Performance of Large Language Models on the Universality Problem of Regular Expressions

Undergraduate Student Research Report, 2025

By: Muhammad Arham Mussawar

Supervisors: Stavros Konstantinidis and Argy Papageorgiou

LLM Consultant: Nikita Neveditsin

Introduction:

As large language models (LLMs) increasingly expand their capabilities beyond natural language understanding and into domains requiring formal reasoning, it becomes essential to examine their performance on classical computational problems. Some of these problems are not only algorithmically challenging but also provide a structured framework for evaluating reasoning skills in a controlled and theoretically grounded setting. Among the many possible starting points, this study focuses on the universality problem for regular expressions, a foundational decision problem in automata theory. Regular expressions are ubiquitous across computer science, appearing in applications ranging from text processing to network security [2], and their familiarity makes them an accessible yet non-trivial testbed for evaluating the formal reasoning abilities of LLMs.

The universality problem offers a rigorous and well-characterized entry point for investigating how LLMs engage with tasks that demand more than surface-level pattern recognition. Although the syntactic construction rules for regular expressions are relatively simple — involving operations like union, concatenation, and Kleene star — determining universality requires deeper computational understanding. Specifically, it asks whether a given regular expression defines the entire set of possible strings over an alphabet. This task is computationally non-trivial: the universality problem is classified as **PSPACE-complete** [3], meaning that solving it may require space that grows exponentially with the size of the input, making it challenging even for any computing machine.

Unlike simple pattern-matching tasks, universality requires reasoning about the **absence** of counterexamples and exhaustive coverage, involving

operations like complement and closure under union and concatenation. A model must infer not only what strings are accepted, but critically, whether any possible string might be excluded. These subtle logical requirements make universality particularly revealing success or failure in solving it provides early indicators of an LLM's capacity for structured reasoning, implicit computation over combinatorially large spaces, and logical consistency across exhaustively large domains. Beginning with a problem that is decidable yet computationally demanding allows this project to systematically probe the strategies and limitations of LLMs when faced with algorithmically hard tasks requiring formal reasoning.

Summary of this research: This study investigates how large language models handle the universality problem for regular expressions. We first establish the theoretical background of universality and approximate universality. Next, we construct a benchmark dataset of regular expressions, including both universal and non-universal cases. We then evaluate the performance of several state-of-the-art LLMs under different prompting strategies, measuring both correctness and reasoning quality. Finally, we conduct error analysis to identify common failure modes and propose fine-tuning directions. Together, these results provide insight into the capabilities and limitations of LLMs in tackling algorithmically hard problems.

We found that all four models performed remarkably well on the universality problem, achieving near-perfect accuracy across both universal and non-universal cases, with only minor variations observed in Gemini at certain depths.

0. Preliminaries

Before addressing the universality problem, it is important to review key concepts from automata theory and formal languages that form the foundation of this study.

0.1 Alphabet and Strings

An **alphabet** Σ is a finite, non-empty set of symbols.
A **string** over Σ is a finite sequence of symbols from Σ.

ε is the **empty string**.
The **set** of **all strings** (including the empty string ε) over Σ is denoted by Σ^*.

0.2 Regular Expressions

A **regular expression** is a formal way to describe a set of strings, known as a **regular language**.
If *R* is a regular expression, then $L(R)$ denotes the **language** described by *R*.

Every regular expression defines a unique regular language as follows:

$L(\sigma) = \{\sigma\}$, for any σ in Σ

$L(\varepsilon) = \{\varepsilon\}$,

Union $(R_1 + R_2)$: We have that $L(R_1 + R_2) = L(R_1) + L(R_2)$.

Concatenation (R_1R_2): $L(R_1R_2)$ = a string from R_1 followed by a string from R_2.

Kleene star (R^*): $L(R^*)$ = zero or more repetitions of a string from R.

Example:

Let Σ={0,1} and R=(0+1)*.

Then $L(R)$=Σ*, i.e., the set of all binary strings.

