





Research Article

Enhancing XML-based Compiler Construction with Large Language Models: A Novel Approach

Idrees A. Zahid^{1, 2,*}, Shahad Sabbar Joudar¹, ¹ Information Technology Center, University of Technology, Baghdad, Iraq² Electrical & Computer Engineering, Gannon University, Erie, PA, USA

ARTICLE INFO

ABSTRACT

Article History

Received 03 Jan 2024

Accepted 02 Mar 2024

Published 20 Mar 2024

Keywords

XML, CFG, XML

schema, LL(1), compiler,

grammar, Large

Language Model (LLM).



Considering the prevailing rule of Large Language Models (LLMs) applications and the benefits of XML in a compiler context. This manuscript explores the synergistic integration of Large Language Models with XML-based compiler tools and advanced computing technologies. Marking a significant stride toward redefining compiler construction and data representation paradigms. As computing power and internet proliferation advance, XML emerges as a pivotal technology for representing, exchanging, and transforming documents and data. This study builds on the foundational work of Chomsky's Context-Free Grammar (CFG). Recognized for their critical role in compiler construction, to address and mitigate the speed penalties associated with traditional compiler systems and parser generators through the development of an efficient XML parser generator employing compiler techniques. Our research employs a methodical approach to harness the sophisticated capabilities of LLMs, alongside XML technologies. The key is to automate grammar optimization, facilitate natural language processing capabilities, and pioneer advanced parsing algorithms. To demonstrate their effectiveness, we thoroughly run experiments and compare them to other techniques. This way, we call attention to the efficiency, adaptability, and user-friendliness of the XML-based compiler tools with the help of these integrations. And the target will be the elimination of left-recursive grammars and the development of a global schema for LL(1) grammars, the latter taking advantage of the XML technology, to support the LL(1) grammars construction. The findings in this research not only underscore the significance of these innovations in the field of compilation construction but also indicate a paradigm move towards the use of AI technologies and XML in the context of the resolution of programming traditional issues. The outlined methodology serves as a roadmap for future research and development in compiler technology, which paves the way for open-source software to sweep across all fields. Gradually ushering in a new era of compiler technology featuring better efficiency, adaptability, and all CFGs processed through existing XML utilities on a global basis.

1. BACKGROUND

Lately, Extensible Markup Language (XML) has evolved into a paramount standard in all the fields of computing, where it has been able to grow in the industries due to its human-readable, text-based, and self-descriptive nature [1]. XML is the core markup language that is used across operating systems, programming languages, and many other computing environments. This demonstrates XML's wide range of applicability and interoperability which is also supported by libraries that conform to the XML standards [2][3]. This omnipresence makes the tools capable of getting the documents in XML format and processing their data markups and exchanges without any difficulties [4][5].

In parallel with the growth of XML, the compiler—the inevitable component of the software development process that translates high-level programming languages into machine code—has made considerable progress [6]. Compilers acting as a bridge between human knowledge and computer algorithms are critical in achieving software development goals. An innovation in this section is the ability of XML files to enclose Context-Free Grammar objects (CFGs). That not only helps to develop new compilers, especially for the Unity Grammar Recognizer but also makes it possible to use XML for the creation of compilers more widely. Artificial Intelligence (AI) rapidly improved and exponentially employed in many fields and domains [7]. Text analysis is one of the extensively researched and developed fields. Text and language processing has gained great attention from scientists and researchers because of its impact and importance.

*Corresponding author. Email: iajahid@gmail.com

With the introduction of the epochal Large Language Models (LLMs), including GPT, BERT, and other architectures, there is a new landscape for compiler technology according to recent powerful AI advancements [8]. LLMs are a source of growth for XML-based compiler tools because of the machine's ability to detect grammatical problems, natural language processing, and written text generation that is indistinguishable from humans. The area of research that is of most interest to us deals with the synergy between LLMs and compiler construction. The revolutionizing role that LLMs can play in this field is the main focus of our research. This integration is set to remove the development process obstacles, introduce new parsing algorithms, and do semantic analysis differently and with error correction [9]. The integration of XML's wide spectrum of applications with LLMs' powerful computational capacity exemplifies the inevitable way towards the revolution in compilers' construction. It anticipates a future where compilers become not only more productive and adaptable but also competent enough to deal with the confusing aspects of language and grammar with unbelievable precision. This development is aimed to be achieved in a more agile, precise, and fault-resilient compiler development process, that in turn will pave the way for new and innovative applications in software engineering [9,10,11]

The process of detailed experimentation and integration with different XML standards and LLMs to produce our work is being discussed in the ongoing discussion on the interface of XML standards and LLMs in compiler technology. It clearly demonstrates that the integration of these two approaches has the potential to overcome the limitations associated with conventional compiler construction, and it creates a picture of the future, where these two approaches are combined and the paradigms of programming language processing are modified.

