Predictive Prompt Analysis

Jae Yong Lee
KAIST
Daejeon, South Korea
jaeyonglee0205@kaist.ac.kr

Sungmin Kang
NUS
Singapore
sungmin@nus.edu.sg

Shin Yoo
KAIST
Daejeon, South Korea
shin.yoo@kaist.ac.kr

Abstract

Large Language Models (LLMs) are machine learning models that have seen widespread adoption due to their capability of handling previously difficult tasks. LLMs, due to their training, are sensitive to how exactly a question is presented, also known as prompting. However, prompting well is challenging, as it has been difficult to uncover principles behind prompting — generally, trial-and-error is the most common way of improving prompts, despite its significant computational cost. In this context, we argue it would be useful to perform 'predictive prompt analysis', in which an automated technique would perform a quick analysis of a prompt and predict how the LLM would react to it, relative to a goal provided by the user. As a demonstration of the concept, we present Syntactic Prevalence Analyzer (SPA), a predictive prompt analysis approach based on sparse autoencoders (SAEs). SPA accurately predicted how often an LLM would generate target syntactic structures during code synthesis, with up to 0.994 Pearson correlation between the predicted and actual prevalence of the target structure. At the same time, SPA requires only 0.4% of the time it takes to run the LLM on a benchmark. As LLMs are increasingly used during and integrated into modern software development, our proposed predictive prompt analysis concept has the potential to significantly ease the use of LLMs for both practitioners and researchers.

CCS Concepts

• **Software and its engineering** → **Software creation and management**; • **Computing methodologies** → *Machine learning*.

Keywords

Prompt Engineering, Large Language Models, Sparse Autoencoders

1 Introduction

Large Language Models (LLMs) are statistical models that predict the likelihood of the next token given preceding context, which have a large number of parameters and are trained on large corpus. An interesting characteristic of these models is that they show emergent task-solving capabilities when scaled [18], which has led to their widespread use in software engineering tasks [8]. In the most common use case of LLMs, one will describe the task in natural language, which is known as prompting the LLM.

Early experiments on LLMs demonstrated that the way one prompts an LLM has a significant influence on performance [11].

FSE Companion '25, Trondheim, Norway

ACM ISBN 979-8-4007-1276-0/25/06
https://doi.org/10.1145/3696630.3728516

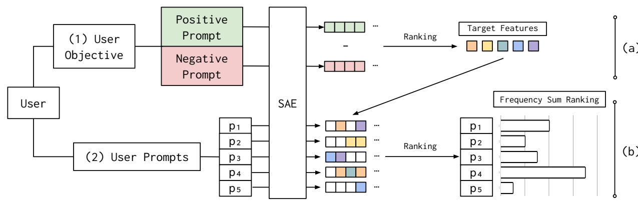

Figure 1: An Overview of SPA, a Prototype Implementing Predictive Prompt Analysis. Upper Half (a): Extracting Target Features. Lower Half (b): Calculating the Ranking.

However, it is often difficult to know which prompts will perform well in practice. As a result, prompt construction often involves significant trial-and-error [4] or prompt optimization based on ground-truth answers [14], both of which require a substantial level of human intervention and computational resources.

In response, we argue for *predictive prompt analysis* — quickly predicting the effect of a prompt before running it on a benchmark, without the need for user-side training or significant user input. By accurately forecasting the effect of prompts in a computationally inexpensive manner, developers could quickly and cheaply iterate on prompt designs that meet their goals. Such analyses would also grow in importance as state-of-the-art LLMs continue to scale, making their computational costs increasingly burdensome.