0.3 Regular Languages and Automata

A regular language is a set of strings that can be described by a regular expression or accepted by a finite automaton. If A is a regular expression or a finite automaton then $L(A)$ denotes the language of A.

There are two types of finite automata:

- **Deterministic Finite Automata (DFA):** exactly one transition per symbol from any state.
- **Nondeterministic Finite Automata (NFA):** multiple possible transitions per symbol (or no transition at all).

Both DFAs and NFAs recognize exactly the class of regular languages [1].

Two automata A and B are equivalent if L(A) = L(B). Language containment L(A) ⊆ L(B) is a related decision problem used in compiler verification and model checking [10].

0.4 Language Operations

Some operations on languages that are useful in this context include:

- **Complement** (L^c): all strings over Σ that are *not* in L.
- **Intersection** and **Union** of languages.
- **Emptiness Checking**: determining whether a given automaton accepts any strings at all.

These operations will be critical when reasoning about universality [2].

1. Problem Description: The Universality Problem

Now, we define the universality problem. The universality problem is a fundamental question in automata theory and formal languages.
The formal definition can be written as:

Given a regular expression R over a finite alphabet Σ, does R describe all possible strings over Σ?
Formally, is $L(R) = \Sigma^*$?

In simpler terms, the task is to determine whether the language defined by R accepts **every string** that can be formed using the symbols from Σ, without any exceptions. This problem has certain properties associated with it:

(1) **Decidability**:
The universality problem for regular expressions is **decidable** [2].
This follows because regular expressions can be converted into finite automata, and finite automata have decidable universality.

(2) **Complexity:**
Although decidable, the problem is **PSPACE-complete** [3].
This means solving it requires a lot of computational resources (memory and time) in the worst case, and it is unlikely to be solvable efficiently for large expressions.

This problem is important both **theoretically** — for understanding the limits of formal language models — and **practically** — in applications like program verification, compiler optimization, and security (e.g., ensuring that certain inputs are always accepted or rejected).

1.1 Approximate Universality

While the standard universality problem asks whether a regular expression (or equivalent automaton) accepts all strings over a given alphabet (i.e., $L(R) = \Sigma^*$), it is often practical to ask a relaxed version of this question—namely, whether the language accepts "almost all" strings [4] under a given probabilistic distribution. This idea leads to the notion of *approximate universality*, wherein the language is deemed universal within a margin of error ε, accepting at least a $(1 - \varepsilon)$ fraction of all strings.

This approximation is particularly relevant in contexts such as coding theory [4], where constructing a maximally inclusive set of strings (e.g., valid codewords) is computationally expensive, but an ε-close approximation is sufficient for practical use. It also allows us to circumvent the PSPACE-hardness of the exact problem using *randomized approximation algorithms* (PRAX), which offer probabilistic guarantees within a tolerance threshold.

2. Standard Solution Strategy for Universality

To decide the universality of a regular expression, a standard algorithmic procedure is typically followed. This procedure systematically reduces the problem to emptiness checking on a finite automaton, leveraging the closure properties of regular languages.

The steps are as follows:

1. **Conversion to NFA**:
   The regular expression R is first converted into an equivalent nondeterministic finite automaton (NFA), using methods such as Thompson's construction [5]. This ensures that the structure of R is preserved while enabling automaton-based analysis.

2. **Conversion to DFA:**
   The resulting NFA is transformed into a deterministic finite automaton (DFA) using the subset construction (powerset construction) [6] method. This step creates a DFA that accepts the same language as the original NFA but with deterministic transitions.

3. **Complementation:**
   The DFA is complemented by swapping its accepting and non-accepting states [2], [8]. As regular languages are closed under complement [2], the resulting DFA accepts exactly the set of strings not accepted by the original regular expression.

4. **Emptiness Checking**:
   Finally, the complemented DFA is examined to determine whether its language is empty [2]:

- If the complement accepts no strings, then $L(R) = \Sigma^*$, and the regular expression is universal.
- If the complement accepts any string, then R does not describe all possible strings.