1.1 XML and XML Schema

XML, which is an acronym for eXtensible Markup Language, is a hierarchical meta markup language that was derived from the Standard Generalized Meta Language (SGML). XML, which has been in use since 1996 and supported by the W3C consortium, spearheaded by Jon Bosak of Sun Microsystems, has become an essential tool for data representation across several areas of application. Unlike HTML which is used for data presentation, XML is used for the expression of data structure. It is then able to facilitate data transmission among different applications. This capability is the essence of XML as a global language widely used by organizations, industries, and academia to organize and classify data across intranets and the internet [12,13,14].

From the Document Type Definitions (DTDs) to XML Schema the evolution is marked by the fact that the language becomes capable of structuring data. While DTDs use Extended Backus-Naur Form (EBNF) grammar and serve as a base structure to define the legal entities in an XML document, they are frequently seen as limited in their domain and applicability. Instead, XML Schema exploits XML syntax itself, which introduces a higher level of complexity with support for namespaces, occurrence constraints, and a massive set of simple and complex data types [15]. The switch to DTDs eliminates the cryptic nature of DTDs and also improves XML's ability to respond to the growing sophistication of data structuring requirements in the modern computing world.

The thing that is highlighted is that XML documents can be either well-formed (they are constructed according to the rules of the XML syntax) or valid (they conform to a particular schema). This distinction shows us that the role of the XML Schema is to validate and maintain the data that are in XML format [16]. XML is structured in a way that it can be easily interpreted and processed by both human and computer systems, consequently making it an extraordinary resource for data markup and exchange.

In essence, the combination of XML and XML Schema enables superior data representation, with a much higher degree of efficiency than previous solutions. The XML features self-explanatory, human-readable and extensible characteristics. Hence; XML is used for a wide variety of applications such as marking up data in different computing environments and also for retrieving and validating complex data structures. With the computing environment ever-changing, the function and significance of XML and its schemas remain vital in how the data is organized and transmitted [17][18].

1.2 Syntax Analysis

Compiler design is a complicated domain where translation, error detection, recovery, and other things are interconnected to translate human-readable notations into low-level instructions [19]. At the core of the compiler is the front end, which is responsible for lexical, syntax, and semantic analysis. These tasks transform input tokens into the structured form known as the parse tree [20]. This architecture corresponds to the code text organized by the compiler and is an important aspect of the code generation. It also plays a vital role in optimization that is part of the compiler's back-end, as optimization emerges as a key player for algorithms and method improvement [21][22]. Parsing, which is one of the most important steps during syntax analysis, relies on different approaches to break down the tokens produced by lexical analysis. As a result, a structure that describes the program's intended logic and functions is created.

The traditional parsing techniques, such as the top-down, bottom-up, and universal ones like the Earley algorithm and the C-Y-K algorithm [23]. All these techniques address different classes of grammar. In particular, the LR and LL grammars are accommodated to match the syntactic constructs of modern programming languages. Nevertheless, these traditional techniques, although successful in dealing with some language subsets, might encounter some limitations when it comes to efficiency and flexibility in production compilers.

The emergence of Large Language Models (LLMs) is an innovative channel to re-invent the compiler construction approach. LLMs, which have the most sophisticated natural language processing capacities, offer a brand-new method of advancement of the traditional parsing methods substantially. The utilization of LLMs could be a gateway to more productive syntax analysis and error detection, as well as the automation of several tasks in code generation [24]. Such an approach not only optimizes the compiler design process but also introduces a degree of versatility and effectiveness unattainable with traditional parsing methods alone.

