Exploring Length Generalization in Large Language Models

Anonymous Author(s)
Affiliation
Address
email

Abstract

The ability to extrapolate from short problem instances to longer ones is an important form of out-of-distribution generalization in reasoning tasks, and is crucial when learning from datasets where longer problem instances are rare. These include theorem proving, solving quantitative mathematics problems, and reading/summarizing novels. In this paper, we run careful empirical studies exploring the length generalization capabilities of transformer-based language models. We first establish that naively finetuning transformers on length generalization tasks shows significant generalization deficiencies independent of model scale. We then show that combining pretrained large language models' in-context learning abilities with scratchpad prompting (asking the model to output solution steps before producing an answer) results in a dramatic improvement in length generalization. We run careful failure analyses on each of the learning modalities and identify common sources of mistakes that highlight opportunities in equipping language models with the ability to generalize to longer problems.

1 Introduction

Many natural problems, such as theorem proving and program synthesis, have a notion of length that strongly correlates with the difficulty of the task. However, in these domains, the number of available problems typically drops rapidly as a function of problem length (see Figure 2 for an example). Therefore, a desirable behavior of the model is to learn from examples of shorter lengths to generalize to longer ones, or at least reduce the number of samples required for longer examples. We refer to this type of problem as *length generalization*.

Recent work on large language models (LLMs) has shown consistent improvement in their performance by scaling model and dataset size. However, such models are still incapable of length generalization. For example, [1] shows that even though scale helps with solving arithmetic problems, scale alone is likely insufficient for learning to solve instances of arbitrary lengths. This implies that models fail to learn the general algorithms that would enable this kind of generalization. Indeed, Razeghi et al. [2] showed that the performance of LLMs on mathematical calculations correlates with term frequency. This suggests that LLMs might have gained their current performance from memorization or pattern matching as opposed to learning the algorithm to perform the calculations.

A recent line of work proposes to use a scratchpad, or chain-of-thought reasoning, when prompting LLMs [3, 4] on multi-step tasks. Breaking down tasks into multiple small steps, and presenting these steps to the model, led to improved performance across a variety of reasoning tasks including word problems, arithmetic, and code execution.

We perform a systematic study of length generalization with transformer-based large language models. We consider problems in which learning an algorithm can in principle enable a model to extrapolate from short examples to problems of arbitrary length. In particular, we focus on two simple algorithmic tasks, *parity* and *variable assignment*, in which the model needs to keep track of a state in order to extrapolate to longer lengths (see Figure 1). These problems are illuminating because their simplicity

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

In Distribution Lengths

```
# Length 3
x = True
y = False
if x == True:
      y = True
print(y)
```

```
# Length 4
a = True
b = False
if a == False:
      b = True
if b == True:
      a = False
print(a)
```

```
# Length 5
k = True
l = False
m = True
k = k and l
k = not k
print(k)
```

Out-of-Distribution Lengths

```
# Length 20
x = True
y = False
z = True
z = x and y
(...)
if x == y:
      z = true
z = z and x
if y == x:
      x = z
print(z)
```

Figure 1: **Examples of variable assignment problems:** Can transformer-based language models learn from short instances of the Variable Assignment task displayed above (left) to extrapolate to much longer instances (right)? **Length generalization** refers to the ability to learn from shorter/easier instances of a problem to acquire the ability to handle longer/harder instances.

Techniques	In-distribution	Out-of-distribution	Improves with scale
Fine-tune	✓✓	✗	✗
Prompting	✗	✗	✗
Fine-tune + Prompting	✓✓	✗	✗
Fine-tune + Scratchpad	✓✓	✗	✗
Prompting + Scratchpad	✓	✓	✓
Fine-tune + Prompting + Scratchpad	✓✓	✓✓*	✓✓

Table 1: Performance on length generalization tasks of three techniques that language models admit: (1) Finetuning, (2) Prompting (or in-context few-shot learning) and (3) Chain-of-Thought reasoning. We find that each techniques (and the combinations thereof) have different modes of failure and present different trade-offs regarding in and out-of-distribution coverage. ✗ signifies poor ✓ signifies nontrivial, ✓✓ signifies near-perfect performance. (*) Refers to task-dependency.

allows us to probe the failure modes as well as contrast the learned solutions with the ground truth algorithm. They provide us with a setting to study how/when these large language models start to fail.

We study combinations of three kinds of techniques for LLMs: finetuning, few shot prompting (also referred to as in-context learning), and use of a scratchpad (also referred to as chain-of-thought), to understand the role of each method and the interplay among the three in length generalization. Interestingly, we observe non-trivial interactions among the three techniques; see Table 1.

