RANDOMLY SAMPLED LANGUAGE REASONING PROBLEMS REVEAL LIMITS OF LLMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Can LLMs pick up language structure from examples? Evidence in prior work seems to indicate yes, as pretrained models repeatedly demonstrate the ability to adapt to new language structures. However, this line of research typically considers languages that are present within common pretraining datasets, or otherwise share notable similarities with seen languages. In contrast, in this work we attempt to measure models' language understanding capacity while circumventing the risk of dataset recall. We parameterize large families of language tasks recognized by deterministic finite automata (DFAs), and can thus sample novel language reasoning problems to fairly evaluate LLMs regardless of training data. We find that, even in the strikingly simple setting of 3-state DFAs, LLMs underperform unparameterized n-GRAM models on both language recognition and synthesis tasks. These results suggest that LLMs struggle to match the ability of basic language models in recognizing and reasoning over languages that are sufficiently distinct from the ones seen at training time, underscoring the distinction between learning individual languages and possessing a general theory of language.

1 INTRODUCTION

Contemporary LLMs have proven themselves to be highly sophisticated natural language completion models that demonstrate many properties of reasoning engines. This has prompted questions surrounding the true intelligence of these models, with some arguing that they possess inherent language learning capabilities (Millière (2024)). In this paper, we explore the question of whether LLMs have the reasoning capacity to understand the structure of a new language. Specifically, we are interested in problems where a model is given a small set of examples from a language and either generates a new sample or determines whether a new sample is from the language or not.

Some work suggests that LLMs broadly understand language structure because they are able to produce syntactically correct samples from languages they have not been trained on (Athiwaratkun et al. (2022)), although LLMs' performance on low-resource languages tends to be lower than their ability on higher resource languages (Bogin et al. (2023)). On the other hand, some critics of LLMs argue that LLMs cannot possess an understanding of language structure as they have learned from data rather than possessing *a priori* universal grammar (Chomsky et al. (2023)). We do not view learning from data as a fundamental limitation, but we *are* concerned with the possibility that a language model might only be able to understand linguistic structures similar to those it has seen in training data. To distinguish between these possibilities, we wish to evaluate LLMs on a benchmark that considers wholly novel languages, eliminating the possibility of dataset leakage.

Ideally, models would be tested on a set of language reasoning problems disjoint from data seen during training and validation. However, as training datasets for LLMs are generally closed and incredibly vast, human-generated problems in natural language are likely to at least partially overlap in syntax or concept with content LLMs have already seen, making probing a model for its ability to reason about the structure of natural languages nearly impossible. Additionally, the problem of determining whether two tasks are semantically identical is itself a nontrivial one. Therefore, ensuring that even an entirely novel invented problem is not a variation on a theme is intractable.

To circumvent this problem, we propose the following general approach: first we define a large, exhaustive, and parsimoniously-defined space of languages that represents all languages of a certain difficulty level. Then, we sample random languages from this space. By sampling randomly, we

can guarantee no bias towards canonical languages that might share structure with common ones in the training dataset. In this work, we use languages recognized by 3-state DFAs as these are the lowest nontrivial difficulty level, but this technique can be generalized to produce benchmarks of any difficulty level.

Experimental results using this approach suggest that contemporary LLMs possess less sophisticated language pattern recognition abilities than expected; underperforming basic, parameter-free n-gram language models on even the simplest languages. These results, combined with LLMs' impressive results on a variety of specific tasks, suggest that LLMs function as ensemble models over language tasks they have seen in their dataset, but do not possess the ability to generalize to entirely novel language reasoning tasks.

In summary, we make the following contributions:

1. We introduce a benchmark for LLM language reasoning evaluation, disjoint from natural language web data.
2. We evaluate a suite of popular LLMs on instances of this benchmark and demonstrate that LLMs underperform compared to simple language model baselines.
3. We analyze the differences in behavior between these models, illustrating the influence of RLHF and chain-of-thought prompting on language reasoning capacity.

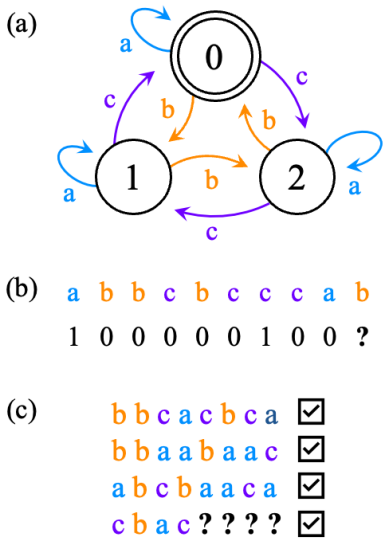


Figure 1: We sample randomly generated languages to test LLMs by sampling deterministic finite automata (DFAs). (a) The DFA shown here, modeling the sum modulo 3 operation (with `abc` representing 0, 1, and 2 respectively), can be used to accept or reject strings from a 3-character alphabet. Accepted strings belong to the grammar, and rejected strings do not. We evaluate models on their ability to (b) act as a transducer, recognizing strings that belong to the DFA-defined grammar, and (c) generate new strings following the grammar.

2 Related Work

2.1 Reasoning with LLMs

Reasoning is one of many "emergent abilities" (Wei et al. (2022a)) possibly possessed by LLMs (Huang & Chang (2022)), although the nonlinear dependence of such emergent abilities on model size is disputed (Schaeffer et al. (2024)). The chain-of-thought prompting technique (Wei et al. (2022b)) has inspired a number of approaches to encourage the latent reasoning ability of models (Yao et al. (2023); Besta et al. (2024); Kojima et al. (2022)), including neuro-symbolic methods (Hua & Zhang (2022); Weir et al. (2023; 2024)). Building on this, other work considers how to optimize exemplars used for in-context learning (Dong et al. (2022)) and chain-of-thought prompting, known as "rationale refinement" (Liu et al. (2021); Fu et al. (2022)). Problem-decomposition is also shown to be effective (Zhou et al. (2022); Khot et al. (2022)).

2.2 LLM reasoning evaluation

LLM reasoning abilities are often tested on natural language benchmarks and commonly seen problems like arithmetic (Cobbe et al. (2021); Amini et al. (2019); Hendrycks et al. (2021)), commonsense reasoning (Bhargava & Ng (2022)), and other, sometimes generative, tasks (Lake & Baroni (2018); Pasupat & Liang (2015); Lin et al. (2019)) and task collections (Srivastava et al. (2022)). LLMs have been shown to lack sufficient reasoning capability across a range of tasks including multi-step planning and complex inference (Valmeekam et al. (2022)). Fan et al. (2023) introduce an LLM reasoning benchmark on algorithmic problems through NP-hard complexity, and Hazra et al. (2024) show that LLMs struggle to complete simple 3SAT problems. Patel et al. (2021) demonstrate that much of LLM mathematical reasoning can be explained by shallow heuristics, and Razeghi et al. (2022) similarly find that term frequency in training data impacts models' in-context learning

ability. In comparison to these, we explore the distinction described by Patel et al. (2021), but push both language simplicity and language unfamiliarity to their limits, by exploring simple languages recognized by randomly sampled DFAs. This enables us to evaluate the ability of LLMs to reason *about language*.

2.3 Language Understanding and LLMs

LLMs can be quite adept at generating programs in general-purpose programming languages (Xu et al. (2022a)). In contrast, adapting models to understand domain-specific languages (Mernik et al. (2005)) introduces unique problems such as navigating idiosyncratic syntax and semantics, and leveraging sparse collections of sample language data. To address these challenges, researchers have considered how well general-purpose LLMs can use language reasoning skills to quickly understand rare or unseen DSLs with only a small set of exemplars (Joel et al. (2024)). While most work in this vein focuses on semantic parsing for downstream applications (Lin et al. (2023)), selecting exemplars (Zhao et al. (2021)), and improving DSL recognition by leveraging more common languages (Bogin et al. (2023)), experiments show strong baseline performance for LLM DSL recognition and parsing out-of-the-box (Wang et al. (2024)), indicating that LLMs may possess emergent language reasoning abilities.

Related lines of work are compositional generalization (Xu et al. (2022b)), which assesses models' ability to organize known units into novel structures, and structural generalization (Yao & Koller (2022)), which assesses models' ability to recognize new structures. Yao & Koller (2022) show that smaller language models like BART and T5 can struggle on these tasks, but to our knowledge there are not comprehensive experiments extending this line of work to LLMs.

3 DFA Reasoning Tasks

3.1 DFAs and Regular Languages

The original Chomsky Hierarchy (Chomsky (1959)) separates language into four types (Figure 2). We focus on the task of understanding Type 3 languages, the simplest form of language in the hierarchy, that are recognized by a Deterministic Finite Automaton (DFA). Examples of languages recognized by DFAs include simple ones like `binary strings with an even number of ones`, and even such examples as `numbers in base 10 divisible by 7`. Type 3 languages are also known as regular languages, which are recognized by regular expressions.