The merging of LLMs in the compiler design releases the capability to transform the landscapes of compiler technology. The integration of conventional compiler architectures with the novel and distinctive capabilities of LLMs is in a position to reshape the way compilers are developed. It ensures a more perceptive and intelligent approach to the translation of human-legible notations into machine code. As well as a development in operational approach for this scenario.

1.3 CFG and LL(1) grammar

Formal languages theory as a discipline is known to have originated from the work of Noam Chomsky who is a linguist [25]. For the first time in the 1950s, an attempt was made by this linguist to precisely characterize the structure of natural languages. In his work, he basically aimed to define the syntax of languages through the use of precise and straightforward mathematical rules. Based on the classification given by Noam Chomsky, there are four divisions of the hierarchy of grammars [26][27]. The four classes in the hierarchy include Regular grammar, Unrestricted grammar, Context-free grammar, and Context-sensitive grammar. The classification of languages provided by Chomsky was a result of the work he did involving the study of the structure of natural languages. The classification provided by Chomsky was based on the complexity of the grammar. As a result of his work in the area of natural language, CFG formalism emerged [28]. Through the CFG formalism, the syntax of programming languages can be easily described. Since then, CFG has emerged as a crucial element in the description of linguistic syntax [29]. In addition, the CFG has also emerged as the most popular means of representing the rules of language syntax for English as well as that of other natural languages. In the work of [30], the authors formally defined a Context-Free Grammar (CFG) as: a 4-tuple (V, Σ, R, S) where:

1. V is a finite set referred to as the variables,
2. Σ is a finite set, different from V , and is known as the terminals,
3. R is a finite set of rules, and each of the rules is a variable and a string of variables and terminals, and
4. $S \in V$ is the start variable.

If u, v , and w are strings of variables and terminals, and $A \rightarrow w$ is the grammar rule, assuming uAv yields uwv , written $uAv \rightarrow uwv$.

Assuming that u produces v , written $u \rightarrow v$, if $u = v$ or if a sequence u_1, u_2, \dots, u_k exists for $k \geq 0$ and $u \rightarrow u_1 \rightarrow u_2 \rightarrow \dots \rightarrow u_k \rightarrow v$.

Example of Context-Free Grammar (CFG):

The parsing of construct keywords such as “int” or “while” is comparatively easy since the choice of grammar production is guided by the keyword. The application of the grammar production must correspond with the input. Here, the focus is on expressions that are more complex and challenging, because of the associativity and precedence of operators. These two are expressed in the grammar below, through simple mathematical expressions. In addition, they also describe terms, expressions, and factors. The terminal symbols in this grammar include: $id + - * / ()$, the nonterminal symbols are *expression*, *term* and *factor*, and *expression* is the start symbol.

expression	\rightarrow	expression + term
expression	\rightarrow	expression – term
expression	\rightarrow	Term
term	\rightarrow	term * factor
term	\rightarrow	term/factor
term	\rightarrow	Factor
factor	\rightarrow	(expression)
factor	\rightarrow	Id

The result below can be obtained when the notational conventions for the above grammar are applied:

E: denotes expression that are made up of terms that are separated by + signs.

T: indicates terms that are made up of factors that are separated by * signs.

F: denotes the factors that can be classified as identifiers or parenthesized expressions.

The table below shows another grammar that falls under the class of LR grammar; the bottom-up parsing approach is more suitable for this kind of grammar. The reason why the top-down parsing approach is not suitable for this kind of grammar is because it is inherently left recursive:

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

The LL(1) grammar which is shown below will be used for top-down parsing since it is a non-left-recursive variant of this expression grammar:

$$\begin{aligned} E &\rightarrow T E' \\ E' &\rightarrow + T E' \mid \varepsilon \\ T &\rightarrow F T' \\ T' &\rightarrow * F T' \mid \varepsilon \\ F &\rightarrow (E) \mid \text{id} \end{aligned}$$

A grammar that possesses a nonterminal A , is considered as left recursive, such that string a has a derivation $A \Rightarrow Aa$. It is impossible for left-recursive grammars to be handled by the top-down parsing methods, and for that reason, the elimination of the left recursion can be achieved through a transformation.