This method provides a rigorous decision procedure based entirely on automata-theoretic operations. While conceptually straightforward, practical implementation can become computationally intensive due to potential exponential growth during the subset construction phase [7].

Thus, the universality problem lies at the intersection of syntactic construction and semantic verification, requiring a detailed analysis of the underlying structure of the language described by the regular expression.

In contrast to exhaustive algorithms, randomized approximation (PRAX) [4] methods estimate universality via statistical sampling. These methods sidestep full determinization and offer polynomial-time approximations with probabilistic guarantees.

3.Methodology

To rigorously evaluate how large language models (LLMs) handle the universality problem for regular expressions, we employ a structured, multi-phase methodology. This approach emphasizes clarity, reproducibility, and theoretical alignment with automata theory. Each phase is designed to probe not only the correctness of model outputs but also the depth and structure of the reasoning processes underlying those outputs.

Next we give a quick summary of the parts of our methodology, and further below we explain each part in a separate section.

3.1 Model Selection:

We select a diverse set of large language models that differ in their **training objectives** (e.g., general-purpose vs. code-oriented), **architectural features** (e.g., multimodal extensions, context length), and **access mechanisms** (e.g., proprietary APIs vs. open-source implementations). This diversity allows us to investigate whether such differences influence formal reasoning performance on the universality problem. Our goal is to include models known for their capacity to engage in formal reasoning and structured problem solving. Priority is given to models that support explanation-driven outputs and have demonstrated strong performance on algorithmic, mathematical, and symbolic tasks [11]-[14]. The selection is intended to represent a range of capabilities while ensuring that the models are capable of interpreting formal language structures such as regular expressions and finite automata.

3.2 Use of Useful Equivalences:

In designing the problem set, we leverage known **equivalences between regular expressions** [2], [8] to generate multiple structurally distinct but semantically identical forms. These transformations are used to ensure that LLMs are not

merely relying on surface-level pattern recognition but are required to reason about the underlying language defined by each expression. Examples of such equivalences include the distributive property of union over concatenation, simplifications of nested Kleene stars, and the elimination of redundant components. By including these equivalent variants, we are able to test whether models understand that different syntactic forms can define the same language and whether they can apply or recognize such transformations during their reasoning process.

3.3 Problem Set Design:

A custom benchmark of regular expressions is constructed to reflect a wide spectrum of structural complexity and semantic behaviors. The dataset includes both universal and non-universal expressions, as well as adversarial cases and expressions designed to test approximate universality. Examples vary in alphabet size and structural composition, and where appropriate, equivalent forms derived from Section 3.2 are incorporated. This diversity ensures that model evaluation captures reasoning over both the syntax and semantics of regular languages.

3.4 Prompting and Testing:

To examine the effect of prompt structure on model performance, we test each model using multiple prompting approaches. These include direct **zero-shot prompts**, illustrative **few-shot examples**, and **chain-of-thought (CoT) prompts** that explicitly guide the model through the steps of transforming regular expressions into automata and analyzing language coverage. These prompting strategies build on established methods in large language model evaluation [11]–[14]. All prompts are standardized across models to reduce variability and allow for fair comparison, and are designed to elicit not just a binary decision but also an accompanying explanation, enabling deeper insight into model reasoning.

3.5 Evaluation Metrics:

Evaluation is conducted using both quantitative and qualitative criteria. Binary accuracy is recorded based on whether the model correctly determines universality. In addition, we assess the quality of explanations using a rubric grounded in automata-theoretic principles, capturing whether the model refers to concepts such as determinization, complementation, and closure properties. Approximate universality is also considered, allowing us to assess cases where models recognize near-complete coverage. Full details of scoring procedures and validation tools are provided in subsequent sections.

3.6 Error Analysis and Fine-Tuning:

To better understand failure modes, incorrect model outputs are categorized into distinct types, including syntactic errors (e.g., misinterpreting regular expression structure), semantic errors (e.g., incorrect reasoning about automata behavior), and logical errors (e.g., making unjustified generalizations). This categorization helps reveal patterns in model performance and provides guidance for refining prompt design or model selection in future work.