As a demonstration that making such a prediction cheaply and without user-side training is possible, we showcase Syntactic Prevalence Analyzer (SPA), a preliminary predictive prompt analysis tool. We design SPA around a simple scenario: the user seeks to have the LLM generate a target syntactic structure, such as a try-except clause, during code synthesis, but there are multiple ways to 'ask' for such a structure; how effective would each prompt be in generating the target structure? SPA predicts an answer to this question, orders of magnitude more quickly than running the prompt on a code synthesis benchmark. SPA uses Sparse Autoencoders (SAEs), which are models that 'cluster' the internal activation patterns of LLMs; specific clusters or 'features' of SAEs can often be mapped to recognizable concepts [2]. SPA first identifies the SAE features related to the user request, then based on these identified features predicts the relative incidence of the syntactic structures, ultimately allowing SPA to predict how well each prompt would meet the goal. Experiments demonstrate that SPA shows strong predictive performance — the actual incidence of the target syntactic structure for each prompt closely followed the predictions of SPA, with a Pearson correlation value of up to 0.994; meanwhile, the time it took to run the predictive analysis was only 0.4% of the total time to run the LLM on the full benchmark, and 18.7% of a baseline that generated 10 code samples per instruction, demonstrating the significant computational efficiency of our approach. Despite our strong early findings, there are two important limitations to SPA. First, in this preliminary work, the behavior we predict, syntactic structure prevalence, is fairly artificial. Second, our approach currently only works on open-weight LLMs for which there is a trained SAE. Nonetheless, we have optimistic initial results suggesting that improved predictive prompt analysis tools could overcome these limitations of SPA; such tools would ease rapid and computationally efficient prompt engineering. In summary, we (i) propose the concept of predictive prompt analysis; (ii) describe the prototype tool SPA implementing it; (iii) provide empirical results demonstrating the strong performance of SPA; and (iv) describe the future directions that are promising.

2 Related Work

Prompt Engineering. Designing and refining how to ask the LLM to do the bidding of the user, without modifying any internal parameters, is known as prompt engineering. As LLMs are computationally expensive, the general practice is to optimize the prompt to achieve the most from a single LLM query [15]. Related techniques include Chain-of-Thought Prompting [19], few-shot prompting [3], and Promptbreeder [5]. However, despite extensive research efforts, finding good prompts is mostly done via trial and error, due to the absence of explicit design principles that would help users construct an effective prompt. Therefore, the search for effective prompts is both challenging and expensive, often necessitating multiple LLM inference runs which are becoming more expensive as the models continue to scale [20]. Predictive prompt analysis, by reducing the computational time and developer waits currently involved in prompt optimization, thus has the potential to significantly ease the prompt engineering process.

Sparse Autoencoders. One promising attempt at interpreting LLMs is the Sparse Autoencoder (SAE). SAEs decompose or 'cluster' the internal activation of LLMs into *features*, which tend to be easier to interpret than the neuron activation patterns [2]. In particular, SAEs take as input the residual stream values of an LLM at a particular token, and generate a sparse encoding where only a few (sparse) features are activated. Prior work shows that these SAE features can often be mapped to identifiable concepts, and thus be used for interpreting LLM behavior [2]. While research regarding SAEs is active [2, 7, 9, 17], they have not yet been used to analyze prompts as we propose. As prompts are the usual way developers interact with LLMs, using SAEs for predictive prompt analysis offers a unique perspective to support them. While predictive prompt analysis is not restricted to the use of SAEs, we use them in SPA as they capture LLM activation patterns without extra user training, and their features may be predictive of LLM behavior in turn.

3 Methodology

As described earlier, consider the scenario where a user wants a syntactic structure to be generated during LLM code synthesis, such that exception handlers are generated to prevent potential errors. The user is struggling between the options in Table 1, as it is unclear *a priori* how consistently each instruction will lead to the LLM generating exception handling. Previously, the only way of answering this was to generate code based on these instructions hundreds or thousands of times. For example, even OpenAI suggests generating

ID	Instructions
1	None
2	It might be helpful to add an exception handler.
3	Write an exception handler.
4	You need to write an exception handler.
5	Please, with all my heart, include an exception handler.

Table 1: Exception Handler Instructions

thousands of results to decide which prompt is better [12], which is both computationally and financially expensive.

To mitigate this substantial cost, we propose SPA, a predictive prompt analysis technique which will predict which instruction is best. SPA takes two inputs: (1) the description of a target syntactic structure in natural language (e.g. "a try-except clause") and (2) multiple prompts to compare, towards the goal of predicting which prompt will most consistently generate the target structure (shown in Tab. 1). With these inputs, SPA goes through two phases: (a) *Extracting the Target Features* and (b) *Calculating the Ranking*, as illustrated in Fig. 1, each described in detail next.

