What Makes Instruction Learning Hard? An Investigation and a New Challenge in a Synthetic Environment

Matthew Finlayson **Kyle Richardson** **Ashish Sabharwal** **Peter Clark**

Allen Institute for AI, Seattle, WA
{matthewf,kyler,ashishs,peterc}@allenai.org

Abstract

The instruction learning paradigm—where a model learns to perform new tasks from task descriptions alone—has become popular in research on general-purpose models. The capabilities of large transformer models as instruction learners, however, remain poorly understood. We use a controlled synthetic environment to characterize such capabilities. Specifically, we use the task of deciding whether a given string matches a regular expression (viewed as an instruction) to identify properties of tasks, instructions, and instances that make instruction learning challenging. For instance, we find that our model, a fine-tuned T5-based text2text transformer, struggles with large regular languages, suggesting that less precise instructions are challenging for models. Instruction executions that require tracking longer contexts of prior steps are also difficult. We use our findings to systematically construct a challenging instruction learning dataset, which we call Hard RegSet. Fine-tuning on Hard RegSet, our large transformer learns to correctly interpret (with at least 90% accuracy) only 65.6% of test instructions, and 11%-24% of the instructions in out-of-distribution generalization settings. We thus propose Hard RegSet as a challenging instruction learning dataset, and a controlled environment for studying instruction learning.[1]

Train			Test		
Instruction (RegEx)	Data (String)	Result (T/F)	Instruction (RegEx)	Data (String)	Result (T/F)
a∗b	aaa	F	(a∗b)∗	aabab	T
a∗b	aab	T	(a∗b)∗	aba	F
(ab)∗	aab	F	(a∗b)∗	aab	T
(ab)∗	abab	T	a∗	aab	F
(ab)∗	aabab	F	a∗	aaa	T

Table 1: We test a model's ability to learn an instruction language (here, of RegExs) by training on examples of instruction + data pairs, then testing on novel instructions. Each RegEx can be seen as an instruction for a different matching task. Note that *no* examples of the test RegExs are seen during training; rather the model must interpret the RegEx instructions themselves to understand the test tasks.

1 Introduction

Recent years have seen an increased interest in instruction learning (Weller et al., 2020) where a model learns to perform unseen tasks at test time in a zero-shot manner given only a prompt containing instructions. This style of learning is an important feature of flexible and general intelligent systems.

Instruction learning stands in contrast to the traditional machine learning paradigm of *example learning*. In example learning, the model has access to input-output pairs during training time. For instance, in sentiment analysis with the goal to classify product reviews, the model has access to many examples of labeled reviews from which to learn the task. In instruction learning, a model that has never seen labeled sentiment analysis data must perform sentiment analysis given only the explicit instruction "Tell me whether this review is positive". In other words, the model learns to interpret the instruction language (here English) in order to execute an instruction for a task it has never seen.

Most recent work on instruction learning has been conducted in the context of natural language instructions (e.g., Wei et al., 2022; Mishra et al., 2022; Sanh et al., 2022; Zhong et al., 2021). In this context, the complexity of natural language makes it difficult to draw clear conclusions about the kinds of instructions transformers can learn to interpret. To remedy this, we adopt a synthetic approach by building a controlled instructional environment based on interpreting regular expressions (RegExs), and use well-studied properties of RegExs to characterize transformer instruction learning capabilities. Importantly, while findings on synthetic data do not necessarily translate to the real world, our work identifies potentially in-

[1]Data: https://github.com/allenai/RegSet

Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, pages 414 - 426
December 7-11, 2022

teresting hypotheses that can be further investigated on natural data. Previous work on datasets such as SCAN (Lake and Baroni, 2018) and RuleTaker (Clark et al., 2020) has also taken a similar approach of simplifying a difficult, fuzzy problem into a synthetic one. Synthetic generation is a low-cost alternative to large-scale dataset building that allows a high level of formal precision that is infeasible with human-authored data. This allows us to distill atomic characteristics of instruction following in the synthetic domain and make informed conjectures about the natural language setting.

A RegEx is a specification of a formal language, i.e., a set of strings of symbols. To avoid the confusion between an instructional language such as English and a formal language specified by a RegEx, we refer to the latter as an *r-language* (for regular language). In our work, we view a RegEx as an instruction for the task of deciding whether a string belongs to the r-language of the RegEx. We choose to study RegExs because they are well known, unambiguous (the r-language decision problem is binary and always well-defined), and easy to compute (there exist linear-time algorithms to recognize whether a string belongs to a RegEx), while also incorporating fundamental computational operations including iteration, disjunction, conjunction, and nesting. This environment allows us to study instruction learning phenomena more precisely.

We construct RegSet, a RegEx instruction learning dataset (§4) with instances of the form (RegEx, string) $\rightarrow$ T/F, and find that a large T5-based text2text model fine-tuned on RegSet is unable to correctly interpret many instructions in the test set. Inspecting the r-languages, RegExs, and string instances, we identify a number of properties that predict which RegEx instructions the model struggles with (§6). Selecting RegExs with these properties, we construct a hard variant of the dataset which we call Hard RegSet (§7). Our fine-tuned model achieves good performance on only 65.6% of the test RegExs in Hard RegSet, leaving room for improvement.

We find that instruction learning models struggle with non-starfree r-languages. This provides evidence that even large Transformers struggle with periodic r-languages, a theoretical limit of transformers' attention mechanism (Hahn, 2020) that Bhattamishra et al. (2020) further study in smaller models, in the example learning setting.

Our findings in §6 suggest four general implications that we expect will extend beyond the synthetic RegEx environment. First, instruction learning is harder when the underlying *tasks* require modular counting (e.g., keeping track of whether a quantity is even or odd). Second, it gets harder as the *instructions* themselves become less precise, that is, they can be executed correctly in several different ways, forcing the execution engine to make and track choices. Third, as expected, it's harder for instructions involving executing and composing many individual operations or steps. Lastly, it's harder when the correct *execution* of instructions requires keeping in memory a larger context of the partial execution thus far and making choices dependent on this longer history.

