Sutton, Charles ; Gehrmann, Sebastian ; Schuh, Parker ; Shi, Kensen ; Tsvyashchenko, Sasha ; Maynez, Joshua ; Rao, Abhishek ; Barnes, Parker ; Tay, Yi ; Shazeer, Noam ; Prabhakaran, Vinodkumar ; Reif, Emily ; Du, Nan ; Hutchinson, Ben ; Pope, Reiner ; Bradbury, James ; Austin, Jacob ; Isard, Michael ; Gur-Ari, Guy ; Yin, Pengcheng ; Duke, Toju ; Levskaya, Anselm ; Ghemawat, Sanjay ; Dev, Sunipa ; Michalewski, Henryk ; Garcia, Xavier ; Misra, Vedant ; Robinson, Kevin ; Fedus, Liam ; Zhou, Denny ; Ippolito, Daphne ; Luan, David ; Lim, Hyeontaek ; Zoph, Barret ; Spiridonov, Alexander ; Sepassi, Ryan ; Dohan, David ; Agrawal, Shivani ; Omernick, Mark ; Dai, Andrew M. ; Pillai, Thanumalayan S. ; Pellat, Marie ; Lewkowycz, Aitor ; Moreira, Erica ; Child, Rewon ; Polozov, Oleksandr ; Lee, Katherine ; Zhou, Zongwei ; Wang, Xuezhi ; Saeta, Brennan ; Diaz, Mark ; Firat, Orhan ; Catasta, Michele ; Wei, Jason ; Meier-Hellstern, Kathy ; Eck, Douglas ; Dean, Jeff ; Petrov, Slav ; Fiedel, Noah: *PaLM: Scaling Language Modeling with Pathways*. 2022

[CYD^{+}23] Cui, Ganqu ; Yuan, Lifan ; Ding, Ning ; Yao, Guanming ; Zhu, Wei ; Ni, Yuan ; Xie, Guotong ; Liu, Zhiyuan ; Sun, Maosong: *UltraFeedback:*

Boosting Language Models with High-quality Feedback. `http://arxiv.org/pdf/2310.01377.pdf`. Version: 2023

[HLWH24] Hao, Shibo ; Liu, Tianyang ; Wang, Zhen ; Hu, Zhiting: *ToolkenGPT: Augmenting Frozen Language Models with Massive Tools via Tool Embeddings.* 2024

[HSW+21] Hu, Edward J. ; Shen, Yelong ; Wallis, Phillip ; Allen-Zhu, Zeyuan ; Li, Yuanzhi ; Wang, Shean ; Wang, Lu ; Chen, Weizhu: Lora: Low-rank adaptation of large language models. In: *arXiv preprint arXiv:2106.09685* (2021)

[JSM+23] Jiang, Albert Q. ; Sablayrolles, Alexandre ; Mensch, Arthur ; Bamford, Chris ; Chaplot, Devendra S. ; Casas, Diego de l. ; Bressand, Florian ; Lengyel, Gianna ; Lample, Guillaume ; Saulnier, Lucile ; Lavaud, Lélio R. ; Lachaux, Marie-Anne ; Stock, Pierre ; Scao, Teven L. ; Lavril, Thibaut ; Wang, Thomas ; Lacroix, Timothée ; Sayed, William E.: *Mistral 7B.* 2023

[KHM+23] Kaddour, Jean ; Harris, Joshua ; Mozes, Maximilian ; Bradley, Herbie ; Raileanu, Roberta ; McHardy, Robert: *Challenges and Applications of Large Language Models.* 2023

[LH19] Loshchilov, Ilya ; Hutter, Frank: *Decoupled Weight Decay Regularization.* 2019

[Ope24] OpenAI: *ChatGPT - OpenAI.* `https://openai.com/chatgpt`, 2024. – Last accessed on 2024-02-29

[PGC+17] Paszke, Adam ; Gross, Sam ; Chintala, Soumith ; Chanan, Gregory ; Yang, Edward ; DeVito, Zachary ; Lin, Zeming ; Desmaison, Alban ; Antiga, Luca ; Lerer, Adam: Automatic differentiation in PyTorch. (2017)

[RSM+23] Rafailov, Rafael ; Sharma, Archit ; Mitchell, Eric ; Ermon, Stefano ; Manning, Christopher D. ; Finn, Chelsea: *Direct Preference Optimization: Your Language Model is Secretly a Reward Model.* 2023

