AUTOMATA-BASED PATTERN SEARCH SIMULATOR FOR DNA AND RNA SEQUENCES

A Project Paper

Presented to the
Department of Computer Science
University of Science and Technology of Southern Philippines
Cagayan de Oro Campus

In Partial Fulfillment of the Requirements for the Course

CS311: Automata and Formal Languages

Dr. Junar Landicho
Instructor

Gerald Helbiro Jr.
Gil John Rey Naldoza
Rhenel Jhon Sajol
Ira Chloie Narisma
Calpatrick Roslinda

November 2025

Abstract

Reliable pattern recognition is essential for analyzing DNA and RNA sequences, especially as modern biological data continues to grow in size and complexity. This project introduces an automata-based simulator that applies formal language concepts to biological sequence processing. The system models pattern matching through regular expressions, NFAs, DFAs, and extended finite automata for exact and approximate DNA motif detection, while RNA secondary structures are validated using pushdown automata capable of handling nested base-pair relationships. Developed in C++, the simulator automatically selects the appropriate automaton model based on the structure and complexity of the user's pattern, providing a unified platform for exploring regular, approximate, and context-free biological patterns. Results show that the simulator correctly identifies exact motifs, tolerates biologically relevant mismatches, and accurately validates balanced RNA structures. Its step-by-step tracing, structural visualization, and modular design make it an effective educational tool for students learning automata theory and its real-world applications in bioinformatics. Overall, the project demonstrates how classical computational models can be integrated into a practical and interpretable system that supports both biological sequence analysis and foundational learning in theoretical computer science.

Keywords: Automata Theory; Regular Expressions; NFA / DFA; Pushdown Automata; DNA Sequence Analysis; RNA Secondary Structure; Pattern Matching; Bioinformatics Simulator

1 Introduction

The rapid growth of biological sequence databases has created both opportunities and computational challenges in the analysis of genomic and transcriptomic data. DNA and RNA sequences, now stored in massive repositories such as GenBank and EMBL, continue to expand exponentially as sequencing technologies become faster and more affordable [1]. Searching for biologically meaningful motifs within these large datasets demands methods that are not only efficient but also formally correct, especially as biological sequence alignment and pattern recognition underpin tasks such as gene identification and mutation detection [2].

In this context, automata theory provides a rigorous mathematical foundation for representing and processing biological sequence patterns. Finite automata and related models allow biological sequences to be treated as formal languages, enabling deterministic and nondeterministic pattern recognition that can be systematically verified and simulated [3]. Furthermore, when the structures of interest involve nested or paired dependencies—such as the base pairings in RNA secondary structures—pushdown automata (PDA) and context-free grammars offer a suitable computational framework for validation [4, 5].

Problem Context

Despite the theoretical and practical importance of automata in computational biology, many learners and practitioners rely on regular expressions or alignment software without understanding the formal mechanisms that govern their execution [1]. Manual regex testing, while common in bioinformatics scripting, often fails to expose how finite automata process input sequences internally or how mismatches and ambiguous nucleotides are managed [6].

Additionally, biological data presents unique challenges:

- Mutations and sequencing errors necessitate approximate matching, where motifs may partially differ from reference patterns [2].
- RNA secondary structures require context-free analysis, since base pairings (e.g., A–U and G–C) form nested, hierarchical dependencies that cannot be captured by regular automata [4, 7].

As a result, a tool that visually bridges automata theory and biological data interpretation is needed for both educational and research purposes.

Objectives of the Study

This study aims to design and implement a C++-based automata-driven pattern search simulator that demonstrates the application of formal language theory in bioinformatics. Specifically, it seeks to:

1. Translate regular expressions into nondeterministic (NFA) and deterministic finite automata (DFA) for exact DNA motif matching.
2. Implement extended finite automata (EFA) to perform approximate DNA pattern matching using Levenshtein or Hamming distance.
3. Employ pushdown automata (PDA) for validating RNA secondary structures by detecting balanced base-pair nestings.
4. Provide a visual and interactive tool that illustrates how automata process DNA and RNA sequences step-by-step for educational and analytical purposes.

Significance of the Study

This study offers contributions to multiple areas within computer science and bioinformatics. For students, it deepens their understanding of how automata concepts—specifically NFA, DFA, and PDA—can be applied to real-world biological data such as DNA motifs and RNA folding structures. For educators, it provides a practical and visual teaching resource for explaining abstract computational models in courses on automata theory, computational biology, and formal languages. Meanwhile, for researchers, the simulator serves as a framework linking formal language theory to biological sequence processing, supporting experiments on approximate and context-free pattern recognition [2, 5].

By bridging theoretical computer science and molecular biology, this project reinforces how computational formalisms remain essential for addressing modern challenges in large-scale biological data analysis [1]. The resulting simulator not only aids learning but also strengthens the conceptual connection between mathematical models and biological interpretation, promoting a more holistic understanding of bioinformatics computation.