In summary, our main contributions are:

- We build the first (to our knowledge) fully synthetic environment for systematically investigating instruction learning.
- We identify properties that make RegEx instruction learning hard for a large T5-based model and suggest broader implications.
- We show that limitations known for small transformers in example learning also apply to large models in the instructional setting.
- We construct a challenging dataset (RegSet) based on our findings to serve as a benchmark for future instruction learning models.

2 Related work

Learning instructions. Large language models like GPT3 (Brown et al., 2020) have some ability to understand task instructions expressed in natural language (Efrat and Levy, 2020). This has led to research on whether models can be fine-tuned to follow instructions reliably, creating the sub-field of instruction learning. Our formulation and approach aligns with Weller et al. (2020) who build ZEST, a benchmark for natural language instruction following, and report that a fine-tuned T5 model (Raffel et al., 2020) does poorly. We differ in the use of a highly-controlled synthetic environment that allows analyzing formal properties of instructions.

Numerous studies (Mishra et al., 2022; Zhong et al., 2021; Wei et al., 2022; Sanh et al., 2022) have followed up with improved models. Though our work differs in domain and goals, we adopt their approach of fine-tuning on instruction-annotated datasets to train an instruction learner.

Earlier work has shown that transformers can emulate (the results of) deductive reasoning over

a set of natural language-like rules with limited expressivity (Clark et al., 2020). Similarly, RegEx instructions include compositions of simple operations such as disjunction and concatenation as well as more complex rules such as iteration (the Kleene star "*" operator) and nesting ("()").

Compositional generalization. Because instruction learning requires the model to generalize to new tasks with new instructions, our work draws on the rich body of literature on compositional generalization. For instance, the SCAN dataset (Lake and Baroni, 2018) revealed significant shortcomings in neural sequence models' systematic generalization ability. Subsequent work has attempted to identify properties of datasets (Keysers et al., 2020; Shaw et al., 2021) and instances (Bogin et al., 2022; Tamari et al., 2021) that make generalization hard. We build upon this by introducing the instructional setting for studying compositional generalization. In addition, we use our setting to test the hypothesis put forward by Bogin et al. (2022) that unseen local structures make generalization harder.

Richardson and Sabharwal (2022) use well-studied computational problems (SAT) to sample hard instances for deductive reasoning tasks. We share the goal of using formal properties to generate hard examples. However, instead of using known hardness characterizations to sample instances, we investigate what makes examples hard.

RegEx expressions and transformers. A number of studies use formal language theory to investigate the computational power of transformers (see Merrill (2021) for a review). This includes theoretical work (Hahn, 2020) and empirical studies (Bhattamishra et al., 2020). Many of these studies focus on specific r-languages and give evidence that certain types of r-languages are hard for transformers to learn. The empirical studies tend to use small, toy-sized transformers (e.g., Bhattamishra et al., 2020). Our work investigates findings from these empirical studies under a setting where instead of learning a single r-language with a model, we task the model with learning how to interpret r-languages in general. Additionally, we use a commonly used large transformer model with much higher capacity than the ones used in these studies.

3 RexEx instruction learning task

3.1 Regular languages and expressions

We use standard terminology for regular languages and expressions (e.g., Harrison, 1978), briefly describing key concepts useful for understanding our results. For completeness, Appendix A includes formal definitions and relevant properties.

We work with strings over the alphabet $\{\mathsf{a}, \mathsf{b}\}$. A set of such strings is called an r-language if it's the empty set; a singleton set containing a, b, or the empty string ε; the union or element-wise concatenation of two r-languages; or the *Kleene star* of an r-language, defined as zero or more occurrences of elements from that r-language.

A regular *expression* (RegEx) is an succinct, non-unique specification of an r-language. Its string nature makes it suitable for text2text models. We denote the language specified or "expressed" by a RegEx r as L_r. The RegEx a represents the singleton r-language $\{\mathsf{a}\}$ (similarly for b and ε), $r|s$ represents the union of L_r and L_s, rs represents the element-wise concatenation of L_r and L_s, and $r*$ represents the Kleene star of L_r. Parentheses indicate the order of operations, e.g., in $(\mathrm{a}|\mathrm{bab})\mathrm{b}*$. We refer to union, concatenation, and Kleene star as *compositional operators*.

3.2 Instruction learning formalism

We formalize *instruction learning*, in contrast to the paradigm of learning from input-output examples.

We view instruction learning as the ML approach to *instruction following*, which seeks to interpret an instruction language to solve *novel tasks*. Both instruction following and instruction learning require an instruction language $\mathcal{R}$ that can be used to describe tasks, and a set of tasks $T = \{t_1, \ldots, t_m\}$. Each task $t_j \in T$ is paired with an instruction $r_j \in \mathcal{R}$ and examples $D_j = \{(x_{ji}, y_{ji}) \mid 1 \leq i \leq N_j\}$. Thus instruction following maps the description r_j of a novel task and an input x_{ji} to the correct output y_{ji} according to the task.

Instruction learning learns this mapping by training a model on data. A key aspect that makes this feasible is compositionality in $\mathcal{R}$, i.e., pieces of $\mathcal{R}$ can be learned at training time and combined in new ways at test time. The training data for instruction learning consists of a subset $T^{\text{train}} \subsetneq T$ of the tasks, where each task $t_j \in T^{\text{train}}$ is specified via its descriptive instruction $r_j \in \mathcal{R}$ and input-output examples D_j. At test time, one is given the

description and an input for an unseen task from $T^{\text{test}} = T \setminus T^{\text{train}}$.

In this notation, standard example learning corresponds to the (degenerate) case where $T^{\text{train}} = T = \{t_1\}$, $T^{\text{test}} = T^{\text{train}}$, and $r_1 = \epsilon$ is the empty string denoting the *null* instruction. In other words, the model has no descriptive instruction available to help learn t_1; it must be learned solely from input-output examples.[2] *Prompt based zero-shot models* can be viewed as a (degenerate) case at the other extreme, where each t_j has the associated prompt as the instruction r_j, but no input-output examples, i.e., $D_j = \phi$. In this case, the model must already understand the instruction (prompt) language $\mathcal{R}$ to perform. In practice, prompt models are large pre-trained language models, and $\mathcal{R}$ is the natural language it is trained on (e.g., English). In turn, standard multi-task learning corresponds to T having multiple tasks t_j where instructions r_j may be either null or a short prefix identifying the task (e.g., Raffel et al., 2020). However, test time tasks are the same as those learnt during training time ($T^{\text{test}} = T^{\text{train}}$).