3.7 Conclusion & Reporting:

A concluding section is included at the end of this paper to provide an overall assessment of the study, summarizing how well the models performed and highlighting directions for future work. In addition, the results of the experiments are compiled into this structured report, which presents model performance metrics, typical reasoning errors, and broader insights into the challenges LLMs face when solving formal algorithmic problems.

4.Model selection

This phase of the project focuses on selecting a targeted subset of large language models (LLMs) for empirical evaluation. The overarching goal is to investigate the extent to which modern LLMs, even though they do not run formal algorithms directly, can emulate or approximate reasoning strategies required to solve the universality problem for regular expressions. As such, model selection is a critical component that must balance capability, accessibility, and diversity in architectural design.

The selected models are expected to engage with tasks that require more than surface-level pattern recognition. Specifically, they must demonstrate a capacity for symbolic manipulation, logical inference, and an understanding of formal language constructs such as regular expressions, finite automata, and language containment. These requirements naturally favor models that have shown strong performance on tasks involving structured reasoning, formal logic, and mathematical problem-solving.

To guide the selection process, we consider four key criteria. First, reasoning competence is prioritized: we select models with demonstrated effectiveness on benchmarks involving multi-step reasoning, algorithmic thinking, and formal correctness, such as mathematical proofs or symbolic logic derivations. Second, transparency and accessibility are important for ensuring reproducibility; we favor models with well-documented capabilities and stable API access. Third, architectural variety is sought to allow comparison across different training regimes. This includes instruction-tuned, code-oriented, and general-purpose models. Comparing these helps us see whether training style affects reasoning ability. Fourth, explainability is a central concern. Since part of our evaluation focuses on the reasoning process itself, models that support or respond well to chain-of-thought prompting are preferred.

At this stage, we make no assumptions about the internal mechanisms of the models or their explicit exposure to formal language theory during training. Instead, our selection is based entirely on observable behaviors in structured problem-solving tasks. The models are not merely tools for solving instances, but are studied as systems that may internalize and approximate formal reasoning in unique, architecture-dependent ways.

Following their selection, each model is subjected to a controlled series of experiments involving both universal and non-universal regular expressions. Performance is assessed based on overall correctness, alignment between explanations and formal theory, and consistency across different prompting strategies. Importantly, we evaluate models not only on their exact answers but also on their ability to approximate correctness when exact universality is infeasible. This allows us to capture a broader spectrum of reasoning behavior, including heuristic or probabilistic strategies that may reflect latent understanding even in the absence of full algorithmic precision.

The four models selected for this study represent a diverse cross-section of current large language model architectures and training philosophies. Each was chosen based on its alignment with the evaluation criteria outlined earlier, including reasoning competence, architectural variety, transparency, and explainability. A brief rationale for selecting each model is provided below.

(1) OpenAI's ChatGPT-4:

Chosen for its widely demonstrated performance on complex reasoning tasks, GPT-4 Turbo serves as a strong upper bound in our experiments [11]. Its instruction tuning, support for chain-of-thought prompting, and consistency across mathematical and code-based benchmarks make it an ideal reference point for formal reasoning tasks [11].

(2) Claude 3 sonnet (v 3.7):

Selected for its architectural novelty and emphasis on safety and interpretability, Claude 3 is known to perform competitively on tasks involving logical inference [12]. Its inclusion allows us to test whether alignment-oriented training translates to improved formal reasoning.

(3) Gemini 2.5 Pro:

Gemini's multimodal and code-informed training suggests a strong aptitude for symbolic tasks [13]. We include it to evaluate whether models trained with broader context windows and code datasets show improved performance on automata-theoretic problems.

(4) DeepSeek-Coder R1 Chat:

Chosen for its code-oriented training and open-source accessibility, DeepSeek performs well on tasks involving symbolic reasoning and formal language analysis [14]. Its inclusion allows us to assess how instruction-tuned, code-focused models generalize to problems like regex universality in a reproducible, transparent setting.

