Chain-Of-Thought Reasoning is a Policy Improvement Operator

Hugh Zhang
Harvard University
hughzhang@seas.harvard.edu

David C. Parkes
Harvard University
parkes@eecs.harvard.edu

Abstract

Large language models have astounded the world with fascinating new capabilities. However, they currently lack the ability to teach themselves new skills, relying instead on large amounts of human-generated training data. We introduce *SECToR* (**S**elf-**E**ducation via **C**hain-of-**T**h**o**ught **R**easoning), a proof-of-concept demonstration that language models can teach themselves new skills using chain-of-thought reasoning. During the self-learning loop, SECToR asks models to solve addition problems using chain-of-thought reasoning before training the next version of the model to solve those same problems directly *without using such reasoning*. This process often results in an improved model which can, when again augmented with chain-of-thought reasoning, solve even harder problems than the original model, allowing the self-learning loop to continue. Language models trained via SECToR autonomously learn to add up to 29-digit numbers without access to any ground truth examples beyond an initial supervised fine-tuning phase consisting only of numbers with 6 or fewer digits. Our central hypothesis is that chain-of-thought reasoning can act as a policy improvement operator, similarly to how Monte-Carlo Tree Search is used in AlphaZero (Silver et al., 2017). We hope that this research can lead to new directions in which language models can learn to teach themselves without the need for human demonstrations.

1 Introduction

Large language models are currently trained on vast corpora of human-generated data (Vaswani et al., 2023; Devlin et al., 2019; Radford et al., 2019; Brown et al., 2020; Chowdhery et al., 2022; Touvron et al., 2023). While large language models have demonstrated many surprising capabilities, the possibility of reaching superhuman performance is a challenging proposition when training solely on existing data. In this paper, we ask whether large language models can autonomously teach themselves new skills rather than solely depending on the availability of suitable data. A positive answer to this question would open the door to a tantalizing possibility. Although the discovery of scaling laws (Kaplan et al., 2020; Hoffmann et al., 2022) for language models has created much excitement for training increasingly larger language models, these models have already consumed a significant fraction of the high-quality (textual) data on the internet. The issue of data exhaustion is an active area of research (Villalobos et al., 2022), especially in light of results showing that repeated training on the same data quickly leads to degeneration (Shumailov et al., 2023). If large language models can effectively learn from data they themselves generate, this could usher in a new era of scaling laws that are solely compute-driven, independent of how much data is available.

Self-training is not a novel concept in AI. For instance, AlphaZero achieved superhuman capabilities in Go, Chess, and Shogi via self-play (Silver et al., 2017). Nevertheless, success with such a process has not yet been demonstrated with language models. Recently, Wei et al. (2022) highlighted the intriguing observation that language models often produce better results when prompted to use *chain-*

Workshop on Instruction Tuning and Instruction Following at NeurIPS 2023.

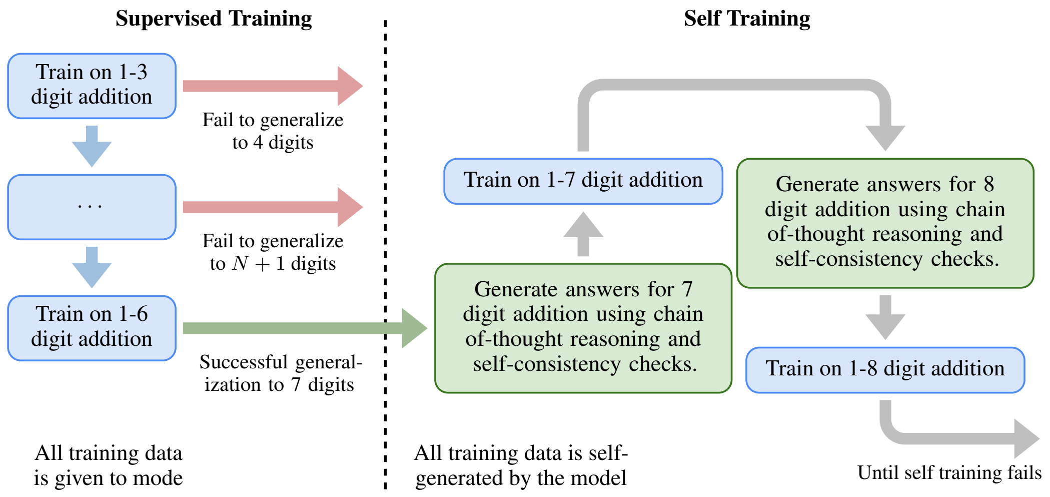

Supervised Training
Self Training
Train on 1-3 digit addition
Fail to generalize to 4 digits
. . .
Fail to generalize to $N + 1$ digits
Train on 1-6 digit addition
Successful generalization to 7 digits
All training data is given to mode
Train on 1-7 digit addition
Generate answers for 7 digit addition using chain of-thought reasoning and self-consistency checks.
Generate answers for 8 digit addition using chain of-thought reasoning and self-consistency checks.
Train on 1-8 digit addition
All training data is self-generated by the model
Until self training fails

Figure 1: SECToR begins with a supervised fine-tuning phase in which the model undergoes training with curriculum learning, mastering simpler addition problems before progressing to more challenging ones. SECToR begins self-training when it generalizes perfectly to $N + 1$ digit addition (after having been trained only on 1 through N digit addition) when equipped with chain-of-thought reasoning. Empirically, this occurred at 7 digits for the 582M parameter model. During self-training, the training process largely mirrors the supervised training phase, except that all training data is autonomously generated by the model using chain-of-thought reasoning and self-consistency checks without any external verification of correctness (see Figure 12).

of-thought reasoning to solve the problem. This approach contrasts with the conventional method where models are prompted to instantly generate an answer, without producing any intermediate steps.

In this paper, we introduce *SECToR* (**S**elf-**E**ducation via **C**hain-of-**Th**ought **R**easoning), which gives a proof-of-concept that large language models can successfully teach themselves new abilities via chain-of-thought reasoning, using addition as a benchmark task. Despite well-known difficulties for language models to perform addition (Liu & Low, 2023; Nye et al., 2021; Lee et al., 2023), language models trained via SECToR teach themselves to add numbers with up to 29-digits *without access to any ground-truth examples* for numbers of length longer than 6 digits.

The observation that lies at the heart of SECToR is that chain-of-thought reasoning can be considered a policy improvement operator: *regardless of the quality* of the underlying model (assuming it satisfies some minimum size and training requirements), models that are prompted to use chain-of-thought reasoning outperform directly sampling an answer from the model. SECToR uses this observation to perform self-learning. During self-learning, SECToR prompts the model to use chain-of-thought reasoning to generate solutions to problems that it could not otherwise solve without such reasoning. Then, this same model is fine-tuned to generate these exact solutions, this time *without using chain-of-thought reasoning*. This leads to an improved version of the model which can now directly solve problems that the previous version of the model required using chain-of-thought reasoning to solve. If this improved model, when again equipped with chain-of-thought reasoning, can solve a larger set of problems than the original model could (even when equipped with chain-of-thought reasoning), the self-learning process can continue. This procedure is highly similar to the procedure that AlphaZero (Silver et al., 2017) used to reach superhuman performance in Chess, Go, and Shogi, except using chain-of-thought reasoning in place of Monte-Carlo Tree Search as the policy improvement operator.

In the specific task of addition, SECToR begins with a pre-trained language model and then performs an initial supervised fine-tuning period in which it learns to perform addition both with and without using chain of thought reasoning but only for addition problems with a small number of digits. After the initial supervised training phase, it then undergoes a self-training phase in which all training data is model-generated, *with zero access to ground-truth data*. During each learning period, we train the language model digit-by-digit using curriculum learning by requiring the model to perform satisfactorily at 1 to N digit addition before introducing $N+1$ digit problems to the dataset. Examples of each of these kinds of tasks are shown in Figure 2. This setup is analogous to prior work such as

AlphaGo (Silver et al., 2016), where the model was initially trained on human-generated data before undergoing self-training.

Task Type	Example
Addition w/o CoT (fast)	Q: 141 + 123 = ? A: 264.
Addition w/ CoT (slow)	Q: 141 + 123 = ? A: The first number's last digit is 1. The second number's last digit is 3. 1 + 3 = 4. The last digit of this sum is 4 so our initial partial answer is 4. We can now recursively solve a smaller problem. Removing the last digit from each number we have 14 and 12. The next subproblem is 14 + 12.

Figure 2: Examples of the two types of addition problems. One can view these two types as the analogues of Type 1 (fast) and Type 2 (slow) reasoning (Kahneman, 2011).

A major challenge that has prevented past efforts of self-learning in language models from succeeding, especially in arithmetic, is a phenomenon that we call *error avalanching*. During self-training, when all training data is generated by the model itself, there is no guarantee that the data is correct. *Error avalanching* occurs when small errors or inaccuracies in a model's output compounds rapidly over time, because each iteration of the model learns from the outputs of the previous model amplifies the existing mistakes. If left unchecked, error avalanching leads to severe degradation of performance within only a few iterations of self-training (see Figure 4). This is consistent with past attempts to get language models to self-learn (for addition as well as other tasks) in which improvement stagnates in only a few steps at most (Zelikman et al., 2022; Lewkowycz et al., 2022; Bai et al., 2022; Huang et al., 2022; Jung et al., 2023). While error avalanching is a fundamental issue in any bootstrapped process, SECToR manages to largely mitigate error avalanching via several forms of self-consistency checks (Figures 4 and 5) that minimize the number of mistakes introduced to the dataset. Nevertheless, SECToR does not continue *ad infinitum*, and training eventually terminates due to accumulated errors. We return to this issue in the discussion.

Results. Our results indicate that the pre-trained 582M parameter ByT5 model (Xue et al., 2022), after a supervised fine-tuning period that only provides examples for 1 to 6 digits, can teach itself to perform up to 29-digit addition with 98+% accuracy – successfully undergoing 22 steps of self-improvement in the process (Table 1). Additionally, a smaller training run with the 300M parameter version of ByT5 showed similar positive results, successfully teaching itself to perform up to 24-digit addition after a supervised learning phase that included examples of up to 8 digits (Appendix E).