Prior work on RegEx learning (Bhattamishra et al., 2020) has focused on learning a model for a *specific* RegEx (e.g., well-studied regular language classes such as parity languages, which can be characterized as a single RegEx). In our notation, this corresponds to the single-task setup mentioned earlier, where $T^{\text{train}} = T^{\text{test}} = \{t_1\}$ and $r_1 = \epsilon$. Similarly, works such as SCAN (Lake and Baroni, 2018) can also be viewed naturally as one of two (degenerate) extreme cases of instruction learning, as follows. Let $z_1, z_2, \ldots$ denote SCAN inputs such as "jump twice". We can view SCAN as involving only one meta-task ($T = \{t_1\}$) with an (implicit) instruction r_1 conveying "convert the input to a sequence of executable steps", and with task inputs $x_{1i} = z_i$. Alternatively, one can view SCAN as having as many distinct tasks $T = \{t_1, t_2, \ldots\}$ as SCAN inputs, the associated instructions being $r_j = z_j$, and the task inputs x_{ji} being null (i.e., $D_j = \phi$) because, in this view, r_j fully determines the output y_{ji} (i.e., instruction r_j can be executed in exactly one way, leading to the output y_{ji}).

Applying our formalism to our setting, we view $\mathcal{R}$ as the language consisting of all RegExs, i.e., an instruction is a RegEx. Each task t_j is associated with a specific r-language L_j and involves deciding whether an input string belongs to L_j. The instruction r_j associated with t_j is a RegEx[3] describing L_j. The input-output examples consist of strings labeled as 1 (string is in L_j) or 0 (not in L_j).

4 RegEx datasets

Our datasets consist of collections of triples according to the formulation given in §3.2, where each triple contains a RegEx r, a string $x \in \{\mathsf{a}, \mathsf{b}\}^*$, and a label $\ell \in \{0, 1\}$ indicating whether $x \in L_r$. Some examples are given in Table 1. The task is to predict ℓ given (r, x) as input.

We sample RegExs r such that all corresponding r-languages L_r are distinct within the dataset. Furthermore, r is chosen such that it uses the minimum number of compositional operators needed to represent L_r. See Appendix B for details of our RegEx sampling method. Constraining the RegExs to express unique r-languages avoids over-representing only a few r-languages, as would happen with a naive sample.

We limit number of compositional operators in each RegEx to at most 6, sampling approximately uniformly at random with respect to the number of operators. For each r, we sample both strings that match r and strings that do not.[4] We limit string length to at most 15 and sample from each length approximately uniformly at random.

Exploration RegSet. To investigate which attributes make RegEx instruction learning hard, we construct an training set of RegExs and randomly sample a large test set from it. We refer to this as Exploration RegSet.

Our training set contains 1000 RegExs, each with 20 strings. We choose our RegEx-to-string ratio by selecting the best model given a budget of 20K training examples. Because the number of strings in (or not in) an r-language is sometimes less than 10, it is impossible to fully balance the data set. However, we choose the maximum number of strings from the minority class up to 10 so that the dataset is as balanced as possible while maintaining 20 strings per RegEx. We also set aside 200 validation RegExs for model selection.

Our test set contains 500 RegExs, disjoint from the training set. We sample more strings to approx-

[2] In standard learning, including an identical instruction with every example is not helpful as it does not form a discriminative feature.

[3] Our instructions are not natural language. In principle, one can convert a RegEx instruction to an equivalent, if cumbersome, English description.

[4] For two special cases, the empty language and the language of all strings, our sampled strings purely "do not match" and "match" the RegEx, respectively.

imate an exhaustive test for each RegEx, sampling $\min(|L_r|, |L_r^c|, 1000)$ strings per RegEx.[5] On average, each RegEx includes 200 strings.

We use the §6 analysis of Exploration RegSet to create a hard version the dataset which we call **Hard RegSet**, detailed in §7.

5 Experimental setup

Model. We select ByT5-Large (1.2B parameters) (Xue et al., 2022) as our base model[6]. ByT5 is a T5-based pre-trained transformer model with character-level tokenization—which is helpful in avoiding issues with tokenizing synthetic strings. We use a pre-trained model because pre-training often imbues performance benefits even for unrelated tasks (Krishna et al., 2021; Maennel et al., 2020).[7]

We feed r x to the model as a string (separated with a space), and train the model to output the strings "True" or "False". We train for 200 epochs, using validation accuracy to select the best model, using a learning rate of $5 \cdot 10^{-5}$ and batch size 32.[8]

Metrics. Consider a RegEx evaluation set D, and a model M. We define M's accuracy on an instance $(r, x) \in D$, denoted $\text{acc}_M(r, x)$, as 1 if M correctly predicts whether x matches r, and 0 otherwise. M's accuracy on a RegEx r is $\text{acc}_M(r) = \text{mean}_{x:(r,x)\in D}\, \text{acc}_M(r, x)$.

To measure how well M learns to interpret each RegEx we use metrics that operate at the level of RegExs. Due to the synthetic—and thus noise-free—nature of our datasets, we expect M to *perfectly* interpret every expression given sufficient training data, at least in the i.i.d. setting and also in reasonable generalization settings. To measure how well M does relative to this upper bound, we define two metrics that give equal weight to all RegExs in D, regardless of how many test strings each has.