The problem associated with the construction of a parse tree for the input string is the top-down parsing which involves creating a parse tree starting from the root, and creating the nodes of the parse tree in preorder. In addition, in top-down parsing, a leftmost derivation is found for the input string. One of the major issues encountered at each step of this method is to determine what production is more suitable for application in a nonterminal. Generally, the top-down method of parsing is referred to as recursive-descent parsing. In this method, the choice of the most suitable production to be applied can be made through the process of backtracking. The recursive descent has an exceptional case which is referred to as predictive parsing. In this exceptional case, the most suitable production is selected by looking ahead at the input of a fixed number of symbols (the next input symbol) [33, 34, 52].

Sometimes, the class of grammars that can be used for the construction of predictive parsers looking κ symbols ahead in the input is referred to as the $LL(\kappa)$ class. There are two functions that support the process of bottom-up and top-down parsing which are, (FIRST and FOLLOW). Utilizing this function, the production that is suitable for application can be selected based on the subsequent input symbol. More so, these two functions are sets of computations through which a “predictive parsing table,” is constructed. This way, the selection of the most appropriate production is done explicitly when bottom-up parsing is implemented [35, 36].

LL(1) is a class of grammar which:

1. The first “L” in LL(1) means that the input should be scanned from left to right.
2. The second “L” is for the production of a leftmost derivation.
3. The “1” means that one input symbol of look ahead should be used at every stage for decision-making about parsing action.

Even though caution must be taken when an appropriate grammar is written for the source language, the class of LL(1) grammars can sufficiently cover the majority of programming constructs [31]. A Context-Free Grammar $G = (V, \Sigma, R, S)$ that has a parsing table in which multiple entries are absent is referred to as LL(1). A language is categorized as an LL(1) if it is derived from an LL(1) grammar. As seen, LL(1) grammars are not characterized by ambiguity, and they are not left-recursive.

Formal definition of LL(1) Grammar:

A grammar G can only be referred to as LL(1), when $A \rightarrow \alpha \mid \beta$ are two different productions of G . The following conditions hold:

For no terminal a , do both α and β derive strings beginning with a .

1. At most, the empty string can be generated by one of α and β .
2. If $\beta \Rightarrow \varepsilon$, then no string that begins with a terminal in FOLLOW(A) is derived by α .
3. Similarly, if $\alpha \Rightarrow \varepsilon$, then no string starting with a terminal in FOLLOW(A) is produced by β .

The first two conditions are the same as the statement that FIRST(α) and FIRST(β) are two different sets. The third condition is the same as stating that if ε is in FIRST(β), then FIRST(α) and FOLLOW(A) are not the same sets, and likewise if ε is in FIRST(α). It is possible to construct predictive parsers for LL(1) grammars, because the right production that should be applied to a nonterminal can be chosen by only having a look at the current input symbol. The constraints of LL(1) can basically be satisfied by the flow-of-control constructs alongside their different keywords.

The FIRST and FOLLOW sets can be incorporated into a predictive parsing table $M[A,a]$. A two-dimensional array (where “A” is a nonterminal and “a” is a terminal or the symbol \$, the input end-marker). The predictive parsing can be constructed using an algorithm that is based on the idea given below:

1. In the event that the subsequent input symbol is in $FIRST(\alpha)$, then the production $A \rightarrow \alpha$ is selected. The problem will only arise if $\alpha = \epsilon$ or, more generally $\alpha \rightarrow \epsilon$.
2. In such a situation, $A \rightarrow \alpha$ should be selected again, if $FOLLOW(A)$ contains the current input symbol, or if the \$ on the input has been reached and \$ is in $FOLLOW(A)$.

Upon completion of the algorithm implementation, production ends in $M[A,a]$, and then $M[A,a]$ should be set to error (which is often denoted by an empty entry in the table). The application of this algorithm can be made to any grammar G so that a parsing table M can be produced. For every LL(1) grammar, each entry of the parsing-table has the capability to outstandingly detect production or error. However, in some grammars, M may possess some entries that are defined in different ways. For instance, in a situation whereby G is characterized by ambiguity or is left-recursive, then M will possess at least one entry that has been defined in multiple ways. In as much as the elimination of left-recursive and left factoring can be easily achieved, there are some kinds of grammars that no matter what kind of modifications are made in them, an LL(1) can never be produced [32][33]. The classification of grammar, as introduced by Noam Chomsky, provides a framework for understanding the complexity of language syntax. Context-Free Grammar (CFG) has been instrumental in the syntax description of programming languages. This incorporation of LLMs into the compiler construction process will be a big deal, especially in grammar analysis where it can be used to automate and optimize code. LLMs, which are trained on large datasets and have a natural knowledge and ability to operate with CFGs, can automate and simplify compiler design and language processing to a higher extent.