2 Approach

2.1 Reasoning as A Policy Improvement Operator

The concept of a policy improvement operator plays a central role in reinforcement learning. In reinforcement learning, a policy is the strategy by which an agent chooses its actions, given the current state of the world. A policy improvement operator is a function that takes in an arbitrary policy and returns an improved policy, which is closer to the optimal policy, with respect to a given reward function. Many reinforcement learning algorithms, such as Q-learning or policy gradients, are built around a policy improvement operator. A crucial property of a policy improvement operator is that it can be used repeatedly —— if one continually applies a policy improvement operator to a policy, it will eventually converge to the environment's optimal policy.

In AlphaZero, Monte-Carlo Tree Search (MCTS) served as the policy improvement operator. Nevertheless, two-player zero-sum games are relatively simple, well-defined environments. In this paper, we explore the hypothesis that *reasoning can serve as a policy improvement operator*, analogous to how MCTS functions in AlphaZero, but for a broader range of environments. To complete the analogy, we consider the language model's conditional distribution over next token as the "policy," the previous context as the "environment." The goal is for each version of the model to generate the data upon which the next version of the model is trained using the policy improvement operator (reasoning).

While large language models have not yet been shown to possess a complete control of general reasoning skills, Wei et al. (2022) showed that a method called "chain-of-thought reasoning" could generate substantial performance improvements merely by asking models to think through a problem step-by-step. In particular, Wei et al. (2022) noticed that often a model *unable to solve a problem* without using reasoning was able to come up with the correct answer if allowed to reason through the problem step-by step. One perspective on chain-of-thought reasoning is that the model is spending computational resources at inference time to augment its own abilities. This is similar to how AlphaZero spends compute to perform MCTS and augment its own game playing abilities. Given that chain-of-thought reasoning can allow models to solve problems that they otherwise could not, it is natural to ask whether repeated application of this method might allow models to self-improve far beyond their original capabilities.

At a high level, SECToR uses chain-of-thought reasoning as a policy improvement operator to assist the model in solving problems that it could not do without using the additional computation. It then constructs a dataset of these new problem, solution pairs, which is then use to train the model to solve these same problems *without using chain-of-thought reasoning*. The hope is that, having learned to directly solve problems that the previous version of the model could only tackle with the additional computational power of chain-of-thought reasoning, this improved model will be able to solve an even larger set of problems when again augmented with chain-of-thought reasoning.

2.2 Error Avalanching

The approach of repeatedly training the model on its own utterances usually quickly results in a phenomenon we call error avalanching. Consider a scenario where the model makes a small error in one iteration. When this output is fed back into the model for the next round of training, the error becomes part of the training data. As the model learns from this flawed data, the error may be reinforced rather than corrected. In subsequent iterations, this error may compound leading to increasingly inaccurate outputs. This is akin to an avalanche, where a small disturbance can trigger a massive slide.

Because we ask the model to learn without giving it any access to the ground truth, it is important to prevent this snowballing of errors, eventually derailing the training process. In past attempts to use similar self-learning approaches, this phenomenon caused the training process to go awry within just a few steps (Zelikman et al., 2022; Jung et al., 2023; Bai et al., 2022; Huang et al., 2022). In order to mitigate the effects of error avalanching, SECToR utilizes several consistency checks largely based on the following idea: we ask the model to generate a large number of "equivalent" variations of any given question and then ask the model to (independently) solve each variation. If the answers are largely consistent, we accept these answers into the training data. Otherwise, SECToR discards the answers. These consistency checks are essential for SECToR to minimize the introduction of incorrect data into the model's self-training loop.

3 Experiments

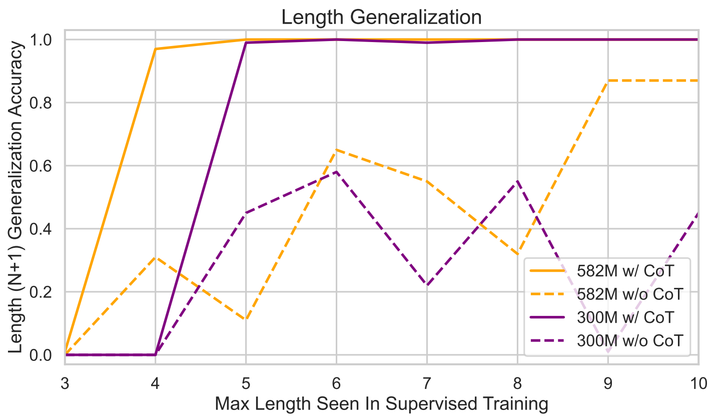


Figure 3: This figure describes the generalization accuracy of models to $N + 1$ digit addition after only training on up to N digit addition. Consistent with past work, we find that models show poor length generalization on standard (fast) addition. However, when models use chain-of-thought reasoning, generalization reaches almost perfect accuracy after training on only a few digits.

3.1 Setup

Addition is a fundamental task in mathematics, but one on which language models have historically struggled to perform. As such, we use it as a toy dataset for evaluating self-learning. As addition can be considered a simple form of problem-solving, a language model being able to teach itself addition may indicate potential for applying general logic and reasoning to solve more complex problems.

We ask whether models can teach themselves how to add very large numbers after having been shown only addition examples with substantially smaller numbers. In our setup, we consider only problems of the form $a + b$ or $a + b + 1$, where a and b are guaranteed to have the same number of digits. a and b are uniformly sampled among all integers with the proper number of digits. We train all models on a cluster of 8 V100 GPUs with 32 GB of memory each. Prior work has indicated the importance of proper tokenization for performing arithmetic (Liu & Low, 2023; Zhou et al., 2022). In order to avoid any such issues, we use the ByT5 family, which operates on the raw character/byte value. A detailed description of the hyperparameters used in training can be found in Appendix C.

3.2 Supervised Training

Without fine tuning, we find that the ByT5 models are quite poor at addition (Appendix D), which is consistent with prior work on benchmarking addition. SECToR thus begins by performing an initial supervised fine-tuning phase consisting of addition problems with only a small number of digits before beginning the self-training loop. This process is described in Figure 1.

During this supervised training period, models are trained to perform addition both with and without chain-of-thought reasoning. In contrast to the self-learning phase of training, all training examples are generated programmatically by an external script during supervised learning. “Fast” addition consists of asking models to immediately output the solution to the addition problem without using any intermediate tokens to reason through the answer. In contrast, “slow” addition consists of asking models to do a single simplification step of turning an N digit problem into an $N - 1$ digit problem and a partial solution, similar to the method of teaching addition to schoolchildren. In this setup, performing “slow” addition refers do performing a single step of simpliciation, not fully solving the problem. Figure 2 depicts examples of both “fast” and “slow” addition. Training examples for each type are prefixed with a special token depicting their type, so the model can differentiate between the two. Learning both how to add numbers directly as well as via chain-of-thought addition can be viewed as a similar idea to that of *process supervision* (Lightman et al., 2023), which recently achieved state-of-the-art performance on the MATH dataset (Hendrycks et al., 2021).

The supervised training phase follows a curriculum learning schedule where a model must first achieve sufficient accuracy on 1 through N digit addition before $N + 1$ digit examples are added to the dataset, starting with $N = 3$. Accuracy is measured by computing an exact token match between the gold reference text and the model output while sampling at temperature 0. Any answer that is not in the correct format is automatically marked as incorrect. To prevent catastrophic forgetting (McCloskey & Cohen, 1989), we mix examples from 1 through N digit addition while training the model on $N + 1$ digit addition. Details of the precise composition of training examples are provided in Section C. In Appendix K, we run an ablation to measure the effects of the curriculum as compared to simply doing a single supervised tuning phase where we learn 1 to N digit addition jointly.

The supervised phase of training concludes when models exhibit length generalization to $N + 1$ digit addition problems when performing “slow”, chain-of-thought augmented addition. In Appendix J, we describe experiments regarding emergent properties of language models, suggesting that larger models need a shorter supervised training period before exhibiting such generalization. This leads to a hypothesis that a sufficiently large pre-trained language model might be able to forgo the supervised training period entirely and begin self-training immediately, perhaps with only a few examples of in-context demonstrations.

3.3 Self-Training

The second period of learning consists of self-training. The distinguishing feature of self-learning is that *all new training data in this phase is generated by the model itself without any external verification of correctness.* Aside from this important difference, self-training largely follows a similar setup to that of the supervised learning period. Here, the goal is to ask whether the model can

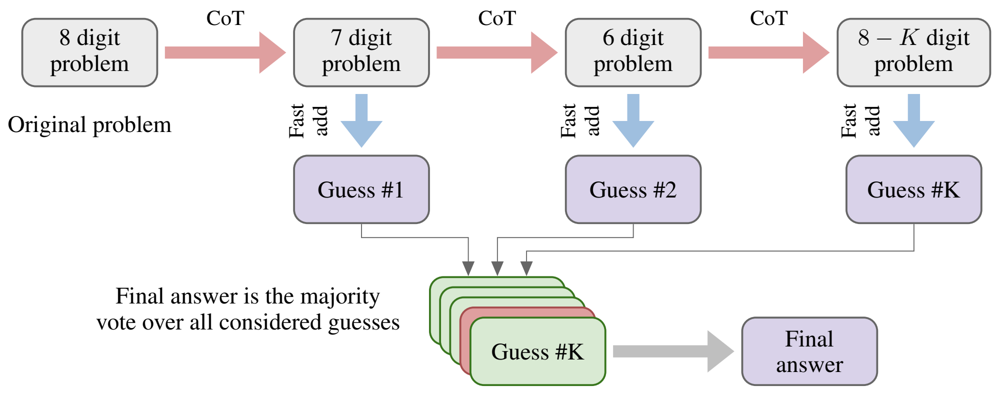


Figure 4: Simplify-then-guess asks the model to simplify the problem K times. After each simplification, the model directly guesses the final solution without any further reasoning steps. These guesses are then aggregated via majority vote to produce the final answer. Simplify-then-guess allows the model to balance the generalization ability of slow, chain-of-thought reasoning, with a consistency check of having K separate guesses of the answer.

continue to teach itself skills even in the absence of human demonstrations, as is often the case in reinforcement learning setups. However, unlike standard reinforcement learning, SECToR does not give the model the privilege of querying the environment: models must learn entirely based on their own conceptions of the environment without any grounding in the real world.