One simple metric of the difficulty of a regular language is the number of states in the corresponding DFA, which represents the amount of memory the automaton has at any point while processing a given sequence.[1] DFAs with 2 states have the property that their set of states is no larger than the output set $\{0, 1\}$, and, therefore, do not have any hidden state. We thus explore 3-state DFAs, as this is the simplest nontrivial case.

3.2 Sequence Completion Task

We first pose a *sequence completion* task, in which models must complete a sequence in a given DFA's language. We study this task primarily because, in practice, most language data that models encounter will be in roughly this format, with several *example sequences* in a given language followed by a *distinct prefix* that needs to be completed via next token prediction.

To generate test cases for this task given a DFA, we first (1) sample 30 example sequences of length 10 that this DFA accepts, and then (2) sample a distinct prefix of length 5 that is not a prefix of any of our 30 example sequences, with the property that there exists some length-≤ 5 *completion* of this prefix that the DFA would accept. The task is to find a completion (not necessarily the same completion found in sampling) of this prefix of between 1 and 5 characters such that the DFA accepts the full sequence. For details on sampling, see Appendix A.2.

We evaluate models by (1) sampling a DFA, (2) sampling 30 problem instances at random (each of which contains 30 example sequences and a distinct prefix), and then (3) computing a binary

[1] There are other metrics of difficulty, but we choose number of states as it is highly parsimonious.

prediction score (whether or not the predicted completion creates a valid string in the language) for each instance separately, then computing a correctness metric as a fraction. We then average this metric over several sampled DFAs to produce our accuracy score.

3.3 TRANSDUCER TASK

While the sequence completion task is the natural one that comes to mind as a basic language task, it has a difficulty-gap problem. Specifically, the issue is that many DFAs, including the one shown in Figure 1, recognize languages that are particularly difficult to identify based on a set of examples, unless you build some kind of world model. Other DFAs end up being trivial to generate a completion for by analyzing common suffixes.[2] To provide a more direct evaluation of non-world-modeling-based pattern recognition, we explore the Transducer task.

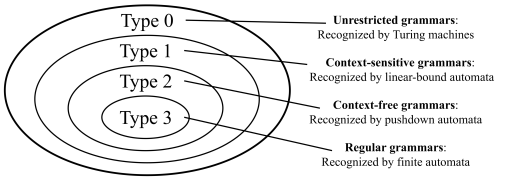


Figure 2: An illustration of Chomsky's hierarchy of languages, ranging from Type 0 to Type 3, which are defined by what formal models can recognize their grammars. In this work, we focus on the simplest language type in the hierarchy, regular grammars, which are recognized by deterministic finite automata (DFAs).

In this task, an input sequence is annotated with an output at each token, the final output is masked, and the masked output is predicted by a language model. We call this a *transducer* task, as the DFA is used as a machine that converts a sequence of inputs into a sequence of outputs. E.g., given the language `does the string have an even number of 'a' tokens` and the input `abcabcaabbccaa`, the annotated string (all that is provided to the model) is `a0b0c0a1b1c1a0a1b1b1c1c1a0a` and the output to predict is `1`. For each problem instance, we provide 30 symbols, and for the first 29, the corresponding transducer output.

This task is significantly more transparent than the sequence completion task as the model has access to intermediate transducer outputs, an (imperfect) proxy for intermediate state.

3.4 BASELINES

To contextualize LLM accuracies, we provide several baseline models with varying degrees of sophistication.

Sequence Completion Task For the Sequence Completion task, we have three kinds of baseline.

- RANDOM$_S$ baseline: produce a random string of length 5 characters. While this might seem redundant as it should have a success rate of 50%, in practice our rejection sampling approach (see Appendix A.2) leads to a slight bias towards DFAs with more accept states. This baseline measures that bias.
- COMMON-SUFFIX$_S$ baseline: find the completion s of length between 1 and 5 that maximizes (# of occurrences as a suffix $\times |s|$). This baseline does not take the distinct prefix into account, and instead tries to find a universal completion that will always end in an accept state for this language.
- n-GRAM$_S$ baseline: we take the last $n-1$ characters of the distinct prefix and search to see if they appear in any of the example sequences at a position where the sequence following is an appropriate length to be a completion (at least 1 but at most 5). We then take a plurality vote among the completions and return this, breaking ties arbitrarily. If there are no matches, we return the result of $(n-1)$-GRAM$_S$. Technically these cover more than n

[2]The difficulty gap exists because a set of recognized sequences of length 10 gives no direct insight into intermediate states between the first and tenth token. As such, to be able to utilize this information for languages like the one in Figure 1 where there are no "resets" (sequences of symbols that necessarily lead to a particular state), a model must be capable of hollistically evaluating the entire sequence, probably requiring a world model. Many other DFAs contain these resets, but do so in such a way that makes it possible to e.g., recognize that all sequences that end in `a` are in the language, making the problem trivial.

characters, since the completion is often > 1 character long; for simplicity, however, we keep the naming consistent with the Transducer baselines.

- BRUTE-FORCE$_S$: take all possible DFAs with 3 states and 3 symbols. Filter for ones that accept all the example sequences. Then try all remaining DFAs on all 3^5 possible 5-length completions and return the completion that the maximal number of DFAs accept, breaking ties arbitrarily.

Note that these baselines are entirely unparameterized and operate identically regardless of the underlying DFA. This makes them direct comparisons to using LLMs in in-context-learning. We do not consider BRUTEFORCE$_S$ to be a reasonable comparison due to its computational complexity, and instead consider it an upper bound on performance on this particular task.

Transducer Task We have similar baselines for the Transducer task.

- NULL$_T$ baseline: for a given DFA, whichever of the following strategies produces a higher accuracy: always predict 0 or always predict 1.
- n-GRAM$_T$ baseline: take the $n-1$ symbols ending at the end of the concatenated transducer sequence (e.g., for $n = 5$ and the above example, this would be `1a0a`). If that sequence does not appear elsewhere in the sequence, return the result of the $(n-1)$-GRAM$_T$ baseline. Otherwise, take the token that appears immediately after each occurrence. If there is a majority, return that, otherwise return the last example.
- BRUTEFORCE$_T$: take all possible DFAs with 3 states and 3 symbols. Filter them for ones that match the given transducer sequence. Take this set and predict the next token. Take a majority vote among these, returning 1 by default if there is no majority.

4 EXPERIMENTS

We evaluated the open-source models Llama 3-8B, Llama 3-70B (AI@Meta (2023)), and Llama 3-8B-Instruct (AI@Meta (2024)), Mistral Nemo Minitron 8B (NVIDIA (2024)), Mistral Nemo Base 2407 (Mistral AI (2024b)) and Mistral Nemo Instruct 2407 (Mistral AI (2024c)), Gemma 7B (Google (2024)), and Falcon 7B (Almazrouei et al. (2023)).

We also evaluated the open-source code models StarCoder2-15B (Lozhkov et al. (2024)), Codestral-22B-v0.1 (Mistral AI (2024a)), Deepseek Coder 33B Instruct (Deepseek (2024)), Qwen2.5-Coder-7B, Qwen2.5-Coder-7B-Instruct, and Qwen2.5-Coder-32B-Instruct (Hui et al. (2024)).

Finally, we evaluated the proprietary models GPT-3.5-turbo-instruct, GPT-3.5 Chat (turbo-0125) (OpenAI (2024a)), GPT-4o-mini (2024-07-18), GPT 4o (2024-05-13) (OpenAI (2024b)), o1-preview (2024-09-12) (OpenAI et al. (2024)), and Claude 3.5 Sonnet (Anthropic (2024)).

For each open source model, we used a local VLLM (Kwon et al. (2023)) server for evaluation and always evaluated on 1000 distinct DFAs. For GPT-4o and Claude, we evaluated on 30 DFAs due to computation costs. For o1-preview we evaluated on only 10 DFAs, and only on the Transducer task (which we felt was a better fit for a reasoning model). For gpt-3.5 and gpt-4o-mini, we evaluated on 100 DFAs. All models were evaluated with temperature 0, except o1-preview[3].

For both tasks, we consider four prompting formats. BASIC provides no context, presenting the problem as a generic sequence generation or next-token prediction task, where output is provided immediately following the input, with no space to think. MORE-EXPL explains that the strings are generated from a DFA with 3 states, but is otherwise identical to BASIC. This remains a sequence generation/next token prediction task. COT provides the same information as MORE-EXPL and additionally invokes chain-of-thought reasoning to help the model reason over the task. Here, the model is given space to reason before providing a tagged answer. RED-GREEN casts the tasks as independent word problems that describe the underlying grammar structure without relying on world knowledge about DFAs and regular languages. It describes an N-state DFA as a house with N rooms, each of which has 3 portals that deterministically go to other rooms (or back to the same room), where the walls of each room are red or green (mirroring transducer output symbols 0 and 1). Similarly to COT, the model is given space to show work before providing a tagged answer.

[3] o1-preview does not allow setting a non-default temperature.

We produce versions of each of these prompts for each task, denoting these with a subscript $_S$ for sequence completion prompts and $_T$ for transducer prompts. Full listings of these prompts can be found in Appendix F, with Table 5 containing a summary of each prompt. While no finite set of prompts will be fully sufficient to capture all possible model behavior, we believe our set of prompts captures common prompting strategies.