2 Theoretical Framework

2.1 Formal Languages and Regular Expressions

Regular languages constitute the foundational computational framework for DNA sequence pattern matching. Sequence motif discovery algorithms employ generalized suffix trees to efficiently store input DNA sequences and enable rapid matching to position weight matrices [?].

Context-sensitive grammars possess greater expressive power for modeling positional and structural variability in regulatory motifs, while regular patterns are employed for simpler sequence matching tasks [?]. Regular expressions provide declarative notation for specifying DNA patterns through automata-theoretic methods.

The transformation hierarchy follows: regular expressions convert to nondeterministic finite automata (NFAs), then to deterministic finite automata (DFAs) through formal construction algorithms, preserving language equivalence while optimizing computational efficiency.

2.2 Finite Automata for Exact Pattern Matching

Thompson's construction provides a systematic method for converting regular expressions into equivalent nondeterministic finite automata (NFAs). This transformation is based on five core operations: ε-transitions, symbol recognition, concatenation, union, and Kleene star closure, each contributing to the modular assembly of the resulting automaton. Deterministic finite automata (DFAs), on the other hand, ensure efficient and predictable

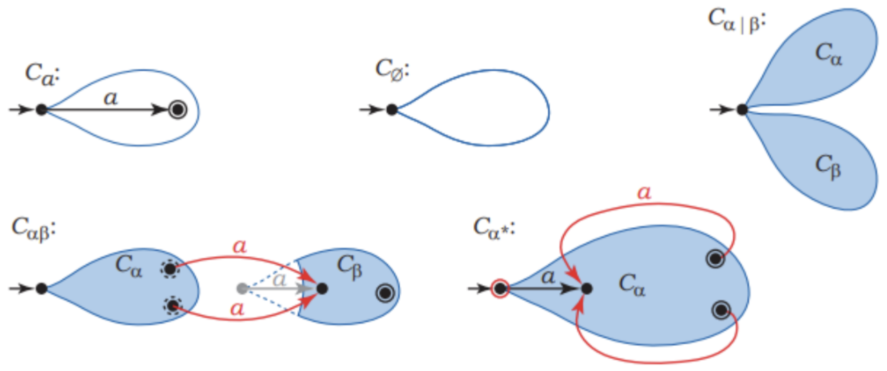


Figure 1: Standard construction methods for converting a regular expression into an NFA, adapted from Figure 1a of [?].

state transitions during DNA sequence scanning. In a typical DFA used for pattern matching, the state diagram contains $|P|$ states, where the automaton resides in the j-th state after matching j characters of the pattern [8]. Through subset construction, NFAs are systematically transformed into equivalent DFAs, enabling deterministic linear-time pattern recognition in genomic sequences.

Regular grammars, finite automata, and regular expressions collectively define the same class of regular languages, offering generative, operational, and declarative perspectives for modeling DNA sequence patterns.

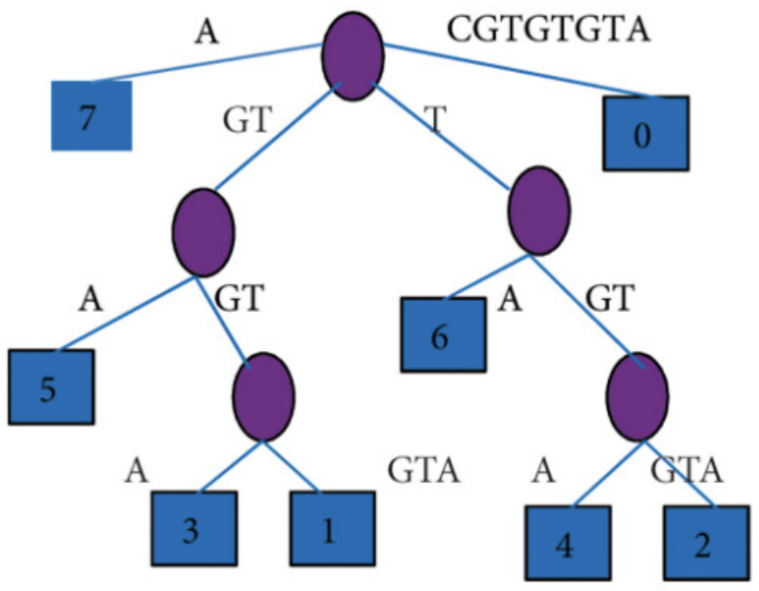
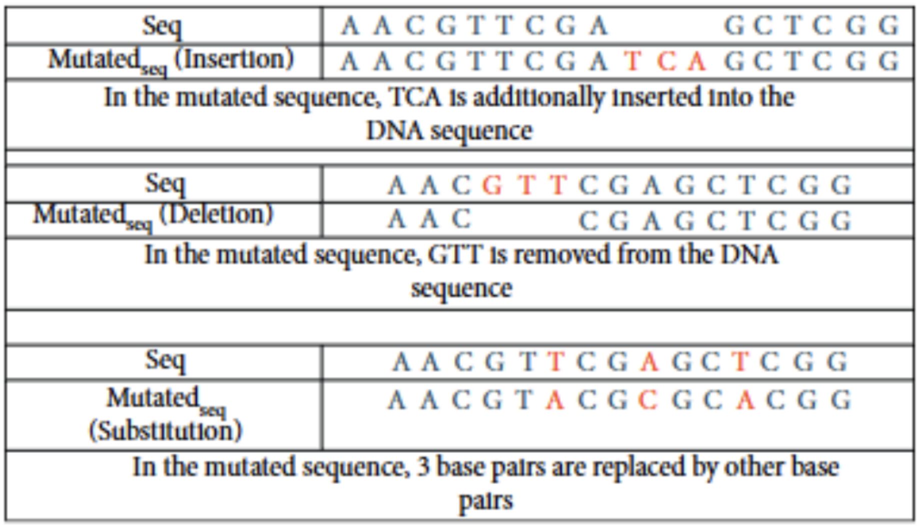