3.3.1 Generating New Addition Examples

The self-training phase requires the model being able to generate both "fast" and "slow" styles of addition for numbers larger than it has seen in training. This is possible largely due to the observation that the reasoning capabilities of models generalize exceptionally well to longer addition lengths beyond what was seen during training. Figure 3 shows that the 582M parameter model begins to generalize to novel digit lengths after training only on 1 through 4 digit addition and reaches almost perfect generalization accuracy to $N + 1$ digit addition after training on 1 through 6 digit addition. Similar results can be obtained for the 300M parameter model, albeit with a longer supervised training period (see Appendix J for a longer discussion on emergence). This generalization capabilities form the heart of how language models are able to perform self-learning with SECToR.

As described in Section 3.2, the curriculum learning setup requires that models successfully learn both fast and slow styles of 1 through N digit addition before engaging in learning $N + 1$ digit addition. Additionally, the precondition for ending the supervised phase of learning and beginning self-training is that models achieve near-perfect generalization accuracy on slow, chain-of-thought augmented $N + 1$ digit addition. This precondition immediately suggests a method of generating training examples for "slow" $N + 1$ digit addition: simply directly sampling from the model using greedy decoding. Generating "fast" style training examples is a more complex process. Figure 3 shows that longer after models generalize using chain-of-thought, generalization without such reasoning remains poor. To generate solutions to $N+1$ digit addition problems, SECToR uses a novel decoding method called *simplify-then-guess* which utilizes a model's abilities to perform both fast and slow addition for 1 through N digit addition (Figure 4). *Simplify-then-guess* is inspired by the approaches of least-to-most prompting (Zhou et al., 2023) and self-consistency (Wang et al., 2023b). In least-to-most prompting, models are prompted to decompose problems into simpler sub-problems before solving each one in sequence. Simplify-then-guess follows a similar process, but adds in a built in self-consistency check inspired by Wang et al. (2023b). After each simplification of the problem, simplify-then-guess asks the model to directly guess the final solution without using any further reasoning steps. The final answer is a majority vote over all intermediate guesses. For example, if a model is tasked with solving an 8-digit addition problem, it will first simplify the problem into a 7 digit addition problem before taking its first guess of a solution. It then repeats this process with the 7-digit problem it just generated, and so on. This process is described in Figure 4.

The primary advantage of simplify-then-guess over least-to-most prompting, is that with least-to-most prompting, a single error at any point in the process corrupts the entire solution. In contrast, simplify-then-guess has a built-in error check in that, because the accuracy of each "guess" is unaffected by any reasoning errors that occur after the guess is made. This error reduction is critical for mitigating error avalanching. In SECToR, simplify-then-guess generates K separate guesses for an addition problem by applying between 1 and K simplification steps before fast adding the remaining addition problem. It then takes a majority vote between the generated guesses to construct a final guess for the answer. In this paper, we use $K = 5$, which was found to be a good balance between computational speed and accuracy.

3.4 Commutativity Checks

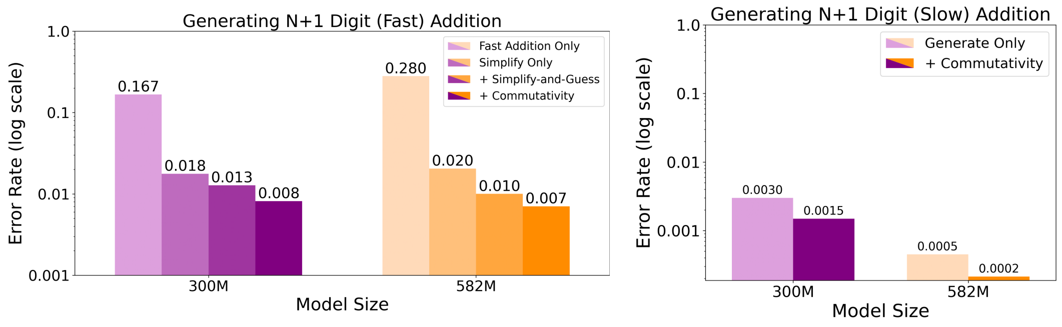


Figure 5: Generating training data for $N + 1$ digit "fast" addition via directly sampling from the model (via Fast Addition) leads to extremely high error rates. Generating training data purely using chain-of-thought reasoning to simplify a problem (Simplify Only) fares better, but not as well as simplify-then-guess, which utilizes a model's ability to perform both fast and slow types of addition. This error is then further reduced by a self-consistency check based on commutativity. Error rate when generating new "slow" addition training examples is substantially lower, both before and after using the commutativity self-consistency check due to the observation that models exhibit strong length generalization when equipped with chain-of-thought reasoning.

Nevertheless, simplify-then-guess alone is not enough to mitigate error avalanching. We perform an additional self-consistency check based on commutativity. Specifically, we ask the model to solve independently (via simplify-then-guess) both a problem $a + b$ and its twin $b + a$. If the generated solutions are not equal, we discard these problems from the dataset. For fast-type addition problems, this check requires that a problem and its twin have answers that are identical, as measured by an exact string match. For slow-type addition problems, this commutativity check is slightly more difficult due to the observation that the chain-of-thought reasoning utterance for $A + B$ is not the same as the answer for $B + A$. Instead, SECToR checks that the answers generated by simplifying for one step, followed by immediately fast adding the subproblem emit identical answers for both a problem and its commutative twin.

3.5 Results

We report the results for a single training run for the 582M parameter model. A similar training run with the 300M parameter model in described in Appendix E. Additional replication experiments are included in Appendix F. The 582M model began self training after 6 digits (Figure 10). Self learning terminated after successfully learning up to 28 digit addition problems, having failed to successfully learn continue the training loop with 29 digit addition. Figure 6 describes the generalization accuracy of the 582M parameter model over the course of training. Table 1 reports the addition accuracies achieved by the final version of the model. We conduct a more careful analysis errors of the model in Appendix I.

Length	1-29	30	31	32	33	34	35+
Accuracy (%)	98-100	88	79	52	26	2	0

Table 1: Accuracy (out of 100 examples) of the final checkpoint of the 582M model after training. For example, this table shows that the post-training 582M model can add 30 digit numbers with 88% accuracy without using any chain-of-thought reasoning.

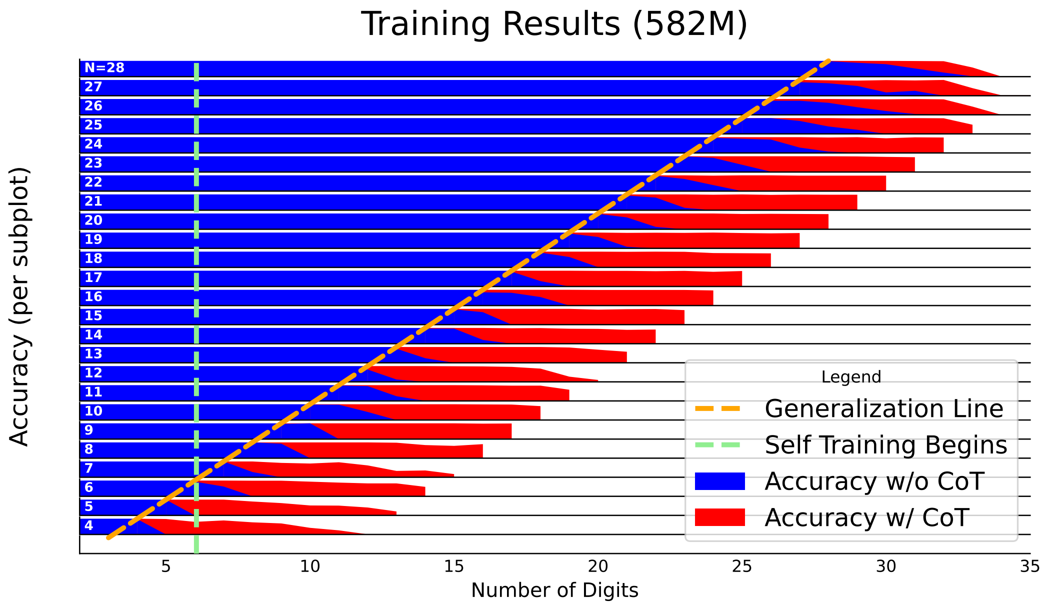


Figure 6: This figure describes the training run of the 582M parameter model. Each row label N denotes a model which has completed training on 1 through N digit addition. The shaded regions depict a model's generalization accuracy beyond N-digits, measured up to a maximum of $N + 8$ digits. Blue represents accuracy without using chain-of-thought reasoning, while the red allows it. Model accuracy both with and without chain-of-thought continues to grow after self-training begins at 7 digits, despite no additional external data being provided after that point. By the end of self-training, the model is capable of accurately adding 30 digit numbers even without using chain-of-thought reasoning.

4 Related Work

Teaching Transformers Arithmetic. While many attempts have been made to teach large language models to perform addition, to our knowledge, none have demonstrated that language models do so by teaching themselves. Lee et al. (2023) found significant challenges with length generation with transformers with very few parameters. Liu & Low (2023) fine-tuned the LLAMA family of models (Touvron et al., 2023), successfully teaching the models 8-digit addition using a supervised fine-tuning, while methods such as Zhou et al. (2022) try to teach addition via in-context learning instead of fine-tuning. Nye et al. (2021) augments language models with the ability to emit intermediate computations to a temporary "scratchpad," allowing the model to perform up to 10-digit addition, whereas (Jelassi et al., 2023) shows that language models can add better when their architecture is modified to use relative position embeddings.