Contributions Our main contributions are as follows:

- We define and characterize the problem of length generalization using notions such as state tracking, execution depth, and per-step error rate. We study and carefully design two tasks, *parity* and *variable assignment*, that measure length generalization (Section 2).
- We find that in the finetuning regime, scaling data, model sizes, and compute does not improve length generalization (Section 3.1). We also observe that even when the model attains perfect in-distribution accuracy, it performs poorly in out-of-distribution domains. Surprisingly, different hyperparameter choices for finetuning have a large effect on length generalization performance, while having minimal effect on the final in-distribution performance (Section 3.3).
- We show finetuning with scratchpad also fails to generalize to longer problems, in contrast to what is suggested by previous works [3]. We look into three potential failure cases: positional encoding, the presence of distractors, and end of token prediction, and conclude that distractors are the main culprit of failures for length generalization (Section 4).
- We establish that in the in-context learning regime, use of a scratchpad shows a qualitatively different behavior and significantly alleviates the decay of performance on longer problems. This capability is a significant finding as it implies that for LLMs, there are certain skills, like length generalization, that can be learned through in-context learning rather than through finetuning, even in the presence of infinite data. This is in stark contrast to the common norms of machine learning (Section 5).

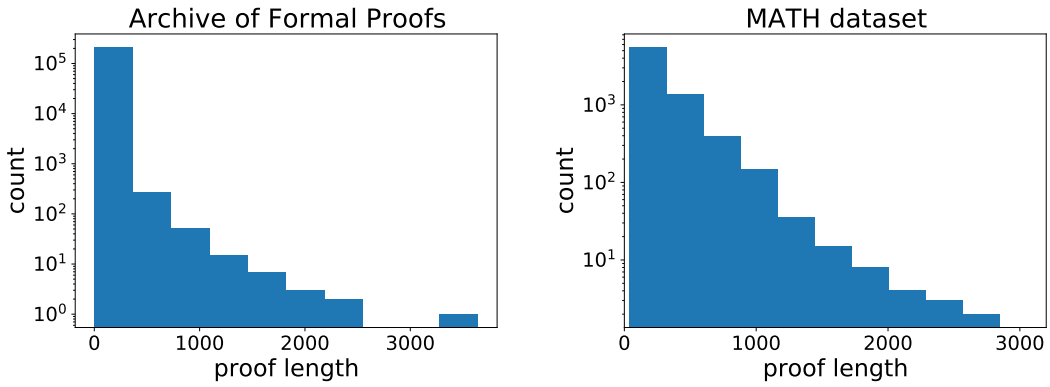

Figure 2: **Real world datasets have heavy tails in length: (left)** Histogram of lengths for proofs presented in the Archive of Formal Proofs **(right)** Histogram of the number of tokens for solutions in the MATH dataset. [5]

2 Length Generalization

Many sequence tasks—especially ones that require reasoning capabilities—have problem instances that differ in terms of their *lengths*. Shorter instances are often easier to state, process, and handle, and require less compute to find the answer. By contrast, longer instances are more challenging to parse and require more compute. Tasks that have a *reasoning component* are especially well represented in this category — multi-hop reasoning [6], program execution [7], deductive reasoning [8] and theorem proving [9], to name a few. Note that having to deal with differing problem lengths poses two significant challenges. First, it is often the case that one encounters longer problem instances than the ones ever encountered during training, and is required to extrapolate. Second, even though longer problem instances have much more variety, real-world datasets often contain few long instances (see Figure 2. Both of these challenges are exacerbated if learning agents are not able to generalize across and beyond the lengths they learn from during training. This paper is about investigating to what extent transformer based language models are able to observe short problem instances and extrapolate to longer ones.

Instance Length as Number of Steps in a Markov Process It is possible to define *problem length* in many different ways to capture different aspects of problem difficulty. Does there exist a notion of length that would expose the same length-generalization-related problem structure observed in qualitatively very different settings? Such a framing would enable researchers to design algorithms and interventions that have the potential to generalize across a broad range of tasks. To this end, we take the approach of characterizing length in the context of a *deterministic Markov process*. From this perspective, length is simply the number of state transitions experienced by an initial world state, which is specific to each individual task. In other words, the data-generation process can be described as sampling an (2) *initial state* and a (2) *variable number of state transformations* to be applied sequentially on the initial state. The agent is provided both the initial state and the transformations, and is trained to predict the final state. This framing applies to a wide range of sequence problems, if not all of them—ranging from more mechanical tasks such as code and algorithm execution and theorem proving, to less structured tasks, such as solving math problems and summarizing detective novels.

In our empirical investigation we focus on two synthetic tasks: *parity* and *variable assignment*. These two tasks avoid problem-specific subtleties that could mislead our analyses, while strongly capturing the deterministic Markov process structure.

