Correct and Optimal: the Regular Expression Inference Challenge

Mojtaba Valizadeh[1] , **Philip John Gorinski**[2] , **Ignacio Iacobacci**[2] and **Martin Berger**[1,3]
[1]University of Sussex
[2]Huawei Noah's Ark Lab, London
[3]Montanarius Ltd
Valizadeh.Mojtaba@gmail.com, {philip.john.gorinski, ignacio.iacobacci}@huawei.com,
contact@martinfriedrichberger.net

Abstract

We propose *regular expression inference (REI)* as a challenge for code/language modelling, and the wider machine learning community. REI is a supervised machine learning (ML) and program *optimisation* task, and poses the problem of finding minimal regular expressions from examples: Given two finite sets of strings P and N and a cost function $\mathsf{cost}(\cdot)$, the task is to generate an expression r that accepts all strings in P and rejects all strings in N, while no other such expression r' exists with $\mathsf{cost}(r') < \mathsf{cost}(r)$. REI has advantages as a challenge problem: (i) regular expressions are well-known, widely used, and a natural idealisation of code; (ii) REI's asymptotic worst-case complexity is well understood; (iii) REI has a small number of easy to understand parameters (e.g. P or N cardinality, string lengths of examples, or the cost function); this lets us easily finetune REI-hardness; (iv) REI, with its emphasis on optimisation, is an unsolved problem for deep learning based ML. Recently, an REI solver was implemented on GPUs, using program synthesis techniques. This enabled, for the first time, fast generation of minimal regular expressions for complex REI instances. Building on this advance, we generate and publish the first large-scale datasets for REI, and devise and evaluate several initial heuristic and machine learning baselines. We invite the community to participate and explore ML methods that learn to solve REI problems. We believe that progress in REI directly translates to progress in code/language modelling.

1 Introduction

We propose regular expression inference (REI) as a challenge for machine learning (ML) communities.

Regular expression inference is the task of finding a regular expression (RE) r given a positive set of strings P and negative set of strings N as well as a cost function $\mathsf{cost}(\cdot)$, such that r is *precise* – it accepts all strings in P while rejecting all strings in N – and *minimal* w.r.t. the cost function.

We assume the reader is familiar with REs; here is an example of one that specifies the language of all strings with characters from the alphabet $\{0, 1\}$ that start with 10:

$$10(0 + 1)^*$$

The exact nature of $\mathsf{cost}(\cdot)$ will play a key role later, for now a naive understanding of cost, for example the length of r as a string, is sufficient.

Regular expression inference always has a trivial solution. Consider inferring a regular expression from

$$\begin{array}{ll} \text{P:} & 10,\ 101,\ 100,\ 1010,\ 1011,\ 1000,\ 1001 \\ \text{N:} & \epsilon,\ 0,\ 1,\ 00,\ 11,\ 010 \end{array}$$

Clearly, the union of all the positive examples

$$10 + 101 + 100 + 1010 + 1011 + 1000 + 1001$$

is correct but trivial, and can be seen as *overfitting*. The problem becomes highly non-trivial if we ask for a *minimal* RE such as $10(0 + 1)^*$, assuming a uniform cost function.

In this work, we present the regular expression inference challenge (REIC) as an open-ended challenge to address REI with machine learning/deep learning. We detail the precise nature of REIC, introduce large-scale datasets suitable for supervised training and evaluation, and present baselines including the use of an instruction-tuned pretrained very large code language model, and a first supervised approach. To our knowledge, we are the first to put REI forward as an organised, well-defined and automatically scored challenge. All data and starter code to recreate our baselines is provided via CodaLab [Pavao *et al.*, 2022] on the REIC site:

https://codalab.lisn.upsaclay.fr/competitions/15096

2 Background & Related Work

Regular Expressions are well-known mathematical structures, invented in the context of modelling biological neurons [Kleene, 1956; Hopcroft *et al.*, 2006]. Abstractly, REs are a constrained mechanism for succinct, finite specifications of finite and infinite languages. While all finite languages are definable by REs, they can only specify simple infinite ones.

REs are equivalent not only to regular languages, the least expressive language type in the Chomsky-Schützenberger

Extended version including Technical Appendix available at https://arxiv.org/abs/2308.07899 hierarchy [Chomsky, 1956], but also finite state automata, whether deterministic (DFAs) or non-deterministic (NFAs), as well as many other important formalisms such as read-only Turing machine [Turing and others, 1936] and monadic second-order logic [Courcelle and Engelfriet, 2012].

REI is a special case of grammar inference (GI), a long-standing research subject in AI. GI emerged from Chomsky-style linguistics, which models the problem of human language learning: how can children infer the grammar of their native language from a small number of examples? A simple model of language learning from examples is the following:

- **Input**: finite sets P and N of strings.
- **Output**: a grammar g (in the sense of Chomsky) that *precisely* captures P and N: all strings in P are accepted by g, and all strings in N are rejected.

This model was first studied in detail by [Gold, 1967] for regular grammars. He showed that, *"in the limit"* this problem cannot be solved from positive examples alone (*i.e.*, $N = \emptyset$). Such grammar inference with regular languages is simplistic as the complex syntactic rules of natural language are not regular, but regular languages are more mathematically tractable, and hence are a suitable starting point.

On the machine learning side, in particular in natural language processing (NLP), grammar inference and learning of Chomsky-style grammars and other formalisms has a long history; for surveys, we refer the reader to [D'Ulizia *et al.*, 2011] and [Muralidaran *et al.*, 2021]. While they do not learn an explicit grammar, large transformer-based models have been shown to learn internal representations akin to implicit grammars [Tenney *et al.*, 2019]. Other inference tasks also have a long-standing tradition in NLP, with many datasets and challenges proposed for Natural Language Inference [Bowman *et al.*, 2015; Williams *et al.*, 2018; Wang *et al.*, 2019; Nie *et al.*, 2020]. Addressing REI with generative models also relates to a long line of research into sequence generation tasks such as presented in [Colin *et al.*, 2016; Budzianowski *et al.*, 2018] and [Guan *et al.*, 2021].