Self-Learning in Language Models. Prior approaches have been tried to perform self-learning in language models. For example, StAR (Zelikman et al., 2022) generates additional data for training by asking models to give rationales for a correct question-answer pair where none exist. It then trains again on those self-generated rationales and demonstrates improved performance on several reasoning-based benchmarks. Impossible distillation (Jung et al., 2023) takes an off-the-shelf LLM and then distills a high-quality dataset by filtering its generations. It then undergoes a self-distillation phase where it does the same process on its own generations, before undergoing one further phase of self-improvement. Huang et al. (2022) use a similar approach of chain-of-thought reasoning and self-consistency checks to improve at several reasoning datasets using very large models. However, the above methods fail to mitigate *error avalanching* in the training process, resulting in only a few steps of self-improvement before the process terminates.

Many other methods rely on self-improvement via methods other than updating the weights. Methods such as Voyager (Wang et al., 2023a) exploit the powerful in-context learning abilities of large language models to self-learn Minecraft abilities. Reflexion (Shinn et al., 2023) uses a similar approach to improve accuracy on the HumEval coding benchmark. SwiftSage (Lin et al., 2023) uses a similar idea of "fast-and-slow" thinking, but does not perform the self-learning loop of fine-tuning a model on its own generations. RAP (Hao et al., 2023) incorporates search into the decoding process. Self-debug (Chen et al., 2023) teaches a model to debug its own generated coding mistakes.

Error Avalanching. Much work has also been done on the difficulties of self-training, especially in language models. Shumailov et al. (2023) give both theoretical and empirical evidence that models repeatedly trained on their own outputs, without any filtering mechanisms, quickly degenerate into nonsense because of error avalanching. SECToR mitigates this by using chain-of-thought reasoning and self-consistency checks minimize the number of errors that accrue in the training process. Zhang et al. (2023b) coins the term *hallucination snowballing*, which can be viewed as a specific form of *error avalanching* in the context of generating factual content. Dziri et al. (2023) give some theoretical justification that errors may snowball exponentially. Wang et al. (2023b) used self-consistency to improve reasoning by sampling various independent methods of solving the problem before using a majority-vote system among the sampled solutions to construct the final answer.

5 Discussion

We demonstrate that chain-of-thought reasoning can serve as a policy improvement operator and show a proof-of-concept demonstration that language models can teach themselves addition. In contrast to prior efforts in which self-improvement fails after only a few steps at most, models trained with SECToR manage to stay on pace for over twenty steps. Nevertheless, numerous avenues remain unexplored in the context of self-learning with language models.

Limitations. While SECToR demonstrates the possibility of self-learning in addition with language models, it is far from showing that models can self-learn in general. A natural question is whether methods like SECToR can generalize to more complex tasks, such as multiplication or perhaps even general mathematics or programming. Secondly, models trained with SECToR do not improve forever. We speculate that a larger model, or a stronger consistency check, might allow for the models to continue improving beyond 29 digits. Finally, while SECToR is data efficient, it is highly compute inefficient, requiring a large amount of compute to generate the next iteration's training data. We leave the development of more efficient methods for SECToR to future work.

Safety. While SECToR is merely a proof-of-concept demonstration of the possibility of self-learning in language models, this line of research brings both tremendous opportunities as well as potential risks. One risk is that self-learning may amplify preexisting biased or erroneous information in the model during the self-training loop. This is not a concern when considering purely objective domains such as addition, but may be an issue if self-learning is more broadly applied to other domains with less objectivity. Additionally, as models gain proficiency in autonomous learning, the boundaries of their capabilities may become less and less predictable, raising questions of how such models can be controlled and used in a safe manner. Alleviating these concerns is an important direction for future research.

References

Reza Yazdani Aminabadi, Samyam Rajbhandari, Ammar Ahmad Awan, Cheng Li, Du Li, Elton Zheng, Olatunji Ruwase, Shaden Smith, Minjia Zhang, Jeff Rasley, and Yuxiong He. DeepSpeed-Inference: Enabling Efficient Inference of Transformer Models at Unprecedented Scale. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '22, pp. 1–15, Dallas, Texas, November 2022. IEEE Press.

Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, Carol Chen, Catherine Olsson, Christopher Olah, Danny Hernandez, Dawn Drain, Deep Ganguli, Dustin Li, Eli Tran-Johnson, Ethan Perez, Jamie Kerr, Jared Mueller, Jeffrey Ladish, Joshua Landau, Kamal Ndousse, Kamile Lukosuite, Liane Lovitt, Michael Sellitto, Nelson Elhage, Nicholas Schiefer, Noemi Mercado, Nova DasSarma, Robert Lasenby, Robin Larson, Sam Ringer, Scott Johnston, Shauna Kravec, Sheer El Showk, Stanislav Fort, Tamera Lanham, Timothy Telleen-Lawton, Tom Conerly, Tom Henighan, Tristan Hume, Samuel R. Bowman, Zac Hatfield-Dodds, Ben Mann, Dario Amodei, Nicholas Joseph, Sam McCandlish, Tom Brown, and Jared Kaplan. Constitutional AI: Harmlessness from AI Feedback, December 2022. URL `http://arxiv.org/abs/2212.08073`. arXiv:2212.08073 [cs].

Emily M. Bender and Alexander Koller. Climbing towards NLU: On Meaning, Form, and Understanding in the Age of Data. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 5185–5198, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.463. URL `https://aclanthology.org/2020.acl-main.463`.

Yonatan Bisk, Ari Holtzman, Jesse Thomason, Jacob Andreas, Yoshua Bengio, Joyce Chai, Mirella Lapata, Angeliki Lazaridou, Jonathan May, Aleksandr Nisnevich, Nicolas Pinto, and Joseph Turian. Experience Grounds Language. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 8718–8735, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.703. URL `https://aclanthology.org/2020.emnlp-main.703`.

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 H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 1877–1901. Curran Associates, Inc., 2020. URL `https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf`.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching Large Language Models to Self-Debug, April 2023. URL `http://arxiv.org/abs/2304.05128`. arXiv:2304.05128 [cs].

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. PaLM: Scaling Language Modeling with Pathways, October 2022. URL `http://arxiv.org/abs/2204.02311`. arXiv:2204.02311 [cs].

Rémi Coulom. Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. In H. Jaap van den Herik, Paolo Ciancarini, and H. H. L. M. (Jeroen) Donkers (eds.), *Computers and Games*, Lecture Notes in Computer Science, pp. 72–83, Berlin, Heidelberg, 2007. Springer. ISBN 978-3-540-75538-8. doi: 10.1007/978-3-540-75538-8_7.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL `https://aclanthology.org/N19-1423`.

Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang Lorraine Li, Liwei Jiang, Bill Yuchen Lin, Peter West, Chandra Bhagavatula, Ronan Le Bras, Jena D. Hwang, Soumya Sanyal, Sean Welleck, Xiang Ren, Allyson Ettinger, Zaid Harchaoui, and Yejin Choi. Faith and Fate: Limits of Transformers on Compositionality, June 2023. URL `http://arxiv.org/abs/2305.18654`. arXiv:2305.18654 [cs].

Guhao Feng, Bohang Zhang, Yuntian Gu, Haotian Ye, Di He, and Liwei Wang. Towards Revealing the Mystery behind Chain of Thought: A Theoretical Perspective, June 2023. URL `http://arxiv.org/abs/2305.15408`. arXiv:2305.15408 [cs, stat].

Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong, Zhen Wang, Daisy Zhe Wang, and Zhiting Hu. Reasoning with Language Model is Planning with World Model, May 2023. URL `http://arxiv.org/abs/2305.14992`. arXiv:2305.14992 [cs].

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. *NeurIPS*, 2021.

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, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. Training Compute-Optimal Large Language Models, March 2022. URL `http://arxiv.org/abs/2203.15556`. arXiv:2203.15556 [cs].

Jiaxin Huang, Shixiang Shane Gu, Le Hou, Yuexin Wu, Xuezhi Wang, Hongkun Yu, and Jiawei Han. Large Language Models Can Self-Improve, October 2022. URL `http://arxiv.org/abs/2210.11610`. arXiv:2210.11610 [cs].

Samy Jelassi, Stéphane d'Ascoli, Carles Domingo-Enrich, Yuhuai Wu, Yuanzhi Li, and François Charton. Length Generalization in Arithmetic Transformers, June 2023. URL `http://arxiv.org/abs/2306.15400`. arXiv:2306.15400 [cs].

Jaehun Jung, Peter West, Liwei Jiang, Faeze Brahman, Ximing Lu, Jillian Fisher, Taylor Sorensen, and Yejin Choi. Impossible Distillation: from Low-Quality Model to High-Quality Dataset & Model for Summarization and Paraphrasing, May 2023. URL `http://arxiv.org/abs/2305.16635`. arXiv:2305.16635 [cs].

Daniel Kahneman. *Thinking, Fast and Slow*. Thinking, Fast and Slow. Farrar, Straus and Giroux, New York, NY, US, 2011. ISBN 978-0-374-27563-1 978-1-4299-6935-2. Pages: 499.

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, January 2020. URL `http://arxiv.org/abs/2001.08361`. arXiv:2001.08361 [cs, stat].

Diederik Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *International conference on learning representations (ICLR)*, San Diego, CA, USA, 2015. tex.optmonth: 12.

Nayoung Lee, Kartik Sreenivasan, Jason D. Lee, Kangwook Lee, and Dimitris Papailiopoulos. Teaching Arithmetic to Small Transformers, July 2023. URL `http://arxiv.org/abs/2307.03381`. arXiv:2307.03381 [cs].

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, Yuhuai Wu, Behnam Neyshabur, Guy Gur-Ari, and Vedant Misra. Solving Quantitative Reasoning Problems with Language Models, June 2022. URL http://arxiv.org/abs/2206.14858. arXiv:2206.14858 [cs].

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's Verify Step by Step, May 2023. URL http://arxiv.org/abs/2305.20050. arXiv:2305.20050 [cs].