Our main metric is **Mean RegEx Performance at k**, defined as the fraction of RegExs on which M's accuracy is at least k (treated as a percentage):

$$\text{perf}_M@k = \underset{r\in D}{\text{mean}}\; I\big(\text{acc}_M(r) \geq k\big) \qquad (1)$$

where $I(\cdot)$ denotes the indicator function[9] and, with slight abuse of notation, we use $\{r \in D\}$ as a shorthand for $\{r \mid (r, x) \in D\}$. We will drop the subscript M from perf_M when the model is clear from the context. perf@100 thus refers to the fraction of RegExs r that are learned *perfectly* (as assessed by all strings tested for r in D). Since this metric is somewhat strict from a machine learning perspective, we use **perf@90** as our main metric, and also track a more lenient metric, perf@80.

We also report a secondary metric, **Mean RegEx Accuracy**, defined as M's accuracy on a RegEx r, averaged across all r present in D: $\text{acc}_M = \text{mean}_{r\in D}\, \text{acc}_M(r)$. As before, we drop the subscript M from acc_M when the model is clear from the context. Note that this metric does not distinguish the case of two RegExs learned to accuracies of 90% and 50% from two RegExs learned to an accuracy of 70% each. Further, when each RegEx in the evaluation set has the same number of strings, acc_M simplifies to the standard instance-level accuracy of the dataset rather than a RegEx-level metric. For these reasons, this metric is less desirable, but we include it for completeness.

Attribute	Exploration	Hard
Regexs (#)	1,000	1,000
Instances/RegEx (mean)	20	20
Starfree (%)	86.1	0
R-lang. size (med)	8	368
Compositions (mean)	4.9	6.0
ES (mean)	3.4	5.4
String length (mean)	7.6	10.5

Table 2: Attribute statistics for Exploration and Hard training sets. Number of unseen sub-expressions is omitted as it is defined for a test set w.r.t. a training set.

Attribute classes	Class 1	Class 2
Starfree/non-starfree	91.9	**71.9**
Small/big r-language	95.1	**57.5**
Low/high composition	95.2	80.3
None/has unseen exprs	95.9	87.5

Table 3: Summary of perf@90 results for language-level (top 2) and expression-level (bottom 2) attributes. Language-level attributes we measure have a larger impact on performance than expression-level attributes.

6 Results: Which instructions are hard?

We train our model on the Exploration Train set and evaluate on the Exploration Test set. The model achieves a high mean RegEx accuracy acc of 97.1%. The number of RegExs learned to at least 90% accuracy (perf@90), however, is a more modest 89.6%. By design, the Exploration test

[5] L_r^c denotes the complement of L_r

[7] In our experiments, we find that in-context learning with GPT-3 (Brown et al., 2020) did no better than random guessing, even when RegExs were converted to a more natural-language-like form.

[8] Not surprisingly, ByT5 without such training does no better than random guessing.

[9] $I(c)$ is 1 if the condition c is satisfied and 0 otherwise.

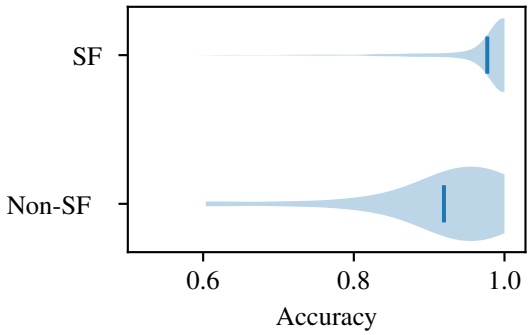

Figure 1: Non-starfree r-languages (Non-SF) are harder for our model than starfree r-languages (SF). RegEx accuracy depicted using a violin plot—the blue region represents accuracy distribution across RexExs, with a vertical blue bar showing the mean accuracy.

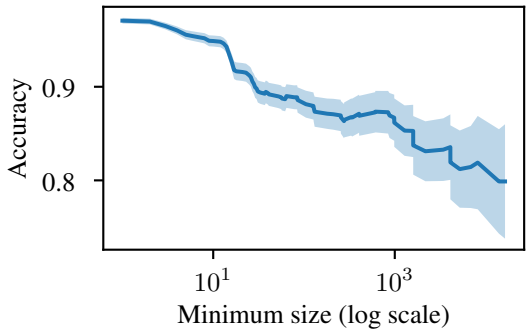

Figure 2: The size of an r-language is highly predictive of difficulty: mean accuracy decreases as language size increases.

set's large size allows us to analyze errors in detail and assess which attributes contribute to the hardness of instruction learning. Table 2 summarizes attribute statistics within the dataset, and Table 3 summarizes perf@90 scores for various attributes. We next define these attributes and discuss our findings.

6.1 Language-level attributes

We coin "language-level attributes" to refer to r-languages properties that are invariant to how the r-language is expressed. For instance, a$*$ and aa$*|$(aa)$*$ express the same r-language, and thus "presence of a union operator" is not a language-level attribute. On the other hand, the number of strings in the r-language is a language-level attribute because it remains the same regardless of how the language is expressed as a RegEx.

6.1.1 Non-starfree r-languages are hard.

A well-known r-language complexity measure is whether it is *starfree* (McNaughton and Papert, 1971). An r-language is starfree if there exists a RegEx for the r-language whose operators include only union, concatenation, and set complement (notably, these operators exclude the Kleene star). For instance, a$*$ can also be expressed as $(\emptyset^c \text{b} \emptyset^c)^c$,[10] so the r-language expressed by a$*$ is starfree. Previous work (Bhattamishra et al., 2020) has shown that small transformer models struggle to generalize when trained on a non-starfree r-language (whereas LSTMs are able to generalize perfectly).

[10]The r-language corresponding to r^c is the set complement of L_r, the r-language of r.

We find that non-starfree expressions are significantly harder to interpret for our model, as shown in Figure 1. We conjecture that these RegExs are harder for our model for the same reason small transformers fail to learn and generalize simple non-starfree r-languages such as $(aa)*$ (Bhattamishra et al., 2020). This also aligns with the theoretical prediction that transformers struggle to model periodicity—a common feature among non-starfree r-languages (Hahn, 2020). As our own addition to this body of work, our results show that, despite increased capacity, large models (as used in practice, without simplifying assumptions) still struggle with non-starfree r-languages under the instructional setting. Our results suggest that even large transformers struggle with instructions that require modeling periodicity and modular counting, e.g., keeping track of whether a quantity is even or odd.