1.4 Large Language Model Advancement

The emergence and ongoing innovation of Artificial Intelligence (AI) in its different domains have dramatically changed the AI environment. A step towards something far more advanced than machine learning. This advancement utilized the enhanced models in healthcare, building and construction, and specific algorithm developments as well as natural language processing and many other domains [34,35,36]. Improvements on language processing and its application have had a tremendous effect [37]. The ability of these models to work with large diverse and extensive datasets during their training has empowered them with an unmatched capability to understand, generate, and interpret human language. As a result, they have kick-started major breakthroughs in NLP and the development of compilers. The benefits of LLMs spread to different sectors such as customer support with more intelligent and conversational bots, transforming education systems into personalized learning mediums and improving narrative writing with AI-based text converters generating high-quality, contextually relevant pieces.

LLMs have brought significant progress which built the foundation for such tasks as sentiment analysis, entity recognition, and language translation with an amazing accuracy level. The fact that these advancements not only demonstrate the models' ability to understand the details of human speech. At the same time, it also utilizes this knowledge in actual practical applications. Moreover, the integration of LLMs in the compiler construction brought to life an intelligent age of compiler tools that are not only smart but also quite efficient [38]. By using LLMs as the means of analyzing and optimizing programming languages, both researchers and developers have managed to automate various intricate tasks such as grammar improvement, code generation, and even the detection and correction of syntax mistakes. This is a significant milestone in compiler design which has resulted in compiler efficiency rise and therefore reduction in time and resources required for software development while also enhancing the flexibility and precision of compilers in handling different software tasks. Along with the rise of LLMs, the auto-suggestions of code and the error corrections have also been brought about in compiler designs. LLMs with deep learning capacity have become an asset for compilers to provide programmers with more intuitive assistance that makes corrections and optimizations to code that aligns with code quality standards. This synergy between LLMs and compiler tools is already not only speeding up the development process but is also making programming available not only for people with deep knowledge of computer science but also for those with no experience [39]. The further development of LLM is likely to provide NLP and compiler technologies a platform to research and implement AI in a more efficient manner. The extent to of LLMs to augment machine-language perception, supplemented by the integration of these tools in the automation and optimization of the computationally complex tasks, is a subtle manifestation of AI's transformative power. The emergence of LLMs which never ceases to evolve brings the dawn of a new age of innovations in technology, where a greater number of AI-driven tools and systems are able to be deployed. Those models are intelligent, adaptable, and useful in an increasing number of challenges across multiple domains.

2. RELATED WORK

The ubiquitous nature of XML in different domains is a further testimony to its great role in the modern computing era regarding data markup and representation. The design principles, including self-description, readability by humans, and versatility, have made XML the industry's standard for the interchange of data across operating systems, programming languages, and the broadest variety of computing devices. Moreover, the usability of XML is to a great extent circumscribed by libraries that provide universal accessibility and manipulation. This, therefore, lets XML documents be read and interpreted by any tool, thus reaffirming its importance as a foundational data exchange protocol.

Language Data Description (DFDL) is presented by [40], which further extends the capabilities of XML beyond the basic functionalities of XLink and XPointer. This expansion is crucial in that it is responsible for assembling and connecting data from heterogeneous raw-data stores by looking into the declarative nature of compilers in terms of the Combination Reduction Systems Extended (CRSX) for formal operational semantics and compiler optimization. These developments open the door to the concept of FDXML (For Data Only eXtensible Markup Language), which basically represents that any object being parsed can easily integrate into an XML pipeline thus increasing the utility of XML for parsing and labeling documents.

The landscape of formal languages has seen notable advancements, such as packrat parsing and Parsing Expression Grammar (PEG), paralleling BNF formalism but focusing on string recognition over construction. This shift towards more precise specifications of backtracking recursive-descent parsers addresses the longstanding challenges of direct and indirect left-recursion, enriching the parsing strategies available for complex language constructions.

