# Small Language Models for Application Interactions: A Case Study

Beibin Li, Yi Zhang, Sébastien Bubeck, Jeevan Pathuri, Ishai Menache

Microsoft, Redmond

May 2024

#### **Abstract**

We study the efficacy of Small Language Models (SLMs) in facilitating application usage through natural language interactions. Our focus here is on a particular internal application used in Microsoft for cloud supply chain fulfilment. Our experiments show that small models can outperform much larger ones in terms of both accuracy and running time, even when fine-tuned on small datasets. Alongside these results, we also highlight SLM-based system design considerations.

## 1 Introduction

Large Language Models (LLMs) are becoming pervasive in assisting humans with a wide variety of tasks, such as writing documents, presenting work, coding and health assistant. Generative LLMs are being rapidly integrated in user-facing software, for answering questions and increasing productivity through simple, language based interactions with technology. One of the key operating principles behind LLMs is exploiting their ability to generalize to unseen tasks by providing examples through the prompt itself – an approach commonly known as in-context learning. While LLMs are being designed to support larger prompt sizes, processing very large prompts might be expensive and incur non-negligible latencies.

In this paper, we consider the alternative of using Small Language Models (SLMs), which are being developed nowadays and open-sourced by several companies. While SLMs may fall behind LLMs for general unseen tasks, we examine using fine-tuned versions thereof to handle a specific and *fixed* set of *tasks*. In particular, we consider using SLMs in order to enable natural-language interactions with a given *application*; an application can be simply viewed as software that includes a set of functions (or APIs) that are invoked by the user to carry out different requirements.

Organizations deploy customized applications in a variety of sectors, including healthcare [1], supply chains [2], and agriculture [3]. Our focus here on a particular application that has been developed in Microsoft for managing the server fulfillment process for Azure's supply chain – namely, the process of sending hardware from warehouses to the data centers. The APIs of this application include adding business and operational constraints for fulfillment, generating a fulfillment plan and extracting details and insights about the plan. We build a language model system on top of this application to enable users to accomplish essential tasks, each of which utilizing one or more APIs. For example, users may interact with our system in plain English to generate a new plan, or answer what-if queries (e.g., "what are the implications of disabling a certain warehouse"?). In this paper, we experiment with different language model technologies that can be used in our system. Our experiments show that SLMs such as Phi-3, Llama 3 and Mistral v0.2 can achieve higher degrees of accuracy than state-of-the-art LLMs, while providing outputs significantly faster. Similar observations regarding the potential benefits of SLMs have been recently reported in [4] for a variety of benchmarks. Importantly, our paper shows that SLMs may reach these superior capabilities with a modest size of training data (see Figure 1).

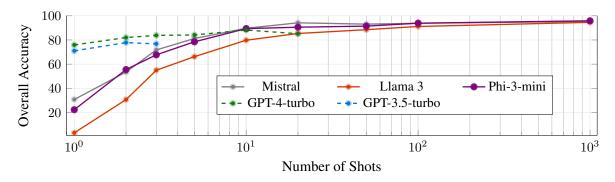


Figure 1: Overall accuracy as a function of the number of the per-task examples. For LLMs, the examples are given in the prompt, and for SLMs they are used for the offline fine tuning.

Our results, while corresponding to only a single case study, may point to an attractive option of using SLMs for interacting with applications. Aside from the potential performance benefits, the SLMs can be hosted locally on a single machine. Such system architectures can be particularly appealing for edge computations (in warehouses, farms, vehicles, etc.) where data transfers and Internet connectivity might be a bottleneck.

## 2 Background and Motivation

## 2.1 Small Language Models

Concurrent with the advent of the Transformer architecture [5], a concept known as "scaling laws" began to take shape [6]. These laws suggest that as we increase computational resources or the size of neural networks, we can expect a corresponding improvement in performance [7]. This principle has guided the expansion LLMs, with notable examples including GPT-3 and GPT-4 [8, 9]. Further refinements to these scaling laws have led to significant leaps in model capabilities [10].

Recently, some research has pivoted towards another dimension of improvement: the enhancement of *data quality*, which reshapes the scaling law and ultimately led us to a series of Small Language Models [11, 12, 13, 14, 15]. These small models operate with orders of magnitude less parameters but are capable of rivaling their larger counterparts on key performance benchmarks. It's a well-established fact that better data yields better outcomes, as from dataset development [16]. High-quality data also results in smaller datasets [17, 18] and the opportunity for more iterations over the data [19].

