

Comparative Analysis of Dynamic Programming, Divide-and-Conquer, and Earley Parsing Approaches for JSON Grammar Recognition

Maulana Ghazzam Adil Al Faiq – 24060124130083

Department of Informatics

Universitas Diponegoro

Semarang, Indonesia

maulanaghazzam4126@students.undip.ac.id

Abstract—JavaScript Object Notation (JSON) is one of the most widely used data exchange formats in modern computing systems. Ensuring that JSON documents conform to their syntactic specifications requires efficient grammar recognition techniques. This study presents a comparative performance analysis of three grammar recognition algorithms: dynamic programming based CYK Parsing, divide-and-conquer based Top-Down Predictive Parsing, and generalized Earley Parsing. A CFG representation of JSON was constructed based on RFC 8259 and transformed as necessary to satisfy the requirements of each parsing algorithm. The algorithms were evaluated using JSON datasets of varying structural complexity and compared in terms of execution time, memory consumption, and scalability. Experimental results show that Top-Down Predictive Parsing achieved the best overall performance, consistently recording the lowest execution time and memory usage across all dataset categories. Earley Parsing demonstrated moderate performance while maintaining the flexibility to process arbitrary context-free grammars. In contrast, CYK Parsing incurred the highest computational cost and exhibited limited scalability when processing highly complex JSON documents. The findings indicate that Top-Down Predictive Parsing is the most suitable approach for JSON grammar recognition when an LL(1)-compatible grammar is available, whereas Earley Parsing offers a practical alternative when greater grammatical flexibility is required.

Keywords—JSON, context free grammar, grammar recognition, dynamic programming, CYK parsing, divide and conquer, top-down predictive parsing, Earley parsing

I. INTRODUCTION

JavaScript Object Notation, or widely known as JSON, is one of the most used interchange formats in the modern world of computing systems thanks to its simplicity, readability, and compatibility with various programming languages. JSON is extensively utilized in both web and mobile applications, application programming interfaces (APIs), cloud computing platforms, and distributed systems for the exchange and storage of data. The increasing dependence on JSON for communication between software components highlights the importance of efficient and reliable mechanisms for validating and recognizing JSON documents before they are processed.

Validation of JSON documents can be examined through the principles of Formal Language and Automata Theory, in which a JSON document is first analyzed by a lexical analyzer (lexer), which converts the input stream into a sequence of tokens. These tokens are subsequently processed by a syntax validator, commonly referred to as a grammar recognizer, to determine whether the token sequence conforms to the grammatical rules of the JSON language. Such a two-stage approach, consisting of lexical analysis followed by syntax analysis, is widely employed in compiler design and language processing systems to ensure the syntactic correctness of input data.

Over the years, several algorithms have been developed for grammar recognition, with each applying different algorithm strategies whilst exhibiting its own distinct performance characteristics. Algorithms such as CYK (Cocke-Younger-Kasami) with its dynamic programming characteristics, Top-Down Predictive Parsing with its recursive divide-and-conquer classical strategy, and Earley Parsing with its generalization characteristics play a critical role in language processing systems by ensuring that input data conforms to the syntactic rules defined by a grammar.

The properties of parsing algorithms, including their computational complexity, grammar requirements, and recognition capabilities, have been extensively discussed in the literature [1, 2, 3, 4, 5]. However, comparative studies focusing on their performance specifically in the context of JSON grammar recognition remain limited to the best of the researchers' knowledge. Since JSON exhibits recursive structures such as nested objects and arrays, it provides an appropriate case study for evaluating the effectiveness of different parsing paradigms. Understanding the strengths and limitations of these approaches may contribute to the selection of suitable parsing techniques for data validation systems, language processors, and compiler-related applications.

This study aims to compare the performance of Dynamic Programming through the CYK algorithm, Divide-and-Conquer through Top-Down Predictive Parsing, and Earley Parsing in recognizing JSON grammars. Specifically, this study evaluates the algorithms in terms of execution time, memory consumption, and scalability across JSON inputs of varying complexity and size.

II. SCOPE AND LIMITATIONS

This study focuses on the comparative analysis of three parsing approaches for JSON grammar recognition, namely the CYK algorithm, Top-Down Predictive Parsing, and Earley Parsing. The research utilizes a context-free grammar representation of JSON based on the syntax specifications defined in JSON standards. The selected algorithms are implemented and evaluated using identical datasets to ensure consistency and fairness in performance measurement. The comparison is conducted using performance metrics including execution time, memory utilization, and scalability with respect to varying input sizes.

The datasets used in the study consist of syntactically valid JSON documents containing different structural characteristics, including objects, arrays, nested objects, nested arrays, and combined hierarchical structures. The grammar used for recognition is restricted to syntactic validation and does not incorporate semantic constraints or application-specific interpretations of JSON data. A variant of context-free grammar conformed to the Chomsky-Normal-

Form (CNF) and LL(1) constraint is also generated alongside the normal to enable CYK and TDP parsing.