Further exploration into ALL(*) parsing strategies by [41] marries the efficiency and predictability of traditional top-down LL(k) parsers with the robustness of GLR-like mechanisms. This hybrid approach, validated by the widespread use of ANTLR 4, demonstrates the feasibility of moving grammar analysis to parse time, thereby accommodating any non-left-recursive CFG. This flexibility is instrumental in parsing complex languages, such as Bangla, where conventional parsing methods are augmented with XML for dictionary searches and parse tree generation, as explored by [42].

Recent discussions also venture into the validity of HTML outputs from web applications, approximating outputs as context-free grammars. This generalized validation algorithm for context-free grammars, modeled after DTD languages, showcases the evolving methodologies in ensuring the syntactic and structural integrity of web-generated content.

In the domain of High-Level Synthesis (HLS) techniques, intelligent methodologies such as RDF rules, compiler-compilers, and XML validation emerge as cornerstones for correct-by-construction implementations. These methods are the distinct example of the combining of RDF rules, logic programming, and XML validation in order to meet the reliability and performance requirements of hardware compilers [43].

The development of Cetus' source-to-source compiler with XML capabilities demonstrates a key step in using XML to enrich compiler architectures. This method leverages the Intermediate Representation of Cetus to be a DOM tree in XML, which enhances the search of advanced code features using XPath expression, thereby showing one of the ways XML proves to be very useful in compiler optimization and code analysis [44]

The dispute on XML validation manifests as a range of conflicting philosophies, from strict grammar descriptions to the more permissive constraint languages such as Schematron. This dispute shows the changing nature of the XML validation techniques whereby the two demands of the strict adherence to schemas and the practical application domain are considered [45].

While several technological advancements have been witnessed in the area, using XML to enhance the capabilities of the LL(1) grammars and compiler construction is an appealing approach. In this research, we are trying to fill the void between classical compilers and the apparent possibilities of LLMs. The use of LLM technology for grammar optimization, the application of natural language processing (NLP) for code generation, and sophisticated parsing algorithms will be the milestones we aim for when considering transformative impacts for XML-based compiler tools. Apart from the fact that the use of LLMs with XML to create compilers will result in faster development and optimization, a new age of compiler technology will start, entailing higher efficiency, adaptability, and user-friendliness in compiler building.

3. METHODOLOGY

The incorporation of LLMs in the testing and conversion of CFGs shows a methodological framework that utilizes LLMs' skills for parsing and grammar revitalization. This methodology is centralized into two main steps, each one depicted in the following flowcharts, that are used for the conversion from left-recursive grammars to LL(1) grammars, implementing the specific features of LLMs.

3.1 Testing CFGs toward Left-Recursive and Converting to Non-Left-Recursive

The first phase utilizes LLMs in order to eradicate left recursion which is a must before grammar transformations into the LL(1) grammar can take place. This process involves the following steps, as depicted in Figures 1 and 2:

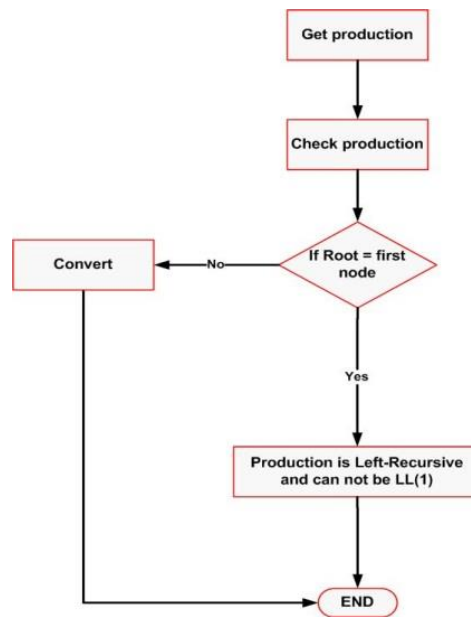


Fig. 1. Testing CFGs toward Left-Recursive

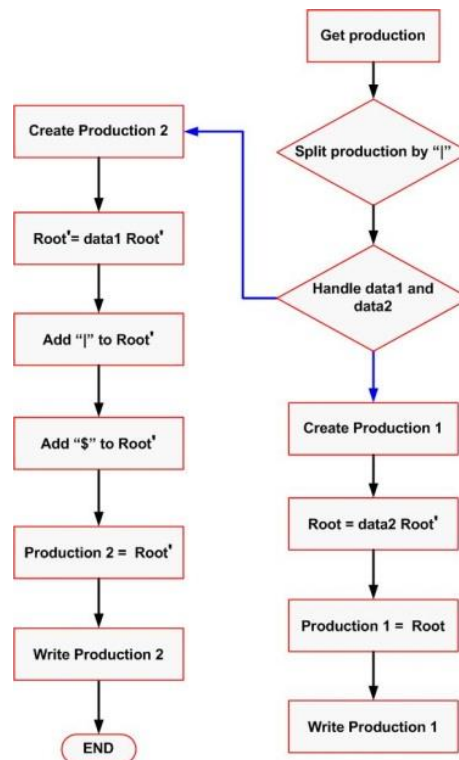


Fig. 2. Convert Left-Recursive CFG to Non Left-Recursive

1. **LLM-Assisted Detection of Left Recursion:** Leveraging the LLMs' advanced abilities to analyze CFGs for the recurring left-recursive patterns. This step ensures that LLM identifies the left recursive elements with a good accuracy level through their capability to comprehend and break down complex grammar structures.
2. **Automated Conversion to Non-Left-Recursive Grammars:** Then, after the left recursions have been detected, LLMs are automatically applied to rewrite the CFGs in a non-left-recursive form. This step entails the grammar prompting approach, in which LLMs construct new grammar structures that preserve the original grammar's semantics but are devoid of left recursion.

3.2 Testing CFG towards LL(1)

Upon converting CFGs to non-left-recursive formats, the methodology advances to testing and validating these grammars for compliance with LL(1) conditions. This involves two sub-stages, detailed in Figures 3 and 4:

1. **Generation of FIRST and FOLLOW Sets:** LLMs can be programmed so as to calculate FIRST and FOLLOW sets. the non-left-recursive CFGs. This step is critical because it forms the basis of constructing lexical parsing tables and many more. Enabled through the LLM’s computational prowess in terms of processing large data volumes and grammar rules efficiently.
2. **Construction of Predictive Parsing Tables:** Employing LLMs to construct predictive parsing tables based on the FIRST and FOLLOW sets. This stage leverages LLMs' pattern recognition and data structuring capabilities, automating the table construction process and ensuring accuracy in identifying valid grammar productions.

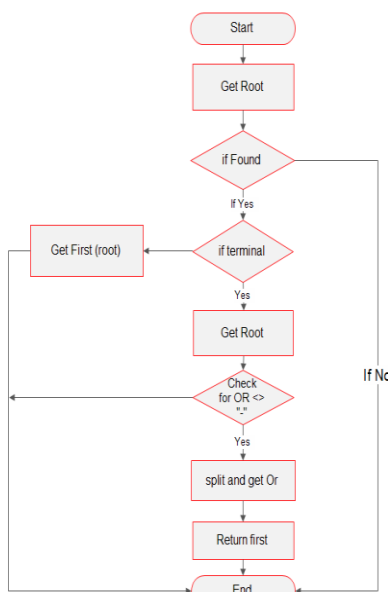


Fig. 3. Get First

Tow sub-stage was used for testing. The first step here is to get FIRST and FOLLOW sets as seen in the flowchart in Figure 3. Afterward, the sets which have been collected for testing CFG towards LL (1) grammar are used in constructing the predictive parsing table as presented in Figure 4.