Model	Size	IT?	Code?	S.C.	SR	Tr.	TR
Baselines							
BRUTEFORCE	–			100.0 (99.9–100.0)	1	96.4 (96.2–96.7)	1
6-GRAM	–			**91.7 (91.0–92.4)**	2	**93.5 (93.1–93.9)**	2
5-GRAM	–			91.2 (90.4–91.9)	3	93.4 (93.0–93.7)	3
4-GRAM	–			89.6 (88.7–90.4)	4	91.1 (90.6–91.6)	4
3-GRAM	–			87.0 (86.1–87.8)	5	87.0 (86.4–87.6)	16
2-GRAM	–			83.3 (82.2–84.2)	8	74.5 (73.6–75.3)	25
COMMON-SUFFIX	–			84.7 (83.6–85.6)	6	–	–
RANDOM$_S$/NULL$_T$	–			53.3 (51.7–54.7)	26	68.9 (68.2–69.6)	26
Open Source Completion							
llama3-8B	8.0B			73.8 (72.4–75.1)	18	87.5 (86.9–88.0)	14
llama3-70B	70.6B			71.4 (70.0–72.7)	23	87.7 (87.2–88.3)	12
llama3.1-8B-Instruct	8.0B	✓		75.3 (74.0–76.6)	16	85.9 (85.3–86.5)	18
mistral-nemo-minitron-8B	8.4B			**78.7 (77.5–79.8)**	12	**88.6 (88.0–89.1)**	5
mistral-nemo-base-12B	12.2B			75.5 (74.3–76.6)	15	87.9 (87.4–88.4)	10
mistral-nemo-instruct-12B	12.2B	✓		72.2 (70.9–73.4)	22	88.0 (87.5–88.5)	8
gemma-7b	8.5B			72.6 (71.3–73.7)	20	82.1 (81.4–82.7)	22
falcon-7b	7.2B			69.0 (67.6–70.2)	24	84.9 (84.3–85.5)	20
Open Source Code							
starcoder2-15b	16.0B		✓	73.5 (72.0–74.7)	19	87.7 (85.8–89.5)	13
codestral-22B	22.2B		✓	78.0 (76.8–79.1)	13	86.6 (86.0–87.1)	17
deepseek-coder-33b-instruct	33.3B	✓	✓	76.7 (75.3–77.8)	14	85.6 (85.0–86.2)	19
qwen-2.5-coder-7B	7.6B		✓	**79.5 (78.4–80.5)**	9	88.2 (87.6–88.7)	7
qwen-2.5-coder-instruct-7B	7.6B	✓	✓	79.5 (78.3–80.5)	10	**88.3 (87.8–88.8)**	6
qwen-2.5-coder-instruct-32B	32.8B	✓	✓	79.2 (78.0–80.3)	11	87.9 (87.4–88.4)	9
Proprietary							
gpt-3.5-instruct	?	✓		67.3 (63.1–71.5)	25	**87.8 (85.9–89.6)**	11
gpt-3.5-chat	?	✓		N/A	–	66.8 (63.4–69.8)	27
gpt-4o-mini	?	✓		72.4 (68.1–76.3)	21	79.8 (77.3–82.2)	23
gpt-4o	?	✓		74.4 (69.9–78.6)	17	83.7 (80.1–86.9)	21
claude-3.5	?	✓		**84.0 (79.3–88.4)**	7	87.1 (83.9–90.2)	15
o1-preview	?	✓		–	–	76.5 (69.4–84.3)	24

Table 1: Results for our experiments. We present model metadata alongside model results on both the Transducer and Sequence completion tasks. Each cell contains the mean performance across DFAs for the best-performing prompt (see Table 2 for details), with 95% confidence intervals of the mean in parentheses. "N/A" is used whenever the model returned an invalid result at least 25% of the time. (IT = Instruction-Tuned, TR/SR = Transducer/Sequence Completion rank, the ordinal rank of the given model on the given task.)

5 RESULTS

Main results for all tasks are presented in Table 1. For all LLMs, we ignore non-answers, i.e., if for a given DFA a model gets 25 correct answers, 1 incorrect answer, and responds with an unparseable result on 4, this counts as a 25/26, not a 25/29. We then report the mean across DFAs and 95% bootstrap confidence intervals.

5.1 SEQUENCE COMPLETION

As seen in Table 1, this task is nearly always fully determined, that is, it can be solved with $\sim$100% accuracy in theory, as demonstrated by BRUTEFORCE$_S$ results. Of course, BRUTEFORCE$_S$ is ex-

tremely computationally expensive, and, as such, we primarily focus on the n-GRAM$_S$ heuristics as our baselines. Still, we find that n-GRAM$_S$ heuristics tend to outperform LLMs.

As seen in Table 2, we find that giving the model the opportunity to logically reason about the prompt via chain-of-thought and present a conclusion has inconsistent results. Specifically, we find that BASIC$_S$ is the best prompt for gpt-4o-mini, but not gpt-4o, where the best performing prompt is RED-GREEN$_S$. We find that claude-3.5 is entirely unable to follow the sequence completion prompts BASIC$_S$ and MORE-EXPL$_S$, and performs best at the COT$_S$ prompt.

Additionally, we find that in this task, code-specific open-source models tend to perform better than sequence completion models, suggesting some generalized ability to produce strings from novel languages demonstrated by example. Overall, the relative performances of LLMs and prompts comport somewhat well to heuristics on which models and prompting strategies should work best. Nonetheless, LLMs underperform simple n-GRAM heuristics.

One potential problem with using this task for cross-model comparisons is the relevance of tokenization. Unfortunately, we found that forcing uniform tokenization by using commas in the prompt uniformly reduced accuracy, see Appendix E for details.

Model	BASIC	MORE-EXPL	COT	RED-GREEN
Sequence Completion				
gpt-4o-mini	**72.4 (68.1–76.3)**	70.5 (66.4–74.6)	58.0 (53.4–62.4)	59.1 (54.9–63.2)
gpt-4o	72.1 (65.9–78.2)	N/A	67.4 (60.8–73.8)	**74.4 (69.9–78.6)**
claude-3.5	N/A	N/A	**84.0 (79.3–88.4)**	80.0 (74.9–85.2)
Transducer				
gpt-4o-mini	**79.8 (77.3–82.2)**	76.7 (74.2–79.3)	65.2 (63.1–67.4)	74.5 (72.0–77.0)
gpt-4o	**83.7 (80.1–86.9)**	82.6 (79.1–85.9)	67.8 (63.1–72.3)	82.6 (78.8–86.3)
claude-3.5	86.9 (83.3–90.0)	**87.1 (83.9–90.2)**	76.4 (72.9–79.9)	82.9 (78.9–86.9)

Table 2: Results for models where we investigated multiple prompts (we only used BASIC on other models). We bold the best prompt for each model. Non-COT prompts consistently work better for the Transducer task, with more mixed results on sequence completion.

5.2 TRANSDUCER

Unlike sequence completion, this task is generally not fully determined, with the BRUTEFORCE$_T$ model only achieving 96.4% accuracy. However, relative results should still be valid as the impossible problem instances are present with equal probability for all models.

We find that in general all LLMs underperform a 4-GRAM$_T$ model, demonstrating that they are unable to adequately solve this task. The relative performance of the models also does not correspond to their overall scale, with open source LLama-3 and Mistral Nemo 8B parameter models outperforming Claude and GPT-4o. Even within a model class we find no clear pattern: GPT-4o and o1-preview[4] are outperformed by GPT 3.5, Llama 3-70B has similar performance to Llama 3-8B, and the Mistral Nemo 12B models perform similarly to Nemo Minitron 8B. Coding models also demonstrate no advantage on this task.

The generally lower performance of chat-oriented models suggests this task is better suited to non-chat models. To investigate that this is not specific to the BASIC prompt, we investigate other prompts for chat models. As seen in Table 2, our chain-of-though and word problem prompts, which attempt to leverage the full reasoning capabilities of chat models, also fare poorly, performing similarly or worse to the BASIC prompt on the Transducer task in all cases.

Overall, we conclude that LLMs are unable to perform the DFA transducer inference task to a reasonable degree. This failure cannot be attributed to a lack of world modeling ability, as n-GRAM$_T$ models do not construct world models. Instead, it seems the LLMs are unable to detect patterns when

[4]The particularly poor performance of o1-preview may be due to the model not supporting temperature 0. In other experiments, we found that GPT models with temperature 1 tended to perform poorly. See Appendix B for more details.

those patterns are drawn from an unfamiliar source, even a relatively simple one. For a detailed analysis of a case study, see Appendix C.