Several limitations are acknowledged in this research. First, the study focuses exclusively on grammar recognition and does not address semantic analysis, schema validation, or data interpretation. Second, the study focuses only on one representative algorithm from each selected parsing paradigm: Dynamic Programming (CYK), Divide-and-Conquer (Top-Down Predictive Parsing), and Generalized Parsing (Earley Parsing). Other parser families, including LR, LALR, GLR, and PEG-based parsers, are not considered. Third, the performance evaluation is conducted within a controlled experimental environment, which may not fully represent all real-world deployment conditions. Finally, the study concentrates on valid JSON inputs and does not extensively evaluate error recovery mechanisms or malformed JSON documents. Consequently, the findings of this study should be interpreted within the context of syntax recognition and parser performance rather than comprehensive JSON processing systems.

III. LITERATURE REVIEW

A. JavaScript Object Notation (JSON)

JavaScript Object Notation (JSON) is a text format for the serialization of structured data derived from the object literals of the JavaScript programming language [6]. At its core, a JSON document is essentially a dictionary comprised of key-value pairs, where the value can again be a JSON document, thus allowing an arbitrary level of nesting [7]. Fig 1. shows an example of a simple JSON document, containing an object named "Image", which contains a nested object named "Thumbnail" and an array named "IDs".

```
{
  "Image": {
    "Width": 800,
    "Height": 600,
    "Title": "Daily Consumption",
    "Thumbnail": {
      "Url": "https://picsum.photos/id/63/5000/2813.jpg",
      "Height": 120,
      "Width": 160
    },
    "Animated": false,
    "IDs": [83, 44, 23, 202, 28, 138, 193]
  }
}
```

Fig. 1. Example of a simple JSON Document

Due to its simplicity, lightweight structure, and readability by both humans and machines, JSON has become one of the most widely adopted formats for data exchange on the Internet. This is evident with the use of JSON in the modern web services to communicate between applications and their users through an Application Programming Interface (API). Additionally, JSON format is widely used in database systems based on the NoSQL paradigm, reinforcing its significance in contemporary computing environments [7].

B. Context-Free Grammar

Context-Free Grammar is a formal mechanism used to describe the syntactic structure of languages. It is formally defined as 4-tuple

$$G = (V, T, S, P)$$

where V is a finite set of nonterminal symbols, T is a finite set of terminal symbols, S is the start symbol where $S \in V$,

and P is a finite set of production rules that define derivations between variables and terminal strings [3]. Fig. 2 shows an example of a context-free grammar commonly used to define simple arithmetic expressions [1]. In this grammar, the nonterminal symbols are

$E \ T \ F$

and the terminal symbols are

$+ \ - \ * \ / \ (\)$

with the following production rules

$$\begin{aligned} E &\rightarrow E + T \mid E - T \mid T \\ T &\rightarrow T * F \mid T / F \mid F \\ F &\rightarrow id \mid (E) \end{aligned}$$

Fig. 2. Example of a Context-Free Grammar for Simple Arithmetic Representation

CFGs are widely utilized in formal language theory and compiler systems due to their capability to represent recursive and hierarchical syntactic structures. Recursive production rules enable grammars to describe nested language constructs such as arithmetic expressions, programming statements, and hierarchical data formats. Since JSON documents frequently contain nested objects and arrays, their syntax can be effectively modeled using context-free grammar representations.

Furthermore, CFGs serve as the theoretical foundation for many parsing algorithms used in grammar recognition, including CYK, Top-Down Predictive Parsing, and Earley Parsing. Consequently, the effectiveness of these algorithms depends on their ability to process grammars that describe the syntactic structure of languages such as JSON.

C. Grammar Recognition

Grammar recognition refers to the process of determining whether a given input string can be generated by a specified grammar. In formal language theory, recognition serves as a fundamental operation for validating the syntactic correctness of language constructs. Unlike syntax-directed translation or semantic analysis, grammar recognition focuses exclusively on verifying membership within a language defined by a formal grammar [1].

In compiler construction, grammar recognition enables source code validation before subsequent stages such as semantic analysis and code generation are performed. Similarly, data formats such as JSON require syntax recognition to ensure that incoming data conforms to established structural specifications. Efficient grammar recognition contributes to system reliability by preventing malformed inputs from propagating through software systems.