3.1 Extracting Target Features

```
User: Write me an example of <<OBJECTIVE>>.
AI: Sure. Here is an example of <<OBJECTIVE>>.
```
def example():
        return
```
```

Figure 2: Template of positive prompt

To precisely extract the relevant SAE features, we employ 'positive' and 'negative' prompts. Figure 2 shows the template of a positive prompt, while removing "of «OBJECTIVE»" would make a negative prompt. Both prompts are needed to distinguish relevant SAE features related to the provided objective from generic features. For instance, on the prompt in Figure 2, generic SAE features that activate on "python code" or the word "write" are activated along with task-relevant features, limiting the effectiveness of the extraction process. Hence, a negative prompt, which has the same structure without mention of the objective, helps excluding these generic features. We extract features that activate from the example code, highlighted in yellow, to find features that influence the LLM in response to the instruction.

Formally, we rank features by the difference in activation strength between prompts, d^{τ}:

$$d^{\tau} = \sum_{x \in T^{\tau}_{\text{pos}}} A^{\tau}_{x} - \sum_{y \in T^{\tau}_{\text{neg}}} A^{\tau}_{y} \tag{1}$$

where T^{τ}_{pos} and T^{τ}_{neg} are the sets of activated tokens on a feature τ for positive and negative prompts, respectively, and A^{τ}_{x} is the activation value of feature τ on the token x. We employ the ranking process on the basis that features with highly differing activation values are the most relevant with the provided context [17]; Ranking by d^τ, SPA selects the top k features to acquire an SAE feature set TF, which are ideally related to the target syntactic structure.

3.2 Calculating the Ranking

Once relevant features have been extracted, they are used to predict the effectiveness of the instructions. For each instruction, a set of prompts S is formed for code synthesis, in which the instruction is combined with randomly sampled problems from a code synthesis dataset. While SPA could operate based on a single example code synthesis problem in principle, we aggregate the results of analysis over multiple problems as in Eq. 2 to reduce noise.

The activation patterns of the LLM over prompts S are then processed by the SAE, yielding the final prediction. First, we define the normalized activation frequency as $f^\tau = \frac{t_{\text{activated}}}{t_{\text{total}}}$, where $t_{\text{activated}}$ is the number of tokens for which the target feature τ is activated and t_{total} is the total number of tokens. Using this, the prediction P for each instruction is defined as:

$$P = \sum_{\tau \in TF} \sum_{p \in S} f_p^\tau \tag{2}$$

where TF is the target feature set from Section 3.1, S is the sample set of prompts, and f_p is the normalized frequencies calculated from the encoded prompt p. Ultimately, instructions are ranked in descending order of P and presented to the user, as we expect that instructions causing the relevant features to activate will more likely generate the target syntactic structures.

4 Experimental Design

4.1 Research Questions

This study answers the following research questions:

- *RQ1 (Efficacy)*: How effective is SPA in predicting the effect of prompts?
- *RQ2 (Efficiency)*: What is computational overhead of SPA in predicting the effect of prompts?

4.2 Experimental Setup

Experiments were conducted using AMD EPYC 9124 16-Core Processor X86_64 CPUs, and four NVIDIA GeForce RTX 4090 GPUs. We used the Google Gemma-2-2b-it [16] model with Gemma-2-2b SAE [10] that has 16384 trained features. SAEs are trained on specific layers of LLMs to analyze activation patterns at that layer; we used SAEs trained on layer 1 (first layer), 9, 16, and 25 (last layer) to further analyze the impact of layer depth on the prediction performance of our technique.

Three syntactic objectives are used: generating try-except clauses, comments, and print statements. For each objective, both instruction sets and fewshot prompts were manually curated through author consensus, simulating a prompt engineering process of involving instructions with varying tones. We experiment with six prompt sets, also listed in Tab. 2. *Exception* and *Print* are composed of five instructions varying degrees of authoritative tone (see Fig. 1), while *Comment* is composed of eight instructions with three designed to suppress comment generation as the LLM tended

Table 2: Average Correlations Across Different Scenarios and Layers Compared to the Sampled Inference

Scenario	Sampled Inference	Layer #			
		1	9	16	25
Exception	0.743	0.972	0.633	0.978	0.790
Exception Fewshot	0.744	0.956	0.952	0.959	0.954
Comment	0.743	0.687	0.887	0.751	0.860
Comment Fewshot	0.740	0.941	0.937	0.908	0.926
Print	0.740	0.960	0.971	**0.994**	0.913
Print Fewshot	0.725	0.953	0.957	0.923	0.957
Average	0.740	0.911	0.890	**0.918**	0.900

to produce comments even when given no instructions. For non-fewshot scenarios, an empty prompt was inserted to observe the inherent tendency of LLM generations. *Exception Fewshot* range from 0-shot to 4-shot, whereas *Comment Fewshot* and *Print Fewshot* are composed up to 3-shot due to memory constraints of our environment. The instructions for *Exception* can be found in Tab 1.