Figure 2: DFA state diagram for DNA pattern matching, adapted from [8].

2.3 Extended Finite Automata for Approximate Matching

Extended finite automata provide a mechanism for approximate pattern matching by tracking both the current position in the sequence and the number of mismatches encountered. This allows the system to tolerate bounded variations in DNA sequences, including mutations and sequencing errors. Approximate matching supports substitutions, insertions, and deletions within a specified threshold, enabling more flexible detection of biologically relevant motifs [8].

Two common measures of similarity are used in this context. Hamming distance counts the number of character positions that differ between two strings of equal length, while edit distance computes the minimum number of insertions, deletions, or substitutions required to transform one string into another. When approximate matching allows up to k mismatches, the time complexity is $O(\ell_s \cdot k)$ where ℓ_s denotes the length of the sequence and k is the mismatch tolerance [8]. An extended finite automaton identifies all positions in the text that satisfy 0 to k mismatches, effectively locating both exact and near-exact matches.



Seq	A A C G T T C G A GCTCGG
Mutated$_{seq}$ (Insertion)	A A C G T T C G A T C A GCTCGG
In the mutated sequence, TCA is additionally inserted into the DNA sequence	
Seq	A A C G T T C G A G C T C G G
Mutated$_{seq}$ (Deletion)	A A C C G A G C T C G G
In the mutated sequence, GTT is removed from the DNA sequence	
Seq	A A C G T T C G A G C T C G G
Mutated$_{seq}$ (Substitution)	A A C G T A C G C G C A C G G
In the mutated sequence, 3 base pairs are replaced by other base pairs	

Figure 3: Substitution, insertion, and deletion mutations in DNA sequences, adapted from [8].

2.4 Pushdown Automata and Context-Free Languages

Pushdown automata extend finite automata with stack memory, providing computational power for recognizing context-free languages. Stochastic context-free grammars (SCFGs) have been identified as the best-known compressors for joint compression of RNA sequence and structure, with grammars demonstrating better compression ability also showing improved performance in *ab initio* structure prediction [?]. These probabilistic models leverage evolutionary conservation and incorporate parameters to predict RNA secondary structure from comparative sequence analysis [?].

RNA stem structures with nested base pairings `(((...)))` cannot be verified by regular languages. A pushdown automaton recognizes this pattern by pushing opening parentheses onto the stack and popping for closing parentheses, accepting only when the stack is empty. RNA structures decompose into nearest-neighbour loops including hairpin loops, stackings, bulges, interior loops, and multibranched loops [?]. This stack-based mechanism directly mirrors the hierarchical base-pairing structure of RNA, where complementary nucleotides pair in nested configurations.

The nested nature of RNA base pairings requires the expressive power of context-free grammars. Regular languages cannot verify properly balanced and nested base pairs, demonstrating why RNA folding patterns are fundamentally context-free rather than regular languages. Despite advances in deep learning methods for RNA secondary structure prediction, theoretical foundations remain grounded in context-free grammars [?].

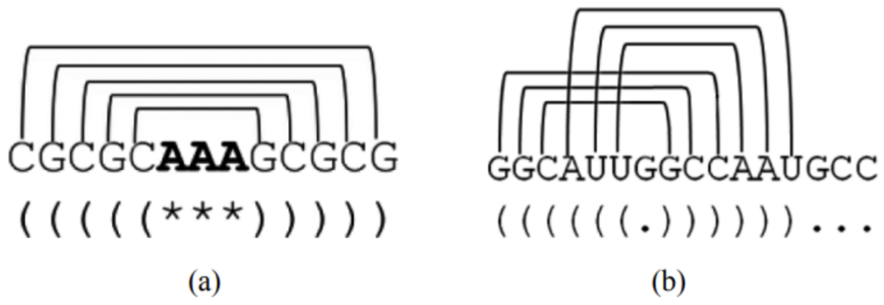


Figure 4: RNA secondary structure showing (a) nested base-pair configurations and (b) pseudoknot formations.