Due to their efficiency and adaptability, SLMs have emerged as an important alternative to LLMs in many resource-criticial industrial scenarios. Their surprisingly low parameter count makes them ideal for applications requiring 1) promptly response, 2) little computation overhead, 3) on-device processing of private data and customized requests.

**The Phi models:** One series of powerful SLMs is the "Phi" models, which operates under the principle that "textbooks are all you need" [11]. These models leverage a highly curated dataset to train an efficient yet capable model. In particular, Phi-3, the latest member of the Phi series [15], achieves remarkable performance with a small amount of parameters (ranging from 3.8B to 14B).

### 2.2 Cloud supply chain fulfillment

In this section, we provide some general background on the application for which we built natural language support. We then describe the interaction mode through tasks and provide examples.

Table 1: Example task types and code for them.

Category	Task	Code Snippet (w/ APIs)			
Data Extraction	What is the standard deviation of supplier S's	ans = retrieve("""SELECT STDDEV(quantity) FROM inventory WHERE supplier_id = 'S'			
inventory in the last T weeks?		AND record_date >= NOW() - INTERVAL T WEEK;""")			
	What was the fraction	<pre>logger.log(f"The std is {ans}") total = retrieve(f"""SELECT SUM(quantity)</pre>			
of cross-geo shipments in		FROM shipment WHERE date >=			
	the last T weeks	NOW() - INTERVAL '{T} weeks'""")			
		cross = retrieve(f"""SELECT SUM(quantity)			
		FROM shipment WHERE date >=  NOW() - INTERVAL '{T} weeks'  AND src_geo != dest_geo;""")			
					if total:
					logger.log(cross / total * 100)
				else:	
				logger.log("No shipments at all")	
Plan Generation	Optimize plan	<pre>model.optimize()</pre>			
	Update (Execute) plan	plan.update()			
	Dock demand D on its	ideal_date = retrieve("""SELECT idd FROM			
	ideal dock date!	<pre>demand WHERE id = 'D';""")</pre>			
		demand.add_constraint(			
		demand_id="D", date=ideal_date,			
		enforce="Exact Match")			
		<pre>model.optimize()</pre>			
		if model.feasible:			
		plan.update()			
		logger.log("Plan updated with cost",			
		model.objVal)			
		else:			
		logger.log("Sorry, impossible to "			
		"dock demand D at its ideal date.")			
What-if Analysis	What are the cost impli-	supplies = retrieve("""SELECT id FROM			
	cations of not using sup-	<pre>supplier WHERE region = 'R';""")</pre>			
	pliers from region R on	<pre>if len(supplies):</pre>			
	Month M?	[ supply.add_constraint(			
		<pre>supply_id=id, demand="*",</pre>			
		date="2024-M-*", enforce="Prohibit";			
		for id in supplies ]			
		model.optimize()			
		<pre>logger.log(f"Cost will be {model.objVal}")</pre>			
		model.reset()			
		else:			
		logger.log("No supplies in R.")			

The application. We consider the fulfillment management application which has been designed internally at Microsoft for managing cloud supply chain fulfillment, a critical operational phase in cloud resource management in light of the constant growth of the cloud (see [20] and references therein); we henceforth refer to that application simply as App, as our design methodology is general and can be applied to other applications. At a high level, App manages the decision making process for satisfying the demand for cloud hardware. A demand request comes from internal Microsoft organizations, such as Microsoft 365 or Azure, and it is specified in racks of server units, alongside a desired dock (or deployment) date. App is in charge of generating a *fulfilment plan* for a configurable time-horizon (typically in the order of weeks). For each demand request, the plan includes (i) *supplier*. The hardware supplier (including warehouse location) that will be used to fulfill the demand, (ii) *Shipping*. The shipping method that will be used to transfer the racks of hardware from the warehouse, and the target datacenter that will host these racks, and (iii) *Scheduling*. The timing of the shipment; note that the a combination of shipping and scheduling determines the dock date. App uses an optimization algorithm to automatically generate a plan while accounting for different business and operational constraints [20].

**Motivation behind language models.** App is consumed by planners who periodically generate plans and oversee that the execution of the decisions is completed as planned. Planners may be interested in obtaining insights from historical plan data, understanding certain decisions, as well asking what-if questions. What-if questions may correspond to understanding the potential impact on the plan if certain conditions are changed (e.g., a supplier becomes unavailable, a certain shipping date must be enforced, etc.). Before the LLM disruption, planners would interact with engineering teams for understanding issues and answering what-if questions [20]. As we elaborate below, the role of language models is to facilitate the planners' job by enabling direct and efficient interaction with App.