As a benchmark to evaluate our technique we used the sanitized version of Mostly Basic Python Problems (MBPP) which has 427 natural language specifications and corresponding code, a manually verified set of crowd-sourced Python programming problems [1]. For SPA, we sampled 10 random problems ($|S| = 10$) from MBPP and merged them with each instruction. As mentioned in Section 3.1, five target features were extracted from the prompts ($|TF| = 5$). Pearson correlation for SPA's predictions was computed against LLM inference outputs on the entire MBPP dataset, averaged over three runs, with the total number of times the target syntactic structure was generated overall tallied. A strong correlation would suggest that SPA effectively captures the causal dependencies between prompt and LLM behavior, thus validating its reliability as a quantitative metric for the task in question. We also compare against a partial inference baseline that involves sampling 10 inference outputs from the same 10 problems used by SPA and counting target syntactic structures. For both the partial inference and SPA predictions, the average was calculated on Fisher Z-transformed correlations over five attempts [6].

5 Results

5.1 RQ1: Effectiveness of *SPA*

In Table 2, we observe that the partial inference baseline we compare against achieves an average correlation of 0.740. In contrast, over all layers, SPA has a better average correlation with the occurrence of the target syntactic structure. Among the layers, layer 16 shows the most best correlation across all scenarios, also with the highest correlation: 0.994 for *Print*. Based on these results, it appears to be more effective to utilize SAEs trained on layers positioned approximately $\frac{2}{3}$ of the way (layer 16) into the network, similar to previous work on SAEs that find layers between $\frac{1}{2}$ and $\frac{5}{6}$ of the way to be the most interpretable [7, 17].

Table 3: Average Computation Time in Seconds

Scenario	Total Inference	Sampled Inference	SPA		
			Ext.	Pred.	Total
Exception	9428	195	0.212	38.1	**38.3**
Exception Fewshot	9409	239	0.218	44.6	**44.8**
Comment	14022	308	0.212	55.1	**55.3**
Comment Fewshot	9102	224	0.221	36.4	**36.6**
Print	8850	175	0.213	40.9	**41.1**
Print Fewshot	8541	209	0.214	35.3	**35.5**
Average	9892	225	0.215	41.7	**42.0**

Answer to RQ1: SPA achieves the highest average correlation of 0.918 using SAE trained with layer 16, outperforming the average correlation of the sampled inference output, 0.740.

5.2 RQ2: Efficiency of *SPA*

Table 3 shows the time taken for prediction, in seconds, for different approaches of evaluating prompts. Notably, while the sampled inference requires an average of 225 seconds, SPA required only 42.0 seconds, a decrease of 81.3% despite showing better predictive performance, underscoring its efficiency and demonstrating the potential of predictive prompt analysis in reducing both the computational and financial overhead in prompt engineering. Furthermore, the average total inference time for 427 data points in MBPP was 9892 seconds, more than 235 times longer than SPA. In terms of the time cost of each phase of SPA, feature extraction took 0.215 seconds on average, while generating predictions for all instructions took 41.7 seconds on average.

Answer to RQ2: SPA is significantly more efficient than the sampled inference and total inference, showing a decrease of 81.3% and 99.6% of computation time, respectively.

6 Qualitative Analysis

In this section, we analyze the extracted features to investigate the effectiveness of the target feature extraction process described in Section 3.1. As SPA does not internally assess the quality of features other than their activation values, manual inspection of the features is helpful in understanding the process. For analysis we use Neuronpedia, a platform that allows visualization of SAE data [1], to find the maximum activating examples from training data of Gemma-2 composed of web documents, code, and math problems, on our extracted target features.

Inspecting the results from layer 16 and from the 'Exception' scenario, for which the correlation is high, SPA identified SAE feature #2423 as relevant, activates on the token "throw" in snippets such as `error.response = response; \n throw`. This feature is clearly related with exception-related behavior, suggesting that it accurately captures specific characteristics of the objective. Similarly, for the `python print statement` objective, SPA identified SAE feature #4961, which activates on snippets like `System.out.println`. This inspection shows that SPA can identify target-relevant features, which in turn help predict the prevalence of syntactic structures.