Parsing algorithms provide practical mechanisms for grammar recognition by systematically analyzing the structure of an input string according to grammatical production rules [1]. Depending on their design, parsing algorithms may operate using top-down, bottom-up, deterministic, non-deterministic, dynamic programming, or generalized recognition strategies [1, 2]. The selection of an appropriate parsing algorithm often depends on factors such as grammar complexity, input size, computational efficiency, and implementation requirements. Consequently, evaluating the performance of different recognition algorithms remains an

important area of research within compiler theory, formal languages, and language processing applications.

D. Algorithms

1) CYK (Cocke-Younger-Kasami) Algorithm

The Cocke-Younger-Kasami (CYK) algorithm is a bottom-up classical algorithm for grammar recognition based on the dynamic programming paradigm. The algorithm systematically constructs a triangular parsing table in which each cell contains the set of nonterminal symbols capable of generating a particular substring of the input. The filling process begins with terminal productions and progressively combines previously recognized substrings according to the grammar rules until the entire input string is evaluated. By storing intermediate results and reusing previously computed subproblems, CYK avoids redundant computations and achieves polynomial-time recognition performance [4].

A key requirement of the CYK algorithm is that the grammar must first be transformed into Chomsky Normal Form (CNF), where every production rule is restricted to either two nonterminal symbols or a single terminal symbol on its right-hand side [3]. In other words, a production must either be in the form

$$A \rightarrow BC$$

or

$$A \rightarrow a$$

where A , B , and C are nonterminal symbols and a is a terminal symbol.

Such restriction allows CYK to recognize arbitrary context-free grammars, including ambiguous grammars. Furthermore, the algorithm guarantees recognition completeness by systematically examining all possible derivations. However, the CNF transformation process may increase grammar size and consequently introduce additional computational overhead [4].

The time complexity of the CYK algorithm is commonly expressed as

$$T(n) = O(n^3),$$

where n denotes the length of the input string. Its space complexity is

$$S(n) = O(n^2)$$

due to the storage requirements of the parsing table [4].

Because CYK is fundamentally based on dynamic programming principles, it serves as the representative Dynamic Programming approach in this study. Its structured table-driven recognition mechanism provides an appropriate basis for comparison with divide-and-conquer and generalized parsing techniques in the context of JSON grammar recognition.

2) Top-Down Predictive Parsing

Top-Down Predictive Parsing, also known as Predictive Recursive Descent Parsing, is an algorithm in which each nonterminal symbol of a grammar is implemented as a recursive procedure. The algorithm begins with the start symbol and recursively expands production rules while consuming input tokens until the

input is either successfully recognized or a syntax error is encountered [1, 2]. This recursive decomposition of the parsing problem naturally reflects the divide-and-conquer paradigm, where complex language constructs are broken down into smaller and more manageable subconstructs.

Top-Down Predictive Parsing is a deterministic parsing algorithm, meaning the parser can uniquely determine the next production rule to apply without requiring backtracking. Consequently, the grammar passed through this algorithm must satisfy the LL(1) constraint, where a nonterminal A with productions $A \rightarrow \alpha \mid \beta$ can always be distinguished between α and β using a single lookahead symbol. This constraint ensures that the parser can select the matching production rule unambiguously.

For grammars that satisfy LL(1) constraints, Top-Down Predictive Parsing generally exhibits linear-time complexity

$$T(n) = O(n),$$

where n represents the length of the input string [1, 2]. Additionally, the parser requires only a small amount of auxiliary memory beyond the recursion stack, making it computationally efficient for practical applications.

One of the primary advantages of Top-Down Predictive Parsing is its simplicity and ease of implementation. The recursive structure of the parser closely mirrors the structure of the grammar, resulting in readable and maintainable code. Furthermore, JSON syntax naturally contains recursive constructs such as nested objects and arrays, making it well suited for recursive parsing techniques.

As a representative Divide-and-Conquer approach, Top-Down Predictive Parsing is included in this study to evaluate the effectiveness of recursive grammar decomposition for JSON grammar recognition.

3) Earley Parsing

Earley Parsing is a generalized context-free grammar recognition algorithm proposed by Jay Earley in 1970. Unlike many parsing techniques that impose restrictions on grammar structure, Earley Parsing is capable of recognizing any context-free grammar without requiring transformations such as left factoring, left-recursion elimination, or Chomsky Normal Form conversion [5]. This flexibility makes Earley Parsing particularly suitable for applications involving complex or irregular grammar structures.

The algorithm operates through three fundamental procedures known as the Predictor, Scanner, and Completer. The Predictor generates possible productions that may be encountered based on the current parsing state, the Scanner matches expected terminal symbols against the input stream, and the Completer updates partially recognized productions when a constituent has been successfully parsed [5]. Together, these operations construct a collection of parsing states, commonly referred to as an Earley chart, which records the recognition progress at each input position.

An Earley state is commonly represented in the form

$$A \rightarrow \alpha \cdot \beta$$

where the dot ($\cdot$) indicates the current parsing position within a production rule. Symbols to the left of the dot have already been recognized, while symbols to the right remain to be processed [5].