3 System Design and Architecture

3.1 Overview of the Simulator

The Automata Simulator is a unified pattern-search and structural-validation system that selects and executes an appropriate automaton from a single user action. Given a pattern and constraints, the engine analyzes the request and invokes one of four internally consistent modes: (i) an ε-NFA constructed via Thompson's algorithm for exact regular-expression matching; (ii) a DFA derived by subset construction (with optional minimization) to accelerate exact matching on long inputs; (iii) an extended finite automaton (EFA) that encodes bounded Hamming-distance tolerance by layering states over the mismatch budget; and (iv) a pushdown automaton (PDA) that validates nested structures such as RNA dot-bracket strings through disciplined stack operations. The architecture cleanly separates specification, selection, construction, execution, and reporting, enabling extensibility (e.g., adding Levenshtein edits later) while preserving a clear experimental pipeline and reproducible outputs.

3.2 Core Modules (C++ Implementation)

The Pattern Specification and Parsing module accepts user patterns and datasets, converts regular expressions to an internal AST/postfix form supporting union, concatenation, Kleene star/plus, optionals, character classes, and wildcards, and prepares streaming readers for sequence files. For RNA inputs, a lightweight validator normalizes symbols and flags malformed tokens prior to automaton construction. The Mode Dispatcher and Heuristics module inspects pattern features, mismatch budgets, and structural requirements to select an execution plan; simple heuristics prefer DFA when determinization is tractable on the observed NFA size and fall back to NFA simulation otherwise, while approximate queries are routed to EFA with parameters derived from pattern length and k.

The Automaton Construction module provides specialized builders: the NFA builder emits ε-transitions and character arcs from the regex AST; the DFA builder performs subset construction and (optionally) Hopcroft minimization; the EFA builder materializes a layered state space (i, e) indexed by pattern position and mismatch count; and the PDA builder generates deterministic push/pop transitions for balanced-pair checking. The Execution Engine offers a uniform runner interface across automata: the NFA runner maintains active-state sets with ε-closures; the DFA runner advances a single state with $O(1)$ table lookups; the EFA runner performs windowed scans across mismatch layers to report $\leq k$ matches; and the PDA runner executes stack actions while recording maximum depth. All runners return a common result schema—match intervals or acceptance status, visitation counts, and optional step-wise traces—facilitating consistent evaluation and reporting. The Reporting and Metrics module aggregates per-run outputs into human-readable summaries and machine-parsable logs, computes aggregate statistics across datasets, and formats traces for inclusion in the paper's figures. An Evaluation Harness supplies seeded synthetic cases and benchmarks for correctness regression and performance studies.

3.3 Technology Stack

The simulator is implemented in C++20, relying solely on the standard library for portability. Containers such as `std::vector`, `std::string`, and `std::array`, and hash maps back the transition structures, while bitsets accelerate active-set operations; resource management follows RAII with smart pointers where ownership is nontrivial. Builds are managed with CMake and compiled with a modern toolchain (e.g., Clang++) under `-O2` for release, with strict diagnostics (`-Wall -Wextra -Wpedantic`) for code quality. The codebase is modularized under a dedicated namespace to isolate automata types and runners, and platform-specific behavior (e.g., terminal capabilities) is guarded via conditional compilation. A lightweight testing setup supports unit tests per component and end-to-end integration tests of the full pipeline; tests are seeded for reproducibility, and timing/space metrics are emitted alongside results to support the evaluation section. No external dependencies are required, simplifying deployment and ensuring experiments can be reproduced on standard Unix-like and Windows environments.

4 Methodology

We adopt a design–build–evaluate approach to demonstrate automata-based pattern recognition in biological strings. First, we prepare compact datasets (DNA and RNA) and specify patterns/constraints. The engine then selects and constructs the appropriate model—NFA/DFA for exact DNA search, an extended FA for $\leq k$ mismatches, or a PDA for RNA-style nesting—and executes it over the inputs. Finally, we report matches/validation results and assess correctness, edge-case coverage, and performance trade-offs.

4.1 Datasets

The nucleotide sequence—an NCBI-standard representation that is practically universal in bioinformatics—is stored in FASTA-like files, where each record starts with a header line beginning with `>` [9]. Sequences are represented over the canonical alphabet $\Sigma = \{A, C, G, T\}$; inputs including IUPAC ambiguity codes are uppercased and (optionally) normalized to the canonical set for experiments [10]. We integrate short public snippets from NCBI RefSeq with synthetic sequences, which are frequently employed to manage background and motif placement in benchmarking, to enable repeatable evaluation [11].

