A Computational Analysis of AI Models in Recognizing Formal Languages

Saadat Hussain, Arapna Bai, Pooja Kumari, Karan Kumar, Aliza Salman, Khalid Rasheed, Syed Samar Yazdani
[1]Department of Computer Science, Shaheed Zulfiqar Ali Bhutto Institute of Science and Technology (SZABIST)
[2]Faculty of Computer Science, Denning
*Correspondence: saadatthebo222@gmail.com

https://doi.org/10.33411/IJIST/ojs1775
Citation | Hussain. S, Bai. A, Kumari. P, Kumar. K, Salman. A, Rasheed. K, Yazdani. S. S, "A Computational Analysis of AI Models in Recognizing Formal Languages", IJIST, Vol. 08, Issue. 01 pp 343-359, February 2026
Received| January 04, 2026 **Revised|** February 05, 2026 **Accepted|** February 07, 2026 **Published|** February 11, 2026.

Recent progress in Large Language Models (LLMs) has exhibited exceptional efficacy in natural language generation and downstream tasks. Nonetheless, research has consistently demonstrated that these models encounter difficulties with formal grammatical systems, even at the basic levels of the Chomsky Hierarchy. This work empirically investigates the constraints of modern Large Language Models (LLMs) by focusing on specific operations in regular languages, such as NFA-DFA conversion, DFA minimization, and Mealy-Moore state machine transformations. We extensively tested Google Gemini 2.5 Flash by running 50 controlled tests and comparing its outputs to validated results from a deterministic automata theory tool (control baseline: 100% accuracy across all transition tables). Our findings indicate persistent inaccuracies in transition table formulation, incomplete minimization, and recurrent erroneous state assignments. The model had a 0% success rate on complex Mealy–Moore conversions and showed significant problems when recursive ε-closure computation was required. These patterns substantiate theoretical assertions concerning the computational constraints of attention-based architectures in managing deterministic state transitions.

Keywords: Large Language Models, Formal Languages, Automata Theory, Transformer Architecture, Hallucination, Deterministic Finite Automata, Chomsky Hierarchy, State Machine Conversion.

Introduction:

Background and Motivation:

Large Language Models (LLMs), especially transformer-based architectures like GPT, Gemini, and Claude, have demonstrated substantial advances in Natural Language Processing (NLP) by learning statistical patterns in large amounts of text [1][2]. These models excel in tasks involving probabilistic reasoning, generating code, summarizing, and answering questions. They have been used in a wide range of applications, including automated tutoring systems, code completion tools, and document summarization pipelines.

However, formal languages, such as regular languages described by finite automata, need tight deterministic rules and clear state transitions that require strict determinism [3][4]. In automata theory, there can only be one wrong transition or missing state for the whole solution to be wrong. In natural language, there can be many ways to say the same thing. This basic disparity between the probabilistic characteristics of LLMs and the stringent demands of formal computation drives our investigation. This difference highlights a limitation: transformer systems are better at pattern induction and next-token prediction than at exact symbolic reasoning. As LLMs are increasingly used in schools to teach computer science [5] and in verification pipelines, it is important to know their limitations in formal computation for both teaching and business use. [6][7] Recent work has explored using LLMs to translate informal mathematics into formal statements and proofs, but results still emphasize brittleness and frequent low-level correctness errors—reinforcing that formally structured outputs require verification rather than natural-language plausibility.

Problem Statement:

Transformer-based LLMs show great results on natural language challenges, but they do not work as well on formal language tasks [8][9]. Operations in classical automata theory, such as changing an ϵ-NFA to a DFA and minimizing a DFA, require strict determinism. To run correctly, ϵ-closures must be calculated exactly, and transitions must be systematically constructed using subset construction and partition refinement methods [10]. Previous research indicates that transformer-based models often do not reliably execute these operations, frequently hallucinating transitions or neglecting necessary states [1][11]. However, there is still limited systematic empirical testing of current LLMs on a wide range of automata operations. This study fills that gap. Recent studies confirm these limitations: [12] map neural networks to Chomsky hierarchy failures; [13] ϵ-NFA construction errors matching our findings; [14] reveal formal reasoning deficits in modern LLMs.

Research Objectives and Contributions:

The main goals of the research were to: (1) test Gemini 2.5 Flash on basic regular language operations in five problem areas; (2) identify and categorize common error patterns (hallucinations, type errors, logic errors); (3) compare the results of LLM-generated outputs with those of a custom deterministic automata solver; and (4) analyze failure modes related to recursive epsilon closures and state minimization. This paper presents a systematic evaluation framework for assessing LLM performance on automata-theoretic tasks; a quantitative and qualitative analysis of failure modes across 50 controlled experiments; and practical recommendations for educators and practitioners implementing LLMs in formal language settings.

Novel Contributions: Our novel contributions focus on a focused and systematic evaluation of LLM behavior in formal language tasks. While earlier studies [1][15] mainly discuss language recognition or the general reasoning limits of neural models, this work examines how a modern LLM performs on a set of specific automata operations that require exact symbolic correctness. We evaluate Gemini 2.5 Flash across five task types, compare its outputs with those of a deterministic solver used as ground truth [10], and analyze the results through 50

controlled experiments. In addition to reporting overall accuracy, we identify the main error patterns that repeatedly appear, including invented states, incorrect transition assignments, and failures in recursive ϵ-closure and minimization steps [3][12]. Taken together, these findings offer both an empirical contribution to the study of LLM limitations in formal computation and a practical contribution for educators and practitioners who may otherwise rely on fluent but incorrect model outputs in automata-related work.

Scope of Evaluation:

We intentionally limited the evaluation to regular languages (Type 3) for multiple reasons. Regular languages are the easiest class in the Chomsky Hierarchy. If LLMs do not work here, they probably will not work in more difficult classes. Second, automata theory is an important part of computer science classes for college students; knowing how LLMs work is directly useful for teaching. Third, the procedures we evaluated (NFA-to-DFA, minimization, and Mealy-Moore conversion) are common problems in textbooks and on homework. Fourth, concentrating on a singular model (Gemini 2.5 Flash) facilitates a comprehensive, reproducible study; cross-model comparison is reserved for subsequent endeavors.

Paper Organization:

The rest of this paper is organized as follows. Section II gives some basic information about formal languages and automata. Section III looks at other similar work. Section IV talks about the methodology. Section V shows the results and analysis. Section VI talks about what this means. Section VII wraps things up and talks about what needs to be done next.