Bill Yuchen Lin, Yicheng Fu, Karina Yang, Prithviraj Ammanabrolu, Faeze Brahman, Shiyu Huang, Chandra Bhagavatula, Yejin Choi, and Xiang Ren. SwiftSage: A Generative Agent with Fast and Slow Thinking for Complex Interactive Tasks, May 2023. URL http://arxiv.org/abs/2305.17390. arXiv:2305.17390 [cs].

Tiedong Liu and Bryan Kian Hsiang Low. Goat: Fine-tuned LLaMA Outperforms GPT-4 on Arithmetic Tasks, May 2023. URL http://arxiv.org/abs/2305.14201. arXiv:2305.14201 [cs].

Gary Marcus. Deep Learning: A Critical Appraisal, January 2018. URL http://arxiv.org/abs/1801.00631. arXiv:1801.00631 [cs, stat].

Michael McCloskey and Neal J. Cohen. Catastrophic Interference in Connectionist Networks: The Sequential Learning Problem. In Gordon H. Bower (ed.), *Psychology of Learning and Motivation*, volume 24, pp. 109–165. Academic Press, January 1989. doi: 10.1016/S0079-7421(08)60536-8. URL https://www.sciencedirect.com/science/article/pii/S0079742108605368.

Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, Charles Sutton, and Augustus Odena. Show Your Work: Scratchpads for Intermediate Computation with Language Models, November 2021. URL http://arxiv.org/abs/2112.00114. arXiv:2112.00114 [cs].

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language Models are Unsupervised Multitask Learners. pp. 24, 2019.

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy Lillicrap, and David Silver. Mastering Atari, Go, Chess, and Shogi by Planning with a Learned Model. *Nature*, 588(7839):604–609, December 2020. ISSN 1476-4687. doi: 10.1038/s41586-020-03051-4. URL https://www.nature.com/articles/s41586-020-03051-4. Number: 7839 Publisher: Nature Publishing Group.

Noah Shinn, Federico Cassano, Beck Labash, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language Agents with Verbal Reinforcement Learning, June 2023. URL http://arxiv.org/abs/2303.11366. arXiv:2303.11366 [cs].

Ilia Shumailov, Zakhar Shumaylov, Yiren Zhao, Yarin Gal, Nicolas Papernot, and Ross Anderson. The Curse of Recursion: Training on Generated Data Makes Models Forget, May 2023. URL http://arxiv.org/abs/2305.17493. arXiv:2305.17493 [cs].

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the Game of Go with Deep Neural Networks and Tree Search. *Nature*, 529(7587):484–489, January 2016. ISSN 0028-0836, 1476-4687. doi: 10.1038/nature16961. URL http://www.nature.com/articles/nature16961.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, and others. Mastering the Game of Go Without Human Knowledge. *Nature*, 550(7676):354–359, 2017. Publisher: Nature Publishing Group.

Ronen Tamari, Chen Shani, Tom Hope, Miriam R L Petruck, Omri Abend, and Dafna Shahaf. Language (Re)modelling: Towards Embodied Language Understanding. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 6268–6281, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.559. URL `https://aclanthology.org/2020.acl-main.559`.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open Foundation and Fine-Tuned Chat Models, July 2023. URL `https://arxiv.org/abs/2307.09288v2`.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need, August 2023. URL `http://arxiv.org/abs/1706.03762`. arXiv:1706.03762 [cs].

Pablo Villalobos, Jaime Sevilla, Lennart Heim, Tamay Besiroglu, Marius Hobbhahn, and Anson Ho. Will We Run Out of Data? An Analysis of the Limits of Scaling Datasets in Machine Learning, October 2022. URL `https://arxiv.org/abs/2211.04325v1`.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An Open-Ended Embodied Agent with Large Language Models, May 2023a. URL `http://arxiv.org/abs/2305.16291`. arXiv:2305.16291 [cs].

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-Consistency Improves Chain of Thought Reasoning in Language Models, March 2023b. URL `http://arxiv.org/abs/2203.11171`. arXiv:2203.11171 [cs].

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. arXiv, 2022. doi: 10.48550/arXiv.2201.11903. URL `http://arxiv.org/abs/2201.11903`. arXiv:2201.11903 [cs].

Yiran Wu, Feiran Jia, Shaokun Zhang, Hangyu Li, Erkang Zhu, Yue Wang, Yin Tat Lee, Richard Peng, Qingyun Wu, and Chi Wang. An Empirical Study on Challenging Math Problem Solving with GPT-4, June 2023. URL `http://arxiv.org/abs/2306.01337`. arXiv:2306.01337 [cs, stat].

Linting Xue, Aditya Barua, Noah Constant, Rami Al-Rfou, Sharan Narang, Mihir Kale, Adam Roberts, and Colin Raffel. ByT5: Towards a token-free future with pre-trained byte-to-byte models, March 2022. URL `http://arxiv.org/abs/2105.13626`. arXiv:2105.13626 [cs].

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of Thoughts: Deliberate Problem Solving with Large Language Models, May 2023. URL `http://arxiv.org/abs/2305.10601`. arXiv:2305.10601 [cs].

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. STaR: Bootstrapping Reasoning With Reasoning, May 2022. URL `http://arxiv.org/abs/2203.14465`. arXiv:2203.14465 [cs].

Mengxue Zhang, Zichao Wang, Zhichao Yang, Weiqi Feng, and Andrew Lan. Interpretable Math Word Problem Solution Generation Via Step-by-step Planning, June 2023a. URL `http://arxiv.org/abs/2306.00784`. arXiv:2306.00784 [cs].

Muru Zhang, Ofir Press, William Merrill, Alisa Liu, and Noah A. Smith. How Language Model Hallucinations Can Snowball, May 2023b. URL `http://arxiv.org/abs/2305.13534`. arXiv:2305.13534 [cs].

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, and Ed Chi. Least-to-Most Prompting Enables Complex Reasoning in Large Language Models, April 2023. URL `http://arxiv.org/abs/2205.10625`. arXiv:2205.10625 [cs].

Hattie Zhou, Azade Nova, Hugo Larochelle, Aaron Courville, Behnam Neyshabur, and Hanie Sedghi. Teaching Algorithmic Reasoning via In-context Learning, November 2022. URL `http://arxiv.org/abs/2211.09066`. arXiv:2211.09066 [cs].

Supplementary Materials

A Approach

A.1 Reasoning as A Policy Improvement Operator

The concept of a policy improvement operator plays a central role in reinforcement learning. In reinforcement learning, a policy is the strategy by which an agent chooses its actions, given the current state of the world. A policy improvement operator is a function that takes in an arbitrary policy and returns an improved policy, which is closer to the optimal policy, with respect to a given reward function. Many reinforcement learning algorithms, such as Q-learning or policy gradients, are built around a policy improvement operator. A crucial property of a policy improvement operator is that it can be used repeatedly —— if one continually applies a policy improvement operator to a policy, it will eventually converge to the environment's optimal policy.

In AlphaZero, Monte-Carlo Tree Search (MCTS) served as the policy improvement operator. Nevertheless, two-player zero-sum games are relatively simple, well-defined environments. In this paper, we explore the hypothesis that *reasoning can serve as a policy improvement operator*, analogous to how MCTS functions in AlphaZero, but for a broader range of environments. To complete the analogy, we consider the language model's conditional distribution over next token as the "policy," the previous context as the "environment." The goal is for each version of the model to generate the data upon which the next version of the model is trained using the policy improvement operator (reasoning).

While large language models have not yet been shown to possess a complete control of general reasoning skills, Wei et al. (2022) showed that a method called "chain-of-thought reasoning" could generate substantial performance improvements merely by asking models to think through a problem step-by-step. In particular, Wei et al. (2022) noticed that often a model *unable to solve a problem* without using reasoning was able to come up with the correct answer if allowed to reason through the problem step-by step. One perspective on chain-of-thought reasoning is that the model is spending computational resources at inference time to augment its own abilities. This is similar to how AlphaZero spends compute to perform MCTS and augment its own game playing abilities. Given that chain-of-thought reasoning can allow models to solve problems that they otherwise could not, it is natural to ask whether repeated application of this method might allow models to self-improve far beyond their original capabilities.

At a high level, SECToR uses chain-of-thought reasoning as a policy improvement operator to assist the model in solving problems that it could not do without using the additional computation. It then constructs a dataset of these new problem, solution pairs, which is then use to train the model to solve these same problems *without using chain-of-thought reasoning.* The hope is that, having learned to directly solve problems that the previous version of the model could only tackle with the additional computational power of chain-of-thought reasoning, this improved model will be able to solve an even larger set of problems when again augmented with chain-of-thought reasoning.

A.2 Error Avalanching

The approach of repeatedly training the model on its own utterances usually quickly results in a phenomenon we call error avalanching. Consider a scenario where the model makes a small error in one iteration. When this output is fed back into the model for the next round of training, the error becomes part of the training data. As the model learns from this flawed data, the error may be reinforced rather than corrected. In subsequent iterations, this error may compound leading to increasingly inaccurate outputs. This is akin to an avalanche, where a small disturbance can trigger a massive slide.

Because we ask the model to learn without giving it any access to the ground truth, it is important to prevent this snowballing of errors, eventually derailing the training process. In past attempts to use similar self-learning approaches, this phenomenon caused the training process to go awry within just a few steps (Zelikman et al., 2022; Jung et al., 2023; Bai et al., 2022; Huang et al., 2022). In order to mitigate the effects of error avalanching, SECToR utilizes several consistency checks largely based on the following idea: we ask the model to generate a large number of "equivalent" variations of any given question and then ask the model to (independently) solve each variation. If the answers are largely consistent, we accept these answers into the training data. Otherwise, SECToR discards the answers. These consistency checks are essential for SECToR to minimize the introduction of incorrect data into the model's self-training loop.

B Additional Related Work

B.1 Chain Of Thought Reasoning

Chain of thought reasoning was introduced by Wei et al. (2022) as a method of improving performance in solving reasoning problems. Since then, this method has been expanded upon and found to improve performance across many domains (Wu et al., 2023; Zhang et al., 2023a; Feng et al., 2023; Yao et al., 2023; Lightman et al., 2023).