One of the primary advantages of Earley Parsing is its ability to process arbitrary context-free grammars while naturally supporting recursive and ambiguous language constructs. Unlike the previous mentioned algorithms, Earley Parsing does not require the grammar to satisfy LL(1) constraints nor the need to conform into Chomsky Normal Form [5].

The computational complexity of Earley Parsing varies depending on the grammar being processed. For arbitrary context-free grammars, the worst-case time complexity is

$$T(n) = O(n^3),$$

while unambiguous grammars can be recognized in

$$T(n) = O(n^2).$$

For many deterministic grammars, Earley demonstrated that the algorithm can achieve near-linear performance,

$$T(n) = O(n)$$

where n denotes the length of the input string [5].

Despite its flexibility, Earley Parsing generally incurs greater implementation complexity and runtime overhead than specialized deterministic parsers. Nevertheless, its ability to process arbitrary context-free grammars without grammar transformation makes it an important benchmark for grammar recognition research.

As a representative generalized parsing approach, Earley Parsing is included in this study to evaluate the effectiveness of chart-based grammar recognition for JSON syntax validation.

IV. METHODOLOGY

This study employs an experimental comparative research design to evaluate the performance of three grammar recognition algorithms: CYK, Top-Down Predictive, and Earley Parsing. The algorithms are implemented under a common testing environment and evaluated using JSON documents of varying sizes and structural complexities. Performance is measured in terms of execution time, memory consumption, scalability.

To illustrate the experimental procedure employed in this study, a research framework was developed and is presented in Fig. 3. The experiment begins with a collection of JSON documents that constitute the dataset. Each document undergoes lexical analysis, during which a lexer converts the raw character stream into a sequence of tokens. The resulting token stream is then supplied to three separate parsers: the CYK parser, the Top-Down Predictive parser, and the Earley parser.

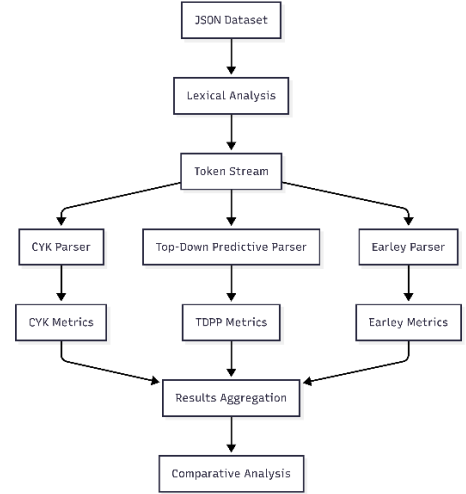


Fig. 3. Performance Experiment Flowchart

Each parser processes the token stream according to its corresponding recognition algorithm and produces a recognition result. During execution, performance metrics such as execution time and memory consumption are recorded for each parser. The collected metrics are subsequently aggregated and analyzed to facilitate a comparative evaluation of the three parsing approaches.

A. JSON Grammar Construction

Here, we construct a grammar representation of JSON derived from the RFC 8259 and related literature [6, 7]. In order to facilitate parser implementation and performance evaluation, several simplifications are introduced. First, whitespace characters are not explicitly represented in the grammar. Second, object and array delimiters are assumed to appear without surrounding whitespace. Third, key-value pairs are represented using a standardized format in which a colon separating a key and value is followed by a single whitespace character.

Furthermore, lexical analysis is considered a separate preprocessing stage and is therefore excluded from the grammar representation. During lexical analysis, JSON documents are transformed into token streams consisting of lexical tokens such as **STRING**, **int**, **float**, **true**, **false**, **null**, and structural symbols including braces, brackets, commas, and colons. Consequently, the grammar used in this study operates on tokens generated by the lexer rather than on individual characters. This separation allows the evaluation to focus exclusively on the grammar recognition capabilities of the selected parsing algorithms.

The constructed grammar consists of nonterminal symbols and terminal symbols. The nonterminal symbols are represented by non-bolded characters with the following members:

{JSON, VALUE, KEY, OBJECT, MEMBERS, ARRAY, ELEMS, NULL, BOOLEAN, NUMBER}

while the terminal symbols generated by the lexical analyzer are represented by bolded characters with the following members:

{**{**, **}**, **[**, **]**, **,**, **:**, **STRING**, **null**, **false**, **true**, **int**, **float**}

As discussed previously, lexical analysis is performed separately from grammar recognition. Consequently, lexical tokens such as **STRING**, **int**, and **float** are treated as terminal symbols rather than being further decomposed into character-level productions.

The start symbol of the grammar is **JSON**, from which all valid derivations originate. The complete set of production rules used to represent the JSON grammar is shown in Fig. 4.