2.1 Tasks

Parity: The parity task is an age-old learning problem that requires the trained agent to predict whether a bit-string has an even or odd number of ones in it. For example, the parity of the bitstring $[0, 1, 1, 0, 1]$ is "odd" (or 1) as opposed to "even" (or 0), because there is an odd number of 1s in the bit-string. The parity task admits a sequential solution that enables length generalization in a straightforward way: simply process the bits left-to-right and record the parity of the bits processed

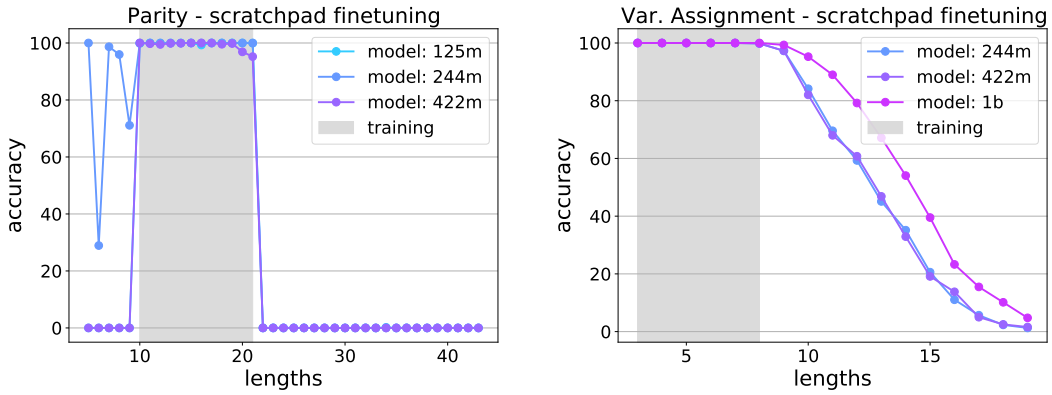

Figure 3: **Scratchpad finetuning displays poor length generalization:** Scratchpad finetuning displays qualitatively similar length generalization pathologies as vanilla finetuning.

so far as the state. The default notion of length in the parity task is the number of bits in the input. However, we also experiment with a version where the number of bits is kept constant, and the number of 1s (i.e. the parity flipping bit) is systematically varied. The number of 1s stands for the number of state changes contained in the input bit-string, and actually appears to capture a more relevant notion of length for transformer models (see Section 3.1).

Boolean Variable Assignment Task: The Boolean Variable Assignment task is designed to capture arbitrarily long, potentially branching unidirectional execution flows. An instance of this task can be seen in Figure 1. The inputs consist of semantically correct (i.e. bug-free) Python programs in which each line contains a boolean variable assignment operation. The output is simply the value of the variable presented in the final line of the program. The sequential solution to this task is to simply execute the program line by line while keeping track of the state of all variables.

The data generation procedure involves randomly generating execution flows that involve Boolean operations; see Supplementary Material (SM) for details. We focus our evaluations on two variants of this dataset. (1) The *diverse variable assignment* split consists of all the boolean operators available and is intended to contain maximally diverse programs. (2) The *chain-like variable assignment split* consists only of operations that compose the values of already defined variables. This results in long chains of dependencies between the initial values of the variables and the queried one, ensuring that there are no redundant operations in the program (i.e. operations that can be removed without affecting the output of the program). This split emphasizes the sequential nature of the variable assignment problem.

3 Standard Finetuning Fails at Length Generalization

We begin by demonstrating that finetuning transformer models on length-generalization tasks results in poor out-of-distribution performance. In experiments we use `BaseModel` [1] decoder-only models. These checkpoints were trained using general natural language data. We use the AdaFactor optimizer [10] during finetuning, and tune the learning rate, batch size and dropout.

3.1 Scale Doesn't Improve Length Generalization

Parity: We finetuned three pretrained `BaseModel` models with 244m, 422m, and 1b parameters on the parity task, where the training distribution included randomly sampled bitstrings of length 10 to 21. We then evaluated the performance on bitstrings of length 3 to 40; see Figure 4. We find that model scale has a little effect on length generalization.

Variable Assignment: We finetuned the same models on the *chain-like* Variable Assignment Task, described in Section 2. We kept the in-distribution lengths at 3 to 11, and evaluated the test performance on lengths 3 to 19. The results can be seen in Figure 4. Just like in the parity task, while the

[1] We omit the name of the model for the purpose of anonymization.