B.2 Self-Training

AlphaZero was one of the original works demonstrating the concept of self-learning, whereupon the training process employed Monte-Carlo Tree Search (MCTS) as a policy improvement operator. Specifically, AlphaZero used MCTS to improve upon the original policy of the game, before distilling this improved policy into the original policy. Coulom (2007) proved that, regardless of the quality of the original policy, running MCTS would provide a policy that was closer to the Nash equilibrium of the game. However, AlphaZero required access to a perfect simulator of the environment in order to perform MCTS. AlphaZero's followup work, MuZero (Schrittwieser et al., 2020), removed the requirement of direct access to a simulator by learning a model of the world, but MuZero still required continual query access to the true simulator to ensure that this world model remained accurate during training. While a simulator is easily accessible for such small, constrained environments, it is often not possible for general domains in the real world, where environments may be ill-defined or otherwise difficult to accurately simulate (e.g. real-world robotics, writing a Pulitzer Prize winning novel). It remains an open question whether an analogue policy improvement operator exists for a broader class of environments. Additionally, while MCTS is a powerful tool for learning in board games, it is not a general policy improvement operator. SECToR builds upon this prior work and can be used to train models to perform up to 30-digit addition without *any* queries to the ground truth world model after the initial self-training period.

C Hyperparameters

All models are fine-tuned using the Adam optimizer (Kingma & Ba, 2015) and the DeepSpeed library (Aminabadi et al., 2022) with a constant learning rate of 10^{-4}. We used a batch size of 2048 for our experiments with the 300M parameter model and 1024 for the 582M parameter model, which were the maximum possible sizes that fit in memory given our computational resources. Furthermore, we used 16-bit training via bfloat16 to conserve memory. The entire training process for the 582M parameter model took around 24 hours on a cluster of 8 V100s.

During the supervised training phase, when learning how to add 1 through N digits, we generated 10000 unique examples for N-digit chain-of-thought addition and 1000 unique examples for chain-of-thought addition for each digit from 1 - N to reduce catastrophic forgetting. For fast addition problems, we would generate 30000 unique examples for N digit addition and 3000 unique examples for all smaller numbers.

During the self-training phase, all numbers were reduced by a factor of 10, since it was substantially more costly to generate training data during this regime. If there are not enough unique training examples to satisfy these conditions[1], then we duplicate the problems until there are a minimum of $\frac{1}{3}$ of the size of the data otherwise required.

Models were allowed to proceed to $N + 1$ digit addition when they have achieved sufficient performance on 1 through N digit addition. Satisfactory performance is defined as achieving at least 75% accuracy length N addition problems without using chain-of-thought reasoning and 100% accuracy (on length N addition problems) when using chain-of-thought reasoning on a held out test set of size 128 of each type of problem. SECToR does not require perfect accuracy on fast addition problems because SECToR's built in self-consistency checks are more robust to errors of this kind than with errors in the chain-of-thought reasoning process.

[1]There are only 100 unique 1-digit addition problems.

D Out of Box Accuracy of ByT5 models on Addition

To defend against the possibility that the ByT5 models secretly already know how to perform addition before SECToR, we evaluate the out-of-box accuracy of these models on simple addition. Table 2 shows that models have little-to-no ability to perform addition out of the box.

Model Size \ Addition Length	**1**	**2**	**3**	**4**
300M	0.015	0.004	0.0	0.0
582M	0.04	0.005	0.002	0.0
1.23B	0.0	0.005	0.0	0.0
3.74B	0.0	0.002	0.0	0.0

Table 2: Out-of-the-box addition accuracy (with no fine-tuning) of the ByT5 Xue et al. (2022) family of models. All accuracies are measured using 1000 randomly generated addition problems.

All models were prompted with 10 correct examples of addition with the specified number of digits before being asked to complete the 11th problem. An example prompt is pasted in its entirety below.

```
10 + 97 = 107
17 + 82 = 99
21 + 68 = 89
35 + 29 = 64
75 + 68 = 143
10 + 28 = 38
48 + 60 = 108
88 + 46 = 134
83 + 49 = 132
11 + 62 = 73
25 + 24 =
```

E Results with the 300M ByT5 Model

The 300M model began self training after 8 digits. The training run is described in Figure 11. Its last saved checkpoint was after successfully learning up to 24 digit addition problems, having failed to successfully learn continue the training loop with 25 digit addition. Figures 7 and 8 describe the generalization accuracies of the 300M parameter model over the course of training. Table 3 reports the addition accuracies achieved by the final version of the model. Note that even though the model was unable to continue training on the 25-digit iteration of learning, its previous checkpoint is still able to do addition problems larger than 24 digits with reasonable accuracy.

Length	1-19	20	21	22	23	24	25	26	27+
Accuracy (%)	98-100	95	98	98	99	98	91	33	0

Table 3: Accuracy (out of 100 examples) of the final checkpoint of the 300M model after training. For example, this table shows that the post-training 300M model can add 24 digit numbers with 98% accuracy without any chain-of-thought reasoning.

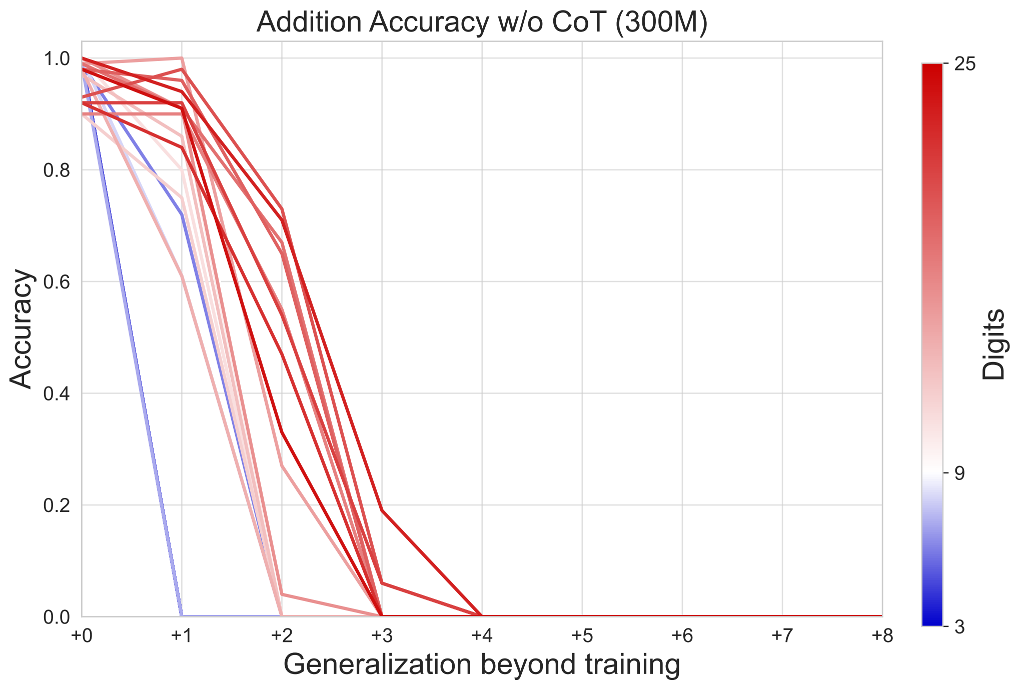


Figure 7: This figure describes the generalization accuracy of the model's addition capabilities over the course of training over the training run of the 300M model. Blue lines indicate the supervised training phase, while red lines indicate the self-training phase. We can see that even at the end of training, models do not show much generalization in their addition capabilities without using chain of thought.