5.3 Comparison of Benchmarks

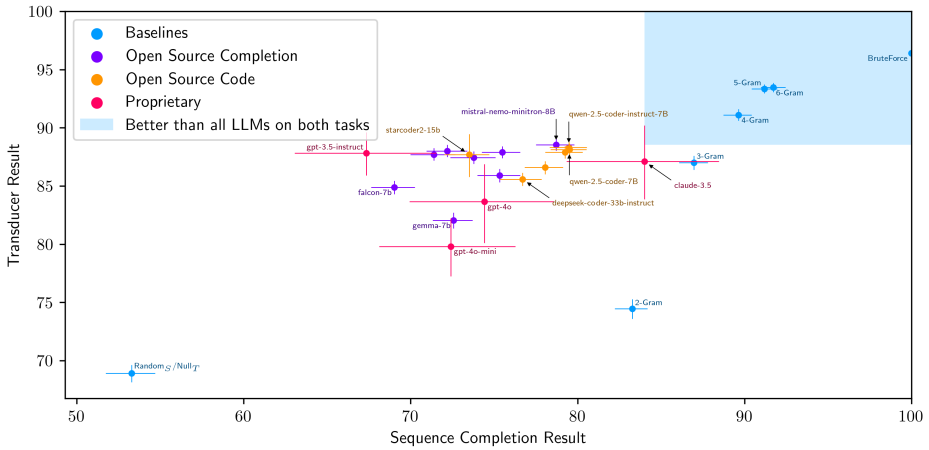


Figure 3: Transducer and sequence completion results plotted against each other. Points represent the mean over several DFAs and intervals represent 95% confidence intervals. Points are colored by model type, with the best and worst model by each metric in each category labeled, as well as all baseline and proprietary models.

Figure 3 displays the relationship between model performance on the Sequence Completion and Transducer benchmarks. While at a high level, there is a positive correlation between the two, there are a few notable differences. For one, the Code models perform notably better than other open source models on Sequence Completion, but not on Transducer. Additionally, on Transducer, a ceiling on performance is observed, where LLMs cluster together between 3-GRAM$_T$ and 4-GRAM$_T$ performance; this clustering does not appear on the Sequence Completion benchmark.

6 Conclusion

Our findings highlight significant weaknesses in large language models' ability to generalize to entirely novel language reasoning problems, even simple ones solely involving next-token prediction on basic languages recognized by 3-state DFAs. These results, combined with that of previous work demonstrating that large language models can quite accurately perform a variety of language tasks, suggests that LLMs solve language problems via a mechanism distinct from general language reasoning ability. Our use of n-gram baselines and next-token prediction tasks allows us to exclude the possibility that the issue is primarily related to LLMs' lack of world modeling or any inherent limitations of next-token prediction models. We believe our results suggest that LLMs have learned individual models of particular languages, but not a general theory of language.

Interestingly, in our transducer experiments, LLMs consistently perform better by directly predicting the next token than by explicitly reasoning through the problem. While our conclusions are limited by the finite nature of our prompt set, this suggests that they do, in fact, possess some latent understanding of language, but this understanding is inferior to basic n-gram models for $n > 3$.

Many potential foundation model applications involve tasks that are not expressed in familiar human languages or pre-existing programming languages. More specifically, in tasks where there is a need to produce an output in a precise, atypical, format, we should be skeptical of the ability of LLMs to in-context-learn this format. For these tasks, it may be prudent to seek a new approach.

IMPACT STATEMENT

Aside from the social consequences of this work as related to advancing the field of Machine Learning in general, this work has the goal of advancing the field of benchmarks in Machine Learning. While we view this as a positive objective, as it ensures that models are being evaluated fairly, it might have negative consequences insofar as benchmarking techniques might be best left unpublished to prevent deliberate or unintentional overfitting.

REFERENCES

AI@Meta. Llama 3 model card, 2023. URL https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md.

AI@Meta. Llama 3.1 8b instruct, 2024. URL https://huggingface.co/meta-llama/Llama-3.1-8B-Instruct.

Ebtesam Almazrouei, Hamza Alobeidli, Abdulaziz Alshamsi, Alessandro Cappelli, Ruxandra Cojocaru, Merouane Debbah, Etienne Goffinet, Daniel Heslow, Julien Launay, Quentin Malartic, Badreddine Noune, Baptiste Pannier, and Guilherme Penedo. Falcon-40B: an open large language model with state-of-the-art performance. 2023.

Aida Amini, Saadia Gabriel, Peter Lin, Rik Koncel-Kedziorski, Yejin Choi, and Hannaneh Hajishirzi. Mathqa: Towards interpretable math word problem solving with operation-based formalisms. *arXiv preprint arXiv:1905.13319*, 2019.

Anthropic. Claude 3.5 sonnet, 2024. URL https://www.anthropic.com/news/claude-3-5-sonnet.

Ben Athiwaratkun, Sanjay Krishna Gouda, Zijian Wang, Xiaopeng Li, Yuchen Tian, Ming Tan, Wasi Uddin Ahmad, Shiqi Wang, Qing Sun, Mingyue Shang, et al. Multi-lingual evaluation of code generation models. *arXiv preprint arXiv:2210.14868*, 2022.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, et al. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 17682–17690, 2024.

Prajjwal Bhargava and Vincent Ng. Commonsense knowledge reasoning and generation with pretrained language models: A survey. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 12317–12325, 2022.

Ben Bogin, Shivanshu Gupta, Peter Clark, and Ashish Sabharwal. Leveraging code to improve in-context learning for semantic parsing. *arXiv preprint arXiv:2311.09519*, 2023.

Noam Chomsky. On certain formal properties of grammars. *Information and control*, 2(2):137–167, 1959.

Noam Chomsky, Ian Roberts, , and Jeffrey Watumull. Noam chomsky: The false promise of chatgpt. *The New York Times*, 2023. URL https://www.nytimes.com/2023/03/08/opinion/noam-chomsky-chatgpt-ai.html.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Deepseek. Deepseek coder 33b instruct, 2024. URL https://huggingface.co/deepseek-ai/deepseek-coder-33b-instruct.

Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Jingyuan Ma, Rui Li, Heming Xia, Jingjing Xu, Zhiyong Wu, Tianyu Liu, et al. A survey on in-context learning. *arXiv preprint arXiv:2301.00234*, 2022.

Lizhou Fan, Wenyue Hua, Lingyao Li, Haoyang Ling, and Yongfeng Zhang. Nphardeval: Dynamic benchmark on reasoning ability of large language models via complexity classes. *arXiv preprint arXiv:2312.14890*, 2023.

Yao Fu, Hao Peng, Ashish Sabharwal, Peter Clark, and Tushar Khot. Complexity-based prompting for multi-step reasoning. In *The Eleventh International Conference on Learning Representations*, 2022.

Google. gemma-7b, 2024. URL `https://huggingface.co/google/gemma-7b`.

Rishi Hazra, Gabriele Venturato, Pedro Zuidberg Dos Martires, and Luc De Raedt. Can large language models reason? a characterization via 3-sat. *arXiv preprint arXiv:2408.07215*, 2024.

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.

Wenyue Hua and Yongfeng Zhang. System 1+ system 2= better world: Neural-symbolic chain of logic reasoning. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pp. 601–612, 2022.

Jie Huang and Kevin Chen-Chuan Chang. Towards reasoning in large language models: A survey. *arXiv preprint arXiv:2212.10403*, 2022.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Kai Dang, et al. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*, 2024.

Sathvik Joel, Jie JW Wu, and Fatemeh H Fard. A survey on llm-based code generation for low-resource and domain-specific programming languages. *arXiv preprint arXiv:2410.03981*, 2024.

Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. Decomposed prompting: A modular approach for solving complex tasks. *arXiv preprint arXiv:2210.02406*, 2022.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213, 2022.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

Brenden Lake and Marco Baroni. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *International conference on machine learning*, pp. 2873–2882. PMLR, 2018.

Bill Yuchen Lin, Wangchunshu Zhou, Ming Shen, Pei Zhou, Chandra Bhagavatula, Yejin Choi, and Xiang Ren. Commongen: A constrained text generation challenge for generative commonsense reasoning. *arXiv preprint arXiv:1911.03705*, 2019.

Kevin Lin, Patrick Xia, and Hao Fang. Few-shot adaptation for parsing contextual utterances with llms. *arXiv preprint arXiv:2309.10168*, 2023.

Jiachang Liu, Dinghan Shen, Yizhe Zhang, Bill Dolan, Lawrence Carin, and Weizhu Chen. What makes good in-context examples for gpt-3? *arXiv preprint arXiv:2101.06804*, 2021.

Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, Tianyang Liu, Max Tian, Denis Kocetkov, Arthur Zucker, Younes Belkada, Zijian Wang, Qian Liu, Dmitry Abulkhanov, Indraneil Paul, Zhuang Li, Wen-Ding Li, Megan Risdal, Jia Li, Jian Zhu, Terry Yue Zhuo, Evgenii Zheltonozhskii, Nii Osae Osae Dade, Wenhao Yu, Lucas Krauß, Naman Jain, Yixuan Su, Xuanli He, Manan Dey, Edoardo Abati, Yekun Chai, Niklas Muennighoff, Xiangru Tang, Muhtasham

Oblokulov, Christopher Akiki, Marc Marone, Chenghao Mou, Mayank Mishra, Alex Gu, Binyuan Hui, Tri Dao, Armel Zebaze, Olivier Dehaene, Nicolas Patry, Canwen Xu, Julian McAuley, Han Hu, Torsten Scholak, Sebastien Paquet, Jennifer Robinson, Carolyn Jane Anderson, Nicolas Chapados, Mostofa Patwary, Nima Tajbakhsh, Yacine Jernite, Carlos Muñoz Ferrandis, Lingming Zhang, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. Starcoder 2 and the stack v2: The next generation, 2024.