Neural models have been successfully applied in all of these areas and related tasks. Recent advances in deep learning (DL), especially the transformer architecture [Vaswani *et al.*, 2017], have led to neural models now dominating virtually any NLP task. In the wake of this, *code synthesis*, i.e., the task of generating programming language code based on natural language instructions, has become an interesting next step; transformers in general, and GPT-like models [Radford *et al.*, 2019; Brown *et al.*, 2020] in particular are dominating this task as well, achieving top scores on datasets such as MBPP [Austin *et al.*, 2021] and APPS [Hendrycks *et al.*, 2021]. We posit there is a direct connection between code synthesis and regular expression generation.

REI as a Yardstick for Quantifying ML Progress. Grammar inference has two core quantifiable dimensions:

- **Correctness:** what fraction of data is classified correctly by the learned representations?
- **Optimisation:** how far is the cost of a learned representation away from the achievable minimum?

Recently, GI was brought into focus by [Delétang *et al.*, 2023; van der Poel *et al.*, 2023] who use benchmarks directly related to the Chomsky hierarchy. Both observe strong correlations between DL architectures and position of a benchmark in the hierarchy. What all existing GI benchmarks have in common is their focus on *correctness*; this leaves *optimisation*, a core goal of ML on the road toward AGI, unquantified.

The challenge presented here changes this. To our knowledge, we present the first benchmark to focus on quantification of a learner's ability to **optimise while remaining correct**. We do this in a simple and natural way: like previous work on GI, we ask learners to produce a grammar (REs) from positive and negative examples. Unlike previous work, we quantify how far away the learned grammar is from the possible minimum cost. A grammar is *optimal* if it achieves minimal cost while correctly classifying all examples.

We deliberately frame REI as a *challenge* benchmark, which we introduce here. We provide training data and an evaluation harness, as well as first approaches to the task using heuristics as well as modern DL techniques, in particular, LLM-based inference. These suggest that REI is indeed a hard optimisation problem, and we suggest using our benchmark as a stepping stone for quantifying and improving the optimisation abilities of contemporary ML models.

There are multiple reasons for REI being a compelling challenge that helps us understand and quantify important dimensions of modern ML.

The computational complexity of this problem is well understood: [Gold, 1978; Angluin, 1978; Pitt and Warmuth, 1993; Kearns and Valiant, 1994] showed that (a small variant of) this problem is NP-hard, and NP-hard even to approximate. They also showed that hardness does not depend on the underlying representation of regular languages, whether DFAs, NFAs, REs, or indeed any other. Those hardness results assure us that there is no easy shortcut that has been overlooked so far. We can also easily adjust the hardness of the problem by increasing simple parameters such as the size of input examples. Moreover, not only are NP-hard problems extremely well studied, but most properties of regular languages/expressions are known, such as their expressive power and limitations, their relationships with automata and grammar formalisms, the computational complexity of expression minimisation, equivalence checking, and more.

Another advantage of REI is that hardness can easily be adjusted in other ways. For example by searching over harder classes of languages (context-free or context-sensitive), or simpler ones, e.g., REs with limited star-height. Another form of simplifying REI is by allowing the learner to query the system to be learned, leading to Angluin-style active automata learning [Angluin, 1987], a form of ML that is actively used in formal verification and computer security [Vaandrager, 2017]. REI is also directly related to other applications, e.g., circuit complexity, bio-informatics, and network security. All those applications and connections with NP-hard problems mean that we have at our disposal a large and heterogeneous toolset to approach the inference challenge. Conversely, each approach that does well on it can be adapted to help with many other important problems.

A further advantage of the challenge is that REI is, to the best of our knowledge, an unsolved problem for deep learning based ML. While neural regular expression synthesis has been investigated [Firoiu *et al.*, 1998; Locascio *et al.*, 2016; Zhong *et al.*, 2018; Park *et al.*, 2019; Li *et al.*, 2021; Chen *et al.*, 2023], no existing work allows for configurable cost function, or achieves minimality; indeed, none even guarantee that all strings in P and N are classified correctly.

From the point-of-view of furthering research in transformer-based large language models (LLMs), REIC has another advantage: each problem instance is small, at most a few hundred tokens. Hence, limited attention window size is irrelevant; REI is hard already when it fits comfortably inside even small attention windows. Therefore, it enables fine-grained and principled investigations into LLM-learning inside the attention window, and neatly separates the problems of quantifying the generalisation performance of transformers inside the attention window from issues arising from the limitations of window size. This is especially relevant as REI can be seen as an idealisation of code synthesis; most contemporary tools for this task are transformer-based LLMs [Yuie *et al.*, 2021; Chen *et al.*, 2021; OpenAI, 2022; Github, 2022; Li *et al.*, 2023; Tunstall *et al.*, 2023].

3 The Regular Expression Inference Challenge

Formally, given a finite alphabet Σ, the *regular expressions over* Σ, short $\mathsf{RE}(\Sigma)$ are given by the grammar below, where option (?), Kleene-star (*) and complement ($\sim$) are unary operators; concatenation ($\cdot$), intersection (&), union (+), and restriction ($-$) are binary operators:

$$r, r' ::= \begin{cases} \emptyset & \text{The empty set} \\ \epsilon & \text{The empty string} \\ a & \text{Alphabet character } a \in \Sigma \\ r? & \text{Shorthand for } \epsilon + r \\ r^* & \text{0 or more repetitions of } r \\ \sim r & \text{Complement} \\ r \cdot r' & \text{Concatenation} \\ r \,\&\, r' & \text{Intersection, logical conjunction} \\ r + r' & \text{Union, logical disjunction} \\ r - r' & \text{Restriction} \end{cases}$$

We use standard abbreviations, e.g., rr' for $r \cdot r'$. With each $r \in \mathsf{RE}(\Sigma)$ we associate the *language of* r, $\mathsf{Lang}(r)$:

- $\mathsf{Lang}(\emptyset) = \emptyset$, $\mathsf{Lang}(\epsilon) = \{\epsilon\}$, $\mathsf{Lang}(a) = \{a\}$
- $\mathsf{Lang}(r?) = \mathsf{Lang}(\epsilon + r)$
- $\mathsf{Lang}(r^*) = \bigcup_{n \geq 0} \mathsf{Lang}(r^n)$ where $r^0 = \epsilon$ and $r^{n+1} = r \cdot r^n$
- $\mathsf{Lang}(\sim r) = \Sigma^* \setminus \mathsf{Lang}(r)$
- $\mathsf{Lang}(r \cdot r') = \mathsf{Lang}(r) \cdot \mathsf{Lang}(r')$
- $\mathsf{Lang}(r \,\&\, r') = \mathsf{Lang}(r) \cap \mathsf{Lang}(r')$
- $\mathsf{Lang}(r + r') = \mathsf{Lang}(r) \cup \mathsf{Lang}(r')$
- $\mathsf{Lang}(r - r') = \mathsf{Lang}(r) \setminus \mathsf{Lang}(r')$

This induces an equality on REs: r is equivalent to r' iff $\mathsf{Lang}(r) = \mathsf{Lang}(r')$. For example, $r + r$ and r have the same language; likewise $\mathsf{Lang}(r^{**}) = \mathsf{Lang}(r^*)$. Note that each equivalence class has an infinite number of inhabitants.

A *cost function* is a map $\mathsf{cost}(\cdot) : \mathsf{RE}(\Sigma) \to \mathbb{N}$ such that there are constants $c_1, \ldots, c_8 > 0$ with[1]:

- $\mathsf{cost}(\emptyset) = \mathsf{cost}(\epsilon) = \mathsf{cost}(a) = c_1 \; \forall a \in \Sigma$
- $\mathsf{cost}(r?) = \mathsf{cost}(r) + c_2$
- $\mathsf{cost}(r^*) = \mathsf{cost}(r) + c_3$
- $\mathsf{cost}(\sim r) = \mathsf{cost}(r) + c_4$
- $\mathsf{cost}(r \cdot r') = \mathsf{cost}(r) + \mathsf{cost}(r') + c_5$
- $\mathsf{cost}(r \,\&\, r') = \mathsf{cost}(r) + \mathsf{cost}(r') + c_6$
- $\mathsf{cost}(r + r') = \mathsf{cost}(r) + \mathsf{cost}(r') + c_7$
- $\mathsf{cost}(r - r') = \mathsf{cost}(r) + \mathsf{cost}(r') + c_8$

We call each c_i the *cost* of the corresponding regular operators. An important special case is the *uniform cost* given by setting $c_1 = \cdots = c_8 = 1$. Expressions in $\mathsf{RE}(\Sigma)$ form tree structures defining the scope of operators; when writing expressions as linear strings, we use parentheses where necessary to clarify scope, they don't count towards the RE's cost.

Finally, regular expression inference is a supervised learning problem and has the following structure:

- **Input:** Two finite sets of strings P and N of positive and negative examples, and a cost function $\mathsf{cost}(\cdot)$ for REs.
- **Output:** A regular expression r that is both:
  - **Precise:** Meaning that r accepts all strings in P and rejects all strings in N.
  - **Minimal:** No regular expression with a cost less than $\mathsf{cost}(r)$ is precise[2].

We call the pair P, N a *PN-set*, the combination with a cost function an *instance*, and r the *solution* of the instance.

We can adjust the hardness of REI. For example, as every REI problem P, N can be solved at a cost not exceeding that of the trivial solution, *i.e.*, the union of all strings in P, we can trivially rule out operators (other than union, concatenation and characters) by setting their cost to something above the cost of the trivial solution. We also note that we can make REI simpler, e.g., if instead of using the cost function as an input parameter, we fix a single cost function for all instances.

4 Datasets

We now detail the data released as part of the challenge. The data is produced automatically using an extension of the GPU-accelerated REI solver of [Valizadeh and Berger, 2023]: we added three additional operators ($\sim, \&, -$). This allowed us to introduce datasets DS3 and DS4, as explained below. The data is split into official training and test sets; the former,

[1]We use this family of cost-functions for several reasons: (i) existing works on the complexity of REI use "uniform cost", (ii) [Valizadeh and Berger, 2023] uses the same family, (iii) our cost-functions are 8-tuples of integers, hence easy to learn—it is unclear how to specify more complex cost functions, and this reassures us that REI hardness is not an effect of how the cost functions are given.

[2]Note there may be other precise r' with $\mathsf{cost}(r') = \mathsf{cost}(r)$.

Dataset	Σ	RE Characters & Ops	Cost Function	#PN Sets	#Instances	#Train	#Test
DS1	$\{0,1\}$	$\{\epsilon, a, ?, *, \cdot, +\}$	uniform	$6,053$	$6,053$	$5,447$	606
DS2	$\{0,1\}$	$\{\epsilon, a, ?, *, \cdot, +\}$	variable	$6,029$	$120,580$	$110,329$	$10,251$
DS3	$\{0,1\}$	$\{\emptyset, \epsilon, a, ?, *, \sim, \cdot, \&, +, -\}$	uniform	$6,054$	$6,054$	$5,448$	606
DS4	$\{0,1\}$	$\{\emptyset, \epsilon, a, ?, *, \sim, \cdot, \&, +, -\}$	variable	$6,013$	$120,260$	$108,234$	$12,026$

Table 1: The four datasets generated for the REI challenge. Here and elsewhere #S denotes the cardinality of S.

but not the latter, come with associated solutions, *i.e.*, for each PN-set and cost function, a minimal regular expression.

Using an algorithmic solver to generate the data for the machine learning challenge guarantees that it is correct: the solver will output an expression for a given PN-set and cost function *if and only if* it is indeed minimal w.r.t. that input[3].

We provide four separate datasets, differing in key characteristics along two dimensions:

- **Operators.** Datasets 1 and 2 (DS1, DS2) allow only the RE operators for option (?), Kleene-star ($*$), concatenation ($\cdot$), and union ($+$). This means complement ($\sim$), intersection (&) and restriction ($-$) are not available. Datasets 3 and 4 (DS3, DS4) allow all operators.
- **Cost functions.** We use only the uniform cost for Datasets 1 and 3, while making it variable for Datasets 2 and 4, which provide a number of solutions per PN-set, given different random cost functions.