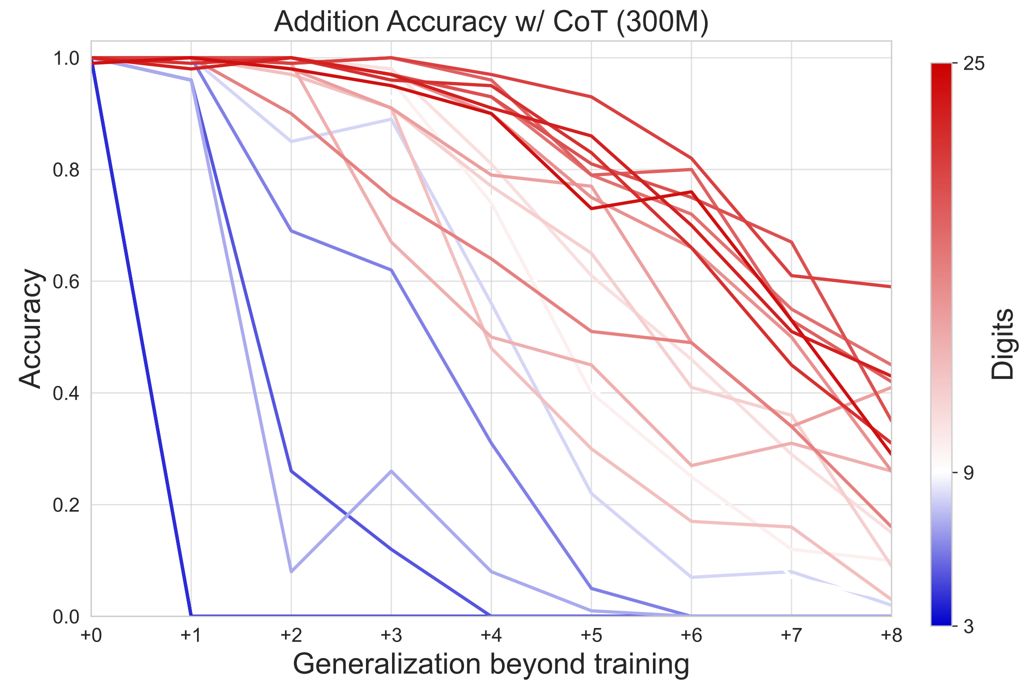
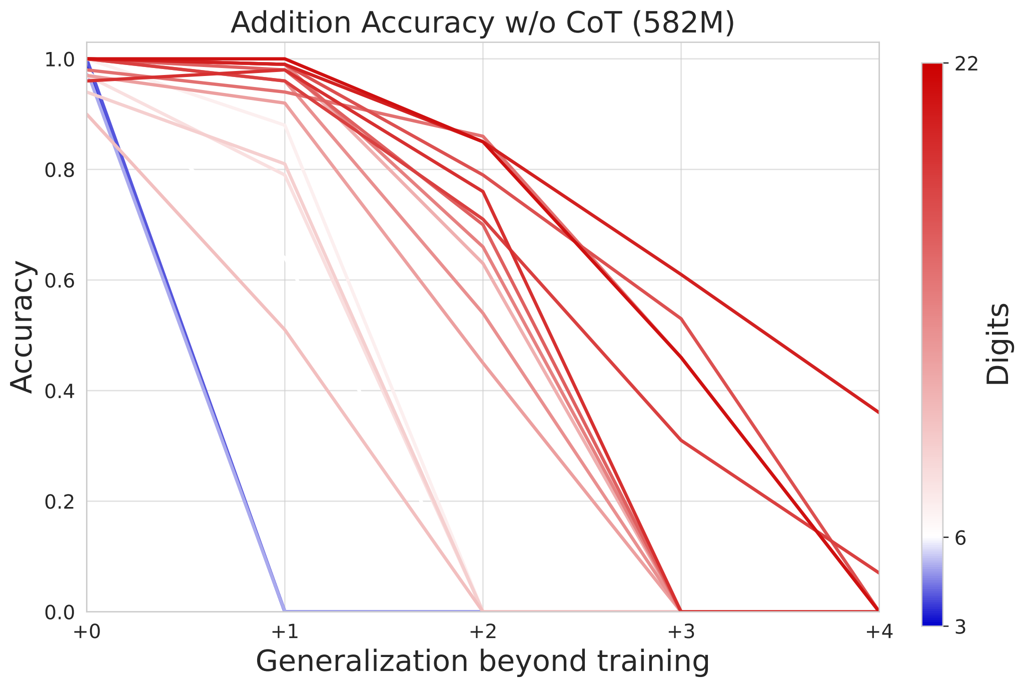
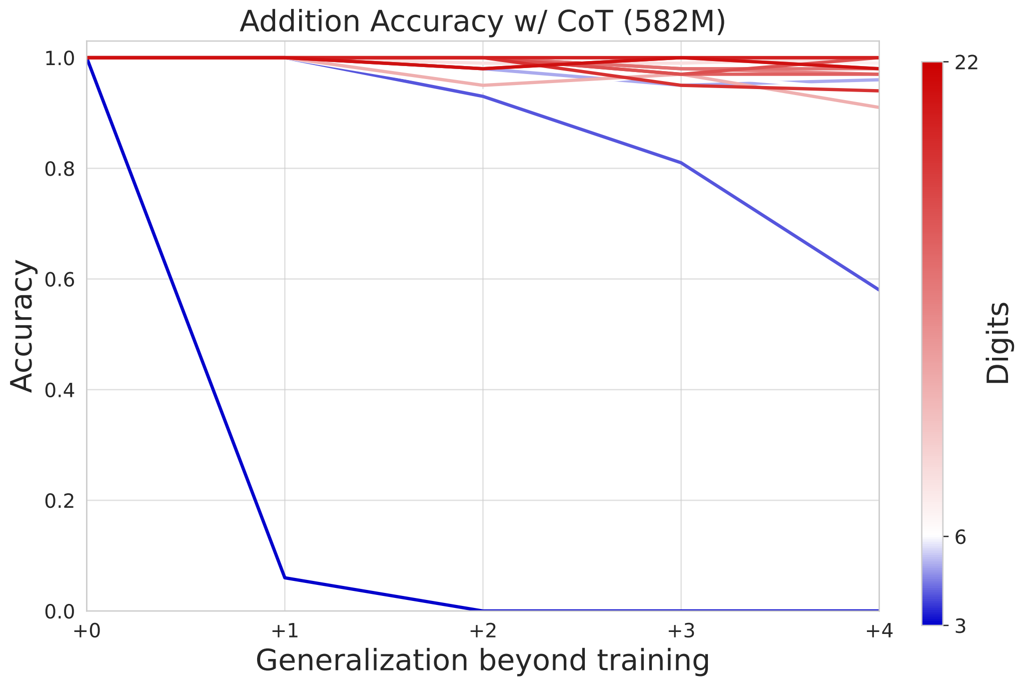


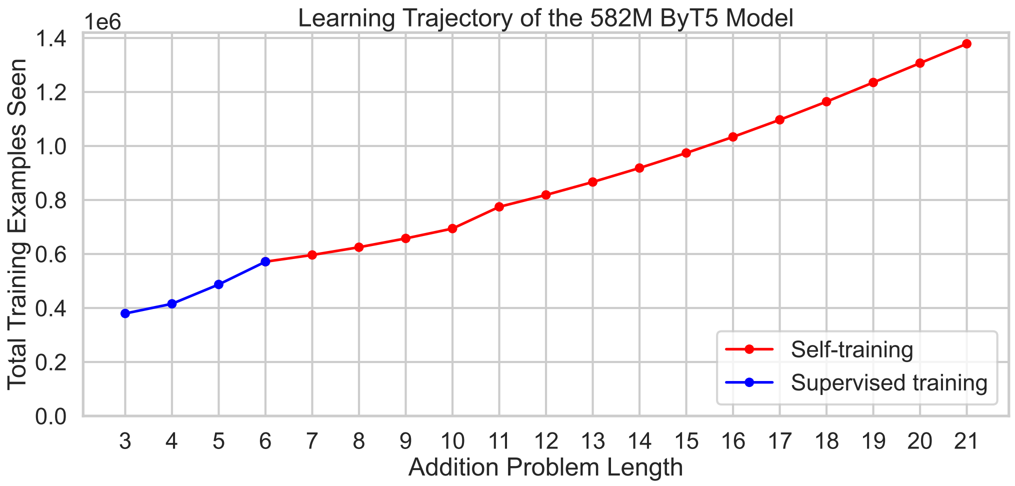
Figure 8: This figure describes the generalization accuracy of the model's addition capabilities over the course of training over the training run of the 300M model. Blue lines indicate the supervised training phase, while red lines indicate the self-training phase. While initially, models show little generalization to lengths greater than than what they have seen in training, models quickly learn to generalize well beyond their training distribution using chain-of-thought, allowing them to continue teaching themselves addition problems beyond what they have seen before.

F Control Experiments

A control experiment with the 582M parameter model had a supervised training phase of 1 through 5 digits and a self-learning phase of 6 through 21 digits. The training run is depicted in Figures **??**, **??** and **??**.






Learning Trajectory of the 582M ByT5 Model
1e6
Total Training Examples Seen
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
Addition Problem Length
Self-training
Supervised training

G Similarities Between AlphaZero's MCTS and Simplify-then-guess

Simplify-then-guess also has spiritual similarities to how AlphaZero often does not perform MCTS to the end of the game, instead terminating search after a certain depth and returning the output of the value network. The analogy for simplify-then-guess is that SECToR runs K steps of simplification (i.e. search) before taking a guess at the answer (i.e. querying the value network) instead of simplifying all the way to 1-digit addition (i.e. running MCTS to the end of the game). This takes direct advantage of the inductive curriculum-style training in which a model learns to add 1 to N digit numbers before being asked to generate training data for $N + 1$ addition numbers.

H Future Directions

H.1 Self-learning vs. Learning to Self-Learn.

While SECToR is a process by which models teach themselves new concepts, they arguably do not *learn* to teach themselves new concepts. In our experiments, SECToR provides scaffolding around the model which, while never performing any aspect of addition, assists the language model in learning. For example, SECToR stops fine tuning once accuracy hits a certain threshold and this threshold was not chosen by the model itself. One can imagine using ideas from recent work, such as Toolformer, to give a language model access to tools or API commands that allow it to fine tune itself and add datapoints to the dataset. If successful, one could imagine a process analogous to SECToR, except that the process would consist solely of sampling from the model and executing the generated commands with no additional assistance. If so, one could imagine a model learning to teach itself a wide variety of concepts, including those in which it has to self-discover the learning process, as well as the new concepts itself.

H.2 Grounding.

Grounding is an area of active discussion in the research community, with many criticizing language models for their perceived lack of grounding in the real world. Some have suggested that future models may need to be embodied to effectively learn in the real world (Marcus, 2018; Bender & Koller, 2020; Tamari et al., 2020; Bisk et al., 2020). We believe our results with SECToR raise the possibility that large language models can succeed without grounding in domains that benefit from the existence of very strong self-consistency checking (e.g., mathematics, programming, etc.). While the model in the present paper is arguably grounded in true arithmetic during the supervised training phase, during the self-training phase, of which the majority of training occurs, models trained with SECToR receive no information from the external world and are, in this sense, ungrounded. Nevertheless, they manage to teach themselves addition problems that are orders of magnitude larger than they have ever seen during supervised fine-tuning. Might it be possible for models to bootstrap their learning in other domains without access to an incremental source of external signal or grounding?

I Error Analysis

In this section, we examine what types of errors the models make on addition. We evaluate the final successful model checkpoint of the 582M parameter model on 30 digit addition. Note that as per Section 3.5, this is beyond what the model has ever seen during training, including self-training. Nevertheless, Table 1 suggests that models can generalize to perform such addition even without using chain-of-thought reasoning.

We plot the accuracy of the model on 30 digit addition against the number of carries required to perform such addition. Surprisingly, we notice little correlation between the number of carries a model must perform and its overall accuracy, suggesting that models are not simply learning to solve the "easy" addition problems.