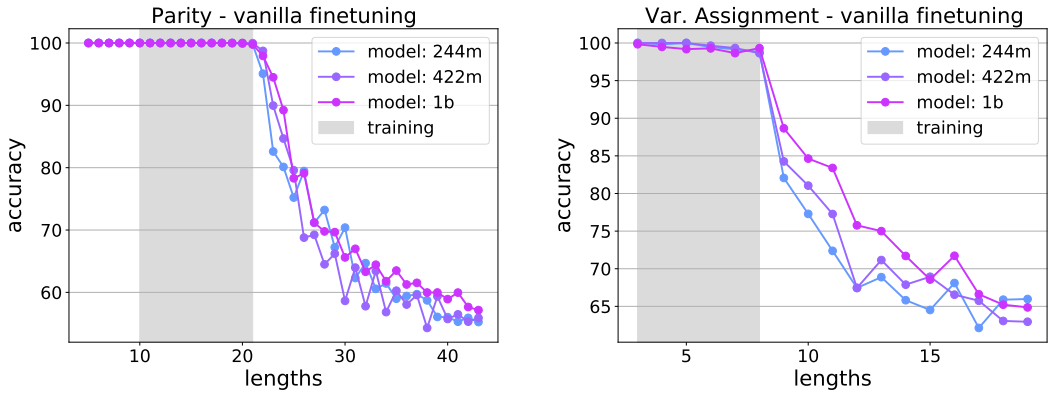

Figure 4: **Finetuned Length Generalization performance doesn't improve with scale:** Models of differing sizes fail at length generalization on both Parity and Variable Assignment tasks, displaying identical generalization pathologies.

in-distribution performance is (near) perfect, out-of-distribution performance degrades rapidly as length increases. To get a sense of just how weak the out-of-distribution performance is, we also trained a 422m model on the same dataset, except we shuffled the operations before feeding it to the model. This removes the sequential dependency between the operations, and helps us establish a strong baseline that only predicts the answers based on non-sequential, spurious correlations. The accuracy-length curves for the baseline can be found in SM.

3.2 Transformers Prefer Parallel Strategies over Sequential Ones

The results presented in Section 3.1 establish that, when presented with sequential length generalization problems, transformers are biased toward learning non-sequential "shortcut" solutions that fail at longer problem instances. We ran additional experiments to gain a better understanding of the nature of this generalization pattern.

On the parity task, we ran finetuning on a different distribution of bit-strings: Instead of first randomly sampling the number of bits in the input bit-string, then sampling the values of the bits, we fixed the total number of bits in the input, and only varied the *number of ones* in the bit-string uniformly. We trained with 10 to 20 ones in the input distribution and tested on an interval containing 1 to 30 ones. This serves two purposes: (1) We can test generalization to *shorter* instances as well, (2) the number of tokens (now fixed at 30) is now disambiguated from number of state changes, which for parity is equal to the number of ones. The difference between in and out-of-distribution performance is even more stark for this data distribution (Figure 5): while in-distribution performance was 100%, OOD performance was close to 0% [2] This suggests that the transformers are learning a non-sequential solution to the parity problem that involves counting the number of ones in the input, and then thresholding the output. This strategy doesn't generalize to longer sequences, and doesn't allow for knowledge transfer between problems of different lengths. This is not surprising, given that self-attention is an equivariant transformation capable of performing pooling operations like max-pooling [11].

On the variable assignment dataset, we finetuned a 255m `BaseModel` model on the *diverse* split of the variable assignment dataset of programs up to 25 lines, and evaluated on the same data generating distribution up to 50 lines. We measure the evolution of the model's accuracy with respect to training iterations on different program lengths (quantified by number of lines). The results can be seen in SM.

We again observed that a different notion of length (which we call "computational graph depth") captures the difficulty of problem instances better than number of program operations. A variable assignment program can be represented as a computational graph where each node corresponds to

[2] We didn't constrain the output of the models to 0s and 1s - an accuracy of 0% means that the model consistently outputted a token that is neither a 0 nor a 1.

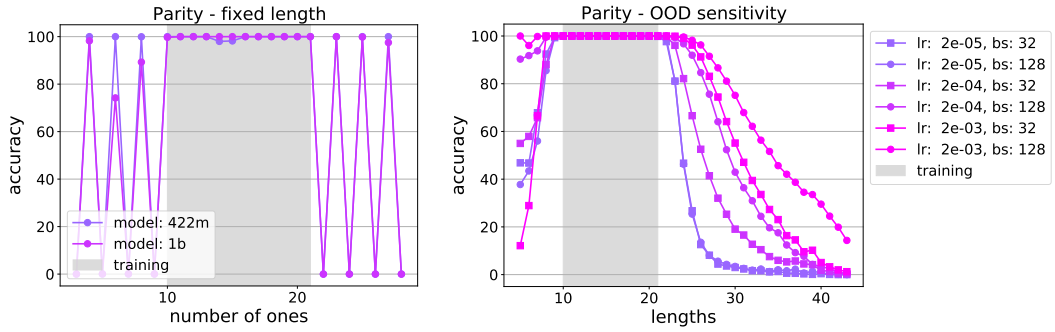

Figure 5: **(left) Complete lack of length generalization:** Transformers trained on the parity task have difficulty generalizing to bit-strings that have a different number of $1s$. **(right) Sensitivity to hyperparameters:** Trained networks sharing architecture, data and in-distribution loss can have very different length generalization performances.