5.Use of Useful Equivalences

Regular expressions, despite their compact syntax, can often be written in multiple syntactic forms that define the same language. These equivalences, rooted in the algebraic properties of regular languages, provide critical tools for both simplification and transformation. In this study, we leverage a set of known equivalences not only for preprocessing but also as an implicit test of model understanding: can a large language model (LLM) recognize that two structurally different expressions represent the same language?

To this end, we employ a curated collection of equivalence rules that are standard in formal language theory and widely used in compiler construction and automata-based verification. These include [2],[8]:

- **Identity and Nullability:**

  $\alpha + \emptyset \sim \alpha$

  $\emptyset \cdot \alpha \sim \emptyset$

  $\epsilon \cdot \alpha \sim \alpha$

  $\alpha^* \sim \epsilon + \alpha \cdot \alpha^*$

- **Idempotence and Commutativity:**

  $\alpha + \alpha \sim \alpha$

  $\alpha + \beta \sim \beta + \alpha$

- **Associativity:**

  $\alpha + (\beta+\gamma) \sim (\alpha+\beta) + \gamma$

  $\alpha (\beta\gamma) \sim (\alpha\beta) \cdot \gamma$

- **Distributivity:**

  $\alpha \cdot (\beta+\gamma) \sim \alpha \cdot \beta + \alpha \cdot \gamma$

  $(\alpha+\gamma) \cdot \beta \sim \alpha \cdot \beta + \gamma \cdot \beta$

- **Kleene Star Properties:**

  (α*)* ∼ α*

  (ϵ+α)* ∼ α*

  (α+β)* ∼ (α*β*)*

  ϵ + 0 + (00+000)* ∼ 0*

6.Problem set design

To evaluate the formal reasoning capabilities of large language models (LLMs) on the universality problem, we constructed a custom dataset of regular expressions over the binary alphabet $\Sigma = \{0, 1\}$ using a two-phase strategy:
(i) stochastic generation of syntactically diverse expressions by depth, and
(ii) targeted construction of semantically guaranteed universal expressions.

This approach adapts the randomized approximation framework for NFA universality from [4], ensuring that the dataset captures a broad range of structural patterns and universality behaviors while remaining reproducible.

Phase 1 — Random Generation by Depth

We implemented a recursive expression generator in Python using the FAdo library [9].
The generator operates over $\Sigma = \{0, 1\}$ with no ε or $\emptyset$ in the base step.
At depth 1, only atomic symbols are chosen (CAtom('0') or CAtom('1')).
For greater depths, the generator randomly selects one of four production types:

- **symbol** — atomic symbol from Σ
- **star** — Kleene closure of a recursively generated subexpression (CStar)
- **union** — disjunction of two recursively generated subexpressions (CDisj)
- **concat** — concatenation of two recursively generated subexpressions (CConcat)

Depth corresponds to the maximum nesting level of these operators and serves as a proxy for syntactic complexity.
Each generated regular expression was converted to an NFA via FAdo's .toNFA() method [9].
Universality was then approximately assessed using FAdo's prax_univ_nfa

implementation [9] of the PRAX algorithm, instantiated with a Dirichlet length distribution (t = 2.001) and tolerance parameter ε = 0.01:

GenWordDis(Dirichlet(t=2.001), {'0','1'}, EPSILON)

Expressions returning True from prax_univ_nfa were labeled as approximately universal; others as non-universal.
This classification follows the probabilistic universality-index testing framework of [4].

Phase 2 — Guaranteed-Universal Construction

To balance the dataset and ensure representation of known universal expressions at every depth, we generated additional expressions using *safe algebraic templates* that preserve universality.
Starting from the canonical universal expression (0 + 1)*, we applied operations such as:

- Union with Σ* or with ε-accepting expressions
- Concatenation with ε-accepting subexpressions
- Kleene star of an already universal expression

These templates, grounded in regular-expression equivalences (see Section 5), guarantee that the resulting language remains Σ* [8], regardless of subexpression choice.