We posit that such a division makes the problems in REIC more variable; we explicitly encourage participants to contribute to results on only some, or all of the four datasets.

When splitting the four generated datasets into training and test data, we aim for a $90/10$ split. For each of the datasets, we ensure that the test portion only contains regular expressions that *do not occur* in the training part. However, sometimes solutions to a PN-set in the training set might also be solutions to an instance in the test set.

4.1 Data Generation

Our data generation approach has two parts, random PN-set and random cost function generation.

PN-set Generation. Random PN-sets are generated with a small set of parameters:

- Σ: The set of characters used in P and N.
- le: The maximum length of example strings in P and N (0 is always minimum length).
- p, n: number of positive negative examples respectively, *i.e.*, size of P and N.

For these parameters, we use two PN-generation schemes from strings over Σ, with complementary properties.

- TYPE 1: We uniformly and independently sample p strings from the set of all strings of size $0, 1, ..., le$, and n strings for N. TYPE 1 is heavily biased towards long strings, since there are exponentially more long strings than short ones. Since short strings, in particular ϵ and the characters in Σ, play an important role in regular languages, bias towards long strings is sometimes suboptimal, and we use a complementary second scheme.
- TYPE 2: This scheme gives more weight to shorter strings and works in two steps: first, uniformly and independently sample a target string length i, and then uniformly and independently sample a string of length i. We use this procedure to generate strings for P and N until the sizes p and n are reached.

In both cases, we ensure that P and N are disjoint, to rule out challenge problems without solutions. As an illustration, assuming an alphabet $\Sigma = \{0, 1\}$, an example of a generated Type 1 PN-set with parameters $le = 6, p = 12, n = 10$ is

P: $10110,\ 001110,\ 010111,\ 1,\ 00101,\ 011111,\ 011,\ 000000,\ 0111,\ 10011,\ 001100,\ 001001$

N: $000,\ 00010,\ 010010,\ 010011,\ 01010,\ 100001,\ 1011,\ 110000,\ 11001,\ 1110$

With the same alphabet, Type 2 generation with parameters $le = 8, p = 10, n = 7$ produces, for example

P: $000,\ 0101,\ 1110010,\ 11,\ 1,\ 111,\ \epsilon,\ 11101100,\ 0,\ 11001$

N: $00010,\ 0010011,\ 010,\ 01001000,\ 011,\ 10,\ 11010$

The four datasets discussed above contain instances generated (roughly) evenly with TYPE 1 and TYPE 2 over the alphabet $\{0, 1\}$. For TYPE 1, parameters p and n range over all of $1, \ldots, 10$, and le ranges over all of $0, \ldots, 7$; for TYPE 2, we use the same ranges for p and n, and le from 0 to 10.

For the challenge presented here, we restrict ourselves to strings over a binary alphabet for two reasons: First, as our baselines in the next section indicate clearly, REI with a binary alphabet is already hard to learn. Secondly, getting ground truth for substantially bigger alphabets, e.g., ASCII, is currently infeasible, because the performance even of the SOTA GPU-based REI solver we use [Valizadeh and Berger, 2023] is heavily constrained by alphabet size.

Cost Function Generation. As mentioned above, our datasets are split w.r.t. cost functions: In DS 1 and 3 the cost function is fixed to the uniform cost. In the remaining DS 2 and 4 the cost function is a parameter. To generate costs, we randomly and independently sample from $1, \ldots, 49$ for each c_i of the cost function. For DS 4, we use all c_i; since DS 2 uses a reduced set of operators, we only sample costs of relevant operators. For both datasets, we generate 19 random cost functions in addition to the uniform cost, for each PN-set. Hence k PN-sets give us $20 * k$ inference problems. Table 1 shows an overview of the data generated for REIC.

[3]PN-set and cost instances for which the solver fails are not included in the released data.

After generating PN-sets and cost functions, we use the REI solver from [Valizadeh and Berger, 2023] to generate reference minimal REs for each instance.

5 REIC Metrics and Baselines

Challenge scoring. The REI challenge is scored using the following metrics. All scoring is automated. All scores below are calculated for all instances in a dataset; if an instance solution is not syntactically valid, it will also negatively contribute to other scores.

- **Compile Ratio**: The ratio of generated REs that are syntactically valid, i.e., that can be parsed by a RE parser.
- **Precise Absolute**: Number of instances for which the generated RE correctly classifies the entire PN-set.
- **Precise Ratio**: The ratio of instance REs that correctly accept and respectively reject the PN-sets.
- **Positive Ratio**: The ratio of positive examples that are correctly accepted.
- **Negative Ratio**: The ratio of negative examples that are correctly rejected.
- **PN Ratio**: The ratio of combined accepted strings in P with rejected strings in N.
- **Minimal Instances**: The number of instances for which the generated RE is precise and minimal.
- **Minimal Ratio – Precise**: The ratio of minimal REs over precise generated REs.
- **Minimal Ratio – Global**: The ratio of minimal REs over all test instances.
- **Cost Ratio**: The average ratio of precise generated REs' costs over the gold-standard minimal RE costs.

Our leaderboard lists all scores for participating models[4]. The leader is determined solely by global Minimal Ratio[5].

Baselines. We devise REI as a challenge to entice the machine learning/deep learning community to explore this hard task. Addressing REI with ML/DL methods is as of yet an unsolved problem; while perfect algorithmic solvers exist (in fact, one such solver was used to generate the data for REIC), no such perfect learned model is known.