a variable, and each edge corresponds to an operation. Computational graph depth is the length of the longest dependency chain that connects to the queried variable node. We present two results that suggest that the relevant notion of length on the variable assignment task for transformers is computational graph depth. First, by inspecting the order in which the trained transformer correctly solves this task we find that performance is strongest on examples with small computational graph depth, even if these examples are long in terms of number of operations, rather than on examples with fewer operations and greater computational graph depth. Second, the transformer does a good job of handling programs with an out-of-distribution number of operations, but for which computational graph depth is in-distribution.

3.3 In-Distribution Generalization Doesn't Predict OOD Generalization on Length Generalization Tasks

Prior work on out-of-distribution generalization establish that in many tasks, in-distribution loss is a strong predictor of out-of-distribution generalization [12]. Our experiments on the parity task indicate that the distribution shift induced by changing problem lengths falls outside of the aforementioned category. Figure 5 shows how the same model trained on the same data achieving roughly the same in-distribution cross entropy loss behaves on OOD data.

It has been shown in prior work that it's possible to get pretrained LLMs to solve a given task by not only outputting the answer, but also the solution steps behind it. Nye et al. [13] use scratchpad finetuning to achieve strong in-distribution performance on execution based tasks such as code execution and computing polynomials. While they also report modest length generalization results on integer arithmetic, we find that scratchpad finetuning suffers from similar length generalization pathologies than vanilla finetuning does. The results on parity and variable assignment tasks can be seen in Figure 4. The precise scratchpad strategies used for these tasks are described in detail in SM.

4 Scratchpad Finetuning Still Fails at Length Generalization

Error analysis: To understand the causes of failure in training scratchpad strategies, we focused on two architectural choices that could account for the poor performance: (1) how transformers encode position information, and (2) whether the transformers are trained to predict an end-of-sequence (EOS) token. `BaseModel` models use T5 position biases [14] to handle position information. If the network is only trained with short instances, position biases that handle longer positional distances might not be trained, explaining poor length generalization. Similarly, Newman et al. [15] report that networks trained with EOS token prediction often suffer from generalizing to longer problem instances, because of the models' tendency to emit EOS tokens prematurely, as well as the EOS tokens' effect on the representations that get learned.

We tested the extent to which these effects can explain lack of length generalization as follows. We padded both the input bit-strings and the scratchpad content with dummy padding tokens to

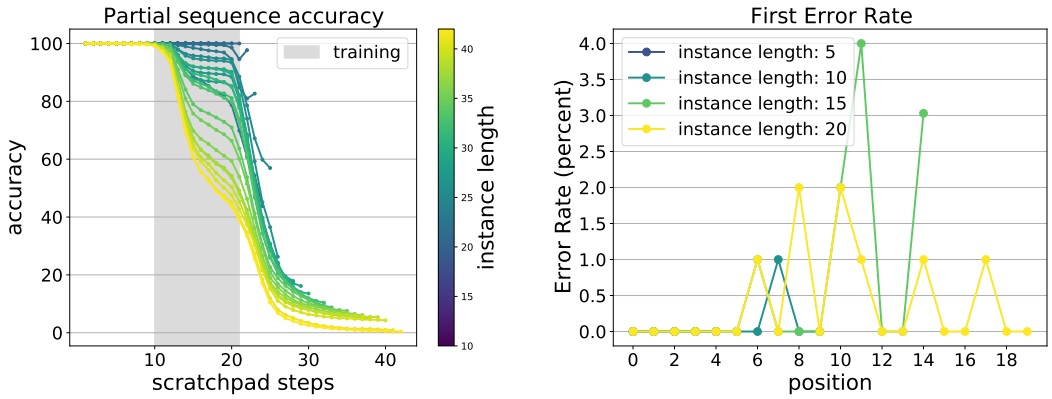

Figure 6: **(left) Effect of input length on per-step scratchpad accuracy:** Points corresponds to the accuracy (y-axis) of the first x scratchpad steps (x-axis) on parity instances of variable length (color). If the input length is out-of-distribution, even in-distribution scratchpad steps are inaccurate, implying the model hasn't learned an attention pattern that generalizes to longer bit-strings. **(right) Roughly constant per-step error rate:** The per-step error rates of the `BaseModel` 128b model, few-shot finetuned on the coin-flip version of the parity task remain constant across the scratchpad steps. This is in contrast with zero-shot scratchpad finetuned models, where the per-step error rates increase.

make the token count the same. We also augmented the input and scratchpad targets with the same number of padding tokens on the left and right so that the relevant bit to attend to when executing the sequential scratchpad strategy corresponds to the same T5 position bias bin. Examples of the updated input-target pairs can be seen in SM. While this intervention helps, the trained models still display significant length generalization issues.