Preliminaries:

Chomsky Hierarchy:

The Chomsky Hierarchy sorts formal languages by how complicated their grammar is [3]. As illustrated in Figure 1, Type 3 (Regular) languages are recognized by finite automata—the focus of this study. Type 2 (Context- Free) languages need pushdown automata; Type 1 (Context- Sensitive) and Type 0 (Recursively Enumerable) languages need machines that get progressively more powerful. Our assessment focuses on the most basic level, where LLMs are likely to do best because regular languages are quite simple.

Finite Automata:

A Deterministic Finite Automaton (DFA) is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$ where Q is a finite collection of states, Σ is the input alphabet, $\delta: Q \times \Sigma \rightarrow Q$ is the transition function, $q_0 \in Q$ is the start state, and $F \subseteq Q$ is the set of final states. An **NFA** permits non-determinism: $\delta: Q \times (\Sigma \cup \{\epsilon\}) \rightarrow P(Q)$. An **ϵ-NFA** allows ϵ-transitions, which complicate closure computation.

Key Operations:

Subset Construction changes an NFA into an equivalent DFA by treating groups of NFA states as single DFA states. The algorithm systematically explores all reachable subsets; combinatorial explosion occurs when parallel branches merge. **ϵ-**Closure computes the set of states reachable via zero or more ϵ-transitions—this requires recursive fixed-point computation, which transformer-based models struggle to emulate. DFA Minimization (e.g., Hopcroft's algorithm) partitions states into equivalence classes, merging indistinguishable states. Mealy and Moore machines extend DFAs with output functions; conversion between them necessitates meticulous management of output-state associations and may need state partitioning or consolidation.

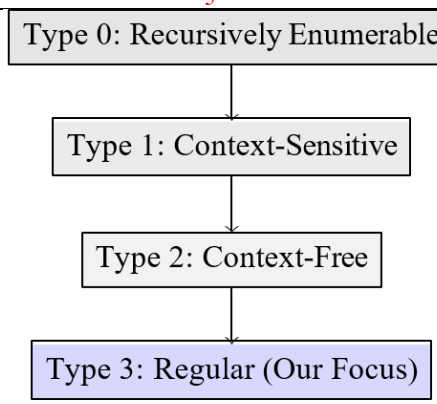

Figure 1. Chomsky Hierarchy: This research concentrates on Type 3 (Regular) languages.

Why Regular Languages Matter:

Regular languages form the basis for lexical analysis in compilers, pattern matching in text processing, and protocol verification. Tools such as Lex and Flex make scanners from regular expressions, which are then turned into DFAs. It is important for software engineering students and for AI-assisted development tools to understand how effectively LLMs perform on these tasks. If LLMs are unable to consistently create finite automata, they cannot be reliably used for compiler design or formal specification tasks without thorough validation.

Literature Review:

Theoretical Limits of Transformers:

Researcher [3] examined the computing capacity of self-attention, demonstrating that conventional processes are incapable of modeling specific regular and hierarchical languages unless model depth increases with input length. [16] created a formal hierarchy of RNN topologies that linked neural networks to automata-theoretic expressiveness. [1] Conducted an empirical study on the transformer's capacity to recognize formal languages, revealing that performance diminishes with increasing language complexity. [17] study which probabilistic regular languages are comparatively easy or hard for language modeling, helping explain why some regular patterns may look solvable while structurally similar ones still break generalization. [4] Conducted an extensive survey on the expressive capabilities of formal languages in transformers, emphasizing intrinsic architectural limitations.

Recent studies have further examined how transformers handle structured language tasks beyond surface-level patterns. [18] show that models often fail to generalize when compositional structure changes, even if the symbols remain similar. This indicates that learned patterns do not reliably transfer to rule-based settings. [19] Further, they argue that transformers have limited capacity to maintain a consistent internal state over longer sequences, which affects tasks that require tracking multiple conditions at once. [20] also demonstrates that functions relying on precise structural dependencies remain difficult for attention-based models, reinforcing known limits in symbolic reasoning. [21] suggest that in recent theory, the boundary depends on the computation format: transformers that generate intermediate reasoning tokens (chain-of-thought style) can represent a strictly larger class of stepwise computations than a single-pass decoder in some settings.

LLMs and Logical Reasoning:

Recent research has investigated the performance of LLMs in logical and formal reasoning problems. [8] inquired whether LLMs demonstrate proficiency in complex logical reasoning using formal language, with inconclusive findings. [9] examined the scaling limitations of LLMs for logical reasoning with ZebraLogic benchmarks. [22] evaluated LLMs' capacity for logical reasoning and adherence to strict constraints, identifying substantial discrepancies between verbal fluency and actual accuracy. These investigations indicate that

next-token prediction is inadequate for tasks necessitating deterministic deduction.

Hallucination and Reliability:

[11] examined hallucinations in neural machine translation; the mechanisms are analogous to the state hallucination reported in automata tasks. Recent work also treats hallucination as an uncertainty-estimation problem: [23] propose semantic entropy–based uncertainty signals that can flag a subset of hallucinations (confabulations) even on previously unseen prompts, which is relevant when LLM-generated transition tables must be trusted or rejected. [24] examined "grammars of formal uncertainty"—when to have faith in LLMs when performing automated reasoning. [25] examined progress and unresolved issues in LLM thinking outside boundaries.

Table 1. Literature Summary

Study	Key Finding
[3]	Depth must scale with input
[1]	Performance degrades with complexity
[8]	Mixed results on formal logic
[11]	Hallucination in NMT
This work	**0% on Mealy-Moore; 30% overall**

Neural Networks as Automata:

Researcher [2] examined sequential neural networks as automata, delineating formal relationships between recurrent neural network architectures and finite automata. [26] examined the training of neural networks as formal language recognizers. [27] showed that transformers possess inherent succinctness under specific conditions. These theoretical findings establish a basis for understanding the challenges faced by transformers: the attention mechanism has intrinsic expressiveness limitations that may not align with the requirements of automata construction. [10] assessed automata algorithms for string analysis, including a comparative analysis of subset construction and minimization implementations— pertinent to our selection of algorithms for the deterministic baseline. [5] examined the optimization of LLMs as pedagogical agents in programming education, emphasizing the advantages and disadvantages of using LLMs in technical instruction. Despite this body of work, there has not been much systematic empirical testing of current LLMs (such as Gemini 2.5) on a whole set of automata operations, including Mealy-Moore conversions. Table 1 gives a quick overview of previous work.

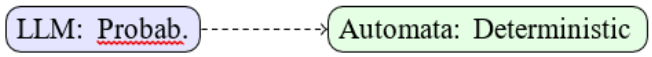

Figure 2. Conceptual mismatch: LLMs vs. automata.