[1]https://www.neuronpedia.org/

7 Limitations and Future Directions

SPA currently relies on open-weight LLMs, such as Gemma, which allow the extraction of internal activations to train and use SAEs. However, state-of-the-art LLMs such as GPT and Claude do not provide such access, restricting the scope of SPA. To surmount this, we hope to experiment with the transferability of predictive prompt analysis from one LLM to another, similarly to the adversarial example transfer in image classifiers [13]. Our preliminary experiments on Gemma-2b and Gemma-2-2b have shown promise, raising the possibility that we could perform predictive analysis on open-weight models and apply the results to closed-weight models.

Another limitation is that SPA focuses on predicting the occurrence of syntactic structures, while developers may be more interested in comparatively abstract or semantic properties, such as general syntactic well-formedness or correctness. Thus, predictive prompt analysis tools that work on more semantic behavior would ideally be devised. Early results on try-except clause prediction experiment indicate features related to the abstract concept of 'trying again', which would cause the LLM to suggest multiple solutions instead of a single solution. Being able to identify such features would improve the utility of predictive prompt analysis by granting greater control over the prompt construction process.

8 Conclusion

This work introduces the concept of predictive prompt analysis, in which a quick analysis is performed to predict how an LLM will behave in response to a prompt. Computationally efficient predictive prompt analysis techniques would ease development of LLM-based applications by accelerating the trial-and-error process of prompt engineering. Our prototype tool, SPA, predicts prompt effect on generating syntactic structures by first identifying relevant features from SAEs, then using those features to analyze how the LLM will respond to prompts — all without the need for training. SPA shows strong predictive performance while being computationally efficient: initial results across six selected scenarios show that the predictions made by SPA of the relative prevalence of target syntactic structures achieve the highest average correlation of 0.918 with actual inference outputs, with up to a 99.6% reduction in computation time. Despite its current limitations, SPA shows the feasibility of predictive prompt analysis and its potential to ease development of LLM-based software.

Data Availability. The code and data used in the paper is available from this link.

Acknowledgements

Jae Yong Lee and Shin Yoo are supported by the National Research Foundation of Korea (NRF) funded by the Korean Government MSIT (RS-2023-00208998), the Engineering Research Center Program funded by the Korean Government MSIT (RS-2021-NR060080), and the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (RS-2022-II220995).

References

[1] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. 2021. Program Synthesis with Large Language Models. arXiv:2108.07732 [cs.PL] https://arxiv.org/abs/2108.07732

[2] Trenton Bricken, Adly Templeton, Joshua Batson, Brian Chen, Adam Jermyn, Tom Conerly, Nick Turner, Cem Anil, Carson Denison, Amanda Askell, Robert Lasenby, Yifan Wu, Shauna Kravec, Nicholas Schiefer, Tim Maxwell, Nicholas Joseph, Zac Hatfield-Dodds, Alex Tamkin, Karina Nguyen, Brayden McLean, Josiah E Burke, Tristan Hume, Shan Carter, Tom Henighan, and Christopher Olah. 2023. Towards Monosemanticity: Decomposing Language Models With Dictionary Learning. *Transformer Circuits Thread* (2023). https://transformer-circuits.pub/2023/monosemantic-features/index.html.

[3] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, and Prafulla et al. Dhariwal. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[4] Hai Dang, Lukas Mecke, Florian Lehmann, Sven Goller, and Daniel Buschek. 2022. How to Prompt? Opportunities and Challenges of Zero- and Few-Shot Learning for Human-AI Interaction in Creative Applications of Generative Models. arXiv:2209.01390 [cs.HC] https://arxiv.org/abs/2209.01390

[5] Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. 2025. Promptbreeder: self-referential self-improvement via prompt evolution. In *Proceedings of the 41st International Conference on Machine Learning* (Vienna, Austria) *(ICML'24)*. JMLR.org, Article 541, 64 pages.