JSON	$\rightarrow$ VALUE
VALUE	$\rightarrow$ OBJECT ARRAY STRING NUMBER BOOLEAN NULL
PAIR	$\rightarrow$ STRING : VALUE
OBJECT	$\rightarrow$ { MEMBERS } {}
MEMBERS	$\rightarrow$ PAIR PAIR , MEMBERS
ARRAY	$\rightarrow$ [ELEMS] []
ELEMS	$\rightarrow$ VALUE VALUE , ELEMS
NULL	$\rightarrow$ null
BOOLEAN	$\rightarrow$ false true
NUMBER	$\rightarrow$ int float

Fig. 4. Production Rules of CFG Representation of JSON

B. Grammar Normalization

Although all three parsing algorithms operate on context-free grammars, they impose different structural requirements on the grammar representation. Consequently, the JSON grammar presented in Fig. 4 undergoes normalization procedures prior to parser implementation. Specifically, the grammar used for CYK Parsing is converted into Chomsky Normal Form (CNF), while the grammar used for Top-Down Predictive Parsing is transformed to satisfy LL(1) constraints. Earley Parsing does not require any grammar transformation and therefore utilizes the original grammar directly.

1) Chomsky Normal Form (CNF)

As stated previously, CYK requires the grammar to be in Chomsky Normal Form. With the original grammar presented in Fig. 4, only the nonterminal symbols **NULL**, **BOOLEAN**, **NUMBER**, and one production rule of nonterminal symbol **VALUE** satisfies the CNF requirement, where production results in only one terminal symbol. However, many productions contain a mixture of terminal and nonterminal symbols or contain more than two symbols on the right-hand side, thereby violating the CNF constraints.

For example, consider the production rule

$$\mathbf{OBJECT} \rightarrow \{\mathbf{MEMBERS}\}$$

which consists of three symbols on the right hand: two terminal symbols encapsulating a nonterminal symbol or two terminal symbols, which violates the CNF requirement.

To address this, terminal symbols appearing alongside nonterminal symbols were replaced with auxiliary nonterminal symbols, while productions containing more than two symbols on the right-hand side were separated into multiple binary productions. Hence, the production was transformed into

$$\mathbf{OBJECT} \rightarrow \mathbf{LCB} \mathbf{V1}$$

$$\mathbf{V1} \rightarrow \mathbf{MEMBERS} \mathbf{RCB}$$

where

$$\mathbf{LCB} \rightarrow \{$$

$$\mathbf{RCB} \rightarrow \}$$

Similar transformations were applied to all productions that violated the CNF constraints. The resulting grammar was then used as the input grammar for the CYK parser. The complete CNF grammar is provided in the Appendix.

2) LL(1) Constraint

The same principle applies to Top-Down Predictive Parsing, where the grammar must satisfy the LL(1) constraint. The original grammar presented in Fig. 4 does not fully satisfy the constraint, as several nonterminal symbols contain alternative productions that begin with the same symbol.

Consider the production rule

$$\mathbf{MEMBERS} \rightarrow \mathbf{PAIR} \mid \mathbf{PAIR}, \mathbf{MEMBERS}$$

Both alternatives start with the nonterminal symbol **PAIR**. As a result, the parser cannot distinguish between the two productions by examining only the next input symbol. While this ambiguity can be resolved through backtracking, predictive parsers are designed to operate deterministically without backtracking. Therefore the grammar is instead normalized to compensate.

The transformation process involves left-factoring, in which the common prefix is extracted into a separate production, ensuring that all production of the nonterminal symbol starts with a unique symbol. Hence, the production was transformed into:

$$\mathbf{MEMBERS} \rightarrow \mathbf{PAIR} \mathbf{MAUX}$$

$$\mathbf{MAUX} \rightarrow , \mathbf{MEMBERS} \mid \epsilon$$

Through this transformation, the parser first recognizes the common prefix **PAIR** before determining whether additional members follow. Similar transformations were applied to all productions that violated LL(1) constraints. The resulting grammar was then used as the input grammar for the TDP parsing. The complete LL(1)-normalized grammar is provided in Appendix.

C. Dataset Preparation

The dataset used in this study consists of 100 syntactically valid JSON documents collected from publicly available online sources, designed to ensure conformance with the grammar specification presented in Fig. 4.

To evaluate parser behavior under varying structural conditions, the dataset was partitioned into four complexity categories, each containing 25 documents:

1. **SIMPLE**: JSON documents in this category are relatively small in size and contain little to no nesting. The documents primarily consist of flat key-value pairs and simple arrays, resulting in low structural complexity.
2. **MODERATE**: JSON documents contain a limited number of nested objects and arrays. The overall

structure remains manageable while introducing moderate hierarchical complexity.