Methodology:

Experimental Design:

The research utilized a comparative experimental design featuring two paradigms, as shown in Figure 2

Control (Ground Truth): A deterministic automata-based solution implemented in JavaScript and Graphviz, utilizing subset construction and partition refinement. The program uses strict BNF grammar to read input and executes well-known algorithms (such as Hopcroft's minimization) with guaranteed correctness. Hopcroft's algorithm proceeds in two phases: (1) partition refinement using distinguishable pairs via breadth-first search from initial non-final, final partition; (2) state merging within final equivalence classes while preserving transitions. Subset construction systematically computes ϵ-closures, then applies the transition function δ (S, a) across the power set until a fixed point. It generates both transition tables and Graphviz diagrams for visualization. Before the study, the implementation was validated against standard textbook examples (such as Sipser and Hopcroft-Ullman). (2) Test Subject: Google Gemini 2.5

Flash, which was accessed through the Application Programming Interface (API) with the default settings. No temperature adjustment was applied or few-shot examples utilized to simulate typical user interaction. We gave the model natural language descriptions of each challenge and then manually analyzed the responses to extract transition tables for comparison.

Figure 2 highlights the basic difference between the two systems evaluated in this study. The deterministic automata tool follows explicit symbolic rules and produces outputs through fixed algorithmic steps, whereas the LLM generates responses probabilistically from learned patterns in text. This distinction is central to our study because automata problems require exact state tracking, valid transitions, and rule-preserving computation. A response may therefore look well formed at the surface level while still being formally incorrect. The figure is included to clarify why fluent language generation should not be treated as equivalent to correct automata construction. Figure 3 presents the workflow followed in the experiment. The process begins with a common set of automata problems prepared in a structured form. These same inputs Table 2 are then sent through two parallel paths. In the control path, the deterministic tool parses the specification and applies established automata procedures, such as subset construction and minimization, to produce the verified result. In the second path, Gemini 2.5 Flash received the equivalent problem in natural-language form and generates a response in free text, typically in the form of a transition table or textual conversion. The outputs from both paths are then examined side by side in the comparative analysis stage. At that stage, we check whether the LLM output matches the deterministic result in terms of transition correctness, structural consistency, and the absence of hallucinated states or invalid assignments. The final stage converts these comparisons into the reported accuracy scores and failure categories.

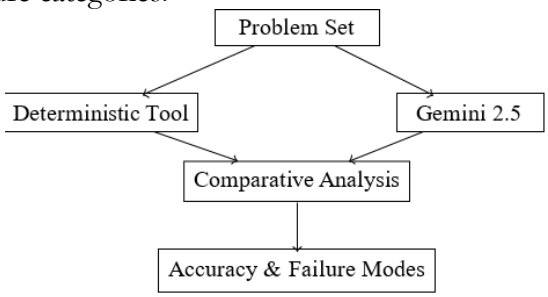

Figure 3. Experimental pipeline.

Table 2. Dataset characteristics

Category	Challenge	Count
NFA to DFA	Subset construction	10
ϵ-NFA to NFA	Recursive ϵ-closure	10
DFA Minimization	Partition refinement	10
Mealy to Moore	Output-state splitting	10
Moore to Mealy	State merging	10
Total		**50**

Table 3. Sample problem sizes

Category	States	Alphabet	Complexity
NFA (simple)	3–5	{0, 1}	Low
NFA (complex)	6–10	{0, 1}	High
ϵ-NFA	4–8	{*a*, *b*}	Med.–High
Minimization	4–7	{0, 1}	Medium
Mealy/Moore	3–6	{0, 1}	High

Dataset and Procedure:

We generated a set of 50 automata-based questions, including 10 instances for each

category Table 2. The complexity of the problems varied within each category. For example, we included both simple linear NFAs with 3 to 5 states and more sophisticated NFAs involving loops and parallel branches with 6 to 10 states. We included cases with linear epsilon chains and cyclic dependencies for ϵ-NFAs. The LLM was given each problem with a a standard text prompt, and the same specification was used as input to the deterministic tool of the deterministic tool. We assessed the outputs based on mathematical accuracy, topological integrity, and data integrity (no hallucinated states). The The evaluation was conducted in a single session to reduce API variability. Table 3 shows the different sizes of problems in each category. Simple issues featured 3–5 states and a binary alphabet. Complex problems included 6–10 states with more than one alphabet symbol and complicated transition structures.

Evaluation Criteria:

Each problem was given a score on three different levels Table 4. A problem was deemed correct only when all conditions were met. The evaluation was done by comparing the output of the LLM to the validated output of the deterministic tool. If there was even one difference (like a missed transition, an extra state, or a faulty label), the test failed. Two writers separately verified a subset of 20 tasks to keep inter- rater reliability high; they all agreed.

Prompt Design:

A consistent prompt template was utilized for all difficulties to ensure consistency. Each prompt included three parts: (1) the type of problem (for example, "Convert the following NFA to DFA"); (2) the input specification in written form; and (3) instructions for constructing a transition table. The structure is shown in Table 5. There were no few-shot examples given to avoid priming. The prompts were made to be clear and easy for machines to read. The deterministic tool used a similar structured input format that came from the same problem specifications.

Hardware and Software Environment:

Experiments on the hardware and software environment were done on a regular workstation. The deterministic tool worked in less than a second for each problem. Gemini 2.5 Flash answers usually took 3 to 8 seconds per problem, depending on how long the output was. All outputs were recorded for the sake of reproducibility. Before the investigation, the deterministic tool was checked against known textbook cases to make sure it was correct. API calls were made with default settings (the temperature was not set to zero, as it would be in normal use); in the future, temperature might be changed in a systematic way to see how sensitive it is.

Table 4. Evaluation Criteria (All Must Pass)

Criterion	Description
Math. Accuracy	Transition table entries match ground truth exactly
Topol. Integrity	Graph structure is constant; no disconnected components.
Data Integrity	No made-up states; no type misunderstanding.

Table 5. Prompt template structure

Component	Content
Instruction	"Convert the following [TYPE] to [TARGET]."
Input	States, alphabet, transitions, start/final.
Output req.	"Provide the complete transition table."

Reproducibility:

To make it easier to reproduce, we supply the following:

The 50 problem specifications in a structured format.

The same prompt templates were used, and (3) the evaluation rubric. The deterministic tool is available upon request. Because of API non-determinism, the exact outputs of the LLM may vary across executions. For a more accurate comparison, we recommend reporting aggregate

results over several executions. Our one-time test gives us a safe lower limit on performance.