Marjan Mernik, Jan Heering, and Anthony M Sloane. When and how to develop domain-specific languages. *ACM computing surveys (CSUR)*, 37(4):316–344, 2005.

Raphaël Millière. Language models as models of language. *arXiv preprint arXiv:2408.07144*, 2024.

Mistral AI. Codestral-22b-v0.1, 2024a. URL https://huggingface.co/mistralai/Codestral-22B-v0.1.

Mistral AI. Mistral-nemo-base-2407, 2024b. URL https://huggingface.co/mistralai/Mistral-Nemo-Base-2407.

Mistral AI. Mistral-nemo-instruct-2407, 2024c. URL https://huggingface.co/mistralai/Mistral-Nemo-Instruct-2407.

NVIDIA. Mistral-nemo-minitron-8b-base, 2024. URL https://huggingface.co/nvidia/Mistral-NeMo-Minitron-8B-Base.

OpenAI. Gpt 3.5 turbo, 2024a. URL https://openai.com/index/new-embedding-models-and-api-updates/.

OpenAI. Gpt-4o system card, 2024b. URL https://arxiv.org/abs/2410.21276.

OpenAI, :, Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, Alex Iftimie, Alex Karpenko, Alex Tachard Passos, Alexander Neitz, Alexander Prokofiev, Alexander Wei, Allison Tam, Ally Bennett, Ananya Kumar, Andre Saraiva, Andrea Vallone, Andrew Duberstein, Andrew Kondrich, Andrey Mishchenko, Andy Applebaum, Angela Jiang, Ashvin Nair, Barret Zoph, Behrooz Ghorbani, Ben Rossen, Benjamin Sokolowsky, Boaz Barak, Bob McGrew, Borys Minaiev, Botao Hao, Bowen Baker, Brandon Houghton, Brandon McKinzie, Brydon Eastman, Camillo Lugaresi, Cary Bassin, Cary Hudson, Chak Ming Li, Charles de Bourcy, Chelsea Voss, Chen Shen, Chong Zhang, Chris Koch, Chris Orsinger, Christopher Hesse, Claudia Fischer, Clive Chan, Dan Roberts, Daniel Kappler, Daniel Levy, Daniel Selsam, David Dohan, David Farhi, David Mely, David Robinson, Dimitris Tsipras, Doug Li, Dragos Oprica, Eben Freeman, Eddie Zhang, Edmund Wong, Elizabeth Proehl, Enoch Cheung, Eric Mitchell, Eric Wallace, Erik Ritter, Evan Mays, Fan Wang, Felipe Petroski Such, Filippo Raso, Florencia Leoni, Foivos Tsimpourlas, Francis Song, Fred von Lohmann, Freddie Sulit, Geoff Salmon, Giambattista Parascandolo, Gildas Chabot, Grace Zhao, Greg Brockman, Guillaume Leclerc, Hadi Salman, Haiming Bao, Hao Sheng, Hart Andrin, Hessam Bagherinezhad, Hongyu Ren, Hunter Lightman, Hyung Won Chung, Ian Kivlichan, Ian O'Connell, Ian Osband, Ignasi Clavera Gilaberte, Ilge Akkaya, Ilya Kostrikov, Ilya Sutskever, Irina Kofman, Jakub Pachocki, James Lennon, Jason Wei, Jean Harb, Jerry Twore, Jiacheng Feng, Jiahui Yu, Jiayi Weng, Jie Tang, Jieqi Yu, Joaquin Quiñonero Candela, Joe Palermo, Joel Parish, Johannes Heidecke, John Hallman, John Rizzo, Jonathan Gordon, Jonathan Uesato, Jonathan Ward, Joost Huizinga, Julie Wang, Kai Chen, Kai Xiao, Karan Singhal, Karina Nguyen, Karl Cobbe, Katy Shi, Kayla Wood, Kendra Rimbach, Keren Gu-Lemberg, Kevin Liu, Kevin Lu, Kevin Stone, Kevin Yu, Lama Ahmad, Lauren Yang, Leo Liu, Leon Maksin, Leyton Ho, Liam Fedus, Lilian Weng, Linden Li, Lindsay McCallum, Lindsey Held, Lorenz Kuhn, Lukas Kondraciuk, Lukasz Kaiser, Luke Metz, Madelaine Boyd, Maja Trebacz, Manas Joglekar, Mark Chen, Marko Tintor, Mason Meyer, Matt Jones, Matt Kaufer, Max Schwarzer, Meghan Shah, Mehmet Yatbaz, Melody Y. Guan, Mengyuan Xu, Mengyuan Yan, Mia Glaese, Mianna Chen, Michael Lampe, Michael Malek, Michele Wang, Michelle Fradin, Mike McClay, Mikhail Pavlov, Miles Wang, Mingxuan Wang, Mira Murati, Mo Bavarian, Mostafa Rohaninejad, Nat McAleese, Neil Chowdhury, Neil Chowdhury, Nick Ryder, Nikolas Tezak, Noam Brown, Ofir Nachum, Oleg Boiko, Oleg Murk, Olivia Watkins, Patrick Chao, Paul Ashbourne, Pavel Izmailov, Peter Zhokhov, Rachel Dias, Rahul Arora, Randall Lin, Rapha Gontijo Lopes, Raz Gaon, Reah Miyara, Reimar Leike, Renny Hwang, Rhythm Garg, Robin Brown, Roshan James, Rui Shu, Ryan Cheu, Ryan

Greene, Saachi Jain, Sam Altman, Sam Toizer, Sam Toyer, Samuel Miserendino, Sandhini Agarwal, Santiago Hernandez, Sasha Baker, Scott McKinney, Scottie Yan, Shengjia Zhao, Shengli Hu, Shibani Santurkar, Shraman Ray Chaudhuri, Shuyuan Zhang, Siyuan Fu, Spencer Papay, Steph Lin, Suchir Balaji, Suvansh Sanjeev, Szymon Sidor, Tal Broda, Aidan Clark, Tao Wang, Taylor Gordon, Ted Sanders, Tejal Patwardhan, Thibault Sottiaux, Thomas Degry, Thomas Dimson, Tianhao Zheng, Timur Garipov, Tom Stasi, Trapit Bansal, Trevor Creech, Troy Peterson, Tyna Eloundou, Valerie Qi, Vineet Kosaraju, Vinnie Monaco, Vitchyr Pong, Vlad Fomenko, Weiyi Zheng, Wenda Zhou, Wes McCabe, Wojciech Zaremba, Yann Dubois, Yinghai Lu, Yining Chen, Young Cha, Yu Bai, Yuchen He, Yuchen Zhang, Yunyun Wang, Zheng Shao, and Zhuohan Li. Openai o1 system card, 2024. URL https://arxiv.org/abs/2412.16720.

Panupong Pasupat and Percy Liang. Compositional semantic parsing on semi-structured tables. *arXiv preprint arXiv:1508.00305*, 2015.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. Are nlp models really able to solve simple math word problems? *arXiv preprint arXiv:2103.07191*, 2021.

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.

Rylan Schaeffer, Brando Miranda, and Sanmi Koyejo. Are emergent abilities of large language models a mirage? *Advances in Neural Information Processing Systems*, 36, 2024.

Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam Fisch, Adam R Brown, Adam Santoro, Aditya Gupta, Adrià Garriga-Alonso, et al. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *arXiv preprint arXiv:2206.04615*, 2022.

Karthik Valmeekam, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. Large language models still can't plan (a benchmark for llms on planning and reasoning about change). In *NeurIPS 2022 Foundation Models for Decision Making Workshop*, 2022.

Bailin Wang, Zi Wang, Xuezhi Wang, Yuan Cao, Rif A Saurous, and Yoon Kim. Grammar prompting for domain-specific language generation with large language models. *Advances in Neural Information Processing Systems*, 36, 2024.

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*, 2022a.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022b.

Nathaniel Weir, Peter Clark, and Benjamin Van Durme. Nellie: A neuro-symbolic inference engine for grounded, compositional, and explainable reasoning. *Preprint*, 2023.

Nathaniel Weir, Kate Sanders, Orion Weller, Shreya Sharma, Dongwei Jiang, Zhengping Zhang, Bhavana Dalvi Mishra, Oyvind Tafjord, Peter Jansen, Peter Clark, et al. Enhancing systematic decompositional natural language inference using informal logic. *arXiv preprint arXiv:2402.14798*, 2024.

Frank F Xu, Uri Alon, Graham Neubig, and Vincent Josua Hellendoorn. A systematic evaluation of large language models of code. In *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming*, pp. 1–10, 2022a.

Zhenlin Xu, Marc Niethammer, and Colin A Raffel. Compositional generalization in unsupervised compositional representation learning: A study on disentanglement and emergent language. *Advances in Neural Information Processing Systems*, 35:25074–25087, 2022b.