[6] R. A. Fisher. 1915. Frequency Distribution of the Values of the Correlation Coefficient in Samples from an Indefinitely Large Population. *Biometrika* 10, 4 (1915), 507–521. http://www.jstor.org/stable/2331838

[7] Leo Gao, Tom Dupré la Tour, Henk Tillman, Gabriel Goh, Rajan Troll, Alec Radford, Ilya Sutskever, Jan Leike, and Jeffrey Wu. 2024. Scaling and evaluating sparse autoencoders. arXiv:2406.04093 [cs.LG] https://arxiv.org/abs/2406.04093

[8] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. 2024. Large Language Models for Software Engineering: A Systematic Literature Review. *ACM Trans. Softw. Eng. Methodol.* 33, 8, Article 220 (Dec. 2024), 79 pages. doi:10.1145/3695988

[9] Robert Huben, Hoagy Cunningham, Logan Riggs Smith, Aidan Ewart, and Lee Sharkey. 2024. Sparse Autoencoders Find Highly Interpretable Features in Language Models. In *The Twelfth International Conference on Learning Representations*. https://openreview.net/forum?id=F76bwRSLeK

[10] Tom Lieberum, Senthooran Rajamanoharan, Arthur Conmy, Lewis Smith, Nicolas Sonnerat, Vikrant Varma, Janos Kramar, Anca Dragan, Rohin Shah, and Neel Nanda. 2024. Gemma Scope: Open Sparse Autoencoders Everywhere All At Once on Gemma 2. In *Proceedings of the 7th BlackboxNLP Workshop: Analyzing and Interpreting Neural Networks for NLP*, Yonatan Belinkov, Najoung Kim, Jaap Jumelet, Hosein Mohebbi, Aaron Mueller, and Hanjie Chen (Eds.). Association for Computational Linguistics, Miami, Florida, US, 278–300. doi:10.18653/v1/2024.blackboxnlp-1.19

[11] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2023. Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing. *ACM Comput. Surv.* 55, 9, Article 195 (Jan. 2023), 35 pages. doi:10.1145/3560815

[12] OpenAI. n.d.. Prompt engineering. https://platform.openai.com/docs/guides/prompt-engineering. Accessed: 2025-01-11.

[13] Nicolas Papernot, Patrick McDaniel, and Ian Goodfellow. 2016. Transferability in Machine Learning: from Phenomena to Black-Box Attacks using Adversarial Samples. arXiv:1605.07277 [cs.CR] https://arxiv.org/abs/1605.07277

[14] Reid Pryzant, Dan Iter, Jerry Li, Yin Tat Lee, Chenguang Zhu, and Michael Zeng. 2023. Automatic Prompt Optimization with "Gradient Descent" and Beam Search. arXiv:2305.03495 [cs.CL] https://arxiv.org/abs/2305.03495

[15] Pranab Sahoo, Ayush Kumar Singh, Sriparna Saha, Vinija Jain, Samrat Mondal, and Aman Chadha. 2024. A Systematic Survey of Prompt Engineering in Large Language Models: Techniques and Applications. arXiv:2402.07927 [cs.AI] https://arxiv.org/abs/2402.07927

[16] Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, and et al. 2024. Gemma 2: Improving Open Language Models at a Practical Size. arXiv:2408.00118 [cs.CL] https://arxiv.org/abs/2408.00118

[17] Adly Templeton, Tom Conerly, Jonathan Marcus, Jack Lindsey, Trenton Bricken, Brian Chen, Adam Pearce, Craig Citro, Emmanuel Ameisen, Andy Jones, Hoagy Cunningham, Nicholas L Turner, Callum McDougall, Monte MacDiarmid, C. Daniel Freeman, Theodore R. Sumers, Edward Rees, Joshua Batson, Adam Jermyn, Shan Carter, Chris Olah, and Tom Henighan. 2024. Scaling Monosemanticity: Extracting Interpretable Features from Claude 3 Sonnet. *Transformer Circuits Thread* (2024). https://transformer-circuits.pub/2024/scaling-monosemanticity/index.html

[18] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. 2022. Emergent Abilities of Large Language Models. arXiv:2206.07682 [cs.CL] https://arxiv.org/abs/2206.07682

[19] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2024. Chain-of-thought prompting elicits reasoning in large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems* (New Orleans, LA, USA) *(NIPS '22)*. Curran Associates Inc., Red Hook, NY, USA, Article 1800, 14 pages.

[20] Zixuan Zhou, Xuefei Ning, Ke Hong, Tianyu Fu, Jiaming Xu, Shiyao Li, Yuming Lou, Luning Wang, Zhihang Yuan, Xiuhong Li, Shengen Yan, Guohao Dai, Xiao-Ping Zhang, Yuhan Dong, and Yu Wang. 2024. A Survey on Efficient Inference for Large Language Models. arXiv:2404.14294 [cs.CL] https://arxiv.org/abs/2404.14294