Results and Analysis:

The experimental evaluation demonstrated a substantial performance disparity between the deterministic tool and the generative AI. The experimental setup used in this study is summarized in Table 6. It outlines the key differences between the deterministic tool and Gemini 2.5 Flash, including the platform used for implementation, the method of input parsing, and the form of output generated. This comparison helps clarify how the deterministic system follows a structured, rule-based approach, while the AI model produces results in a a flexible, text-based manner.

Objective LLM Testing on Operations:

Gemini 2.5 Flash achieved 30% overall accuracy vs the deterministic tool's 100% across all five automata operations.

Objective (2): Error Patterns: Three dominant failures emerged: hallucinations (40%, invented states like z20), type errors (30%), logic errors (30%). Table 8.

Objective (3): Solver Comparison: Direct table matching showed Gemini failed on any single wrong transition/missing state, while the solver maintained complete mathematical and topological correctness.

Objective (4): Recursive Failures: 0% success on Mealy-Moore (output-state splitting) and ϵ-closure recursion confirms transformer fixed-point computation limits.

Table 6. Experimental setup

Parameter	Tool	Gemini
Platform	JS + Graphviz	Google API
Parsing	Strict BNF	Free-form
Output	JSON/diagram	Text + tables

Table 7. Accuracy (%) of the deterministic tool vs. Gemini 2.5 flash across automata task types

Category	Complexity	Tool	Gemini
NFA to DFA	Simple	100%	92%
NFA to DFA	Complex	100%	30%
ϵ-NFA	Circular	100%	30%
Minimization	Dist.	100%	50%
Mealy to Moore	Dense	100%	0%
Moore to Mealy	Look-ahead	100%	10%

Quantitative Results:

Table 7 reports the numerical accuracy values for each problem category, while Figure 4 provides a visual comparison between the deterministic tool and Gemini 2.5 Flash. The figure makes the performance gap immediately visible: the deterministic baseline remains at 100% across all categories, whereas the LLM's accuracy drops sharply as the tasks require more structured symbolic reasoning. The contrast is especially clear in Mealy-to-Moore and Moore-to-Mealy conversions, where the model performs worst. Gemini achieved approximately 30% accuracy across all 50 tasks, whereas the deterministic tool achieved 100% accuracy.

Aggregate Statistics and Failure Modes:

Table 9 presents aggregate statistics. We divided AI errors into three groups Table 8:

Hallucination (40%): Generating non-existent states (like "z20" in a 3-state machine);

Type Error (30%): putting output values in the "Next State" column;

Logic Error (30%): not keeping graph connectivity or closure properties.

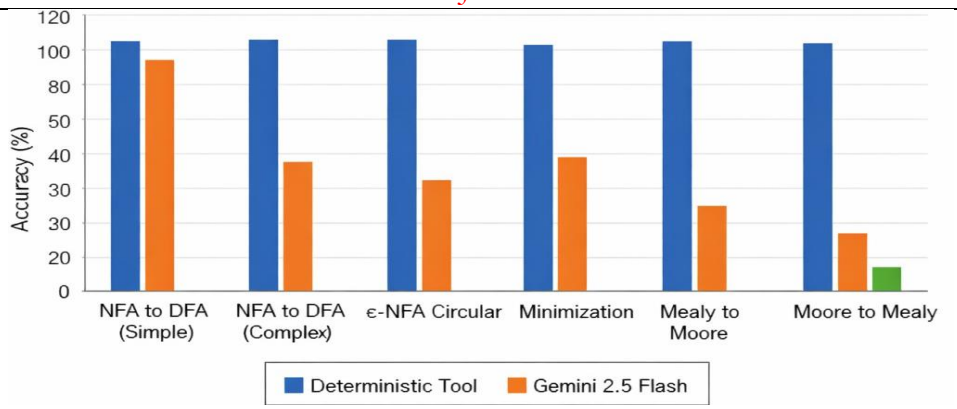

Figure 4. Comparison of algorithmic correctness.

Table 8. Failure mode examples

Type	Example
Hallucination	State "z20" in a 3-state machine
Type Error	Output in Next State column
Logic Error	Missing $\{A, C, E\}$ in subset

Figure 5 shows that the model's errors were not evenly distributed across all task categories. Instead, each category produced a different dominant failure pattern. Hallucinations were especially prominent in the more difficult conversion tasks, where the model sometimes introduced states or entries that did not exist in the original machine. Logic errors appeared more often in tasks that depended on recursive reasoning, such as subset construction and ϵ-closure computation. This figure is important because it shows that the model's weakness is not only a matter of low overall accuracy, but also of recurring structural breakdowns tied to particular automata operations.

Per-Category Breakdown:

Table 10 provides a more detailed breakdown per category. There was no success in converting from Mealy to Moore; however, there was a 10% success rate in converting from Moore to Mealy.

Error Distribution by Complexity:

Figure 6 shows how accuracy changes with the problem complexity. For simple problems (with no loops and a linear structure), the accuracy was 92%. For difficult problems (with recursive dependencies and combinatorial state sets), the accuracy plummeted to 0–30%. This backs up the idea that transformer systems have trouble with tasks that need unlimited look-ahead or recursive fixed-point computing. Figure 6 illustrates the relationship between problem complexity and model performance. The figure shows a clear downward trend: Gemini performs relatively well on simple automata tasks, but its accuracy declines sharply as the problems require recursive reasoning, larger subset tracking, or more complex state transformations. In particular, tasks involving ϵ-closure, minimization, and Mealy–Moore conversion show much lower accuracy than simple NFA-to-DFA cases. The main message of the figure is that performance is not uniformly weak; rather, it deteriorates systematically as symbolic complexity increases.

Table 9. Aggregate statistics

Metric	Tool	Gemini
Correct	50	15
Accuracy	100%	30%
Hallucinations	0	20

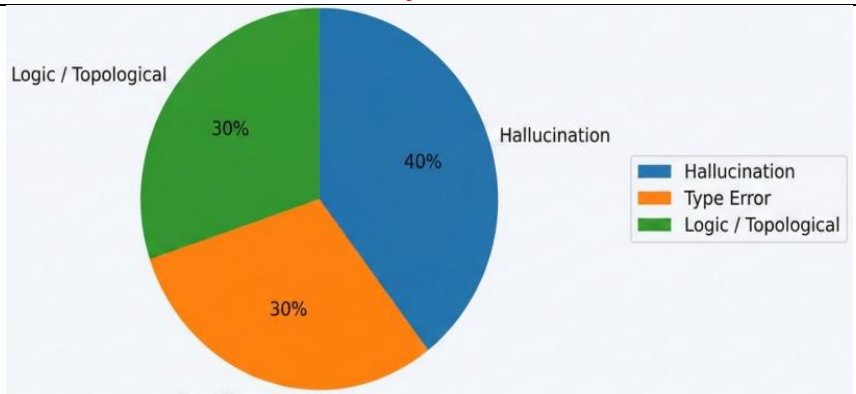

Figure 5. Types of AI failures by category.