6.1.2 Bigger r-languages are harder.

Since an r-language is a set of strings, one attribute of interest is the size of the set. For many r-languages, this size is infinite. For practicality and to distinguish between r-languages of infinite size, we define the *size* of an r-language to be the number of strings in the set with length up to 15. By this definition, the maximum size of an r-language in our dataset is 2^{15}. The median r-language size in the Exploration training set is 8.

Figure 2 shows that our model struggles with r-languages with more strings: perf@90 drops from 95.1 for small r-languages to only 57.5 for large. Small r-languages have a very narrow scope of interpretation (only a few strings match the specification). They can thus be viewed as relatively *precise instructions*. Based on our findings, we

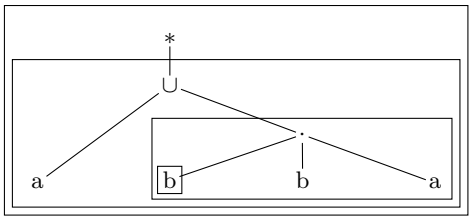

Figure 3: Regex $(a|bba)*$ contains sub-expression $a|bba$, which contains sub-expression bba, etc.

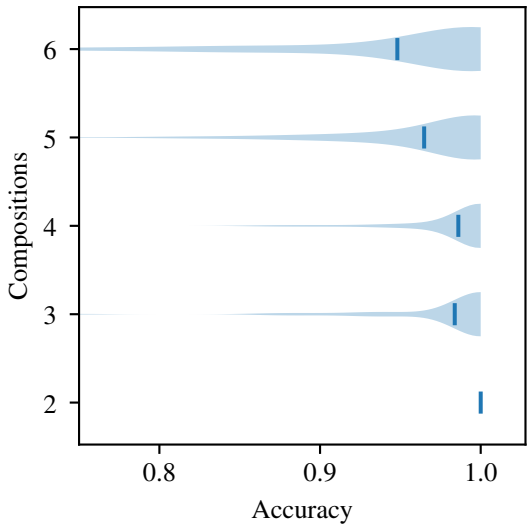

Figure 4: More compositions make expressions somewhat harder. High-composition RegExs (top violins) have slightly lower accuracy than low-composition RegExs (bottom).

postulate that more precise instructions with fewer possible interpretations are, in general, easier for models.

6.2 Expression-level attributes

Expression-level attributes are specific to a RegEx, but not the r-language they express. For instance $a*$ and $aa*|(aa)*$ express the same r-language, but only the latter uses a union operator. Thus having a union operator is an expression-level attribute.

6.2.1 More composed expressions are harder.

RegExs are constructed by recursively composing smaller RegExs together, e.g., Figure 3. To study the effect of the amount of composition on RegEx difficulty, we define the amount of *composition* in an expression as the number of operators used to construct it, e.g., the expression $(a|bba)*$ has 4 compositions: 1 star, 1 union, and 2 concatenations.

Figure 4 shows that accuracy decreases only slightly for expressions with more composition.

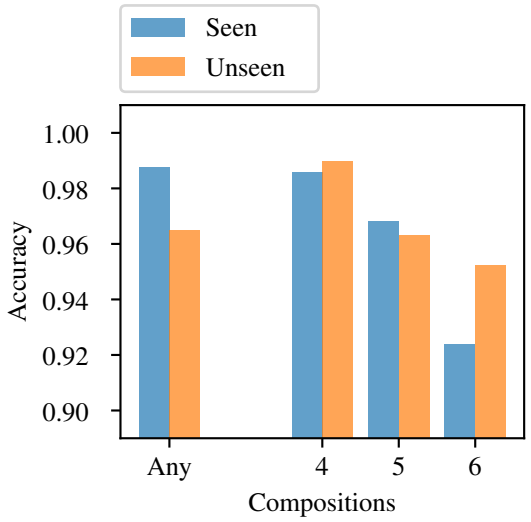

Figure 5: Without controlling for amount of composition (left), unseen sub-expressions appear to make expressions harder. However, controlling for amount of composition (right clusters), these effects disappear. Blue bars indicates mean accuracy for expressions without unseen subexpressions, orange for ≥ 1.

This aligns well with intuition because more composed strings tend to be longer and the complexity of the RegEx string recognition algorithm scales linearly with input length (Thompson, 1968).

Speculating beyond the RegEx domain, we hypothesize that instructions composed of many sub-instructions are more challenging for models like T5 compared to less compositional instructions, but that other factors (like specificity or periodicity) likely play a larger role in determining difficulty.

6.2.2 Unobserved local structures do not contribute significantly to hardness.

Prior work (Lake and Baroni, 2018; Keysers et al., 2020; Bogin et al., 2022) shows that models often fail to generalize to new compositions of atomic components, even when all the components have been seen during training time. To investigate this in our own datasets, we define the *unseen sub-expressions* of a RegEx w.r.t. a dataset as the set of sub-expressions contained in the RegEx that are not contained in any of the RegExs in the dataset. 33.4% of the expressions in the Exploration test set contain sub-expressions not seen in the training set.

Upon first inspection, RegExs with unseen sub-expression appear harder, which would support for our hypothesis. However, there is a confounding variable: deeper RegExs are more likely to

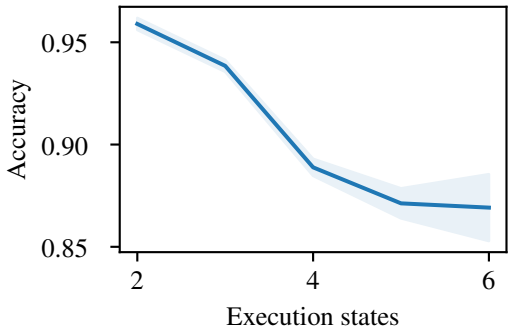

Figure 6: Accuracy decreases for high-ES strings. We group test set instances by ES and plot average accuracy (filled in area shows standard error), leaving off groups with fewer than 100 instances.

have unseen substructures. When we control for the compositions in the RegEx, the observed effect disappears and even reverses[11] (see Figure 5). We conclude that unseen local structures do not make instruction learning hard in the RegEx domain. There are many possible explanations for this, including the possibility our model is able to generalize well in settings like ours with very few atomic symbols and operators.