3. **COMPLEX:** JSON documents contain multiple levels of nested objects and arrays, resulting in deeper hierarchical structures and increased parsing complexity.
4. **HIGHLY COMPLEX:** JSON documents exhibit extensive nesting, large arrays, and highly hierarchical structures. This category is intended to evaluate parser scalability under demanding input conditions.

D. Performance Metrics

To evaluate the performance of the selected parsing algorithms, three performance metrics were considered: execution time, memory consumption, and scalability.

1) Execution Time

The execution time for each parser was computed as

$$T = t_{end} - t_{start}$$

where t_{start} represents the time immediately before the parsing process begins and t_{end} represents the time immediately after parsing is completed. The resulting value was recorded in milliseconds (ms).

2) Memory Consumption

For each parsing operation, the peak allocated memory was recorded and used as the memory consumption metric. This value represents the maximum amount of memory required by the parser while processing a JSON document and was reported in kilobytes (KB).

3) Scalability

In this study, scalability was evaluated by observing changes in execution time and memory consumption across datasets of varying complexity levels. A parser is considered more scalable if its resource requirements increase gradually as the number of tokens and nesting depth of the JSON documents grows.

V. RESULTS AND ANALYSIS

A. Execution Time Test Result and Analysis

TABLE 1. EXECUTION TIME TABLE

Data Complexity	Time Average (ms)		
	TDP	Earley	CYK
Simple	0.028	1.046	0.421
Medium	0.174	3.426	9.008
Complex	0.192	28.135	185.208
Highly Complex	1.264	125.609	142,672.869

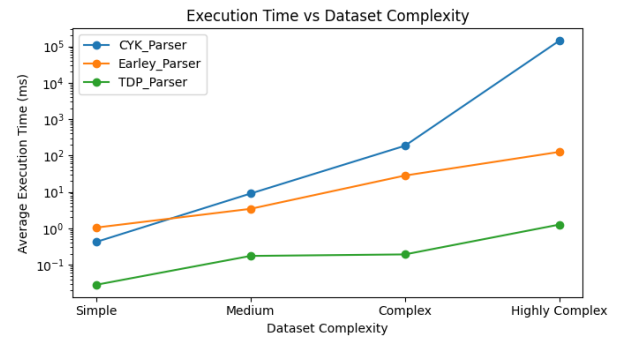


Fig. 5. Average Execution Time Across Different Complexity Levels

Table 1 and Fig. 5 present the average execution times obtained for each parser across the different dataset categories.

The results show that Top-Down Predictive Parsing consistently achieved the lowest execution time among all evaluated algorithms. The parser required an average of only 0.028ms for simple documents, 0.174ms for medium documents, 0.192ms for complex documents, and 1.264ms for highly complex documents. Even for the most complex inputs, the execution time remained below one millisecond, demonstrating excellent computational efficiency.

Earley Parsing exhibited moderate performance. Average execution times increased from 1.046ms for simple documents to 125.609ms for highly complex documents. Although significantly slower than Top-Down Predictive Parsing, Earley Parsing remained capable of processing all dataset categories without requiring grammar transformations beyond the original context-free grammar representation.

CYK Parsing exhibited the highest execution times throughout the experiment. The average execution time increased from 0.421ms for simple documents to a staggering 2 minutes and 22 seconds for highly complex documents. This substantial increase in processing time can be attributed to the dynamic programming table construction required by the algorithm, which evaluates all possible derivations for every substring of the input.

B. Memory Consumed Test Result and Analysis

TABLE 2. MEMORY CONSUMPTION TABLE

Data Complexity	Memory Consumed (KB)		
	TDP	Earley	CYK
Simple	0.25	21.9	64.79
Medium	0.72	86.75	1,047.94
Complex	1.6532	260.74	7,800.49
Highly Complex	10.648	1,702.78	N/A

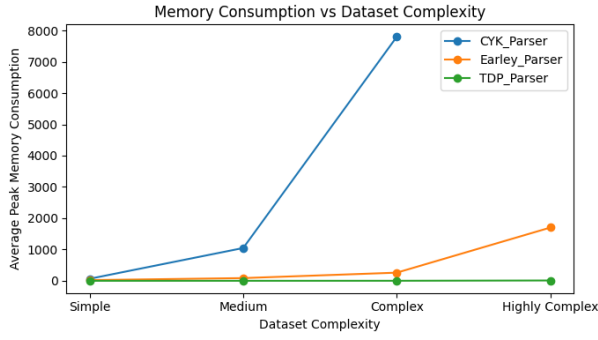


Fig. 6. Average Memory Consumption Across Different Complexity Levels

Table 2 and Fig. 6, indicates Top-Down Predictive Parsing consumed the least memory across all dataset categories. Average memory consumption ranged from only 0.25 KB for simple documents to approximately 10.648 KB for highly complex documents. The minimal memory footprint can be attributed to the parser's deterministic nature, which primarily relies on recursive procedure calls and a limited parsing stack.