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, 2023. *URL https://arxiv. org/pdf/2305.10601. pdf*, 2023.

Yuekun Yao and Alexander Koller. Structural generalization is hard for sequence-to-sequence models. *arXiv preprint arXiv:2210.13050*, 2022.

Zihao Zhao, Eric Wallace, Shi Feng, Dan Klein, and Sameer Singh. Calibrate before use: Improving few-shot performance of language models. In *International conference on machine learning*, pp. 12697–12706. PMLR, 2021.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, et al. Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625*, 2022.

A Details on Sampling

A.1 Sampling of DFAs

We use rejection sampling to sample DFAs. Specifically, we uniformly sample a start state, then for each (source state, symbol) pair, we sample a post-transition state. We also randomly assign each state to be accept or reject with probability 50%. We then reject any DFA that has all accept or all reject states (so only DFAs with 1 or 2 accept states are allowed), or for which certain states are unreachable from the start state.

A.2 Sampling of Sequence Completion Tasks

To sample a sequence completion task, we first sample a DFA as described in Appendix A.1.

To sample a task instance, we sample example sequences and distinct prefix. Each example sequence is sampled uniformly from the space of $\{a, b, c\}^{10}$ and then rejected if the DFA does not accept the sequence. Our distinct prefix and completion are sampled uniformly from $\{a, b, c\}^5 \times \{a, b, c\}^5$, and are rejected if the DFA does not accept the concatenation of the two, or if the prefix is the prefix of any of the previous sequences. We then discard the completion. If we, at any point, reject 50 sequences when attempting to sample a sequence or prefix, we return an error.

We run a "pilot" sampling for a DFA to ensure that it is valid, in which we sample an instance as described above. If there is an error in sampling this pilot instance, we reject the DFA. Otherwise, we proceed to sample our task instances. At this stage, if there is an error in sampling, we reject the instance rather than the DFA. This pilot sample rejection procedure leads to a slight bias towards 2-accept state DFAs over 1-accept state DFAs, as measured by the RANDOM_S baseline.

A.3 Sampling of Transducer Tasks

We sample a DFA as described in Appendix A.1, and then sample random sequences (30 in our experiments) and generate transducer traces. If every transducer trace ends with a 0 or every trace ends with a 1, we reject the DFA and resample.

B Results of o1-preview

We evaluated o1-preview on 10 DFAs, using 30 problem instances per DFA of the Transducer task, as in other Transducer experiments, and the BASIC_T prompt, as this is the most neutral prompt. Table 3 displays results on each DFA. Overall, while these results are not on a particularly large sample, they fairly definitively demonstrate that o1-preview does not achieve strikingly good performance on this task.

DFA	o1-preview	gpt-4o	6-Gram
1	25/30	27/30	26/30
2	23/29	24/30	25/30
3	19/30	23/30	28/30
4	22/30	23/30	28/30
5	29/29	30/30	30/30
6	19/30	24/30	30/30
7	17/29	23/30	25/30
8	23/30	25/30	26/30
9	21/30	28/30	30/30
10	29/30	29/30	30/30

Table 3: Results on each DFA. We find that in 7 cases, o1-preview underperforms gpt-4o, in 2 cases it gets the same number of instances wrong but provides a non-answer on an additional instance, and in 1 case it ties gpt-4o. In no cases does it outperform.

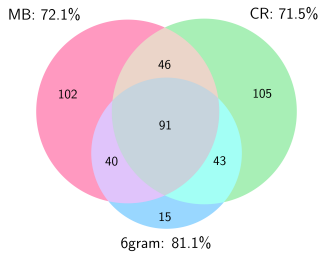
C CASE STUDY: SUM MODULO 3 DFA

We investigate the transducer task on the DFA depicted in Figure 1. This DFA can be interpreted as an arithmetic check, where `a` represents 0, `b` represents 1, and `c` represents 2, and the DFA accepts strings whose sum is equal to 0 modulo 3. For this case study, we focus on the model/prompt combinations MB and CR, defined as

- MB: mistral-nemo-minitron-8B/BASIC$_T$. Selected as it is the best performing combination overall.
- CR: claude-3.5/RED-GREEN$_T$. Selected as it is the best performing combination that provides an explanation (needed later for our qualitative analysis)

Figure 4a depicts the number of errors each model receives on 1000 instances of the transducer task for this DFA. In general, nearly all errors made by the 6-GRAM$_T$ model were also made by at least one LLM model, while the two LLM models often made unique errors. Interestingly, while this task is better-known than most DFAs, we find that all 3 models perform worse on this DFA than their average across random DFAs.

(a)



(b)

	Correct	Incorrect
Total	30	30
`a` is no-op	21	22
`1b` and `1c` lead to 0	14	17
2-periodic	9	14
3-periodic	4	4
2 red rooms	2	3

Figure 4: Results on Sum Modulo 3 DFA. (a) MB=mistral-nemo-minitron-8B/BASIC$_T$, CR=claude-3.5/RED-GREEN$_T$. Venn diagram of errors (out of 1000). Labeled percentages are accuracies. (b) Results of qualitative analysis, out of 30 in both cases.

We also performed a qualitative analysis, investigating CR's outputs on the RED-GREEN$_T$ prompt to see what kind of reasoning it is using; specifically we sampled 30 examples where it had the correct answer, and 30 examples where it had the incorrect answer but the 6-GRAM$_T$ model had the correct answer. Results of this analysis can be found in Figure 4b. We find that, in general, CR is following a 3-GRAM approach, learning rules relating to the conditions under which the previous output and symbol can be used to predict the next output. Specifically, it is able to learn that `a` does not change the output, and that `b` and `c` will lead a `1` state to a `0` state. These results comport with the overall finding of Table 1, where we found that 3-GRAM$_T$ was the largest n-GRAM$_T$ that any LLM outperformed.

The model occasionally makes attempts at more sophisticated pattern match reasoning, but rarely is successful in doing so. It also attempts to identify periodic patterns, but identifies period-2 patterns more than period-3 patterns, despite knowing that there are three "rooms" (states). At no point in any of the 60 reasoning traces analyzed does it realize that this is a version of the Sum Modulo 3 DFA[5], or fully determine the DFA in any other way, but it does show some glimmers of world modeling. Specifically, in a few cases it correctly determines that there are two red rooms; but this does not seem to lead to any further discoveries. It is not superior reasoning that leads to correct solutions, rather the correct examples are more likely to be ones that a 3-GRAM model would infer correctly, i.e., those traces ending in `a`, `1b`, or `1c`, which occur cumulatively in $\frac{5}{9}$ of cases[6].

Despite transformers' high computational capacity, without the ability to pattern match to existing problems, Claude uses an unsophisticated and ineffectual approach.

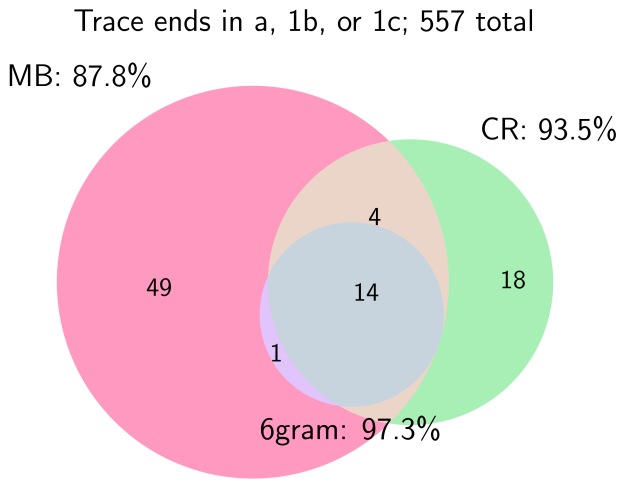
[5] In fact in none of the 1000 reasoning traces do the substrings "sum" or "mod" appear, except once as a part of "assuming"

[6] Looking at the $\sim\frac{5}{9}$ of examples that follow this pattern, we find that CR achieves 93.5%, to the 6-GRAM$_T$'s 97.3%, and on the remaining $\sim\frac{4}{9}$, it achieves only 43.8%, or worse than chance, to the 6-GRAM$_T$'s 60.7%. Detailed Venn diagrams on these conditions can be found in Appendix D.

D MORE DETAILS ON SUM MODULO 3 DFA CASE STUDY

Figure 5 depicts the results of the Sum Modulo 3 experiment, but filtered for two conditions. In the (a) condition, the trace ends in such a way that a 3-GRAM model would be able to determine the output, and the (b) condition is the complement.