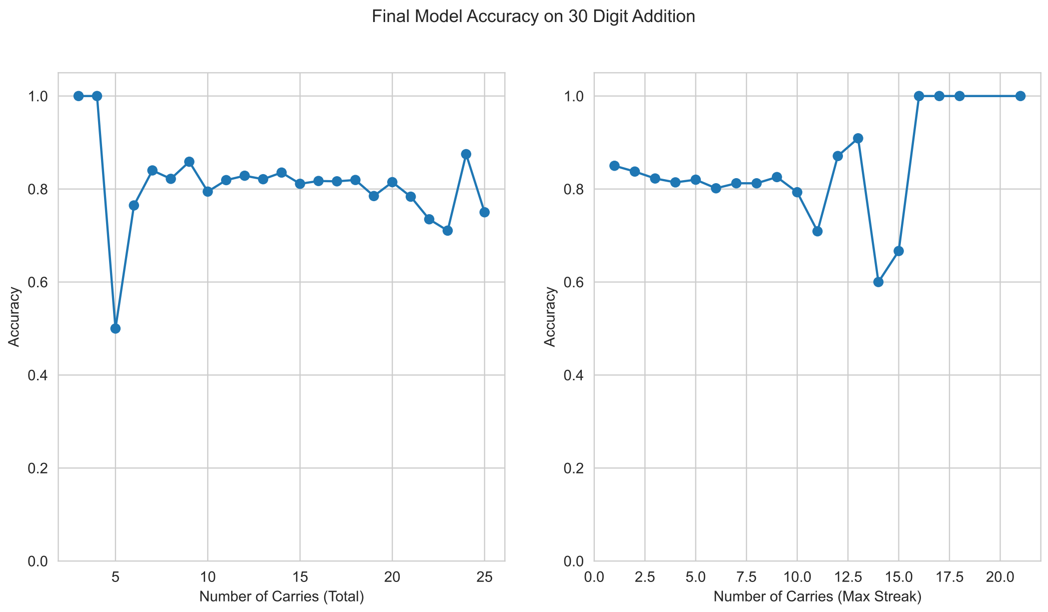


Figure 9: Model accuracy on 30 digit addition against the number of carries required. We observe little relationship between accuracy either the total number of carries required (left) or the longest streak of carries in a problem (right).

J Emergence Experiments

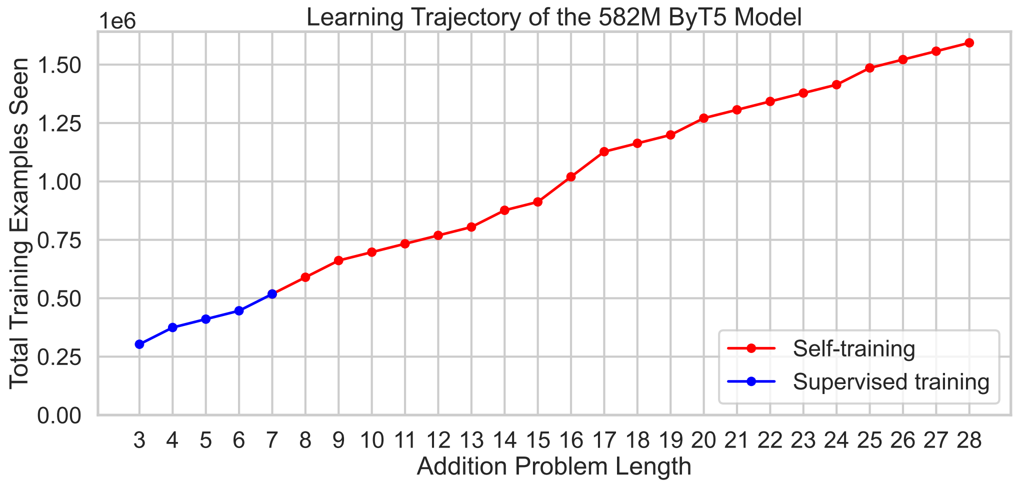


Figure 10: This figure describes how much training data was generated/consumed over the course of the training run of the 582M model.

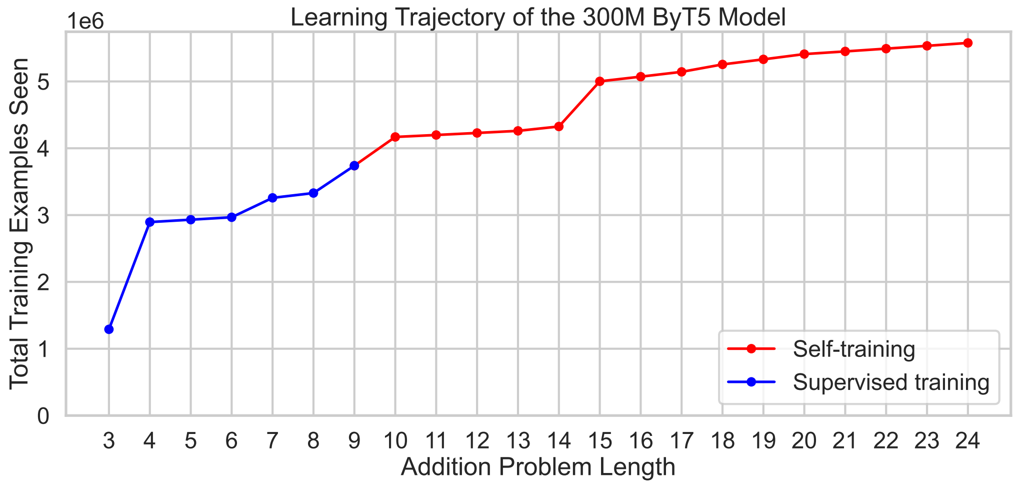


Figure 11: This figure describes how much training data was generated/consumed over the course of the training run of the 300M model.

While the 300M and 582M parameter ByT5 models required an initial supervised learning phase before being able to begin self-learning, we note that there seems to be an inverse correlation between the size of the model and the length of the supervised training period, both in terms of the maximum length of addition problems seen as well as the total training examples. For examples, Figure 11 shows that the 300M model required almost 4 million training examples of up to length 8 before generalizing sufficiently well to begin training. In contrast, Figure 10 shows that the 582M model required only 0.5 million training examples of up to length 6 before generalizing. We hypothesize that a sufficiently large model might be able to forgo the supervised training phase entirely and begin self-training immediately out of the box, possibly with the assistance of in-context learning.

Additionally, while simplify-then-guess outperforms generic "simplification" for generating new $N + 1$ digit examples (Figure 5), we find that "simplify + commutativity" rivals (and sometimes outperforms) simplify-then-guess combined with commutativity in the 582M parameter model, even though it does not in the 300M parameter model. We speculate that this may be due to the errors in the simplification process being less correlated than errors in the simplify-then-guess process, but lead such speculation to future work.

K Importance of Curriculum Learning

A natural question is how important the curriculum learning where the model is required to successfully learning 1 through N digit addition before $N + 1$ digit examples are added in the training step. We run an ablation where we train a 582M parameter ByT5 model on 1 through 6 digit addition in a single supervised fine-tuning step, instead of via the curriculum learning setup done in Section 3.5. We find that this model, when properly trained generalizes to up to 9 digit (slow) addition perfectly, suggesting that curriculum learning is the primary reason why SECToR is able to perform self-learning. Nevertheless, this ablation does not mean that that SECToR is able to run without some form of curriculum learning because a priori, one would have no way of knowing precisely what number of digits were sufficient to generalize other than empirical experiments. Additionally, during the self-training process, curriculum learning is essential by design, as one requires a model capable of performing 1 through N digit addition to generate the training data for $N + 1$ digit addition.

L Additional Figure On Self-Learning

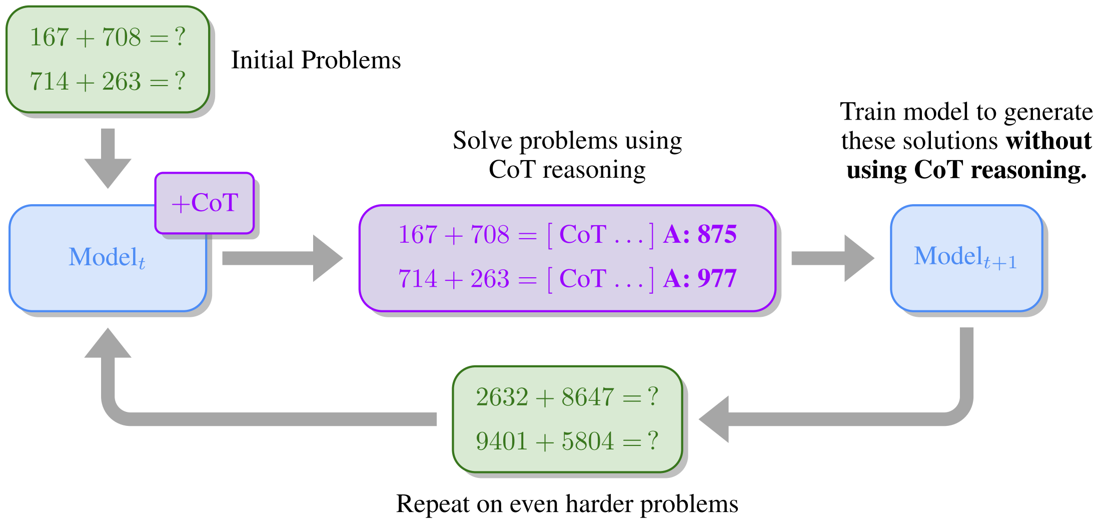

$167 + 708 = ?$
$714 + 263 = ?$
Initial Problems
+CoT
Model$_t$
Solve problems using CoT reasoning
$167 + 708 =$ [CoT . . .] **A: 875**
$714 + 263 =$ [CoT . . .] **A: 977**
Train model to generate these solutions **without using CoT reasoning.**
Model$_{t+1}$
$2632 + 8647 = ?$
$9401 + 5804 = ?$
Repeat on even harder problems

Figure 12: To perform self-learning, SECToR asks models to solve addition problems using chain-of-thought reasoning by decomposing the problem step-by-step. It then trains the next version of the model to solve those same problems directly *without using chain-of-thought reasoning*. This process often results in an improved model which can often solve even harder problems than original model, allowing the self-learning loop to continue.