6.3 Instance-level attributes

Instance-level attributes are attributes that depend on the RegEx or r-language *as well as* the string.

6.3.1 Instances requiring many execution states are hard.

An r-language can be equivalently defined as the strings accepted by a deterministic finite automaton (DFA). We define *execution states* (ES) of a string x with respect to a RegEx r to be the number of unique states in L_r's minimal DFA that are visited while recognizing x. This is closely related to the notion of *state-complexity* (Yu, 2001). Since each r-language has a unique minimal DFA, this property is invariant with respect to the RegEx used to express the r-language. We choose this metric as a way to measure how much space is required to execute the string recognition problem, with the intuition that keeping track of more states is harder.

We observe in Figure 6 that performance decreases for RegEx-string pairs where many distinct states are visited by the minimal DFA recognizing the string. Interestingly, *ambiguity*, which can be viewed as an expression-level version of ES, does not produce a good predictor of hardness.

Intuitively, being in many different states means that when processing the next character (say `a`), one must act differently depending on how one arrived at that point, i.e., depending on the prefix of the string up till that character. More states thus implies more of the prior context or history must be remembered and taken into account when processing the next character. Speculating beyond RegExs, we hypothesize that instruction learning is harder when determining the next valid step (while following the instruction) requires considering a longer context of prior steps.

[11] This is an example of Simpson's Paradox (https://en.wikipedia.org/wiki/Simpsons_paradox).

Regex	Accuracy (%)
$b\|(a\|(a\|b)b)^*$	34
$aa(a(a\|b))^*\|a$	44
$(b(b\|ab))^*a^*$	50
$(a\|bbb)^*b\|a$	57
$(b(a^*aa\|b))^*$	60
$((b\|(a\|b)a)b)^*\|a$	61
$b((b\|a)a)^*a$	62
$b\|(b(a\|b))^*$	72

Table 4: The 8 lowest-scoring RegExs obtained by filtering the Exploration test set for large non-starfree r-languages paired with strings with high ES.

7 Hard RegSet: A new challenge

Based on our findings, we select the attributes that contribute most to difficulty, namely starfreeness (or rather, non-starfreeness), r-language size, and ES; following the idea of *salient variable* sampling (Shin et al., 2019). In selecting attributes we balance the trade-off between accuracy reduction and aggressiveness of the filter and ignore attributes with small effects like number of compositions. Filtering the Exploration test set for non-starfree expressions with size > 64 and ES > 4 yields a reasonable sized set of 785 instances from 16 RegExs with $\text{acc}_M = 72.2$. The 8 lowest-scoring RegExs from this group are shown in Table 4. We use these same settings to generate Hard RegSet.

Hard RegSet is split into train, validation, and test sets[12] which match the sizes of the exploration train, validation, and test sets.

[12] Unlike some prior studies (Hendrycks et al., 2021) that only consider a hard *test* set, we also provide the corresponding identically distributed hard *training* and validation sets. This helps rule out confounding factors such as distribution mismatch being the prime reason for the observed hardness.

7.1 Performance on RegSet

Table 5 summarizes the performance of our model on both Exploration and Hard sets. Recall that our main metric in this instruction learning setup is perf@90, i.e., how many RegEx instructions does the model learn with an accuracy of at least 90%. For completeness, we also include Mean RegEx accuracy (acc). The Random baseline, which predicts 0/1 with an equal probability, has an accuracy of 50% and perf@90 of zero.

We see that the model struggles on Hard RegSet even in the in-distribution setting (IID), achieving a perf@90 score of only 65.5%. As noted earlier, the upper bound is essentially 100% due to the programmatic nature of the task. Closing the 34.5% gap thus remains a challenge.

Further, in our out-of-distribution (OOD) settings, we train the model on the Exploration set and test on the Hard set, and vice versa. Here the model achieves perf@90 scores of only 23.4% and 11.0%, respectively. Even the raw accuracy scores are quite low (77.2% and 66.8%) relative to the Random baseline. In other words, the model really struggles to generalize to OOD RegEx instructions, even though these instructions use the same primitives (the few basic RegEx operators) and have similar syntactic properties (instruction length, etc.) as what the model has seen during training. Notably, the model trained on the Hard set performs very poorly (11.0%) on the Exploration set, demonstrating that its reasonable performance on the Hard set (65.5%) is not a good indication of it actually learning and understanding the primitives of the underlying instruction language, namely, all regular expressions. Thus, generalization to OOD instructions remains an open challenge.

Trng. Set	*Eval. Set*		acc	perf @80	**perf @90**	perf @100
–	Expl.	RND	50.0	0.0	0.0	0.0
–	Hard	RND	50.0	0.0	0.0	0.0
Expl.	Expl.	IID	97.1	96.4	**89.6**	69.9
Hard	Hard	IID	88.9	81.6	**65.6**	15.2
Expl.	Hard	OOD	77.2	52.8	**23.4**	2.0
Hard	Expl.	OOD	66.8	29.3	**11.0**	3.8

Table 5: Performance of the ByT5 model on Exploration and Hard RegSet datasets, in both in-distribution (IID) and out-of-distribution (OOD) settings. RND denotes the uniform random baseline. acc denotes Mean RegEx Accuracy (%) and perf@k the percentage of RegExs with model accuracy at least k% (§5). **perf@90** is our main metric, under which the model struggles (65.6%) on the IID Hard RegSet and performs very poorly (11.0%-23.4%) in the OOD settings.