To bootstrap REIC with baseline solutions, and provide further insights into the challenge, we devise several heuristic baselines, as well as two first DL-based approaches:

- The **Trivial RE** baseline simply generates, for each test instance, the trivial regular expression formed by the union $(+)$ of the strings in its positive set. Note that this always returns a precise RE, albeit usually not minimal.
- **PN Retrieval** is a baseline that, for each test instance, retrieves the closest matching train data instance w.r.t. overlap of their PN sets, and uses the corresponding regular expression for the test. If multiple PN sets the with same overlap are retrieved, the lowest cost RE w.r.t. the test instance cost function is returned. This baseline lets use assess how much of the training data is solvable by learning from closely related data.
- **RE Retrieval** has access to all regular expressions in given training sets (DS1+DS2, DS3+DS4), and tests all of them against each test data instance, selecting for each that RE with highest PN Ratio. In cases of ties, it selects that expression with lowest cost, according to the test instance cost function. This baseline can show us how many test instances are solvable precisely and with minimal cost with expressions seen in the training data.
- **StarChatβ** uses a pretrained code LLM and performs REI with a few-shot prompt. Specifically, we employ *StarChatβ* [Tunstall *et al.*, 2023], an instruction-tuned variant of StarCoderPlus [Li *et al.*, 2023], a SOTA 15.5 billion parameter code LLM. We use prompting to instill information about REI into the model, and let it generate 10 regular expressions for each test instance, keeping the best. Prompts and inference details used for this baseline are shown in the technical appendix.
- **ReGPT** is a GPT-2-like [Radford *et al.*, 2019] model, trained on the training portions of the REIC datasets; details are given below, and all training hyperparameters are given in the technical appendix. We use the model's inference capabilities as a generative LLM to generate regular expressions for the test set instances.

We considered *automaton minimisation* as a baseline, such as taking an NFA $\mathcal{A}$, e.g., one based on the trivial solution, and transforming it into an equivalent minimal DFA $\mathcal{B}$. However, we decided against it for several reasons: First, by definition, $\mathcal{B}$ accepts the same language as $\mathcal{A}$; but this is not necessarily the case for a minimal solution to a REI problem – it just needs to classify the PN-set correctly, regardless of the language it specifies. Hence $\mathcal{B}$ will typically not be minimal for a given PN-set. Second, to the best of our knowledge, no algorithm for DFA minimisation exists that can handle cost functions. Finally, even assuming a notion of minimality in DFAs, e.g., number of states, we doubt the standard translations from automata (possibly cyclic graphs) to REs (trees) preserve minimality. We therefore disregard this as a baseline suited to the challenge at hand.

ReGPT Training and Inference. For ReGPT we train two models from scratch, one on the combined training sets DS1 and DS2, and a second model on DS3 and DS4. All models are implemented in the HuggingFace transformers framework[6] and use the GPT-2 architecture with a total of $\sim$300M parameters. Details are shown in the technical appendix.

We use a `[CLS]` token to indicate the beginning of the input, `[POS]` and `[NEG]` for positive/negative strings, one `[COST_X]` token per operator to indicate the cost function,

[4] As heuristics/algorithms are known to achieve perfect scores, the official leaderboard only lists and ranks ML/DL models.

[5] 'Gaming" the system to get high scores in some metrics is easy; e.g., the Trivial baseline achieves 100% Compile Rate, Precision, and P/N/PN Ratio by definition. Similarly, Minimal Ratio (Precise) and Cost Ratio can easily be exploited by high-precision, low-recall systems. Hence, any score encompassing a (weighted) average score could easily be artificially tuned.

[6] https://huggingface.co/docs/transformers

		CR	Prec	Prec%	P%	N%	PN%	Min	Min% P	**Min% G**	Cost Ratio
Dataset 1	Trivial	100	606	100	100	100	100	0	0.0	0.0	3.34
	PN Ret	100	3	0.5	45.9	89.3	67.8	1	33.3	1.7	1.20
	RE Ret	100	302	49.8	94.0	95.6	95.4	133	44.0	21.9	1.15
	StarChatβ	100	5	0.8	37.2	85.9	63.4	0	0.0	0.0	2.02
	ReGPT-1k	100	60	9.9	80.9	88.5	87.4	33	55.0	5.4	1.10
	ReGPT-10	100	10	1.7	65.6	74.2	73.4	7	70.0	1.2	1.08
Dataset 2	Trivial	100	10251	100	100	100	100	77	0.8	0.8	3.35
	PN Ret	100	10251	100	100	100	100	2133	20.8	20.8	1.08
	RE Ret	100	10251	100	100	100	100	2375	23.2	23.2	1.06
	StarChatβ	99.69	29	0.3	39.6	79.8	61.9	3	10.3	<0.1	1.47
	ReGPT-1k	100	9787	95.5	99.1	99.1	99.2	2013	20.6	19.6	1.08
	ReGPT-10	99.99	9775	95.4	99.0	99.1	99.1	1945	19.9	19.0	1.08
Dataset 3	Trivial	100	606	100	100	100	100	2	0.3	0.3	3.99
	PN Ret	100	9	1.5	63.6	72.5	70.8	3	33.3	5.0	1.20
	RE Ret	100	328	54.1	95.7	94.6	96.0	151	46.0	24.9	1.16
	StarChatβ	100	6	1.0	44.6	79.8	65.8	1	16.7	0.2	2.10
	ReGPT-1k	100	97	16.0	85.5	89.6	89.8	41	42.3	6.8	1.14
	ReGPT-10	100	28	4.6	73.3	77.0	79.0	15	53.6	2.5	1.11
Dataset 4	Trivial	100	12026	100	100	100	100	32	0.3	0.3	4.32
	PN Ret	100	12026	100	100	100	100	1239	10.3	10.3	1.08
	RE Ret	100	12026	100	100	100	100	1412	11.7	11.7	1.07
	StarChatβ	100	52	0.4	44.5	78.9	64.2	1	1.39	<0.1	2.56
	ReGPT-1k	100	10222	84.99	97.0	97.2	97.4	1093	10.7	9.1	1.09
	ReGPT-10	99.99	9590	79.7	94.7	95.0	95.3	957	10.0	8.0	1.09
All Data	Trivial	100	23489	100	100	100	100	111	0.5	0.5	3.86
	PN Ret	100	22289	94.9	97.7	99.0	98.4	3376	15.1	14.4	1.08
	RE Ret	100	22907	97.5	99.7	99.7	99.8	4071	17.8	17.3	1.07
	StarChatβ	99.86	92	0.4	42.1	79.5	63.2	5	5.4	<0.1	2.16
	ReGPT-1k	100	20166	85.9	97.2	97.6	97.7	3180	15.8	13.5	1.09
	ReGPT-10	99.99	19403	82.6	95.3	95.8	96.0	2924	15.1	12.4	1.09

Table 2: Baseline results on the four REIC test datasets, as well as combined test data. PN Ret and RE Ret are PN Retrieval and RE Retrieval baselines, respectively. CR refers to Compile Ratio, Prec/Prec% to total/ratio of REs that precisely accept/reject PN-sets, P/N/PN% to accuracies over respective sets, Min% P/Min% G to ratio of minimal cost for precise REs/across full dataset. "ReGPT-1k" and "ReGPT-10" are our ReGPT model with $1k$ and 10 sampled solutions, respectively.

and `[BOR]` and `[EOR]` to mark the beginning/end of a regular expression; `[EOR]` doubles as the end-of-sequence token. An example ReGPT encoding is shown in the appendix.