**Tasks.** App may be used in order to carry out different *tasks*. The task types can be divided into three categories: data extraction, plan generation, and what-if analysis. Each task is carried out through one or more APIs (App has over forty different APIs). We provide some task examples below.

#### **Data extraction:**

- [supplier] What is the standard deviation of supplier S's inventory in the last T weeks?
- [shipping] What was the fraction of cross-geographical shipments in the last T weeks?
- [scheduling] Will demand D be deployed in its ideal dock date?

#### Plan generation:

• Optimize plan while taking into account a set of new constraints C.

### What-if analysis:

- [supplier] What are the cost implications of not using suppliers from region R on Month M?
- [shipping] By how much would a plan cost increase if we force priority shipping for demand D?
- [scheduling] Is it possible to dock demand D in its ideal dock-date?

**Language-model system overview.** Our system uses a language model to translate a human query to the right task and produce the relevant code for executing it. We henceforth refer to the system as *LM-Sys*. Our implementation currently supports around fifty different task types. For any supported task type, *LM-Sys* will translate a corresponding query (given in English) to a Python snippet, which can use multiple *App* APIs. We provide below a few examples, see also Table 1. For a new plan generation, a user may first ask the system to generate a plan while taking into account a new set of constraints. For example, the task "Generate a plan that does not use any

suppliers from region R on Month M", will first invoke an API to find the IDs of the suppliers from region R demand, and then use these IDs to add constraints prohibiting using the corresponding suppliers in the plan; the last step is generating the plan, which invokes an algorithm that will generate an optimal plan. What-if tasks produce a quantitative analysis of certain conditions are modified. Consequently, their implementation can be very similar to that of plan generation. For example, the task "What would be the cost increase if we do not use any supply from region R in Month M", will use the same code sequence as in the previous example, except that instead of updating the plan, the system will only invoke the algorithm in order to examine the total cost and compare it with that of the current plan.

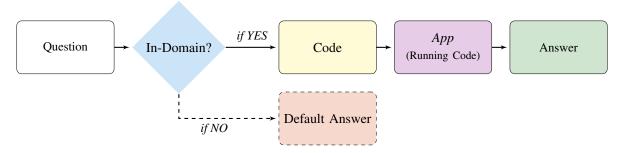


Figure 2: *LM-Sys* logical flow. User queries are first processed to determine if they are in-domain. For in-domain queries, *LM-Sys* generates the relevant code snippet that is executed to provide the answer. Otherwise, the SLM will return a default response (e.g., guide the user about supported tasks.

## 3 Language-model system design

In this section we highlight some design details of *LM-Sys*. Chronologically, our first implementation used LLMs, and only very recently we started incorporating SLMs in production. Figure 2 depicts *LM-Sys* data flow. A screenshot of an interaction log from production is given in Figure 3.

Natural-language interaction challenges. Building on our production experience, we observe that users occasionally pose questions that are irrelevant to the core functionality of the application, such as requests to "help me rewrite this email", "how is the weather today", or simply "how are you". One of our design goals has been to systematically filter out such questions. Another challenge arises from the variety of ways that a certain query can be phrased. For example, the following query, "why is my demand not fulfilled?", "there is no docking for my demand", and "can you dock my demand?" all point to the same task of understanding the reasoning for an unfulfilled demand (e.g., is it possible at all to fulfill the demand, or alternatively, quantify the cost implication of forcing that demand to be fulfilled). As the number of different supported tasks increases, it becomes more difficult to include the different question variations in the prompt itself. Fine-tuning thus becomes an attractive alternative, as one can increase the number of training examples to capture the diversity.

**SLM fine-tuning.** In our SLM-based design, we include the entire task handling flow in a single training example, similar to the "chain-of-thought" concept [21]. The process of generating training examples adheres to a conventional data generation pipeline. It incorporates techniques such as using example templates [20], "text-books are all you need" [11, 12, 15], and prompt evolution from WizardLM [22, 23]. To enhance the diversity and complexity of the data, we employed GPT-4 to rewrite human-posed questions, incorporating alternative writing styles including typos and distractions.