8 Conclusion

Instruction learning is an important step towards general-purpose AI systems. Understanding the limits and capabilities of current instruction learning models is key to improving them. Our Exploration RegSet dataset provide a controlled environment for discovering what makes instruction learning hard for today's transformers. We identify several such attributes in our setting and use our findings to make informed speculations about the difficulty of instruction learning in general. We also use our findings to systematically construct Hard RegSet, a challenging synthetic instruction learning benchmark. Our T5-large based model leaves much room for improvement on this dataset, failing to meet the 90% accuracy bar in over 34% of the RegExs in an IID setting and faring much worse in our OOD generalization settings. We offer our dataset as a challenge to help the community progress towards building reliable instruction learning systems.

9 Limitations

It is likely that there are attributes not considered in this study that are correlated with the difficulty or ease of RegEx instruction learning. This may explain why our model attains perfect accuracy on as many as 15% of the test RegExs from our Hard RegSet benchmark.

While we make informed speculations about which attributes may make instruction learning hard in a general setting (beyond the RegEx environment), validating our general hypotheses in a natural language instruction learning setting remains an open problem which can only be done via future empirical investigations on actual data.

Lastly, it is possible that much larger models (e.g., T5-11B) will have different qualitative trends on Hard RegSet. We have not evaluated such models. That said, as long as our qualitative findings about which attributes make instruction learning hard continue to hold, it should be possible to construct a scaled up version of Hard RegSet (larger RegExs, larger strings) that challenges even these much bigger models.

References

Satwik Bhattamishra, Kabir Ahuja, and Navin Goyal. 2020. On the ability of self-attention networks to recognize counter languages. In *EMNLP*.

Ben Bogin, Shivanshu Gupta, and Jonathan Berant. 2022. Unobserved local structures make compositional generalization hard. *CoRR*, abs/2201.05899.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, T. J. Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeff Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *NeurIPS*.

Peter Clark, Oyvind Tafjord, and Kyle Richardson. 2020. Transformers as soft reasoners over language. In *IJCAI*.

Avia Efrat and Omer Levy. 2020. The Turking Test: Can language models understand instructions? *ArXiv*, abs/2010.11982.

Michael Hahn. 2020. Theoretical limitations of self-attention in neural sequence models. *TACL*, 8:156–171.

Michael A Harrison. 1978. *Introduction to formal language theory*. Addison-Wesley Longman Publishing Co., Inc.

Dan Hendrycks, Collin Burns, Steven Basart, Andrew Critch, Jerry Zheng Li, Dawn Xiaodong Song, and Jacob Steinhardt. 2021. Aligning ai with shared human values. In *ICLR*.

John Hopcroft. 1971. An n log n algorithm for minimizing states in a finite automaton. In Zvi Kohavi and Azaria Paz, editors, *Theory of Machines and Computations*, pages 189–196. Academic Press.

Daniel Keysers, Nathanael Schärli, Nathan Scales, Hylke Buisman, Daniel Furrer, Sergii Kashubin, Nikola Momchev, Danila Sinopalnikov, Lukasz Stafiniak, Tibor Tihon, Dmitry Tsarkov, Xiao Wang, Marc van Zee, and Olivier Bousquet. 2020. Measuring compositional generalization: A comprehensive method on realistic data. In *ICLR*.

Kundan Krishna, Jeffrey P. Bigham, and Zachary Chase Lipton. 2021. Does pretraining for summarization require knowledge transfer? In *Findings of EMNLP*.

Brenden M. Lake and Marco Baroni. 2018. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *ICML*.

Hartmut Maennel, Ibrahim M. Alabdulmohsin, Ilya O. Tolstikhin, Robert J. N. Baldock, Olivier Bousquet, Sylvain Gelly, and Daniel Keysers. 2020. What do neural networks learn when trained with random labels? In *NeurIPS*.

Robert McNaughton and Seymour A Papert. 1971. *Counter-Free Automata (MIT research monograph no. 65)*. The MIT Press.

William Merrill. 2021. Formal language theory meets modern NLP. *arXiv preprint arXiv:2102.10094*.

Swaroop Mishra, Daniel Khashabi, Chitta Baral, and Hannaneh Hajishirzi. 2022. Cross-task generalization via natural language crowdsourcing instructions. In *ACL*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *JMLR*, 21:1–67.

Kyle Richardson and Ashish Sabharwal. 2022. Pushing the limits of rule reasoning in transformers through natural language satisfiability. In *AAAI*.

Victor Sanh, Albert Webson, Colin Raffel, Stephen H. Bach, Lintang A. Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Teven Le Scao, Arun Raja, Manan Dey, M SAIFUL BARI, Canwen Xu, Urmish Thakker, Shanya Sharma Sharma, Eliza Szczechla, Taewoon Kim, Gunjan Chhablani, Nihal V. Nayak, Debajyoti Datta, Jonathan Chang, Mike Tian-Jian Jiang, Han Wang, Matteo Manica, Sheng Shen, Zheng Xin Yong, Harshit Pandey, Rachel Bawden, Thomas Wang, Trishala Neeraj, Jos Rozen, Abheesht Sharma, Andrea Santilli, Thibault Févry, Jason Alan Fries, Ryan Teehan, Stella Rose Biderman, Leo Gao, T. G. Owe Bers, Thomas Wolf, and Alexander M. Rush. 2022. Multitask prompted training enables zero-shot task generalization. In *ICLR*.

Peter Shaw, Ming-Wei Chang, Panupong Pasupat, and Kristina Toutanova. 2021. Compositional generalization and natural language variation: Can a semantic parsing approach handle both? In *IJCNLP*, pages 922–938.

Richard Shin, Neel Kant, Kavi Gupta, Christopher Bender, Brandon Trabucco, Rishabh Singh, and Dawn Song. 2019. Synthetic datasets for neural program synthesis. In *ICLR*.

Ronen Tamari, Kyle Richardson, Aviad Sar-Shalom, Noam Kahlon, Nelson Liu, Reut Tsarfaty, and Dafna Shahaf. 2021. Dyna-bAbI: unlocking bAbI's potential with dynamic synthetic benchmarking. *arXiv preprint arXiv:2112.00086*.

Ken Thompson. 1968. Programming techniques: Regular expression search algorithm. *Commun. ACM*, 11:419–422.