Final Dataset

For each depth $d \in [1, 10]$, we generated 200 regular expressions, 100 Universal and 100 Non-Universal, producing ~ 2,000 random expressions in total.
Depth and universality labels are stored alongside the original expression and its structural parse, enabling stratified evaluation of LLM performance by both syntactic complexity and universality status.
This structure allows analysis of reasoning patterns as depth increases and supports reproducibility in future experiments.

7. Prompting and Testing

To evaluate the ability of large language models (LLMs) to solve the universality problem, we designed a controlled prompting framework that tests multiple prompting strategies across a standardized problem set. The goal was to assess not only the correctness of the models' answers, but also the quality, structure, and consistency of their reasoning. Each model was evaluated on the same set of regular expressions, stratified by depth and universality status (both approximate and exact), ensuring that performance comparisons reflect differences in model reasoning rather than differences in input distribution.

We employed three primary prompting strategies: **zero-shot, few-shot, and chain-of-thought (CoT).** In the **zero-shot** setting, models were directly asked whether a given regular expression is universal over a specified alphabet (typically {0,1}) **without prior examples**. This setup tests the model's innate knowledge of formal language concepts. In the **few-shot** setting, prompts included 2–3 annotated examples of universal and non-universal expressions, each accompanied by a brief explanation. For example:

Prompt: "Is the regular expression (0+1)*(00+11)(0+1)*universal over the alphabet {0,1}? Explain your reasoning."
Answer: "No. This expression requires that every accepted string must contain either the substring '00' or the substring '11'. Therefore, it excludes strings such as 01, 10, or any string without consecutive identical symbols. Since not all binary strings are accepted, the expression is not universal."

By providing such examples, the few-shot strategy simulates in-context learning and probes whether models can generalize from structural patterns. The most detailed testing condition involved **chain-of-thought prompting**, in which models were guided through the formal reasoning process step by step. These prompts encouraged models to (i) convert the regex into an equivalent automaton [5][6], (ii)

describe the structure of accepted strings, (iii) consider whether any strings might be excluded, and (iv) conclude universality status based on language coverage.

All prompts were kept structurally consistent across models and experiments to reduce prompt-induced bias. Care was taken to avoid using surface-level patterns that could trivially reveal universality; instead, prompts were designed to require reasoning about semantic coverage and the absence of counterexamples. For example, expressions like $(0+1)^*$ were mixed with deceptive non-universal expressions such as $(0+1)^*0(0+1)^*$, which accept large but incomplete subsets of Σ^*. Where applicable, model outputs were parsed to extract both a binary answer ("Yes"/"No") and an accompanying rationale, which was later evaluated independently.

By varying the prompt type and capturing both decisions and justifications, we were able to systematically probe whether model performance results from superficial pattern recognition or deeper automata-theoretic reasoning. For instance:

- **Zero-shot prompt:** *"Is the regular expression $(0+1)^*$ universal over {0,1}? Answer Yes or No, and justify your reasoning."*
- **Few-shot prompt:** *Example 1 — (0+1)* is universal because it accepts all binary strings. Example 2 — (0+1)*0(0+1)* is not universal because it excludes strings that end in 1. Now, is (1*0*) universal? Explain your answer.*
- **Chain-of-thought prompt:** *"Convert the regex (0+1)*0(0+1)* into an automaton. Describe what kinds of strings it accepts, then check whether any binary strings are excluded. Conclude whether it is universal."*

These examples illustrate how prompts were designed to elicit different levels of reasoning: from direct judgments in zero-shot, to in-context learning in few-shot,

to structured logical reasoning in chain-of-thought. This also allowed us to measure how model behavior changes under cognitive load (as imposed by syntactic depth or semantic ambiguity), and whether prompting strategy affects consistency, reliability, or error types in LLM-generated outputs.