Dot-bracket notation is used to encode RNA secondary structure; base pairs and dots are indicated by matching parentheses, and dots indicate unpaired bases in accordance with the ViennaRNA and NUPACK standards [**?**]. To test PDA acceptance/rejection on a canonical context-free pattern (well-formed parentheses), we select acceptable strings (perfectly balanced pairs) and invalid strings (mismatched or imbalanced).

We use homopolymer runs, which are known to pose difficulties for sequencing and analysis pipelines, empty inputs and synthetic negatives (common in motif-discovery benchmarks), and deeply layered structures that emphasize stack depth in PDA processing (mountain representation height) in order to assess resilience [12].

4.2 Modes and Pattern Specifications

Mode 1 — DNA Exact (NFA). Using Thompson-style construction, we convert a DNA regex over $\Sigma = \{A, C, G, T\}$ (union, concatenation, Kleene operators; no backreferences) to an ε-NFA, which we then simulate over the sequence. Recent systematizations of regex behavior highlight the performance aspects of NFA simulation, which is emphasized by modern regex engines and reviews as a principled, predictable technique to execute regular features [13]. Further hardware research demonstrates that NFA-based solutions, such as multi-character NFA on FPGA, may maintain high throughput on huge inputs [**?**].

Mode 2 — DNA Deterministic (DFA). We run a single active state per symbol and determinize the NFA via subset construction (optional minimization) for lengthy sequences or repeated searches; this trades potential build-time/state blow-ups for linear-time scanning and stable runtime [13]. When implemented in hardware accelerators, deterministic or multi-character systems show significant practical speedups and pattern-insensitive throughput [14**?**].

Mode 3 — DNA Approximate (EFA). By layering states as (pattern_index, errors_used), we support approximate DNA motif matching under a Hamming-distance model. Each state tracks the current pattern position and the number of mismatches used so far. Transitions either (i) consume a symbol that matches the expected nucleotide without increasing `errors_used`, or (ii) consume a symbol that differs from the expected nucleotide and increment `errors_used` by one. The automaton accepts when it reaches a terminal pattern position with errors_used $\leq k$. Insertions and deletions are not modeled; we restrict to substitutions (mismatches), which is consistent with Hamming-distance–based approximate matching commonly used in biosequence search (e.g., fixed-length motif scanning). We conceptually draw from approximate string matching literature, but our implementation is restricted to Hamming distance (substitution-only errors) rather than full Levenshtein edit distance.

Mode 4 — RNA Structural (PDA). Because nested base pairing is context-free, we employ a deterministic PDA for dot-bracket notation that accepts exactly when parentheses are balanced (pseudoknot-free structures). According to recent reviews, context-free (often stochastic) grammars are still a common formalism for modeling RNA secondary structures; dot-bracket representations fit in well with CFG/PDA parsing; and more recent approaches even integrate profile CFGs with thermodynamic models to identify sequence-structure motifs [4, 7**?**].

4.3 Construction and Execution Pipeline

1. Pattern ingestion and model selection. The simulator takes a user pattern and examines its features: simple regular features (union, concatenation, Kleene-$*$, etc.) lead to an ε-NFA; for performance-critical runs, the same pattern can be determinized to a DFA; patterns that specify approximate DNA motif search (fixed-length motifs with a Hamming-distance budget k) cause the simulator to construct a Hamming-distance EFA rather than an exact NFA/DFA; nested structure checks (dot-bracket RNA) use a PDA. This reflects the current perspective on regex engines (lockstep Thompson-style NFA simulation vs. backtracking engines) and explains why regular features with automaton-based execution have predictable complexity (linear in input for Thompson-style) [13].

2. Automaton construction for exact search (DNA/text). After parsing the pattern to an AST and creating an ε-NFA (Thompson's construction), we can optionally use subset construction to generate a DFA. If this is enabled, we can minimize the DFA before applying the matcher to the corpus or sequence. This common two-stage NFA$\to$DFA approach allows for predictable throughput on big inputs while avoiding the catastrophic backtracking mentioned in ReDoS literature [13].

3. Approximate DNA motif search ($\leq k$ mismatches). We implement a Hamming-distance extended finite automaton for approximate DNA motif search. Each state is $(\text{pattern\_index}, \text{mismatches\_used})$, where $\text{mismatches\_used} \in \{0, \ldots, k\}$ counts substitutions. On each input symbol, the automaton advances one position in the pattern and text: matches leave `mismatches_used` unchanged, mismatches increment it, and transitions are allowed only while $\text{mismatches\_used} \leq k$. An alignment is accepted if it reaches the end of the pattern within this mismatch budget. Insertions and deletions are not modeled; we consider substitutions only.