(a)



(b)

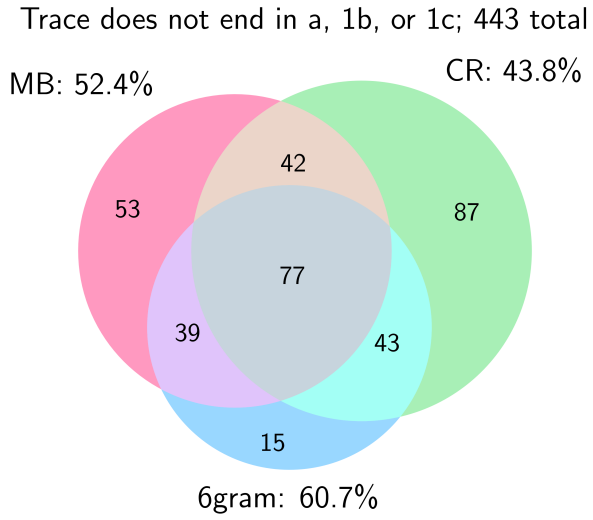


Figure 5: Results on Sum Modulo 3 DFA under trivial / nontrivial conditions. Percentages are accuracy numbers, and venn diagram is error counts. (a) In this condition, CR and the 6-GRAM$_T$ both get very high accuracies, with nearly all 6-GRAM$_T$ also being CR errors. MB does relatively poorly. (b) In this condition, models do significantly more poorly overall, with CR in particular performing worse than chance. Here, errors are more symmetric, with more 6-GRAM$_T$ errors that are not accounted for by either or both model, indicating that a larger fraction of both successes and failures in this condition are down to random chance.

Model	BASIC$_S$	BASIC-COMMAS$_S$
qwen-2.5-coder-7B	**79.5 (78.4–80.5)**	**60.7 (59.3–62.1)**
qwen-2.5-coder-instruct-7B	79.5 (78.3–80.5)	55.5 (54.0–56.9)
qwen-2.5-coder-instruct-32B	79.2 (78.0–80.3)	55.2 (53.7–56.7)
mistral-nemo-minitron-8B	78.7 (77.5–79.8)	59.3 (57.9–60.8)
codestral-22B	78.0 (76.8–79.1)	59.0 (57.5–60.3)
deepseek-coder-33b-instruct	76.7 (75.3–77.8)	54.9 (53.0–56.8)
mistral-nemo-base-12B	75.5 (74.3–76.6)	60.6 (59.1–62.2)
llama3.1-8B-Instruct	75.3 (74.0–76.6)	56.3 (54.4–58.1)
llama3-8B	73.8 (72.4–75.1)	61.5 (60.2–62.9)
starcoder2-15b	73.5 (72.0–74.7)	58.2 (56.7–59.8)
gemma-7b	72.6 (71.3–73.7)	54.0 (51.9–56.0)
gpt-4o-mini	72.4 (68.1–76.3)	64.1 (59.5–68.3)
mistral-nemo-instruct-12B	72.2 (70.9–73.4)	58.2 (56.4–59.8)
gpt-4o	72.1 (65.9–78.2)	66.8 (58.5–74.8)
llama3-70B	71.4 (70.0–72.7)	56.4 (54.7–58.0)
falcon-7b	69.0 (67.6–70.2)	56.1 (54.5–57.6)
gpt-3.5-instruct	67.3 (63.1–71.5)	52.3 (46.5–57.9)
claude-3.5	N/A	N/A
gpt-3.5-chat	N/A	N/A

Table 4: Results on Sequence Completion Task. We compare BASIC$_S$ to the comma-variant BASIC-COMMAS$_S$.

E SEQUENCE COMPLETION TASK PROMPT WITH COMMAS

To avoid tokenization differences with models, we also investigate a version of our Sequence Completion prompt that uses spaces and commas between the elements of the sequence. Unfortunately, results using this prompt were uniformly worse than results on the prompt without spaces and commas. Table 4 shows the results on a variety of models. All are worse with commas than without.

F PROMPT LISTINGS

F.1 SUMMARIES

Table 5 contains summaries of each prompt.

F.2 FULL EXAMPLE LISTINGS

Prompt	T	S
BASIC	You are a sequence completion model. Output the next element of the sequence, and nothing else. <TRANSDUCER PREFIX>,	The following strings come from an alien language that follows a simple grammar. Infer the alien grammar using the example strings. Then, add a suffix to the final string using between 1 and 5 characters such that the full string follows the grammar. Output only the necessary suffix to complete the final string, and nothing else. <EXAMPLES> <PREFIX>
MORE-EXPL	You are a sequence completion model. The following sequence is generated from an unknown but consistent grammar. Identify the patterns within the sequence to determine its next element. Output the next element of the sequence, and nothing else. <TRANSDUCER PREFIX>,	I have a 3-state DFA model that outputs either 0 or 1 after each element I input. 1 indicates that the input string thus far results in a "valid" state, and 0 indicates that it does not. I collect a set of valid strings using this DFA, listed below. Infer the underlying DFA model using these strings and complete the final string, using up to n characters, such that it is also a valid string. Output only the necessary suffix to complete the final string, and nothing else. <EXAMPLES> <PREFIX>
COT	A DFA is a finite-state machine that accepts or rejects a given string of symbols, by running through a n-state sequence uniquely determined by the string. I have a 3-state DFA model that outputs either 0 or 1 after each element I input. 1 indicates that the input string thus far results in a "valid" state, and 0 indicates that it does not. I collect the inputs and outputs into an input sequence and an output sequence. Infer the underlying DFA model to predict the next integer in the output sequence. Reason step by step, and then output the next output integer using <answer> tags, like <answer>0</answer>. Input sequence: <TRANSDUCER PREFIX> Output sequence:	I have a 3-state DFA model that outputs either 0 or 1 after each element I input. 1 indicates that the input string thus far results in a "valid" state, and 0 indicates that it does not. I collect a set of valid strings using this DFA, listed below. Infer the underlying DFA model using these strings and complete the final string, using up to n characters, such that it is also a valid string. Reason step by step, and then output the next necessary suffix for this final string, <answer> tags, like <answer>ab</answer>. Given these valid strings: <EXAMPLES> Complete the following string: <PREFIX>
RED-GREEN	``` You are in a house of rooms and portals. There are 3 rooms in the house, and each room has 3 unique portals labeled A, B, and C. Each portal teleports you to one room of the house (and sometimes the destination is the room the portal is in). Every portal in a given room always behaves the same way. In this house, each of the rooms look exactly the same, except some of the rooms have red walls and some have green walls. However, there are *three* rooms in total, so you cannot determine which room you are in by color alone, and two rooms of the same color may have portals that behave differently. As you move through the house, at each time step you write down what portal you take and the color of the room you arrive (or stay) in. Based on your notes, predict what color room you will end up in after the last step. Tag your final answer like <answer>color</answer>. You walk through a portal labeled "<TRANSDUCER PREFIX>" and end up in a red room. ```	You are outside a house of rooms and portals. There are 3 rooms in the house, and each room has 3 unique portals labeled a, b, and c. Each portal teleports you to one room of the house (and sometimes the destination is the room the portal is in). Every portal in a given room always behaves the same way. In this house, each of the rooms look exactly the same, except some of the rooms have red walls and some have green walls. However, there are *3* rooms in total, so you cannot determine which room you are in by color alone, and two rooms of the same color may have portals that behave differently. You've been into this house many times before. Each time, as you move through the house, you write down what series of portals you take and the color of the room you end up in. You have a collection of paths you've taken where you've ended up in a room with green walls, listed below. Given the final incomplete path at the bottom, write a series of up to 5 remaining steps that will cause you to end up in a room with green walls again. Tag your final answer like <answer>ab</answer>. Given these paths that end in a room with green walls: <EXAMPLES> Complete the following path: <PREFIX>

Table 5: Shortened summary of each prompt

F.2.1 BASIC$_T$

```
You are a sequence completion model. Output the next element of the sequence, and nothing else.

a, 1, b, 1, a, 1, b, 1, b, 1, c, 0, a, 1, c, 1, a, 1, a, 1, a, 1, c, 1, b, 1, c, 0, c, 1, a, 1, b, 1, b, 1, b,
       1, b, 1, a, 1, b, 1, a, 1, a, 1, b, 1, c, 0, a, 1, c, 1, a, 1, b,
```

F.2.2 MORE-EXPL$_T$

```
You are a sequence completion model. The following sequence is generated from an unknown but consistent
      grammar. Identify the patterns within the sequence to determine its next element. Output the next
      element of the sequence, and nothing else.

a, 1, b, 1, a, 1, b, 1, b, 1, c, 0, a, 1, c, 1, a, 1, a, 1, a, 1, c, 1, b, 1, c, 0, c, 1, a, 1, b, 1, b, 1, b,
       1, b, 1, a, 1, b, 1, a, 1, a, 1, b, 1, c, 0, a, 1, c, 1, a, 1, b,
```

F.2.3 COT$_T$

```
A DFA is a finite-state machine that accepts or rejects a given string of symbols, by running through a n-
      state sequence uniquely determined by the string.

I have a 3-state DFA model that outputs either 0 or 1 after each element I input. 1 indicates that the input
      string thus far results in a "valid" state, and 0 indicates that it does not. I collect the inputs and
      outputs into an input sequence and an output sequence. Infer the underlying DFA model to predict the
      next integer in the output sequence. Reason step by step, and then output the next output integer using
      <answer> tags, like <answer>0</answer>.

Input sequence: a, b, a, b, b, c, a, c, a, a, a, c, b, c, c, a, b, b, b, b, a, b, a, a, b, c, a, c, a, b
Output sequence: 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1,
```

F.2.4 RED-GREEN$_T$

```
```
You are in a house of rooms and portals. There are 3 rooms in the house, and each room has 3 unique portals
      labeled A, B, and C. Each portal teleports you to one room of the house (and sometimes the destination
      is the room the portal is in). Every portal in a given room always behaves the same way.