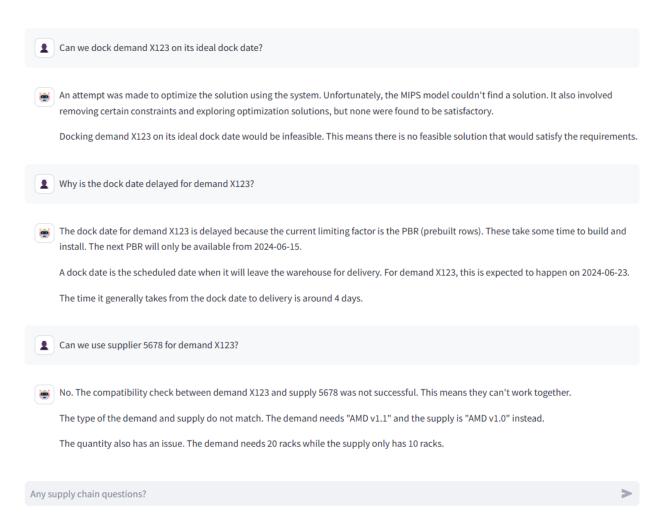


Figure 3: An example of an interaction log from production.

To ensure robust results, the SLM model training employed consistent hyperparameters across all setups. We utilized an effective batch size of 16 and employed the AdamW optimizer with a stable learning rate of 0.0002. Each model was constrained to a maximum of 1024 input tokens and 500 output tokens to meet our application needs. LoRA was integrated into the training process to effectively handle the relatively small datasets available [24]. The overall training was capped at 100,000 steps of gradient updates, so that each model has the same conditions for optimizing its weights no matter how many unique training examples it has seen.

**Output example.** A screenshot of an interaction log from production is given in Figure 3.

### 4 Evaluation

In this section we compare the different language models that we considered. They include different versions of GPT with in-context learning, and multiple fine-tuned SLMs.

Model	Method	Overall Acc. (%)	Coder Acc. (%)	F-1 (%)	Cost (¢)
Phi-2 (2.7B) [12]	not	$89.48 \pm 1.46$	$91.12 \pm 0.66$	$64.28 \pm 35.71$	0.18
Phi-3 mini (3.8B) [15]	Tuned w/ 1000-shot	<b>95.86</b> $\pm$ 0.33	$95.64 \pm 0.34$	100	0.19
Gemma (7B) [25]	100	$92.62 \pm 4.06$	$92.23 \pm 4.26$	$88.73 \pm 25.18$	0.25
Mistral v0.2 (7B) [26]	/w	$95.47 \pm 0.26$	$95.23 \pm 0.27$	100	0.29
Gorilla OpenFunc v2 (7B) [27]	pəu	$93.83 \pm 0.94$	$94.00 \pm 0.92$	$73.40 \pm 26.62$	0.21
Llama 3 (8B) [28]	Tm	$94.59 \pm 0.71$	$94.85 \pm 0.75$	$90.0 \pm 15.66$	0.28
GPT-3.5-turbo	1 (1) -4	71.01	75.21	80.00	0.23
GPT-4-turbo	1-Shot	75.88	74.60	75.00	4.52
GPT-3.5-turbo	2-Shot	77.81	76.63	80.00	0.44
GPT-4-turbo		82.00	81.05	85.71	8.72
GPT-3.5-turbo	3-Shot	76.78	75.56	80.00	0.66
GPT-4-turbo		83.84	82.96	80.00	12.92
GPT-3.5-turbo	5-shot	Out of Memory			
GPT-4-turbo	5-shot	84.15	83.31	85.71	21.32
GPT-4-turbo	10-shot	88.22	87.60	92.31	42.32
GPT-4-turbo	15-shot	87.99	87.37	92.31	63.32
GPT-4-turbo	20-shot	85.17	84.39	92.31	84.32

Table 2: Accuracy and cost per query of the different models. The SLMs results include average + standard deviation over ten runs.

## 4.1 Accuracy

**Metrics.** Table 2 summarizes the accuracy of the different models. We consider three metrics for accuracy. Analyzing production data, we observe that a non-negligible subset of queries (3-5%) is "out-of-domain", namely does not correspond to any of the tasks. As described above, the language model should identify such tasks and provide a 'generic' answer. The *F-1 Score* represents the success ratio of the identification. *Coder accuracy* measures the precision of the code generation for in-domain queries, namely queries that capture supported tasks. Finally, *Overall accuracy* is the ratio of queries (both out-of-domain and in-domain) for which the system produces the right output.