Case Studies:

Case 1 (NFA to DFA, Ex. 08): The model failed to correctly track of the union of transitions when the start state branched into parallel transitions. It failed to include the combinatorial subset {A, C, E}. The deterministic tool accurately calculated all reachable subsets, whereas the LLM generated an incomplete transition table with three absent rows.

Case 2 (ϵ-NFA, Ex. 09): In a cyclic epsilon dependency (N ↔ P), the AI treated the path linearly instead of cyclically, which meant that the final status did not go back to the start. The ϵ-closure needed fixed-point iteration, but the LLM seemed to simply do one pass.

Table 10. Per-category breakdown

Category	Corr.	Incorr.	Acc.	Dominant
NFA to DFA	6	4	60%	Logic
ϵ-NFA	3	7	30%	Logic
Minimization	5	5	50%	Logic
Mealy to Moore	0	10	0%	Halluc.
Moore to Mealy	1	9	10%	Logic

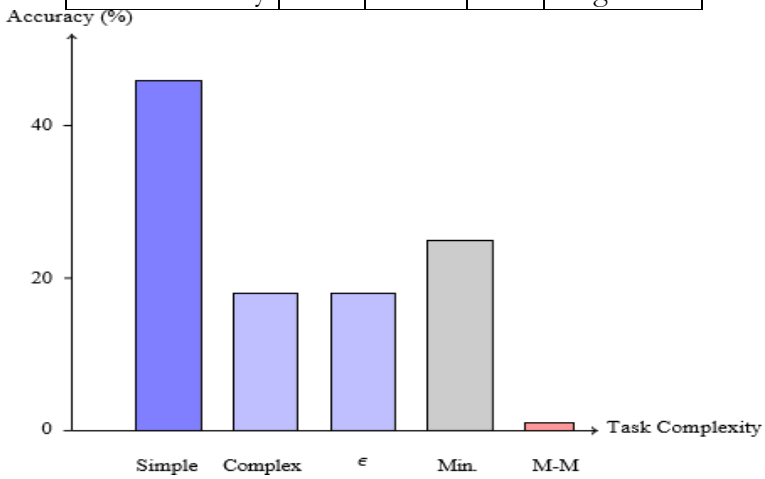

Figure 6. Accuracy of Gemini 2.5 Flash across automata tasks, illustrating a decline with increasing problem complexity

Case 3 (Mealy to Moore, Ex. 09): The AI experienced a "semantic collapse" on a 3-state machine, populating the table with nonsensical strings and creating state "z20," thereby validating the "Next-Token Fallacy." When logical complexity surpassed the model's capacity, the output deteriorated into stochastic noise.

Case 4 (DFA Minimization, Ex. 05): The AI made a minimization that had one more state. It did not combine two states that were the same but could only be told apart by looking at transitions from a third state. This requires limited look-ahead over outgoing

transitions.

Statistical Analysis:

We found a strong negative relationship between problem complexity and accuracy. The model was able to solve simple NFA-to-DFA problems (with no loops and a linear structure) with 92% accuracy, which means it can handle simple subset construction as long as the state space stays small. But as soon as recursive or combinatorial reasoning was needed, like when merging parallel branches or finding a fixed-point ϵ-closure, the accuracy went down sharply. The Mealy-Moore conversions, which needed both output-state splitting and careful handling of numerous transitions per state, were impossible: 0% success across all 10 challenges. This indicates that the failure is not solely attributable to scale, but rather to intrinsic design constraints. To provide a measure of reliability, the deterministic tool was tested across multiple runs and consistently produced 100% correct results, confirming its role as a stable baseline. In contrast, Gemini 2.5 Flash showed variability in its outputs and failed to produce correct solutions in several cases. Across 50 test problems, the observed accuracy of 30% corresponds to an approximate 95% confidence interval of 18%–44%, indicating a clear performance gap. In addition, a subset of the results was independently verified by two authors, with complete agreement observed.

Response Characteristics:

In addition to accuracy, we noticed qualitative trends in the outputs of the LLM. The model often made transition tables that were well-formatted and had the right column headers. This suggests that it learnt the surface structure of automata notation. But the information in those tables was often inaccurate. In certain situations, the model produced explanatory text describing the purpose of the table (like "To convert NFA to DFA, we apply subset construction...") before the table. This text was clear but not necessary for correctness. The length of the responses ranged from 50 to 400 tokens, and larger responses did not lead to more accurate answers. These observations corroborate the perspective that the model is engaged in pattern completion rather than algorithmic execution.

Example Transition Table:

Table 11 shows a representative discrepancy. The LLM output included a fabricated state called 'z20' and wrong transitions. The ground truth used $\{A, C\}$ as a composite state appropriately. The LLM replaced it with the nonsensical "z20," which shows that the model lost track of the state set semantics and regressed to pattern-based token generation.

Discussion:

Summary of Findings:

The results show that generative AI models work more as systems for thinking and explaining than as reliable computing engines [22][24]. The model often produced fluent explanations in normal language, but it was not reliable enough for formal computation. The presence of "z20" hallucinations in Mealy machines constitutes conclusive evidence that the model is not executing an algorithm but rather forecasting text patterns. When the logical complexity was too high for the context window, the output turned into random noise. Related evaluations on unseen theory-of-computation tasks also caution that strong surface performance may not transfer to genuinely novel formal problems, supporting the need for controlled automata tests like the ones used here. [28].

Table 11. Ground truth vs. LLM output

Correct			**LLM (Incorrect)**		
State	0	1	State	0	1
$\{A\}$	$\{B\}$	$\{C\}$	A	B	C
$\{B\}$	$\{B\}$	$\{A, C\}$	B	B	z20
$\{A, C\}$	$\{B\}$	$\{A, C\}$	z20	B	A

Theoretical Implications:

The findings are consistent with the theoretical frameworks established by [3][16] about the expressive limitations of attention-based architectures. Standard transformers have trouble with tasks that need unbounded recursion or exact state tracking. Our empirical evidence corroborates the premise that next-token prediction is inadequate for the building of deterministic automata. The significant drop in accuracy as the problems got harder Figure 6 shows that the failure is consistent across increasing complexity levels.