In this house, each of the rooms look exactly the same, except some of the rooms have red walls and some have
      green walls. However, there are *three* rooms in total, so you cannot determine which room you are in by
       color alone, and two rooms of the same color may have portals that behave differently. As you move
      through the house, at each time step you write down what portal you take and the color of the room you
      arrive (or stay) in. Based on your notes, predict what color room you will end up in after the last step
      .

Tag your final answer like <answer>color</answer>.

You walk through a portal labeled "A" and end up in a green room.
Then, you walk through a portal labeled "B" and end up in a green room.
Then, you walk through a portal labeled "A" and end up in a green room.
Then, you walk through a portal labeled "B" and end up in a green room.
Then, you walk through a portal labeled "B" and end up in a green room.
Then, you walk through a portal labeled "C" and end up in a red room.
Then, you walk through a portal labeled "A" and end up in a green room.
Then, you walk through a portal labeled "C" and end up in a green room.
Then, you walk through a portal labeled "A" and end up in a green room.
Then, you walk through a portal labeled "A" and end up in a green room.
Then, you walk through a portal labeled "A" and end up in a green room.
Then, you walk through a portal labeled "C" and end up in a green room.
Then, you walk through a portal labeled "B" and end up in a green room.
Then, you walk through a portal labeled "C" and end up in a red room.
Then, you walk through a portal labeled "C" and end up in a green room.
Then, you walk through a portal labeled "A" and end up in a green room.
Then, you walk through a portal labeled "B" and end up in a green room.
Then, you walk through a portal labeled "B" and end up in a green room.
Then, you walk through a portal labeled "B" and end up in a green room.
Then, you walk through a portal labeled "B" and end up in a green room.
Then, you walk through a portal labeled "A" and end up in a green room.
Then, you walk through a portal labeled "B" and end up in a green room.
Then, you walk through a portal labeled "A" and end up in a green room.
Then, you walk through a portal labeled "A" and end up in a green room.
Then, you walk through a portal labeled "B" and end up in a green room.
Then, you walk through a portal labeled "C" and end up in a red room.
Then, you walk through a portal labeled "A" and end up in a green room.
Then, you walk through a portal labeled "C" and end up in a green room.
Then, you walk through a portal labeled "A" and end up in a green room.
Then, you walk through a portal labeled "B" and end up in a ...
```
```

F.2.5 BASIC$_S$

```
The following strings come from an alien language that follows a simple grammar. Infer the alien grammar using
      the example strings. Then, add a suffix to the final string using between 1 and 5 characters such that
      the full string follows the grammar. Output only the necessary suffix to complete the final string, and
      nothing else.

cbcbabbcca
abcaaacbaa
aabccbabbb
bbbccbbbca
aababaccba
aaaacbacac
baacbccbaa
cbbaacabcc
baabaacaab
bbbbbcacab
acaabcbbba
acaacbccac
cacbabcbba
abcbcbcbcc
ccaccccaba
bcbcabbcca
baabacabca
caababacac
bacacaccaa
bcacbbbbca
bcbbbcaccc
ccabbcccbb
bccbcabbca
baacbabcbc
ccacabccab
caacbcaaab
cacbaaccac
aaccbcaabb
abacabcaab
bacbcbcaca
caacb
```

F.2.6 BASIC-COMMAS$_S$

```
The following strings come from an alien language that follows a simple grammar. Infer the alien grammar using
      the example strings. Then, add a suffix to the final string using between 1 and 5 characters such that
      the full string follows the grammar. Output only the necessary suffix to complete the final string, and
      nothing else.

c, b, c, b, a, b, b, c, c, a
a, b, c, a, a, a, c, b, a, a
a, a, b, c, c, b, a, b, b, b
b, b, b, c, c, b, b, b, c, a
a, a, b, a, b, a, c, c, b, a
a, a, a, a, c, b, a, c, a, c
b, a, a, c, b, c, c, b, a, a
c, b, b, a, a, c, a, b, c, c
b, a, a, b, a, a, c, a, a, b
b, b, b, b, b, c, a, c, a, b
a, c, a, a, b, c, b, b, b, a
a, c, a, a, c, b, c, c, a, c
c, a, c, b, a, b, c, b, b, a
a, b, c, b, c, b, c, b, c, c
c, c, a, c, c, c, c, a, b, a
b, c, b, c, a, b, b, c, c, a
b, a, a, b, a, c, a, b, c, a
c, a, a, b, a, b, a, c, a, c
b, a, c, a, c, a, c, c, a, a
b, c, a, c, b, b, b, b, c, a
b, c, b, b, b, c, a, c, c, c
c, c, a, b, b, c, c, c, b, b
b, c, c, b, c, a, b, b, c, a
b, a, a, c, b, a, b, c, b, c
c, c, a, c, a, b, c, c, a, b
c, a, a, c, b, c, a, a, a, b
c, a, c, b, a, a, c, c, a, c
a, a, c, c, b, c, a, a, b, b
a, b, a, c, a, b, c, a, a, b
b, a, c, b, c, b, c, a, c, a
c, a, a, c, b,
```

F.2.7 MORE-EXPL$_S$

```
I have a 3-state DFA model that outputs either 0 or 1 after each element I input. 1 indicates that the input
      string thus far results in a "valid" state, and 0 indicates that it does not. I collect a set of valid
      strings using this DFA, listed below. Infer the underlying DFA model using these strings and complete
      the final string, using up to n characters, such that it is also a valid string. Output only the
      necessary suffix to complete the final string, and nothing else.
cbcbabbcca
abcaaacbaa
aabccbabbb
bbbccbbbca
aababaccba
```

```
aaaacbacac
baacbccbaa
cbbaacabcc
baabaacaab
bbbbbcacab
acaabcbbba
acaacbccac
cacbabcbba
abcbcbcbcc
ccaccccaba
bcbcabbcca
baabacabca
caababacac
bacacaccaa
bcacbbbbca
bcbbbcaccc
ccabbcccbb
bccbcabbca
baacbabcbc
ccacabccab
caacbcaaab
cacbaaccac
aaccbcaabb
abacabcaab
bacbcbcaca
caacb
```

F.2.8 COT_S

```
I have a 3-state DFA model that outputs either 0 or 1 after each element I input. 1 indicates that the input
     string thus far results in a "valid" state, and 0 indicates that it does not. I collect a set of valid
     strings using this DFA, listed below. Infer the underlying DFA model using these strings and complete
     the final string, using up to n characters, such that it is also a valid string. Reason step by step,
     and then output the next necessary suffix for this final string, <answer> tags, like <answer>ab</answer
     >.

Given these valid strings:
cbcbabbcca
abcaaacbaa
aabccbabbb
bbbccbbbca
aababaccba
aaaacbacac
baacbccbaa
cbbaacabcc
baabaacaab
bbbbbcacab
acaabcbbba
acaacbccac
cacbabcbba
abcbcbcbcc
ccaccccaba
bcbcabbcca
baabacabca
caababacac
bacacaccaa
bcacbbbbca
bcbbbcaccc
ccabbcccbb
bccbcabbca
baacbabcbc
ccacabccab
caacbcaaab
cacbaaccac
aaccbcaabb
abacabcaab
bacbcbcaca

Complete the following string:
caacb
```

F.2.9 RED-GREEN_S

```
You are outside a house of rooms and portals. There are 3 rooms in the house, and each room has 3 unique
     portals labeled a, b, and c. Each portal teleports you to one room of the house (and sometimes the
     destination is the room the portal is in). Every portal in a given room always behaves the same way.

In this house, each of the rooms look exactly the same, except some of the rooms have red walls and some have
     green walls. However, there are *3* rooms in total, so you cannot determine which room you are in by
     color alone, and two rooms of the same color may have portals that behave differently. You've been into
     this house many times before. Each time, as you move through the house, you write down what series of
     portals you take and the color of the room you end up in. You have a collection of paths you've taken
     where you've ended up in a room with green walls, listed below. Given the final incomplete path at the
     bottom, write a series of up to 5 remaining steps that will cause you to end up in a room with green
     walls again.

Tag your final answer like <answer>ab</answer>.
```

```
Given these paths that end in a room with green walls:
cbcbabbcca
abcaaacbaa
aabccbabbb
bbbccbbbca
aababaccba
aaaacbacac
baacbccbaa
cbbaacabcc
baabaacaab
bbbbbcacab
acaabcbbba
acaacbccac
cacbabcbba
abcbcbcbcc
ccaccccaba
bcbcabbcca
baabacabca
caababacac
bacacaccaa
bcacbbbbca
bcbbbcaccc
ccabbcccbb
bccbcabbca
baacbabcbc
ccacabccab
caacbcaaab
cacbaaccac
aaccbcaabb
abacabcaab
bacbcbcaca

Complete the following path:
caacb
```