**Results.** The results depicted in Table 2 correspond to using one thousand training examples per task for the SLMs fine-tuning. For the LLMs, we report results with a varying number of in-context learning examples. At some point, the prompt size becomes a bottleneck and we cannot practically increase it further. The results indicate that Phi-3 mini and Mistral obtain the best results. The different LLM models fall short of the best SLMs.

We next zoom in on the convergence rate of the different models as a function of the number of training examples. For the GPT models, the examples are given in the prompt, and for the SLMs they are used for offline fine tuning. Figure 1 shows the results, where the x-axis represents the number of shots (or training examples per task); the total number of examples is obtained by multiplying that number by the number of tasks. The results show that GPT-4 is superior until 10 shots, but then plateaus. Phi-3 and Mistral obtain high accuracy with relatively few shots, and then improve gradually with more examples.

#### 4.2 Performance and cost

**Token count and latency.** For gradient-free LLM prompting, input token counts vary based on the number of shots. For example, for GPT-4-turbo there are approximately 4300 for 1-shot, 8500 for 2-shot, and 12400 for 3-shot. There are, on average, 72.24 output tokens with a standard deviation of 22.25. In comparison, the average input token count for Phi-3 is only 18.89 (std=12.88), and the output size is 108.76 (std=30.76) tokens. Due to the significantly lower average input token count, the fine-tuned Phi-3 can handle queries quickly, typically taking only a few seconds, compared to 1-2 minutes for GPT-4-turbo.

**Inference cost.** The cost of GPT models is calculated based on the average number of input and output tokens processed<sup>1</sup>. The cost per query for an SLM is estimated by the renting price of a GPU VM per hour divided by the number of queries that can be processed per hour; we note that this cost model assumes high GPU utilization, which can be achieved, for example, by utilizing Multi-LoRA inference servers [4]. In our calculation, we assume a batch size of one for the SLMs (costs may be lower if we apply mini-batching).

**Training cost.** Finally, the estimated training cost for SLMs is in the order of \$10 for 1000 shot training, which adds a negligible amount to the per-query cost, even when retraining is done every week.

### 5 Related Work

A preliminary version of using LLMs for interacting with the cloud supply chain fulfillment application was reported in [20]; that version supported a much smaller number of tasks. The *Gorilla* benchmark [27] is used to examine language model capabilities in translating a human query to the correct function call. The setting of that benchmark is different than ours, as the set of possible function calls is not known in advance and cannot be used in the training phase.

### References

- [1] Peter Lee, Sebastien Bubeck, and Joseph Petro. Benefits, limits, and risks of GPT-4 as an AI chatbot for medicine. *New England Journal of Medicine*, 388(13):1233–1239, 2023.
- [2] David Simchi-Levi, Philip Kaminsky, and Edith Simchi-Levi. *Designing and managing the supply chain: Concepts, strategies, and cases.* McGraw-hill New York, 1999.
- [3] Angels Balaguer, Vinamra Benara, Renato Luiz de Freitas Cunha, Roberto de M Estevão Filho, Todd Hendry, Daniel Holstein, Jennifer Marsman, Nick Mecklenburg, Sara Malvar, Leonardo O Nunes, et al. Rag vs fine-tuning: Pipelines, tradeoffs, and a case study on agriculture. *arXiv e-prints*, pages arXiv–2401, 2024.
- [4] Justin Zhao, Timothy Wang, Wael Abid, Geoffrey Angus, Arnav Garg, Jeffery Kinnison, Alex Sherstinsky, Piero Molino, Travis Addair, and Devvret Rishi. LoRA Land: 310 Fine-tuned LLMs that Rival GPT-4, A Technical Report. arXiv preprint arXiv:2405.00732, 2024.
- [5] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In Advances in Neural Information Processing Systems, volume 30, 2017.

<sup>&</sup>lt;sup>1</sup>As of May 2024, GPT-3.5-turbo-0125 costs \$0.50 per million input tokens and \$1.50 per million output tokens, while GPT-4-turbo is priced at \$10.00 per million input tokens and \$30.00 per million output tokens.