To gain further insight about the source of the problem, we plotted how the scratchpad target prediction error rates change as a function of (1) how far along one is in constructing the scratchpad, and (2) the length of the input bit-string. The results can be seen in Figure 6. The fact that the model makes mistakes in in-distribution scratchpad steps when the input has an OOD length implies that the attention mechanism isn't capturing the relevant part of the input to form the scratchpad output. See SM for additional analysis.

5 Scratchpad Prompting Significantly Improves Length Generalization

Wei et al. [4] and Nye et al. [13] showed that combining prompting (i.e. in-context learning) with scratchpad strategies present a powerful combination. They demonstrate that pretrained LLMs, without the help of any finetuning, can solve grade school math word problems and execute pieces of code with nontrivial correctness [13], when prompted with the right scratchpad strategy. We corroborate these findings, and report that scratchpad prompting endows pretrained LLMs with the capability of *variable length template matching* (see Figure 8). That is, in-context learning enables the model to "learn" solution steps from a small number of short instances, and apply the same template on significantly longer instances with a high degree of accuracy.

5.1 Few-shot scratchpad

Contrary to vanilla and scratchpad finetuning, we find that few-shot scratchpad strategies sometimes significantly improves LLMs' capability to extrapolate to lengths much further than what pretraining weights grant them in a task-dependent manner.

To evaluate the performance of few-shot conditioning with scratchpad inputs without any finetuning, we phrase the parity problem in natural language as a coin flipping task. An example for the few-shot prompts we used can be seen in Figure 8. Wei et al. [4] also report results on the coin-flip task: we note that the scratchpad format we used differs in that while ours respects the sequential nature of the task (i.e. each coin flip corresponds to a step in the scratchpad solution), Wei et al. [4]'s scratchpad strategy involves summing up the number of coin flips, then deciding on the final output based on the

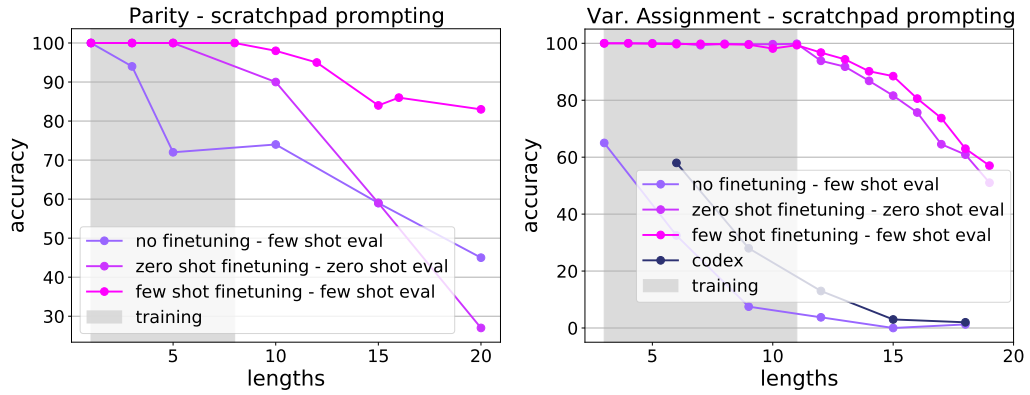

Figure 7: **Few-shot finetuning with scratchpad** displays qualitatively different behaviour on parity and variable assignment tasks. On parity, where the non-finetuned model already performs very well, few-shot-finetuning with scratchpad leads to a significant performance boost over zero-shot finetuning with scratchpad. On variable assignment, where the base model doesn't perform poorly, there's not a significant gap between few-shot finetuning and zero-shot finetuning with scrathpad.

evenness/oddness of the sum. Also, while they only test up to 4 flips, we go up to 20 flips while still attaining nontrivial accuracy levels.

For the variable assignment task, our scratchpad strategy involves copying over the program that's being executed, with comments added in between lines specifying the value of the variable that was assigned in the line above. Instances of this scratchpad strategy can be seen in SM.

Figure 7 shows the performance of the pretrained `BaseModel` 128b model on the coin-flip version of the parity task. Figure 8 shows an instance of how a length 3 prompt can induce the model to correctly output a 20 step scratchpad. We find that with the right scratchpad prompt, LLMs are able to generate correct scratchpad solutions. This reduces the problem to simply filling in the content of the generation correctly by inferring the right state transitions without having to figure out how to extrapolate the solution template.