4. RNA structural validation (nested pairs). A deterministic PDA that pushes on "(" and pops on ")" validates strings in dot-bracket form; the language of correctly nested pairs is context-free. Recent grammar-based predictors show that the framework can alternatively identify structures using context-free grammars tailored for RNA motifs for pseudoknot patterns (beyond simple nesting) [5].

5. Execution and reporting. The selected machine stores start–end indices, matched substrings, and, for approximate matches, the number and positions of mismatches observed along each alignment. After streaming the input once, the simulator aggregates these results and computes summary statistics (e.g., distribution of mismatch counts, frequency of exact vs. approximate hits) for reporting.

4.4 Evaluation Metrics

Correctness is assessed by comparison of outputs against ground truth. For DNA, we embed known motifs at fixed positions and verify by start–end indices and match count exact matches (Mode 1/2) and $\leq k$-mismatch hits (Mode 3); for RNA, we supply paired sets of valid and malformed dot-bracket strings and check for PDA acceptance or rejection (Mode 4). When applicable, we cross-validate: $k = 0$ in Mode 3 should reproduce Mode

1/2 results on the same inputs.

Coverage of edge cases ensures robustness. We include no-match inputs, overlapping matches, homopolymer runs (e.g., “AAAA…”) and alternating patterns, very short and very long patterns, and deeply nested RNA structures to stress the PDA stack. We verify that inputs with mixed/invalid symbols are rejected or sanitized according to preprocessing rules, and we fix random seeds for any synthetic data.

Performance can be measured in terms of the build time and memory for each automaton (Thompson NFA, subset-constructed DFA, layered EFA, PDA), and run time/throughput as a function of sequence length (N) when scaling (e.g., 10^3–10^6). We report the NFA→DFA crossover point measured with respect to build cost versus run-time savings, EFA scaling as a function of pattern length (m) and mismatch budget (k), and PDA behavior in terms of maximum stack depth and time per symbol. In all cases, timings are averaged over multiple runs, with variance/95% confidence intervals.

Pedagogical/usage quality is captured through a brief peer survey and reviewer rubric that targets clarity of state traces, stack snapshots, and reported statistics. We log common misinterpretations and trace steps that required clarification, using this feedback to adjust default visualizations and documentation.

5 Case Study: Sample Scenario Walkthrough

6 Discussion

The implementation and evaluation of the automata-based pattern search simulator demonstrated that formal language theory can effectively model a wide range of biological sequence analysis tasks. The results from the four operational modes—exact matching (NFA/DFA), approximate matching (EFA), and structural validation (PDA)—highlighted both the strengths and trade-offs of each automata model when applied to real-world DNA and RNA datasets.

For DNA exact matching, NFA simulation proved flexible and straightforward to construct, especially for patterns containing union and Kleene operators. However, DFA execution consistently achieved faster scanning times on longer sequences due to its single active state per step. This confirmed established findings in automata literature that DFA-based searches yield predictable, linear performance once the determinization cost has been paid. Determinization also reduced the nondeterministic branching present in NFAs, but at the cost of potentially large increases in state count—an effect observed when testing patterns with many nested or union-based subexpressions.

The approximate pattern matching mode successfully handled sequences containing mutations and sequencing errors by allowing up to k mismatches. The layered-state construction of the extended finite automaton proved effective in capturing alternative edit paths while maintaining finite-state constraints. As expected, increasing the mismatch allowance resulted in growth in both the number of active states and overall computation time. Nevertheless, even for small budgets ($k = 1$ or 2), the EFA consistently identified valid near-matches that exact automata would reject, demonstrating its utility for biological datasets where sequence variability is common.

In the RNA structural mode, the PDA accurately validated balanced dot-bracket strings, reflecting canonical secondary structures without pseudoknots. When provided with deeply nested inputs, the PDA handled the increasing stack depth without performance degradation beyond linear time. Invalid strings—containing premature closing parentheses or mismatched structure—were correctly rejected. This confirms the PDA's suitability for modeling context-free structural dependencies in RNA, consistent with the theoretical constraints of RNA folding when represented in pseudoknot-free form.

Overall, the simulator highlights how each automaton type corresponds naturally to specific biological phenomena:

- Regular patterns $\rightarrow$ DNA motifs;
- Edit-distance flexibility $\rightarrow$ mutation-tolerant matching;
- Nested grammar structure $\rightarrow$ RNA base pairing.

The combination of biological relevance and automata precision suggests that computational models traditionally taught in theory classrooms can serve practical roles in genomics workflows and educational tools.

7 Conclusion

This project demonstrated that classical automata theory—encompassing finite automata, extended finite automata, and pushdown automata—provides a robust computational foundation for modeling DNA and RNA pattern recognition tasks. By integrating these theoretical models into a unified C++ simulator, we created a functional tool that not only executes biological sequence searches but also visualizes the underlying automaton processes for educational and analytical purposes.