- [6] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.
- [7] Joel Hestness, Sharan Narang, Newsha Ardalani, Gregory Diamos, Heewoo Jun, Hassan Kianinejad, Md Mostofa Ali Patwary, Yang Yang, and Yanqi Zhou. Deep learning scaling is predictable, empirically. arXiv preprint arXiv:1712.00409, 2017.
- [8] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners. In Advances in Neural Information Processing Systems, volume 33, pages 1877–1901, 2020.
- [9] OpenAI. GPT-4 Technical Report, 2023. arXiv preprint arXiv:2303.08774 [cs.CL].
- [10] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katherine Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Oriol Vinyals, Jack William Rae, and Laurent Sifre. An empirical analysis of compute-optimal large language model training. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, Advances in Neural Information Processing Systems, 2022.
- [11] Suriya Gunasekar, Yi Zhang, Jyoti Aneja, Caio César Teodoro Mendes, Allie Del Giorno, Sivakanth Gopi, Mojan Javaheripi, Piero Kauffmann, Gustavo de Rosa, Olli Saarikivi, Adil Salim, Shital Shah, Harkirat Singh Behl, Xin Wang, Sébastien Bubeck, Ronen Eldan, Adam Tauman Kalai, Yin Tat Lee, and Yuanzhi Li. Textbooks Are All You Need, 2023.
- [12] Yuanzhi Li, Sébastien Bubeck, Ronen Eldan, Allie Del Giorno, Suriya Gunasekar, and Yin Tat Lee. Textbooks Are All You Need II: Phi-1.5 technical report, 2023.
- [13] Mojan Javaheripi, Sébastien Bubeck, Marah Abdin, Jyoti Aneja, Sebastien Bubeck, Caio César Teodoro Mendes, Weizhu Chen, Allie Del Giorno, Ronen Eldan, Sivakanth Gopi, et al. Phi-2: The surprising power of small language models. *Microsoft Research Blog*, 2023.
- [14] Bingbin Liu, Sebastien Bubeck, Ronen Eldan, Janardhan Kulkarni, Yuanzhi Li, Anh Nguyen, Rachel Ward, and Yi Zhang. TinyGSM: achieving > 80% on GSM8k with small language models. *arXiv preprint* arXiv:2312.09241, 2023.
- [15] Marah Abdin, Sam Ade Jacobs, Ammar Ahmad Awan, Jyoti Aneja, Ahmed Awadallah, Hany Awadalla, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Harkirat Behl, et al. Phi-3 technical report: A highly capable language model locally on your phone. *arXiv preprint arXiv:2404.14219*, 2024.
- [16] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551, 2020.
- [17] Shayne Longpre, Gregory Yauney, Emily Reif, Katherine Lee, Adam Roberts, Barret Zoph, Denny Zhou, Jason Wei, Kevin Robinson, David Mimno, et al. A Pretrainer's Guide to Training Data: Measuring the Effects of Data Age, Domain Coverage, Quality, & Toxicity. *arXiv preprint arXiv:2305.13169*, 2023.

- [18] Da Yu, Sivakanth Gopi, Janardhan Kulkarni, Zinan Lin, Saurabh Naik, Tomasz Lukasz Religa, Jian Yin, and Huishuai Zhang. Selective Pre-training for Private Fine-tuning. *arXiv preprint arXiv:2305.13865*, 2023.
- [19] Niklas Muennighoff, Alexander M Rush, Boaz Barak, Teven Le Scao, Aleksandra Piktus, Nouamane Tazi, Sampo Pyysalo, Thomas Wolf, and Colin Raffel. Scaling Data-Constrained Language Models. *arXiv* preprint arXiv:2305.16264, 2023.
- [20] Beibin Li, Konstantina Mellou, Bo Zhang, Jeevan Pathuri, and Ishai Menache. Large language models for supply chain optimization. *arXiv* preprint arXiv:2307.03875, 2023.
- [21] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. *arXiv* preprint arXiv:2201.11903, 2022.
- [22] Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, and Daxin Jiang. Wizardlm: Empowering large language models to follow complex instructions. *arXiv preprint arXiv:2304.12244*, 2023.
- [23] Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. WizardCoder: Empowering Code Large Language Models with Evol-Instruct, 2023.
- [24] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-Rank Adaptation of Large Language Models. In *International Conference on Learning Representations*, 2022.
- [25] Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, et al. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295*, 2024.
- [26] Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. Mistral 7B. *arXiv* preprint arXiv:2310.06825, 2023.
- [27] Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. Gorilla: Large language model connected with massive apis. *arXiv preprint arXiv:2305.15334*, 2023.
- [28] Meta. Introducing Meta Llama 3: The most capable openly available LLM to date. https://ai.meta.com/blog/meta-llama-3/, 2024.