Few-Shot Finetuning with Scratchpad Strategies: Does combining finetuning, few-shot prompting, and scratchpad strategies improve length generalization?

We find that the answer is **yes** in the case of parity. As seen in Figure 7, few-shot finetuning performs significantly better than the baseline model, both on in- and out-of-distribution lengths. Note that the vanilla (i.e. no shot) finetuning baseline also outperforms the no-finetuning baseline, it actually does worse on the larger lengths — a pathology that doesn't appear with few-shot finetuning.

The results point to a qualitatively different picture for the variable assignment task. Both few-shot finetuning and vanilla finetuning result in similar length generalization behavior (Figure 7). We hypothesize that this distinction is caused by the different pretrained performances that the model displays on these tasks: while length generalization is already very strong with no finetuning on parity, that's not the case for variable assignment. In the latter case, the model is forced to acquire a new skill via finetuning, which displays the same pathologies as zero-shot finetuning with scratchpad. Evaluations on a larger number of tasks is needed to test this hypothesis.

6 Related Works

There have been many attempts to study neural networks' ability to generalize from shorter/easier to longer/harder examples.

Challenges in length generalization: Several existing works have investigated pathologies that arise when models are asked to generalize to processing and generating longer (measured by number of tokens) sequences. Newman et al. [15] find that sequence models trained with and in the absence of the end-of-sequence token display qualitatively different length extrapolation behaviour and learn different representations. Dubois et al. [16] proposes modifications to the commonly used dot-product attention to improve the models' ability to extrapolate to longer sequences. Murray and Chiang

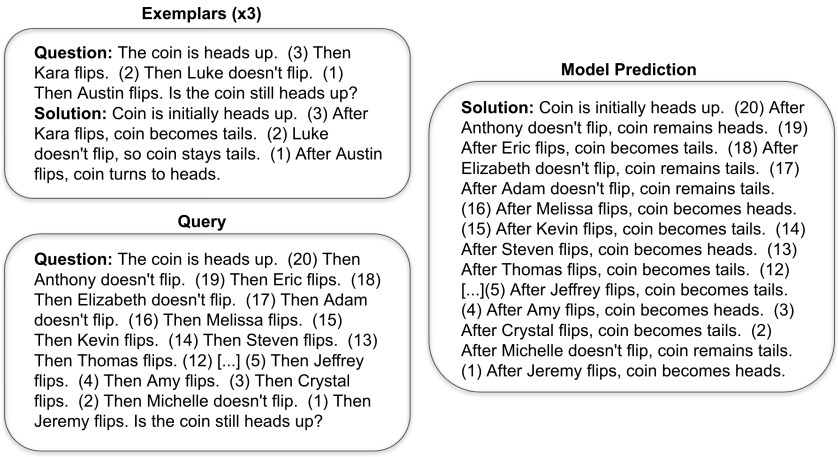

Figure 8: **Few-shot length generalization:** The largest `BaseModel` model is able to map the scratchpad solution template from a few short exemplars onto much longer queries.

[17] demonstrate that neural machine translation models tend to have a bias towards generating shorter-than-desired translations. Yehudai et al. [18] show that length generalization issues are also present in training graph neural networks, where extrapolating across graph size presents a challenge. Ju et al. [19] propose a new attention mechanism to facilitate recurrent processing in transformer models. Press et al. [20] propose modifying transformer attention biases to facilitate generalization beyond the training context length.

Easy-to-Hard generalization: Schwarzschild et al. [21] and Bansal et al. [22] use weight-tied neural networks to generalize from easy to hard examples. Schwarzschild et al. [21] also provide three tasks to benchmark easy-to-hard generalization. Dehghani et al. [23] and Kaiser and Sutskever [24] assess the capabilities of their proposed architectures on easy-to-hard generalization problems.

7 Conclusion

The ability to learn from shorter/easier problem instances to generalize to longer/harder ones is a key capability in a large number of tasks, especially ones requiring reasoning. We defined the concept of *length generalization* and measured language models' length generalization capabilities. After conducting careful experiments using finetuning, scratchpads, and few-shot prompting, we reached the following conclusions: (1) Length generalization is a challenge for language models at least up to the 100B parameter scale. Both vanilla finetuning and finetuning with scratchpads suffer from a lack of length generalization caused by models' tendency to pick up non-sequential pattern that don't generalize to larger lengths.(2) Few-shot scratchpad prompting allows pretrained large language models to pick up scratchpad-templates that extrapolate to arbitrary lengths, leading to dramatic improvements on longer problem instances. Unlike raw finetuning, this approach does scale with model size [4]. (3) Trying to further enhance the performance of few-shot scratchpad prompted LLMs via finetuning yields mixed results, depending on the non-finetuned performance of the base model at the target task. We emphasize that the aforementioned few-shot variable length pattern matching capability — something that doesn't require changing model architecture — offers a qualitatively different approach to handle length generalization in contrast to prior art that introduced architectural modifications to achieve the same goal. This capability is also significant in that it implies that for LLMs, there are certain skills, like length generalization, that can be learned better through in-context learning rather than through finetuning, even in the presence of infinite data.