Key Implications and Recommendations:

Table 12 summarizes the main implications of these findings and the actionable recommendations for different users, including educators, practitioners, and researchers. For **educators** utilizing LLMs in automata courses [5], LLM output should not be used for grading without verification; instead, such tools should be paired with deterministic methods and used mainly for conceptual explanation. For practitioners in verification or compiler-related settings, LLMs should be limited to natural-language interfaces, while all formal computation should remain under verified symbolic tools with strict validation pipelines. For researchers, the results support the development of hybrid systems in which LLM-generated outputs are systematically checked by deterministic solvers. [29] show that neuro-symbolic finite and push- down automata can outperform large multimodal LLMs on structured reasoning tasks, reinforcing hybrid pipelines where symbolic state machines validate generated outputs.

Comparison with Prior Work:

Table 13 compares our findings to previous research. Our research builds on previous studies by assessing a more comprehensive array of automata operations and offering an in-depth assessment of failure modes.

Implications for Computer Science Education:

As LLMs grow more popular in student workflows [5], teachers will have to find ways to use them in a constructive way in computer science classes. Students in automata theory classes can utilize LLMs to "check" their homework or generate examples. Our findings suggest that such utilization can be detrimental: erroneous yet seemingly credible products may perpetuate preconceptions. We suggest that teachers: (1) make it clear to students that LLM-generated automata might be wrong; (2) give students exercises that require them to check outputs with deterministic tools; and (3) consider using LLMs only for natural language explanation (for example, "explain subset construction") and leaving computation to verified tools. The pedagogical value of LLMs in formal domains continues to be an unresolved issue for additional investigation.

Broader Impact:

The results have consequences for the safety and dependability of AI. Using LLMs in safety-critical areas (such as compilers and verification tools) without thorough testing could lead to small flaws that are hard to find. If a compiler front-end does not reduce a DFA correctly, it could reject valid programs or accept invalid ones. Our findings indicate that hybrid systems, which integrate LLM natural language understanding with deterministic solutions, have a more resilient trajectory for advancement. In this kind of system, the LLM would figure out what the user wanted and make a candidate specification. Then, a deterministic tool would check and fix the result before it could be used by anybody else. When teachers use these techniques in formal language classes, they should be honest with students about the limits of LLMs. The "z20" hallucination and other such mistakes raise concerns about the interpretability of LLM outputs in formal settings. In natural language, mistakes are sometimes clear, like words that are not grammatically correct. However, automata mistakes can be hard to spot. For example, a bad transition could make the machine accept or reject a single string improperly, which could go undiscovered without rigorous testing. This shows how important it is to apply automated verification anytime LLM output

is used in formal settings.

Table 12. Recommendations for educators and practitioners

Audience	Do	Do Not
Educators	Verify all LLM output; use hybrid tools	Grade without verification; trust LLM as sole source
Practitioners	Use LLMs for NL interfaces only	Deploy LLM for formal computation
Researchers	Combine LLMs with deterministic solvers	Assume LLMs can replace algorithmic tools

Table 13. Comparison with prior studies

Study	Focus	Models	Key Finding
[1]	Lang. recognition	BERT, GPT-2	Degrades with complexity
[3]	Theoretical limits	Self-attention	Depth scales with input
[8]	Logical reasoning	GPT-4, Claude	Mixed on formal logic
This work	Automata ops.	Gemini 2.5	0% Mealy-Moore; 30% overall

Threats to Validity:

Internal validity: The evaluation criteria were implemented consistently, but subjective judgment may have been necessary in marginal circumstances (e.g., partially correct tables). We reduced this risk by having two authors check the work on a subset.

External validity: The results may not apply to other models (Claude, GPT-4) because they may work differently. The problem set, although varied, may not encompass all automata patterns observed in practice.

Construct validity: We presumed that transition table accuracy is a suitable criterion; certain applications may accept approximate solutions.

Conclusion validity: The sample size (50) is sufficient for preliminary findings, yet larger studies would enhance statistical power.

Limitations:

This work has the following limitations:

Single model scope: This work was confined to a singular AI model (Google Gemini 2.5 Flash), and the results might not apply to other architectures (like GPT- 4o or Claude 3.5 Sonnet).

Restricted language class: The study concentrated solely on Regular Languages (Type 3 in the Chomsky Hierarchy) and does not cover more complex language classes.

Deterministic tool constraints: The deterministic tool is mathematically correct, but it only works with a limited number of pre-programmed algorithms and a strict input syntax. This is different from LLMs, which can handle a wider range of input types.

Prompting not explored: We also did not look at how few-shot prompting or chain-of-thought reasoning would help performance.

Dataset size: The 50-problem dataset is good enough for a first look, but it could be bigger to get more reliable statistical results.

Conclusion:

This research shows that Gemini 2.5 Flash performs poorly on formal automata tasks, achieving 30% overall accuracy compared to the deterministic tool's 100%, with performance dropping to 0% on Mealy–Moore conversions and other tasks requiring recursive reasoning, highlighting clear limitations in symbolic computation. Traditional deterministic algorithms always give correct answers, while probabilistic language production typically does not meet the requirements for formal correctness. The model had a 0% success rate on hard Mealy-Moore conversions, which shows that it cannot be trusted for strict formal translation jobs without checking. Architectural variants that add explicit stack-like state between transformer

layers have been proposed to better model recursive structure and improve performance on Chomsky-hierarchy benchmarks, suggesting a plausible direction for reducing ϵ-closure and state-tracking failures [30]. The consistent drop in accuracy as problems get harder Figure 6 shows that next-token prediction and algorithmic execution do not work well together. Our main results can be summed up like this: (1) The overall accuracy of Gemini 2.5 Flash on automata operations was 30%, while the deterministic baseline was 100%. (2) The difficulty of the task was a strong predictor of failure: simple problems had a 92% accuracy rate, while complex problems had a 0–30% accuracy rate. (3) Hallucination was the most common type of error (40%), showing up as made-up states like "z20" in machines that only had three states. (4) Mealy-Moore conversions were impossible, with a 0% success rate across all 10 problems. (5) Practical implication: LLMs should not be used for formal computation without checking; hybrid systems that combine LLMs with deterministic solvers are a better way to go.

Future Work:

Future research should broaden the parameters of this analysis in various aspects:

Chomsky Hierarchy Expansion: Assess LLMs on Context-Free Grammars (Pushdown Automata) and Context-Sensitive languages to evaluate the limitations concerning infinite tape logic and recursion depth.

Cross-Model Benchmarking: Compare these results to those of other designs, such as GPT-4o, Claude 3.5 Sonnet, and DeepSeek-V3, to see if the faults are only in the tested model or if they are part of the Transformer architecture itself.