During training, we randomly split the combined training data into train and validation sets in a 90/10 split. To select the model checkpoints for final evaluation on the test sets, we greedily generate one solution for each validation instance, and calculate the average of PN Ratio and Global Minimality. For each test set, we select the checkpoint with highest average score on the respective validation set.

For inference on the test set, we use the trained ReGPT models to generate $1k$ solutions for each instance: one greedily, and the remaining solutions using Nucleus Sampling [Holtzman *et al.*, 2019]. We keep the best solution as measured by PN Ratio; in cases of ties, we keep the solution with lower cost according to the test instance cost function.

In our inference setup, sampling $1,000$ REs with ReGPT takes roughly the same time as sampling 10 solutions with StarChatβ. However, for a better comparison, we also report ReGPT performance when generating 10 solutions. We provide training and inference details in the appendix.

Note that these first baselines are in fact not directly optimized for RE minimalisation. We provide these as examples of addressing the novel challenge we present with the currently dominant approach to inference tasks, i.e., training LLMs on large amounts of positive data. While the cost function is encoded in the LLM input, there is no theoretical guarantee that the model will be able to generalise to unseen combinations of PN-sets and cost functions. This is deliberate: We provide a first mainstream approach to the new challenge benchmark, and leave more involved approaches, e.g., Reinforcement Learning using cost functions for the reward signal, to future work and future challenge entries.

5.1 Discussion of Baseline Performance

Table 2 summarises the performance of all our baselines on the four test datasets, as well as on all combined test data.

The heuristic baselines – *Trivial RE, PN Retrieval, and RE Retrieval* – provide valuable insights into the challenge data. Of course, the Trivial baseline achieves 100% in all PN coverage metrics, as well as Compile Rate. Due to data artifacts, it even "finds" some minimal expressions; however, these finds are rare, showing that the minimal RE is almost never trivial. Across all datasets, this baseline achieves the worst Cost Ratio, producing expressions up to more than 4.3 times as costly on average than the minimal regular expression, on Dataset 4.

The two retrieval baselines reveal an interesting pattern across the four datasets. While both PN and RE Retrieval methods are able to find expressions that are 100% *precise* on DS2 and DS4, this metric falls to $\sim$50% (RE) and only $\sim$1% (PN) on the (much smaller) DS1 and DS3. For minimality, retrieving solutions based on PN sets or REs leads to very similar performance on the datasets with variable cost functions, while there is a stark performance difference on those with uniform cost function. This is likely an artefact of data generation, where 20 instances per PN set are given for both DS2 and DS4, with varying cost functions. Therefore, the retrieval baselines are able to find corresponding instances across these larger training and test data; while this leads to perfect precision, the heuristics also reveal that at most $1/4$ (RE Retrieval, 24.9% minimality on DS3) of the challenge is solvable from the data directly, without learning techniques.

ReGPT (1k) produces syntactically valid regular expressions for all test instances, indicating it has learned a good internal representation of the RE grammar. Even when only sampling 10 solutions, it performs at or essentially at 100% compile rate. While it is outperformed by the (very strong) RE Retrieval, ReGPT generates better expressions than can be retrieved by PN Retrieval on the smaller test sets, DS1 and DS3. PN Retrieval is conceptually the closest heuristic to the learned model, which is trained to generate REs conditioned on PN sets; ReGPT outperforming this baseline on the smaller datasets – which are more out-of-distribution compared to the training data than the larger test sets – indicates that there is a learning signal available in the training data that allows a degree of generalisation. ReGPT's strongest test sets are DS2, followed by DS 4, which both contain variable cost functions for the same PN sets. These datasets of course contribute the vast amount of training data during ReGPT training; in addition, Dataset 2 uses the reduced set of permitted operators, explaining the relatively high test performance.

Across all datasets, ReGPT finds cost-efficient solutions; on the challenging sets DS1 and DS3, the average cost of its perfect solutions even is *lowest* across systems. This yields interesting possibilities of using trained models in combination with algorithmic solvers such as that of [Valizadeh and Berger, 2023], for example by providing precise solutions that are *close* to minimal as starting points for the search algorithm, potentially helping with search space reduction.

The StarChatβ baseline shows a severe performance gap on the challenge test sets. This is not surprising: While it is truly large and trained for code synthesis – a task closely related to, if not a superset of regular expression inference – it has never seen REIC data during training. While prompting instruction-tuned LLMs with specific tasks can be a powerful technique, it seems not enough to perform well on the novel, untrained, and provably hard challenge proposed here.

In fact, as StarChatβ's compile rate of (essentially) 100% demonstrates, the model is capable of generating REs as defined for our challenge. However, when it comes to *precision* and *minimality*, the large SOTA pretrained code LLM struggles with the proposed challenge in a prompt-based setting, where it underperforms Compared to the much smaller, but fully supervised ReGPT baseline. This is true even when comparing models that sample the same number of outputs.

It is likely that a fine-tuned version, i.e., a StarChatβ model trained on REIC's training data, will outperform the prompt approach – though competitiveness with ReGPT is another question[7]. However, training such a large model even for finetuning is time-consuming and expensive. Other prompting methods – either with different hand-written prompts, or trained – might also yield improvement; however, the long inference time and resources needed make rapid development of this a challenge in itself. As this work aims to establish REI as a challenge for the research community, we thus leave these approaches as interesting future work.