Societal impact. Language models that are are able to extrapolate beyond their training distribution have the potential to be useful on a variety of useful tasks. In this work we take a step in this direction, by investigating the failure modes of existing models. We do not foresee that the present work will have notable societal impact.

References

[1] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[2] Yasaman Razeghi, Robert L Logan IV, Matt Gardner, and Sameer Singh. Impact of pretraining term frequencies on few-shot reasoning. *arXiv preprint arXiv:2202.07206*, 2022.

[3] Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, et al. Show your work: Scratchpads for intermediate computation with language models. *arXiv preprint arXiv:2112.00114*, 2021.

[4] 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, 2022. URL https://arxiv.org/abs/2201.11903.

[5] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

[6] Haoyu Wang, Mo Yu, Xiaoxiao Guo, Rajarshi Das, Wenhan Xiong, and Tian Gao. Do multi-hop readers dream of reasoning chains? *arXiv preprint arXiv:1910.14520*, 2019.

[7] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

[8] Peter Clark, Oyvind Tafjord, and Kyle Richardson. Transformers as soft reasoners over language. *arXiv preprint arXiv:2002.05867*, 2020.

[9] Yuhuai Wu, Albert Qiaochu Jiang, Jimmy Ba, and Roger Grosse. Int: An inequality benchmark for evaluating generalization in theorem proving. *arXiv preprint arXiv:2007.02924*, 2020.

[10] Noam Shazeer and Mitchell Stern. Adafactor: Adaptive learning rates with sublinear memory cost. In *International Conference on Machine Learning*, pages 4596–4604. PMLR, 2018.

[11] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *International Conference on Machine Learning*, pages 3744–3753. PMLR, 2019.

[12] Vaishnavh Nagarajan, Anders Andreassen, and Behnam Neyshabur. Understanding the failure modes of out-of-distribution generalization. *arXiv preprint arXiv:2010.15775*, 2020.

[13] Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, et al. Show your work: Scratchpads for intermediate computation with language models. *arXiv preprint arXiv:2112.00114*, 2021.

[14] 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. *arXiv preprint arXiv:1910.10683*, 2019.

[15] Benjamin Newman, John Hewitt, Percy Liang, and Christopher D Manning. The eos decision and length extrapolation. *arXiv preprint arXiv:2010.07174*, 2020.

[16] Yann Dubois, Gautier Dagan, Dieuwke Hupkes, and Elia Bruni. Location attention for extrapolation to longer sequences. *arXiv preprint arXiv:1911.03872*, 2019.

[17] Kenton Murray and David Chiang. Correcting length bias in neural machine translation. *arXiv preprint arXiv:1808.10006*, 2018.

[18] Gilad Yehudai, Ethan Fetaya, Eli Meirom, Gal Chechik, and Haggai Maron. From local structures to size generalization in graph neural networks. In *International Conference on Machine Learning*, pages 11975–11986. PMLR, 2021.

[19] Da Ju, Stephen Roller, Sainbayar Sukhbaatar, and Jason Weston. Staircase attention for recurrent processing of sequences. *arXiv preprint arXiv:2106.04279*, 2021.

[20] Ofir Press, Noah Smith, and Mike Lewis. Train short, test long: Attention with linear biases enables input length extrapolation. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=R8sQPpGCv0.

[21] Avi Schwarzschild, Eitan Borgnia, Arjun Gupta, Furong Huang, Uzi Vishkin, Micah Goldblum, and Tom Goldstein. Can you learn an algorithm? generalizing from easy to hard problems with recurrent networks. *Advances in Neural Information Processing Systems*, 34, 2021.

[22] Arpit Bansal, Avi Schwarzschild, Eitan Borgnia, Zeyad Emam, Furong Huang, Micah Goldblum, and Tom Goldstein. End-to-end algorithm synthesis with recurrent networks: Logical extrapolation without overthinking. *arXiv preprint arXiv:2202.05826*, 2022.

[23] Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. *arXiv preprint arXiv:1807.03819*, 2018.

[24] Łukasz Kaiser and Ilya Sutskever. Neural gpus learn algorithms. *arXiv preprint arXiv:1511.08228*, 2015.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [Yes]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [No] The code and the data are proprietary.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [No]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [N/A]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]