Earley Parsing required substantially more memory than Top-Down Predictive Parsing. Average memory consumption increased from 22.96 KB for simple documents to 1,702.78 KB for highly complex documents, with a maximum recorded value of 2,589.48 KB. This behavior is expected because Earley Parsing maintains multiple chart states throughout the recognition process, causing memory requirements to increase with input complexity.

CYK Parsing yet again exhibited the highest memory requirements among the evaluated algorithms. For medium-sized documents, the parser consumed an average of 1,047.49 KB of memory, while complex documents required approximately 7,800.49 KB, with a maximum recorded value of 8,172.62 KB. Reliable memory measurements for the Highly Complex category could not be obtained, as the profiling overhead introduced by Python's memory-tracing instrumentation, combined with CYK's already substantial runtime on these inputs, prevented recognition from being completed within the available session.

These results suggest that memory consumption is a major factor affecting the practical applicability of CYK Parsing for large JSON documents. In contrast, Top-Down Predictive Parsing demonstrates exceptional memory efficiency, while Earley Parsing represents a compromise between memory usage and grammatical flexibility.

C. Scalability Analysis

Top-Down Predictive Parsing demonstrated the most favorable scalability characteristics. Although execution time and memory consumption increased with document complexity, the rate of growth remained relatively small. The parser maintained sub-millisecond execution times and low memory usage even for highly complex JSON documents. This behavior reflects the effectiveness of deterministic parsing when applied to an LL(1)-compatible grammar.

Earley Parsing demonstrated moderate scalability. Resource consumption increased steadily as document complexity increased; however, the parser remained capable of processing all dataset categories successfully. The results suggest that Earley Parsing provides a practical balance between performance and grammatical flexibility.

CYK Parsing exhibited the poorest scalability among the evaluated algorithms. Both execution time and memory consumption increased rapidly as document complexity grew. These findings are consistent with the theoretical characteristics of dynamic-programming-based recognition algorithms, which require substantial storage and computation for large inputs.

D. Comparative Analysis

The experimental results reveal a clear trade-off between computational efficiency and grammatical flexibility. Top-Down Predictive Parsing achieved the best overall performance in terms of execution time and memory consumption. Its deterministic LL(1)-based recognition strategy enabled rapid processing while requiring minimal computational resources. Consequently, it represents the most suitable approach for JSON grammar recognition when grammar normalization is feasible.

Earley Parsing achieved moderate performance while retaining the ability to recognize arbitrary context-free grammars without requiring Chomsky Normal Form conversion or LL(1) normalization. Although slower than Top-Down Predictive Parsing, its ability to process all dataset categories successfully demonstrates its practical applicability for more general parsing tasks.

CYK Parsing exhibited the lowest practical performance among the evaluated algorithms. While theoretically powerful and guaranteed to recognize context-free grammars in Chomsky Normal Form, its substantial execution time significantly limited scalability, with individual Highly Complex documents requiring over two minutes to process. This runtime cost also prevented memory profiling from completing on this category within the available session, leaving CYK's memory behavior at this tier unresolved rather than confirmed to be excessive.

Hence, the experimental findings indicate that Top-Down Predictive Parsing provides the most efficient solution for JSON grammar recognition, provided the grammar can be normalized to LL(1) form. Earley Parsing offers a balanced alternative, accepting the input grammar directly without requiring any transformation. CYK Parsing, despite requiring a comparable normalization step (conversion to Chomsky Normal Form), fails to yield either the efficiency of Top-Down Predictive Parsing or the transformation-free flexibility of Earley Parsing, positioning it as the least practical of the three approaches for this recognition task despite its theoretical generality.

VI. CONCLUSION AND FUTURE WORK

A. Conclusion

This study presented a comparative performance analysis of three grammar recognition algorithms representing different algorithmic paradigms, namely CYK Parsing as a Dynamic Programming approach, Top-Down Predictive Parsing as a Divide-and-Conquer approach, and Earley Parsing as a generalized parsing approach.

Experimental evaluation was conducted using JSON documents of varying structural complexity. The algorithms were compared in terms of execution time, memory consumption, and scalability. The results indicate that all three algorithms were capable of correctly recognizing valid JSON

documents, demonstrating the correctness of the implemented grammar and parser designs.