Hybrid Systems: Create and test hybrid pipelines that combine LLM natural language understanding with deterministic automata solvers to ensure end-to-end correctness.

Fine-Tuning and Prompting: Find out if task-specific fine-tuning or structured prompting (such as chain-of- thought or scratchpad) can help with automata tasks.

Integration of Formal Verification: Look into how to combine LLMs with model checkers and theorem provers to automatically check the correctness of generated automata.

Table 14 shows a summary of the suggested timeline for future development. Cross-model benchmarking and prompting tactics should be the main goals of short-term efforts (6–12 months). Work that lasts for a year or two can deal with the growth of the Chomsky Hierarchy and the design of hybrid systems. Long-term research may investigate the merging of formal verification with innovative architectures. A significant avenue is the creation of benchmark suites for formal language problems, similar to mathematical reasoning benchmarks (e.g., GSM8K, MATH), but specifically designed for automata operations. These benchmarks would allow for systematic comparisons between models over time. We also want the deterministic tool and problem set to be made public so that other people can do this work on their own and build on it.

Appendix: Sample Problem and Solution:

To show how the evaluation approach works, we give an example of an NFA-to-DFA problem. **Input (NFA):** States $\{A, B, C\}$, alphabet $\{0, 1\}$, start state A, final states $\{C\}$ Transitions: $A \xrightarrow{0} B$, $A \xrightarrow{1} C$, $B \xrightarrow{0} B$, $B \xrightarrow{1} A$, $C \xrightarrow{0} B$, $C \xrightarrow{1} C$.

Expected DFA (subset construction): The DFA states are subsets of $\{A, B, C\}$. Start: $\{A\}$. From $\{A\}$ on 0: $\{B\}$; on 1: $\{C\}$. From $\{B\}$ on 0: $\{B\}$; on 1: $\{A\}$. From $\{C\}$ on 0: $\{B\}$; on 1: $\{C\}$. Final states: any subset that has C, like $\{C\}$, $\{A, C\}$, $\{B, C\}$, or $\{A, B, C\}$.

Common LLM error: In our tests, the model sometimes produced $\{A\}$, $\{B\}$, and $\{C\}$ as DFA states without using subset construction, or it left out the composite state $\{A, C\}$ when both A and C were reachable. This shows that the subset lattice was not looked at in a methodical way.

Additional Example: ϵ-NFA Conversion:

Look at an ϵ-NFA with states {N, P, Q}, alphabet {a}, start state N, and final state Q. Transitions: $N \xrightarrow{\epsilon} P$, $P \xrightarrow{\epsilon} N$, $P \xrightarrow{a} Q$. The ϵ-closure of N is {N, P} (cyclic). The LLM in Example 09 did not compute this closure because it thought N and P were independent and did not realize that both could be reached from the start without using input. These kinds of mistakes are typical of models that use linear reasoning instead of fixed-point reasoning.

Summary of Findings by Problem Category:

Table 15 gives a short look at the results for each of the five problem categories. The pattern is always the same: accuracy goes down as the algorithmic complexity of the required operation goes up. subset construction (NFA-to-DFA) works when the state space is small. ϵ-closure and partition refinement do not work when fixed-point or multi-pass reasoning is needed, and Mealy-Moore conversion does not work at all.

Table 14. Proposed future work roadmap

Timeline	Focus	Deliverables
Short-term (6–12 mo.)	Cross-model benchmarking; prompting	Benchmark suite; best practices
Medium-term (1–2 yr.)	Chomsky expansion; hybrid systems	PDA evaluation: prototype tools
Long-term	Formal verification integration	Theorem prover plugins

Table 15. Detailed results by problem category

Category	Correct	Total	Acc.	Tool	Primary Error
NFA to DFA	6	10	60%	100%	Subset union, recursion
ϵ-NFA to NFA	3	10	30%	100%	Closure fragmentation
DFA Minimization	5	10	50%	100%	Shallow partition
Mealy to Moore	0	10	0%	100%	Hallucination
Moore to Mealy	1	10	10%	100%	Look-ahead
Total	**15**	**50**	**30%**	**100%**	

Acknowledgment:

The authors express gratitude to the Department of Computer Science at SZABIST for their institutional assistance. We thank the people who made Graphviz and JavaScript technologies that were used in the deterministic solver. We also want to thank the anonymous reviewers for their helpful comments on an earlier version of our work.

References:

[1] N. G. Satwik Bhattamishra, Kabir Ahuja, "On the Ability and Limitations of Transformers to Recognize Formal Languages," *EMNLP 2020 - 2020 Conf. Empir. Methods Nat. Lang. Process. Proc. Conf.*, 2020, [Online]. Available: https://arxiv.org/abs/2009.11264

[2] William Merrill, "Sequential Neural Networks as Automata," *arXiv:1906.01615*, 2019, [Online]. Available: https://arxiv.org/abs/1906.01615

[3] Michael Hahn, "Theoretical Limitations of Self-Attention in Neural Sequence Models," *Trans. Assoc. Comput. Linguist.*, 2019, [Online]. Available: https://arxiv.org/abs/1906.06755

[4] Lena Strobl, William Merrill, Gail Weiss, David Chiang, Dana Angluin, "What Formal Languages Can Transformers Express? A Survey," *Trans. Assoc. Comput. Linguist.*, 2024, [Online]. Available: https://arxiv.org/abs/2311.00208

[5] Emily Ross, Yuval Kansal, Jake Renzella, Alexandra Vassar, Andrew Taylor, "Supervised Fine-Tuning LLMs to Behave as Pedagogical Agents in Programming

Education," *arXiv:2502.20527*, 2025, [Online]. Available: https://arxiv.org/abs/2502.20527

[6] Yuhuai Wu, Albert Q. Jiang, Wenda Li, Markus N. Rabe, Charles Staats, Mateja Jamnik, Christian Szegedy, "Autoformalization with Large Language Models," *arXiv:2205.12615*, 2022, [Online]. Available: https://arxiv.org/abs/2205.12615

[7] Minghai Lu, Benjamin Delaware, "Proof Automation with Large Language Models," *Proc. - 2024 39th ACM/IEEE Int. Conf. Autom. Softw. Eng. ASE 2024*, 2024, [Online]. Available: https://dl.acm.org/doi/10.1145/3691620.3695521

[8] L. G. Jin Jiang, Jianing Wang, Yuchen Yan, Yang Liu, Jianhua Zhu, Mengdi Zhang, Xunliang Cai, "Do Large Language Models Excel in Complex Logical Reasoning with Formal Language?," *arXiv:2505.16998*, 2025, [Online]. Available: https://arxiv.org/abs/2505.16998

[9] Y. C. Bill Yuchen Lin, Ronan Le Bras, Kyle Richardson, Ashish Sabharwal, Radha Poovendran, Peter Clark, "ZebraLogic: On the Scaling Limits of LLMs for Logical Reasoning," *Proc. Mach. Learn. Res.*, 2025, [Online]. Available: https://arxiv.org/abs/2502.01100

[10] P. Hooimeijer and M. Veanes, "An Evaluation of Automata Algorithms for String Analysis," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 6538 LNCS, pp. 248–262, 2011, doi: 10.1007/978-3-642-18275-4_18.