[SDYD+23] Schick, Timo ; Dwivedi-Yu, Jane ; Dessì, Roberto ; Raileanu, Roberta ; Lomeli, Maria ; Zettlemoyer, Luke ; Cancedda, Nicola ; Scialom, Thomas: *Toolformer: Language Models Can Teach Themselves to Use Tools.* 2023

[TBL+23] Tunstall, Lewis ; Beeching, Edward ; Lambert, Nathan ; Rajani, Nazneen ; Rasul, Kashif ; Belkada, Younes ; Huang, Shengyi ; Werra, Leandro von ; Fourrier, Clémentine ; Habib, Nathan ; Sarrazin, Nathan ; Sanseviero, Omar ; Rush, Alexander M. ; Wolf, Thomas: *Zephyr: Direct Distillation of LM Alignment.* `http://arxiv.org/pdf/2310.16944.pdf`. Version: 2023

[TLI+23] Touvron, Hugo ; Lavril, Thibaut ; Izacard, Gautier ; Martinet, Xavier ; Lachaux, Marie-Anne ; Lacroix, Timothée ; Rozière, Baptiste ; Goyal, Naman ; Hambro, Eric ; Azhar, Faisal ; Rodriguez, Aurelien ; Joulin, Armand ; Grave, Edouard ; Lample, Guillaume: *LLaMA: Open and Efficient Foundation Language Models.* 2023

[TMS+23] Touvron, Hugo ; Martin, Louis ; Stone, Kevin ; Albert, Peter ; Almahairi, Amjad ; Babaei, Yasmine ; Bashlykov, Nikolay ; Batra, Soumya ; Bhargava, Prajjwal ; Bhosale, Shruti u. a.: Llama 2: Open foundation and fine-tuned chat models. In: *arXiv preprint arXiv:2307.09288* (2023)

[WDS+20] Wolf, Thomas ; Debut, Lysandre ; Sanh, Victor ; Chaumond, Julien ; Delangue, Clement ; Moi, Anthony ; Cistac, Pierric ; Rault, Tim ; Louf, Rémi ; Funtowicz, Morgan ; Davison, Joe ; Shleifer, Sam ; Platen, Patrick von ; Ma, Clara ; Jernite, Yacine ; Plu, Julien ; Xu, Canwen ; Scao, Teven L. ; Gugger, Sylvain ; Drame, Mariama ; Lhoest, Quentin ; Rush, Alexander M.: Transformers: State-of-the-Art Natural Language Processing. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations.* Online : Association for Computational Linguistics, Oktober 2020, 38–45

[WWS+23] Wei, Jason ; Wang, Xuezhi ; Schuurmans, Dale ; Bosma, Maarten ; Ichter, Brian ; Xia, Fei ; Chi, Ed ; Le, Quoc ; Zhou, Denny: *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models.* 2023

[XCG+23] Xi, Zhiheng ; Chen, Wenxiang ; Guo, Xin ; He, Wei ; Ding, Yiwen ; Hong, Boyang ; Zhang, Ming ; Wang, Junzhe ; Jin, Senjie ; Zhou, Enyu ; Zheng, Rui ; Fan, Xiaoran ; Wang, Xiao ; Xiong, Limao ; Zhou, Yuhao ; Wang, Weiran ; Jiang, Changhao ; Zou, Yicheng ; Liu, Xiangyang ; Yin, Zhangyue ; Dou, Shihan ; Weng, Rongxiang ; Cheng, Wensen ; Zhang, Qi ; Qin, Wenjuan ; Zheng, Yongyan ; Qiu, Xipeng ; Huang, Xuanjing ; Gui, Tao: *The Rise and Potential of Large Language Model Based Agents: A Survey.* 2023

[YZY+23] Yao, Shunyu ; Zhao, Jeffrey ; Yu, Dian ; Du, Nan ; Shafran, Izhak ; Narasimhan, Karthik ; Cao, Yuan: *ReAct: Synergizing Reasoning and Acting in Language Models.* 2023

[ZCLW23] Zhang, Kexun ; Chen, Hongqiao ; Li, Lei ; Wang, William: *Syntax Error-Free and Generalizable Tool Use for LLMs via Finite-State Decoding.* 2023

[ZCS^{+}23] Zheng, Lianmin ; Chiang, Wei-Lin ; Sheng, Ying ; Zhuang, Siyuan ; Wu, Zhanghao ; Zhuang, Yonghao ; Lin, Zi ; Li, Zhuohan ; Li, Dacheng ; Xing, Eric P. ; Zhang, Hao ; Gonzalez, Joseph E. ; Stoica, Ion: *Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena.* `http://arxiv.org/pdf/2306.05685.pdf`. Version: 2023