Among the evaluated algorithms, Top-Down Predictive Parsing generally exhibited the best performance in terms of execution time and memory consumption. Its deterministic nature and LL(1)-based parsing strategy allowed the parser to recognize JSON documents efficiently with minimal computational overhead. Earley Parsing demonstrated moderate performance while maintaining the advantage of supporting arbitrary context-free grammars without requiring extensive grammar transformations. In contrast, CYK Parsing incurred the highest computational cost due to the construction and evaluation of dynamic programming tables, although it remained effective as a grammar recognition technique.

Overall, the findings suggest that Top-Down Predictive Parsing is the most suitable approach for JSON grammar recognition when an LL(1)-compatible grammar is available. Nevertheless, Earley Parsing and CYK Parsing remain valuable alternatives in situations where greater grammatical flexibility is required.

B. Future Work

Several opportunities exist for extending the present study. First, future research may expand the comparative analysis by including additional parsing algorithms such as LR, LALR, GLR, PEG, and Packrat parsers. Incorporating these techniques would provide a broader understanding of the trade-offs between parser performance, grammar requirements, and implementation complexity.

Second, the current study focuses exclusively on grammar recognition of syntactically valid JSON documents. Future studies may investigate parser behavior in the presence of malformed inputs, error recovery mechanisms, and syntax diagnostics. Such analyses would provide a more comprehensive evaluation of parser applicability in real-world software systems.

Third, future work may consider larger datasets and more diverse JSON structures, including documents obtained from real-world web services, NoSQL databases, and cloud-based applications. Evaluating parser performance under these conditions may provide additional insights into practical deployment scenarios.

Finally, future studies may revisit the Earley Parsing approach by utilizing more efficient variants such as Practical Earley Parsing (PEP) proposed by Aycock and Horspool [8]. PEP introduces several optimizations to the traditional Earley algorithm that reduce parsing overhead while preserving its ability to process arbitrary context-free grammars. Comparing PEP against CYK, Top-Down Predictive Parsing, and conventional Earley Parsing may provide a deeper understanding of the trade-offs between grammatical flexibility and computational efficiency in JSON grammar recognition.

APPENDIX

A. Source Code and Grammars

All programs and context-free grammars used to perform this study are uploaded into the following link:

<https://drive.google.com/drive/folders/1ckmJZ4QGUO5MT-RxdNA9NbGL1gRPURtIT?usp=sharing>.

REFERENCES

- [1] A. V. Aho, *Compilers: Principles, Techniques, & Tools*. Pearson, 2007.
- [2] D. Grune and C. J. H. Jacobs, *Parsing techniques*. 2007. doi: 10.1007/978-0-387-68954-8.
- [3] J. E. Hopcroft, R. Motwani, and J. D. Ullman, "Introduction to automata theory, languages, and computation, 2nd edition," *ACM SIGACT News*, vol. 32, no. 1, pp. 60–65, Mar. 2001, doi: 10.1145/568438.568455.
- [4] F. M. Zanzotto, G. Satta, and G. Cristini, "CYK Parsing over Distributed Representations," *Algorithms*, vol. 13, no. 10, p. 262, Oct. 2020, doi: 10.3390/a13100262.
- [5] J. Earley, "An efficient context-free parsing algorithm," *Communications of the ACM*, vol. 13, no. 1, pp. 57–102, Feb. 1970, doi: 10.1145/357980.358005.
- [6] P. Bourhis, J. L. Reutter, and D. Vrgoč, "JSON: Data model and query languages," *Information Systems*, vol. 89, p. 101478, Dec. 2019, doi: 10.1016/j.is.2019.101478.
- [7] F. Pezosa, J. L. Reutter, F. Suarez, M. Ugarte, and D. Vrgoč, *Foundations of JSON Schema*. 2016, pp. 263–273. doi: 10.1145/2872427.2883029.
- [8] J. Aycock, "Practical Earley Parsing," *The Computer Journal*, vol. 45, no. 6, pp. 620–630, Jun. 2002, doi: 10.1093/comjnl/45.6.620.

STATEMENT

I hereby declare that this paper is an original work of my own and is neither an adaptation, translation, nor plagiarism of another's work. All sources cited and referenced in this paper is listed according to the official scientific writing guidelines and properly acknowledged.

Regarding the use of Artificial Intelligence (AI), I hereby declare that **AI was utilized** solely as an assistive tool during the preparation of this paper, with the following details:

- AI Tool: **ChatGPT**
- Purpose of Use: Assisting in improving paragraph coherence, academic writing style, grammar, and language clarity
- Scope of Use: Assisting in drafting, paraphrasing, proofreading, language refinement, and organizing the structure of the paper. AI was not used to generate research results, conduct experiments, perform data analysis, or formulate conclusions.

The use of AI is limited to only as a writing assistance tool and did not replace my role as the primary author. All contents, analysis, and conclusions written within this paper fully remain entirely my responsibility.

Semarang, 7th of June 2026



Maulana Ghazzam Adil Al Faiq
24060124130083