[11] M. J.-D. Vikas Raunak, Arul Menezes, "The Curious Case of Hallucinations in Neural Machine Translation," *NAACL-HLT 2021 - 2021 Conf. North Am. Chapter Assoc. Comput. Linguist. Hum. Lang. Technol. Proc. Conf.*, 2021, [Online]. Available: https://aclanthology.org/2021.naacl-main.92/

[12] Grégoire Delétang, Anian Ruoss, Jordi Grau-Moya, Tim Genewein, Li Kevin Wenliang, Elliot Catt, Chris Cundy, Marcus Hutter, Shane Legg, Joel Veness, Pedro A. Ortega, "Neural Networks and the Chomsky Hierarchy," *arXiv:2207.02098*, 2023, [Online]. Available: https://arxiv.org/abs/2207.02098

[13] Yiding Hao, Dana Angluin, Robert Frank, "Formal Language Recognition by Hard Attention Transformers: Perspectives from Circuit Complexity," *arXiv:2204.06618*, 2022, [Online]. Available: https://arxiv.org/abs/2204.06618

[14] Debela Gemechu, Ramon Ruiz-Dolz, Henrike Beyer, Chris Reed, "Natural Language Reasoning in Large Language Models: Analysis and Evaluation," *ACL Anthol.*, 2025, [Online]. Available: https://aclanthology.org/2025.findings-acl.192/

[15] Jinxin Liu, Shulin Cao, Jiaxin Shi, Tingjian Zhang, Lunyiu Nie, Linmei Hu, Lei Hou, Juanzi Li, "How Proficient Are Large Language Models in Formal Languages? An In-Depth Insight for Knowledge Base Question Answering," *arXiv:2401.05777*, 2024, [Online]. Available: https://arxiv.org/abs/2401.05777

[16] E. Y. William Merrill, Gail Weiss, Yoav Goldberg, Roy Schwartz, Noah A. Smith, "A Formal Hierarchy of RNN Architectures," *Proc. Annu. Meet. Assoc. Comput. Linguist.*, 2020, [Online]. Available: https://aclanthology.org/2020.acl-main.43/

[17] Nadav Borenstein, Anej Svete, Robin Chan, Josef Valvoda, Franz Nowak, Isabelle Augenstein, Eleanor Chodroff, Ryan Cotterell, "What Languages are Easy to Language-Model? A Perspective from Learning Probabilistic Regular Languages," *arXiv:2406.04289*, 2025, [Online]. Available: https://arxiv.org/abs/2406.04289

[18] Josef Valvoda, Naomi Saphra, Jonathan Rawski, Adina Williams, Ryan Cotterell, "Benchmarking Compositionality with Formal Languages," *ACL Anthol.*, 2022, [Online]. Available: https://aclanthology.org/2022.coling-1.525/

[19] Ta-Chung Chi, Ting-Han Fan, Alexander Rudnicky, Peter Ramadge, "Transformer Working Memory Enables Regular Language Reasoning And Natural Language

Length Extrapolation," *ACL Anthol.*, 2023, [Online]. Available: https://aclanthology.org/2023.findings-emnlp.397/

[20] Michael Hahn, Mark Rofin, "Why are Sensitive Functions Hard for Transformers?," *arXiv:2402.09963*, 2024, [Online]. Available: https://arxiv.org/abs/2402.09963

[21] William Merrill, Ashish Sabharwal, "The Expressive Power of Transformers with Chain of Thought," *arXiv:2310.07923*, 2024, [Online]. Available: https://arxiv.org/abs/2310.07923

[22] Zhiyong Han, Fortunato Battaglia, "Beyond Text Generation: Assessing Large Language Models' Ability to Reason Logically and Follow Strict Rules," *AI*, vol. 6, no. 1, p. 12, 2025, [Online]. Available: https://www.mdpi.com/2673-2688/6/1/12

[23] Sebastian Farquhar, Jannik Kossen, Lorenz Kuhn & Yarin Gal, "Detecting hallucinations in large language models using semantic entropy," *Nature*, vol. 630, 2024, [Online]. Available: https://www.nature.com/articles/s41586-024-07421-0

[24] S. S. Debargha Ganguly, Vikash Singh, "Grammars of Formal Uncertainty: When to Trust LLMs in Automated Reasoning Tasks," *arXiv:2505.20047*, 2025, [Online]. Available: https://arxiv.org/abs/2505.20047

[25] N. T. Mohamed Amine Ferrag, "Reasoning Beyond Limits: Advances and Open Problems for LLMs," *ICT Express*, 2025, [Online]. Available: https://arxiv.org/abs/2503.22732

[26] G. K. Alexandra Butoi, "Training Neural Networks as Recognizers of Formal Languages," *13th Int. Conf. Learn. Represent. ICLR*, 2024, [Online]. Available: https://arxiv.org/abs/2411.07107

[27] A. W. L. Pascal Bergsträßer, Ryan Cotterell, "Transformers are Inherently Succinct," *arXiv:2510.19315*, 2025, [Online]. Available: https://arxiv.org/abs/2510.19315

[28] Shlok Shelat, Jay Raval, Souvik Roy, Manas Gaur, "Beyond Memorization: Testing LLM Reasoning on Unseen Theory of Computation Tasks," *arXiv:2601.13392*, 2026, [Online]. Available: https://arxiv.org/abs/2601.13392

[29] S. Sasaki, D. M. Lopez, and T. T. Johnson, "Neurosymbolic Finite and Pushdown Automata: Improved Multimodal Reasoning versus Vision Language Models (VLMs)," *NeuS*, 2025.

[30] Kechi Zhang, Ge Li, Jia Li, Huangzhao Zhang, Yihong Dong, Jia Li, Jingjing Xu, Zhi Jin, "Recursive Transformer: Boosting Reasoning Ability with State Stack," *Open Rev.*, 2026, [Online]. Available: https://openreview.net/forum?id=2bbDg587uh