Jason Wei, Maarten Bosma, Vincent Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V Le. 2022. Finetuned language models are zero-shot learners. In *ICLR*.

Orion Weller, Nicholas Lourie, Matt Gardner, and Matthew E. Peters. 2020. Learning from task descriptions. In *EMNLP*.

Linting Xue, Aditya Barua, Noah Constant, Rami Al-Rfou, Sharan Narang, Mihir Kale, Adam Roberts, and Colin Raffel. 2022. ByT5: Towards a token-free future with pre-trained byte-to-byte models. *TACL*, 10:291–306.

Sheng Yu. 2001. State complexity of regular languages. *Journal of Automata, Languages and Combinatorics*, 6(2):221–234.

Ruiqi Zhong, Kristy Lee, Zheng Zhang, and Dan Klein. 2021. Adapting language models for zero-shot learning by meta-tuning on dataset and prompt collections. In *Findings of EMNLP*.

A Regular Languages and Expressions

Following standard definitions from formal language theory (Harrison, 1978), we define a language as a set of strings of symbols from some alphabet Σ. The regular languages over an alphabet Σ are defined as follows:

- The empty set $\emptyset$ is a regular language.
- For each symbol $\sigma \in \Sigma$, the singleton set $\{\sigma\}$ is a regular language.
- The singleton set $\{\varepsilon\}$ is a regular language, where ε is the empty string.
- If A and B are regular languages, then their union $A \cup B$ is a regular language.
- If A and B are regular languages, then the set $\{ab \mid a \in A, b \in B\}$ is a regular language. This new set is called the *concatenation* of A and B and is denoted $A \cdot B$
- If A is a regular language, then the set $\{\varepsilon\} \cup A \cup AA \cup AAA \cup \ldots$ is a regular language. This new set is called the *Kleene star* of A and is denoted A^*.

A regular *expression* (RegEx) is a specification of a regular language. We denote the language specified or "expressed" by a RegEx r as L_r. In a RegEx, a symbol from Σ or the empty string represents its own singleton set, *e.g.* RegEx a represents the language $\{\mathsf{a}\}$. Given two RegExs r and s, the RegEx $r|s$ represents $L_r \cup L_s$, the union of the languages expressed by r and s. Likewise, rs represents the concatenation of the two languages, and $r*$ represents the Kleene star of L_r. Parentheses are used to indicate order of operations, *e.g.* $(\mathrm{a}|\mathrm{bab})\mathrm{b}*$. We refer to union, concatenation, and Kleene star as *compositional operators*.

Additionally, it has been shown that regular languages are closed under set complementation, *e.g.* if L is a regular language, then the set $\{\sigma \in \Sigma^* \mid \sigma \notin L\}$ (denoted L^c) is a regular language. It follows that the complement of a any regular language can be expressed without a dedicated complement operator, however the RegEx may be verbose *e.g.* $(\mathrm{a}|\mathrm{b})*((\mathrm{b}(\mathrm{a}|\mathrm{b}))|((\mathrm{a}|\mathrm{b})\mathrm{b}))(\mathrm{a}|\mathrm{b})*$ expresses $L^c_{\mathrm{a}*|\mathrm{b}}$.

B Sampling RegExs

Algorithm 1 details our method for sampling regular expressions. In summary, we find a RegEx with the minimum number operators to express each language expressible with up to D operators, then sample from this set uniformly at random with respect to number of operators. We achieve this by generating 0-operator RegExs before moving to 1-operator RegExs and so forth, tracking which r-languages have been generated. To compare r-languages, we use their minimal DFA representations which can be constructed via Hopcrofts' algorithm (Hopcroft, 1971). Formally, let $\mathcal{L}_n$ be the set of r-languages expressible with n operators but not expressible with less than n operators. For each n, we find all $L \in \mathcal{L}_n$ and randomly choose an RegEx r with n operators that expresses L. We limit the maximum number of compositional operators in each r to 6, choosing RegExs approximately uniformly at random with respect to the number of operators. Specifically, we separate all possible RegExs into bins based on the number of operators they have. From each bin, we randomly sample the same number of RegExs, with the exception of small bins that contain too few RegExs, in which case we sample as many as we can.

Algorithm 1 Sampling N RegExs with at most D operators. R_d is the set of RegExs with d compositions.

procedure SAMPLE(D, N)
  $L \leftarrow \emptyset$
  $S \leftarrow \emptyset$
  for $d \in 0, 1, \ldots, D$ **do**
    for $r \in R_d$ **do**
      if $L_r \notin L$ **then**
        $S \leftarrow S \cup \{r\}$
        $L \leftarrow L \cup \{L_r\}$
      end if
    end for
  end for
  $T \leftarrow \emptyset$
  for $d \in 0, 1, \ldots, D$ **do**
    $n \leftarrow \min\left(\frac{N}{D-d}, |S \cap R_d|\right)$
    $r_1, r_2, \ldots, r_n \sim \mathrm{Unif}(S \cap R_d)$
    $T \leftarrow T \cup \{r_1, r_2, \ldots, r_n\}$
  end for
  return T
end procedure

C Other attributes

We use our results section to discuss RegEx attributes for which we had a significant findings. We did, however explore other attributes, and we define them here.

Ambiguity. We define *ambiguity* as the maximum number of sub-expressions any token refers

to, as a string is processed in either direction. For instance, given RegEx $r = \mathrm{a}|\mathrm{ab}|\mathrm{abb}$ and string $x = \mathsf{abb}$, processing left to right, $\mathsf{a}\ldots$ could refer to any of the three disjoint sub-expressions, $\mathsf{ab}\ldots$ could refer to either of the last two sub-expressions $(\mathrm{ab}|\mathrm{abb})$, and finally, $\mathsf{abb}\ldots$ could refer only to the last sub-expression (abb). The same can be done considering x's elements in reverse order. We take the minimum value for the two directions to be the *ambiguity* of x w.r.t. r. This metric is intended as an expression-level analogue of ES, as we measure the computational space complexity of the execution, only here we do not allow the implicit conversion to a minimal DFA. We do not find that ambiguity has an significant impact on RegEx difficulty.