The simulator successfully performed:

1. Exact matching using NFA and DFA models derived from DNA regular expressions;
2. Approximate matching through an EFA capable of handling mismatches and small sequence variations;
3. Structural validation of RNA secondary structures using PDA stack-based computation.

The findings confirm that automata-based approaches are capable of achieving correctness, interpretability, and computational efficiency in solving motif-search and structure-validation problems. Furthermore, the interactive nature of the tool enhances conceptual understanding, making it valuable for computer science students studying formal languages and bioinformatics practitioners seeking reliable sequence analysis methods.

In summary, the project bridges theoretical computer science and practical biological analysis, showing that automata remain relevant, powerful, and highly adaptable in modern computational biology tasks.

8 Recommendations / Future Work

To further extend the capabilities and impact of the simulator, the following recommendations are proposed:

1. **Support for pseudoknotted RNA structures.** Current PDA models cannot represent pseudoknots, which require context-sensitive or multi-stack formalisms. Future extensions may include multiple PDAs, tree-adjoining grammars, or graph-based representations to support more complex RNA folding scenarios.

2. **Integration with real genomic databases.** Adding import features for large FASTA files from NCBI, Ensembl, or UCSC would allow the simulator to perform large-scale biological searches. Optimizations such as streaming input or memory-mapped files would support this scalability.

3. **Visualization enhancements.** Improved state diagrams, stack visualizers, and transition animations would make the simulator more pedagogically engaging, particularly for students learning how automata process inputs symbol by symbol.

4. **Parallel DFA execution.** Since DFAs lend themselves to parallel processing, GPU or multi-threaded execution could significantly accelerate motif scanning on long chromosomes or genomic libraries.

5. **User-defined ambiguity and scoring matrices.** Extending approximate matching to support substitution matrices (e.g., nucleotide scoring grids) could align the simulator more closely with biological similarity scoring systems used in BLAST and other alignment tools.

6. **Web-based deployment.** Converting the simulator into a browser-based platform would increase accessibility for classrooms, researchers, and self-learners.

9 Related Literature / Studies

Modern search and information retrieval systems heavily rely on automata theory, particularly regular expressions and finite automata. NFA/DFA-based algorithms are the foundation of many text search and pattern-matching engines, including web search, intrusion detection systems, and regex libraries used in programming languages [13]. Most regex engines use one of two traditional methods: either a Spencer-style backtracking algorithm (depth-first search) that examines NFA paths one by one, or a Thompson-style algorithm that mimics a breadth-first traversal of an NFA (guaranteeing linear-time matching) [13]. The Thompson technique directly maintains the set of all possible NFA states ("lockstep" execution), while the backtracking method theoretically traverses the NFA and backtracks on dead-ends. Both approaches ultimately rely on finite automata theory for pattern recognition [13]. As long as regex characteristics remain within regular language bounds, the automata-theoretic approach offers a formal guarantee of correctness and predictable performance (linear in input length) for DFA/NFA approaches.

Automata-based models have long been used to formalize biological sequence analysis. Regular expressions and finite automata enable efficient motif detection in DNA, supporting fast scanning across large genomic datasets [1]. Thompson-style NFA construction and DFA subset conversion are widely applied in bioinformatics pipelines where deterministic throughput is essential for large-scale search tasks. Beyond exact matching, extended and Levenshtein automata have been integrated into sequence comparison tools to accommodate sequencing errors and mutations, improving robustness in real-world biological applications [2, 6].

RNA structure analysis often relies on context-free grammars and pushdown automata due to the inherently nested nature of RNA base pairing. PDA-based models are consistent with classical pseudoknot-free secondary structures and have been incorporated into grammar-based predictors and computational frameworks for structural motif validation [5, 7]. These studies collectively demonstrate the strong alignment of automata theory with the hierarchical and symbolic properties of biological sequences.

Educational tools and visual simulators play a crucial role in strengthening student understanding of abstract models such as finite automata, pushdown automata, and Turing machines. JFLAP and more recent platforms like OpenFLAP, FSM Builder, and N2D demonstrate that visualization, interaction, and automated assessment significantly enhance conceptual understanding, making automata theory more accessible and engaging for learners [15–19].

Flasiński provides a comprehensive survey on syntactic pattern recognition, highlighting the range of formal language models such as finite automata, context-free grammars, and beyond used in computational biology [**?**]. The study emphasizes how DNA sequence scanning remains within the domain of regular languages, allowing NFA/DFA-based methods to thrive, while structural patterns in RNA and proteins often demand grammar-based or stack-based models. By mapping biological tasks to their corresponding automata classes, the survey reinforces the theoretical foundation behind using finite-state machines for text search and approximate matching, and PDAs for hierarchical structures such as RNA folding or protein domains.