8.Evaluation Metrics:

Evaluation is conducted using both quantitative and qualitative measures to capture the breadth of LLM performance on the universality problem. At the quantitative level, we record binary accuracy—the proportion of cases where the model's universality judgment matches the ground truth label, obtained either through exact automata-theoretic computation or high-confidence PRAX classification. To account for the probabilistic nature of approximate universality, we also compute tolerance-aware accuracy, crediting models for correctly identifying expressions as "almost universal" when the measured universality index meets or exceeds the $(1-\varepsilon)$ threshold.

In addition to simple correctness, we track **precision** and **recall** [15]: precision measures the share of predicted universal expressions that are actually universal, while recall measures the share of true universal expressions that are correctly identified. These two metrics together capture how well models balance false positives (incorrectly labeling non-universal expressions as universal) and false negatives.

At the qualitative level, model explanations are scored against a rubric grounded in formal language theory. The rubric checks for reference to essential concepts such as NFA/DFA conversion, closure properties, complementation, and emptiness checking. It also evaluates whether the reasoning steps follow a coherent and logically valid sequence, whether potential counterexamples are considered, and whether the explanation demonstrates awareness of the distinction between exact and approximate universality. For example, a strong explanation for the regex (0+1)* might state: "Yes, this expression is universal because it allows any sequence of 0s and 1s, including the empty string, which covers all strings over the alphabet {0,1}." By contrast, a weaker explanation might simply say: "Yes, because it accepts everything," without justifying why.

9. Error Analysis and Fine-Tuning:

Following initial evaluation, we conducted a systematic error analysis to categorize the types of mistakes made by the models. Errors were grouped into three main categories:

- **Syntactic errors** — misinterpreting the structure of the regular expression.
  Example: A model treats (ab)* as accepting only the string "ab" instead of repetitions such as "abab" or "ababab."
- **Semantic errors** — incorrect reasoning about the language accepted by the corresponding automaton.
  Example: A model claims that (0+1)*0(0+1)* is universal, overlooking that it excludes strings that have no "0's".
- **Logical errors** — drawing invalid conclusions despite otherwise correct intermediate steps.
  Example: A model correctly recognizes that (0+1)* generates all binary strings but then incorrectly concludes that it is *not* universal.

For each error category, representative cases were examined to determine whether failures stemmed from knowledge gaps, reasoning flaws, or prompt misinterpretation.

Patterns observed during error analysis directly informed refinements to both prompts and evaluation design. Frequent **syntactic misreads** prompted the inclusion of explicit parenthesization and step-by-step parsing instructions in chain-of-thought prompts. **Semantic errors** related to closure properties and complement operations motivated the addition of targeted few-shot examples covering these concepts. **Logical inconsistencies** were addressed by adding verification steps to prompts, encouraging models to re-check intermediate conclusions before finalizing answers.

Based on these findings, several fine-tuning strategies were considered to improve performance. These included generating **synthetic training data** focused on known failure patterns, incorporating **equivalence-based reasoning exercises** (from Section 5), and designing **curriculum-style prompts** that gradually increase expression complexity. While full fine-tuning was beyond the scope of this study, these strategies provide a clear roadmap for enhancing model robustness and supporting generalization in future work.

10. Conclusion:

As shown in the tables below, across all four models tested, performance on the universality problem was consistently strong. ChatGPT-4o, Claude 3 Sonnet, and DeepSeek achieved perfect accuracy (100%) on both universal and non-universal regular expressions across all depths tested up to string length 12, showing complete reliability within the evaluated range. Gemini v2.0 also performed at a very high level, matching the other models in most cases, but showed slight drops in universality accuracy at intermediate depths (97% at depth 4 and 99% at depth 5). These small deviations suggest that Gemini may occasionally struggle with certain structural complexities, whereas the other three models remained flawless. In addition to the tests reported in the tables, we also did a few isolated tests that involved non-universal expressions whose language included all strings of length up to 36. With proper initial prompts, the LLMs were able to correctly report that the given regular expressions are indeed non-universal. Overall, the results indicate that state-of-the-art LLMs are highly capable of reasoning about regular expression universality at the tested scale, with only minor variability across architectures.