Finally, focusing on model performance on all combined data, it seems clear that the hardness of REIC indeed lies *in the task of finding minimal solutions* to the given inference problems. The heuristic baselines perform close to 100% precision, and ReGPT's learning approaches 86% for the combined data on the same metric, which we can expect to be improved upon with more sophisticated models. With exception of StarChatβ, all approaches including ReGPT achieve scores in the high 90s on *PN Ratio*, giving further evidence that learning to cover the PN-sets is achievable. However, even the strongest heuristic baseline, RE Retrieval, only finds precise and *minimal* solutions on 17.3% of the full data.

It seems clear that REI provides a challenging setting, for both heuristic and learning-based methods. We hope these results will incentivise the research community to actively participate in the REIC, pushing the envelope in regular expression and deep learning research and understanding.

6 Conclusions

We have introduced the regular expression inference challenge, an open-ended research challenge with a novel goal, *optimisation while remaining correct*, for the machine learning community, inviting participants to develop models that can generate minimal regular expressions w.r.t. PN-sets and cost functions. We motivated the challenge in the light of regular expression and formal language research, machine learning research, and potential practical applications that we believe stand to benefit from regular expression inference. We employed heuristic and learned baselines, indicating that the REI challenge is difficult, and warrants further research into how machine learning and deep learning approaches can be leveraged to better cover this problem.

We invite the research community to participate in the REI challenge hosted on CodaLab, where starter code is also made available to enable reproduction of our baselines:

https://codalab.lisn.upsaclay.fr/competitions/15096

Contribution Statement

Mojtabah Valizadeh and Philip John Gorinski share equal contribution. Martin Berger is the main contact author.

[7]We speculate that, given the small context size for REI, a fully supervised model like ReGPT should not require billions of parameters to properly capture the task.

References

[Angluin, 1978] Dana Angluin. On the complexity of minimum inference of regular sets. *Information and Control*, 39(3):337–350, 1978.

[Angluin, 1987] Dana Angluin. Learning Regular Sets from Queries and Counterexamples. *Inf. Comput.*, 75(2):87–106, November 1987.

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

[Bowman *et al.*, 2015] Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. A large annotated corpus for learning natural language inference. In *Proceedings of EMNLP 2015*, pages 632–642, Lisbon, Portugal, September 2015.

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

[Budzianowski *et al.*, 2018] Paweł Budzianowski, Tsung-Hsien Wen, Bo-Hsiang Tseng, Iñigo Casanueva, Stefan Ultes, Osman Ramadan, and Milica Gašić. MultiWOZ - a large-scale multi-domain Wizard-of-Oz dataset for task-oriented dialogue modelling. In *Proceedings of EMNLP 2018*, pages 5016–5026, Brussels, Belgium, October-November 2018.

[Chen *et al.*, 2021] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[Chen *et al.*, 2023] Qiaochu Chen, Arko Banerjee, Çağatay Demiralp, Greg Durrett, and Işıl Dillig. Data extraction via semantic regular expression synthesis. *Proc. ACM Program. Lang.*, 7(OOPSLA2), oct 2023.

[Chomsky, 1956] Noam Chomsky. Three models for the description of language. *IRE Transactions on information theory*, 2(3):113–124, 1956.

[Colin *et al.*, 2016] Emilie Colin, Claire Gardent, Yassine M'rabet, Shashi Narayan, and Laura Perez-Beltrachini. The WebNLG Challenge: Generating Text from DBPedia Data. In *Proceedings of the 9th international natural language generation conference*, pages 163–167, 2016.

[Courcelle and Engelfriet, 2012] Professor Bruno Courcelle and Dr Joost Engelfriet. *Graph Structure and Monadic Second-Order Logic: A Language-Theoretic Approach*. Cambridge University Press, USA, 1st edition, 2012.

[Delétang *et al.*, 2023] Grégoire Delétang, Anian Ruoss, Jordi Grau-Moya, Tim Genewein, Li Kevin Wenliang, Elliot Catt, Chris Cundy, Marcus Hutter, Shane Legg, Joel Veness, and Pedro A. Ortega. Neural networks and the chomsky hierarchy. In *11th International Conference on Learning Representations*, 2023.

[D'Ulizia *et al.*, 2011] Arianna D'Ulizia, Fernando Ferri, and Patrizia Grifoni. A survey of grammatical inference methods for natural language learning. *Artificial Intelligence Review*, 36:1–27, 2011.

[Firoiu *et al.*, 1998] Laura Firoiu, Tim Oates, and Paul R. Cohen. Learning Deterministic Finite Automaton with a Recurrent Neural Network. In *Proceedings of the ICGI 1998*, ICGI '98, page 90–101, Berlin, Heidelberg, 1998. Springer-Verlag.

[Github, 2022] Github. Your AI pair programmer. https://github.com/features/copilot, 2022. Blog post accessed 8 June 2023.

[Gold, 1967] E. Mark Gold. Language identification in the limit. *Information and Control*, 10(5):447–474, 1967.

[Gold, 1978] E. Mark Gold. Complexity of automaton identification from given data. *Information and Control*, 37(3):302–320, 1978.

[Guan *et al.*, 2021] Jian Guan, Zhexin Zhang, Zhuoer Feng, Zitao Liu, Wenbiao Ding, Xiaoxi Mao, Changjie Fan, and Minlie Huang. OpenMEVA: A benchmark for evaluating open-ended story generation metrics. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 6394–6407, Online, August 2021.

[Hendrycks *et al.*, 2021] Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt. Measuring coding challenge competence with apps. *NeurIPS*, 2021.

[Holtzman *et al.*, 2019] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. In *International Conference on Learning Representations*, 2019.

[Hopcroft *et al.*, 2006] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 2006.

[Kearns and Valiant, 1994] Michael Kearns and Leslie Valiant. Cryptographic Limitations on Learning Boolean Formulae and Finite Automata. *J. ACM*, 41(1):67–95, jan 1994.

[Kleene, 1956] S. C. Kleene. Representation of events in nerve nets and finite automata. In *Automata Studies*, pages 3–41. Princeton University Press, Princeton, NJ, 1956.