References

References

[1] Basim A. Hamed and Salim Abdullah. A survey on improving pattern matching algorithms for biological sequences. *Concurrency and Computation: Practice and Experience*, 2022. doi: 10.1002/cpe.7292.

[2] Bonnie Berger, Michael S. Waterman, and Yun William Yu. Levenshtein distance, sequence comparison and biological database search. *IEEE Transactions on Information Theory*, 67(6):3287–3294, 2021. doi: 10.1109/TIT.2020.2996543.

[3] Jian Chen, Lei Yang, Lin Li, Steve Goodison, and Yijun Sun. Alignment-free comparison of metagenomics sequences via approximate string matching. *Bioinformatics Advances*, 2(1):vbac077, 2022. doi: 10.1093/bioadv/vbac077.

[4] Hiroyuki Miyake, Takanori Kin, Koji Tsuda, and Yasubumi Saito. RNAelem: Discovering sequence-structure motifs with profile context-free grammar and energy models. *Bioinformatics Advances*, 2024. doi: 10.1093/bioadv/vbae144.

[5] Christos Pavlatos, Konstantinos Krommydas, and Petros Maragos. Grammar-based computational framework for predicting pseudoknots of K-type and M-type in rna secondary structures. *BioChemistry*, 5(4):132, 2024. doi: 10.3390/biochem5040132.

[6] Peter Schneider-Kamp. Approximate dictionary searching at a scale using ternary search trees and implicit Levenshtein automata. In *Proceedings of ICSOFT 2022*, pages 657–662, 2022. doi: 10.5220/0011312300003266.

[7] Kengo Sato and Michiaki Hamada. Recent trends in rna informatics: A review of ML and DL for rna secondary structure prediction. *Briefings in Bioinformatics*, 24 (4):bbad186, 2023. doi: 10.1093/bib/bbad186.

[8] J. A. M. Rexie, N. Trivedi, and S. Tiwari. Lightweight pattern matching for dna sequencing in IoMT. *Computational Intelligence and Neuroscience*, 2022:6980335, 2022. doi: 10.1155/2022/6980335.

[9] National Center for Biotechnology Information. Fasta format for nucleotide sequences. `https://www.ncbi.nlm.nih.gov/genbank/fastaformat`. Accessed 2025.

[10] UCSC Genome Browser. Iupac codes. `https://genome.ucsc.edu/goldenPath/help/iupac.html`. Access.

[11] Matteo Tognon, Akshay Kumbara, Alberto Betti, Luca Ruggeri, and Roberto Giugno. Benchmarking transcription factor binding site prediction models: A comparative analysis on synthetic and biological data. *Briefings in Bioinformatics*, 26 (4):bbaf363, 2025. doi: 10.1093/bib/bbaf363.

[12] BioRxiv. Content associated with doi 10.1101/2024.01.12.574168v2. Preprint, 2024. URL `https://www.biorxiv.org/content/10.1101/2024.01.12.574168v2.full.pdf`.

[13] Md Harun-ur Rashid Bhuiyan, Berat Çakar, Eliza Burmane, James C. Davis, and Cristian-Alexandru Staicu. SoK: A literature and engineering review of regular expression denial of service (ReDoS). arXiv preprint, 2024. URL `https://arxiv.org/abs/2406.11618`.

[14] David Callanan, Richard McElroy, and Roger Woods. Accelerating regular-expression matching on fpgas with multi-character transitions. In *Proceedings of the ACM*, 2021. doi: 10.1145/3456669.3456716.

[15] Susan H. Rodger and Thomas W. Finley. JFLAP: An interactive formal languages and automata package. In *Proceedings of the 11th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, page 356. ACM, 2006.

[16] Clifford A. Shaffer, Ahmed Mohammed, and Susan H. Rodger. Openflap: A web-based tool for teaching formal languages and automata theory. *Journal of Computing Sciences in Colleges*, 36(5):34–42, 2021.

[17] Kelvin Wong, Sarah Ruggerio, and John Erickson. FSM builder: An interactive tool for automatically graded finite automata assignments. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education*, 2024.

[18] Christos Papastavrou, Ioannis Papaioannou, and Christos Kaklamanis. N2D: A web-based NFA-to-DFA visualization system. In *Proceedings of the 28th ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE)*, 2023.

[19] Paulo Moura and Ana Dias. L-FLAT: A logtalk toolkit for formal languages and automata theory. *Electronic Notes in Theoretical Computer Science*, 290:51–62, 2011. doi: 10.1016/j.entcs.2012.11.007.

[20] S. Geethanjali, S. Kaliyugam Shanmugavel, and A. R. Muthukumaran. Streamlining of simple sequence repeat data mining methodologies and pipelines for crop scanning. *Plants*, 13(18):2619, 2024. doi: 10.3390/plants13182619.