Future Work: While the models performed exceptionally well on the tested dataset, future research should extend these experiments to more complex regular expressions, including larger alphabets, deeper nesting levels, and adversarially constructed cases. Additional testing could also examine how fine-tuning strategies and more varied prompting techniques influence model robustness under increasing syntactic and semantic complexity.

CHATGPT-4o (Upto string length 12)		
Depth	LLM Accuracy for non-universal	LLM Accuracy for universal
1	100%	100%
2	100%	100%
3	100%	100%
4	100%	100%
5	100%	100%
6	100%	100%
7	100%	100%
8	100%	100%
9	100%	100%
10	100%	100%

Claude 3 sonnet (v 3.7) (Upto string length 12)		
Depth	LLM Accuracy for non-universal	LLM Accuracy for universal
1	100%	100%
2	100%	100%
3	100%	100%
4	100%	100%
5	100%	100%
6	100%	100%
7	100%	100%
8	100%	100%
9	100%	100%
10	100%	100%

DeepSeek (Upto string length 12)		
Depth	LLM Accuracy for non-universal	LLM Accuracy for universal
1	100%	100%
2	100%	100%
3	100%	100%
4	100%	100%
5	100%	100%
6	100%	100%
7	100%	100%
8	100%	100%
9	100%	100%
10	100%	100%

Gemini v2.0 (Upto string length 12)		
Depth	LLM Accuracy for non-universal	LLM Accuracy for universal
1	100%	100%
2	100%	100%
3	100%	100%
4	100%	97%
5	100%	99%
6	100%	100%
7	100%	100%
8	100%	100%
9	100%	100%
10	100%	100%

References:

[1] M. O. Rabin and D. Scott, “Finite automata and their decision problems,” IBM J. Research and Development, vol. 3, no. 2, pp. 114–125, 1959.

[2] J. E. Hopcroft, R. Motwani, and J. D. Ullman, Introduction to Automata Theory, Languages, and Computation, 3rd ed., Addison-Wesley, 2006.

[3] M. De Wulf, L. Doyen, T. A. Henzinger, and J.-F. Raskin, “A New Algorithm for Checking Universality of Finite Automata,” CAV 2006.

[4] S. Konstantinidis, M. Mastnak, N. Moreira, and R. Reis, “Approximate NFA universality and related problems motivated by information theory,” Theoretical Computer Science, vol. 972, p. 114076, 2023, doi: 10.1016/j.tcs.2023.114076.

[5] K. Thompson, “Programming Techniques: Regular expression search algorithm,” Communications of the ACM, vol. 11, no. 6, pp. 419–422, 1968.

[6] M. O. Rabin and D. Scott, “Finite automata and their decision problems,” IBM J. Research and Development, vol. 3, no. 2, pp. 114–125, 1959.

[7] “Powerset construction,” Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Powerset_construction

[8] D. Kozen, “A Completeness Theorem for Kleene Algebras and the Algebra of Regular Events,” Information and Computation, vol. 110, no. 2, pp. 366–390, 1994.

[9] FAdo Documentation, v2.2, 2024. [Online]. Available: http://fado.dcc.fc.up.pt

[10] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*. MIT Press, 1999

[11] OpenAI, GPT-4 Technical Report, arXiv:2303.08774, 2023.

[12] Anthropic, Introducing Claude 3, Anthropic Blog, Mar. 2024. [Online]. Available: https://www.anthropic.com/news/claude-3

[13] Google DeepMind, Gemini 1.5: Unlocking Multimodal Reasoning at Scale, Feb. 2024. [Online]. Available: https://deepmind.google/technologies/gemini/

[14] DeepSeek-AI, DeepSeek-Coder: Open Models for Code Intelligence, arXiv:2401.14196, 2024.

[15] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge University Press, 2008.