[Li *et al.*, 2021] Yeting Li, Shuaimin Li, Zhiwu Xu, Jialun Cao, Zixuan Chen, Yun Hu, Haiming Chen, and Shing-Chi Cheung. TransRegex: Multi-Modal Regular Expression Synthesis by Generate-and-Repair. In *Proceedings of the 43rd International Conference on Software Engineering*, ICSE '21, page 1210–1222. IEEE Press, 2021.

[Li *et al.*, 2023] Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny

Chim, Qian Liu, Evgenii Zheltonozhskii, Terry Yue Zhuo, Thomas Wang, Olivier Dehaene, Mishig Davaadorj, Joel Lamy-Poirier, João Monteiro, Oleh Shliazhko, Nicolas Gontier, Nicholas Meade, Armel Zebaze, Ming-Ho Yee, Logesh Kumar Umapathi, Jian Zhu, Benjamin Lipkin, Muhtasham Oblokulov, Zhiruo Wang, Rudra Murthy, Jason Stillerman, Siva Sankalp Patel, Dmitry Abulkhanov, Marco Zocca, Manan Dey, Zhihan Zhang, Nour Fahmy, Urvashi Bhattacharyya, Wenhao Yu, Swayam Singh, Sasha Luccioni, Paulo Villegas, Maxim Kunakov, Fedor Zhdanov, Manuel Romero, Tony Lee, Nadav Timor, Jennifer Ding, Claire Schlesinger, Hailey Schoelkopf, Jan Ebert, Tri Dao, Mayank Mishra, Alex Gu, Jennifer Robinson, Carolyn Jane Anderson, Brendan Dolan-Gavitt, Danish Contractor, Siva Reddy, Daniel Fried, Dzmitry Bahdanau, Yacine Jernite, Carlos Muñoz Ferrandis, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. Starcoder: may the source be with you! *Transactions on Machine Learning Research*, 2023.

[Locascio *et al.*, 2016] Nicholas Locascio, Karthik Narasimhan, Eduardo DeLeon, Nate Kushman, and Regina Barzilay. Neural generation of regular expressions from natural language with minimal domain knowledge. In *Proceedings of EMNLP 2016*, pages 1918–1923, Austin, Texas, November 2016.

[Muralidaran *et al.*, 2021] Vigneshwaran Muralidaran, Irena Spasić, and Dawn Knight. A systematic review of unsupervised approaches to grammar induction. *Natural Language Engineering*, 27(6):647–689, 2021.

[Nie *et al.*, 2020] Yixin Nie, Adina Williams, Emily Dinan, Mohit Bansal, Jason Weston, and Douwe Kiela. Adversarial NLI: A new benchmark for natural language understanding. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4885–4901, Online, July 2020.

[OpenAI, 2022] OpenAI. Introducing ChatGPT. https://openai.com/blog/chatgpt, 2022. Blog post published November 30, 2022.

[Park *et al.*, 2019] Jun-U Park, Sang-Ki Ko, Marco Cognetta, and Yo-Sub Han. SoftRegex: Generating Regex from Natural Language Descriptions using Softened Regex Equivalence. In *Proceedings of EMNLP-IJCNLP 2019*, pages 6425–6431, Hong Kong, China, November 2019.

[Pavao *et al.*, 2022] Adrien Pavao, Isabelle Guyon, Anne-Catherine Letournel, Xavier Baró, Hugo Escalante, Sergio Escalera, Tyler Thomas, and Zhen Xu. CodaLab Competitions: An open source platform to organize scientific challenges. *Technical report*, 2022.

[Pitt and Warmuth, 1993] Leonard Pitt and Manfred K. Warmuth. The Minimum Consistent DFA Problem Cannot Be Approximated within Any Polynomial. *J. ACM*, 40(1):95–142, jan 1993.

[Radford *et al.*, 2019] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[Tenney *et al.*, 2019] Ian Tenney, Dipanjan Das, and Ellie Pavlick. BERT Rediscovers the Classical NLP Pipeline. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4593–4601, 2019.

[Tunstall *et al.*, 2023] Lewis Tunstall, Nathan Lambert, Nazneen Rajani, Edward Beeching, Teven Le Scao, Leandro von Werra, Sheon Han, Philipp Schmid, and Alexander Rush. Creating a coding assistant with starcoder. *Hugging Face Blog*, 2023. https://huggingface.co/blog/starchat.

[Turing and others, 1936] Alan Mathison Turing et al. On computable numbers, with an application to the entscheidungsproblem. *J. of Math*, 58(345-363):5, 1936.

[Vaandrager, 2017] Frits Vaandrager. Model Learning. *Commun. ACM*, 60(2):86–95, jan 2017.

[Valizadeh and Berger, 2023] Mojtaba Valizadeh and Martin Berger. Search-Based Regular Expression Inference on a GPU. *Proc. ACM Program. Lang.*, 7(PLDI), jun 2023. Draft available at https://arxiv.org/abs/2305.18575, implementation: https://github.com/MojtabaValizadeh/paresy.

[van der Poel *et al.*, 2023] Sam van der Poel, Dakotah Lambert, Kalina Kostyszyn, Tiantian Gao, Rahul Verma, Derek Andersen, Joanne Chau, Emily Peterson, Cody St. Clair, Paul Fodor, Chihiro Shibata, and Jeffrey Heinz. MLRegTest: A Benchmark for the Machine Learning of Regular Languages, 2023.

[Vaswani *et al.*, 2017] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[Wang *et al.*, 2019] Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. Superglue: A stickier benchmark for general-purpose language understanding systems. *Advances in neural information processing systems*, 32, 2019.

[Williams *et al.*, 2018] Adina Williams, Nikita Nangia, and Samuel Bowman. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122, New Orleans, Louisiana, June 2018.

[Yuie *et al.*, 2021] Wang Yuie, Weishi Wang, Shafiq Joty, and Steven C.H. Hoi. CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation. In *EMNLP*, 2021.

[Zhong *et al.*, 2018] Zexuan Zhong, Jiaqi Guo, Wei Yang, Jian Peng, Tao Xie, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. SemRegex: A Semantics-Based Approach for Generating Regular Expressions from Natural Language Specifications. In *Proceedings of EMNLP 2018*, pages 1608–1618, Brussels, Belgium, October-November 2018.