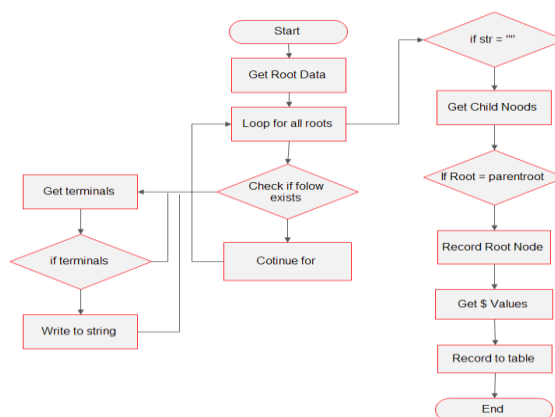


Fig. 4. Get Follow

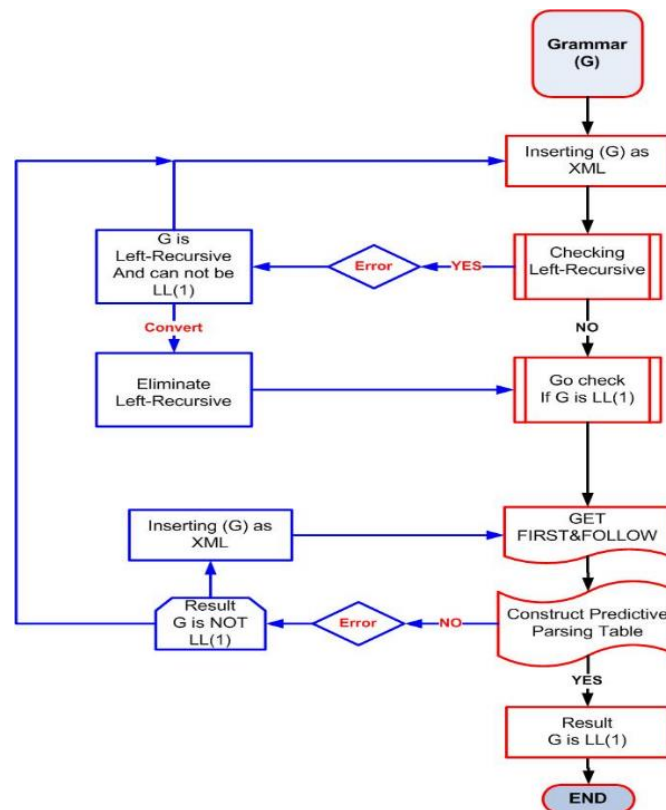


Fig. 5. Completed work plan in one flowchart

4. RESULTS AND DISCUSSION

The application of this methodology is demonstrated through various experiments, showcasing the effectiveness of LLMs in optimizing grammar testing and conversion processes. Experiment results, as seen in Figures 6 through 19, validate the potential of LLMs to automate and refine the development of compiler tools. Notably, the insertion of grammars into the application and the automated conversion of left-recursive CFGs to non-left-recursive, and ultimately to LL(1) grammars, illustrate the substantial improvements in efficiency and accuracy afforded by LLM integration.

The experiments underscore the LLMs' capability to not only detect and eliminate left recursion but also to generate FIRST and FOLLOW sets and construct predictive parsing tables with high precision. This demonstrates a significant advancement in compiler construction methodologies, paving the way for more intelligent, efficient, and automated compiler tools.

This article has proposed a novel approach for schema extraction from XML structures tailored to CFG grammars, specifically those of the LL(1) variety. The methodology reveals itself as an effective way of removing left-recursive grammars from the algorithm, while at the same time providing a simple means of grammar testing and validation, which in turn makes the algorithm LL(1) compliant. The presented schema is specially designed to easily convert LL(1) grammars to XML format, thus increasing parsing efficiency and simplifying syntax analysis during compiler construction. Hence, the schema supports the developer most efficiently.

In addition to the advanced functionalities of LLMs, this study also shows the benefits of incorporating LLM techniques, including grammar prompts, as well as model parallelism, into the XML-based schema of the CFGs. It is expected that this integration will raise grammar optimization automation to another level, create the required natural language processing for the purposes of code generation, introduce advanced parsing algorithms, and simplify the work of developers and optimizers of the compiler tools.

While the utility of the application depends on the user's knowledge of production rules and CFG restrictions, it is seen that the user's understanding of these concepts is a crucial factor. So this research work is primarily aimed at the construction of a parser for LL(1) grammar using ML and showing the potential of such a system. Future researches need to be directed towards the design of a complex global architecture that will be able to incorporate different CFGs. This AI extension will not only extend the utility of the application to different phases of compiler construction but also increase the flexibility of adaptation to complex grammar structures.

Intelligent software is a new trend that has blurred the borders of many domains, it highlights the role of advanced tools for building compilers. However, by broadening the scope to encompass the programming languages used in lexical analysis and other phases, another compiler toolchain application might be developed which could

promote the spread of the web-based application. This will lead to a widespread usage and improvement of these compiler technologies thereby capturing the advantages of artificial intelligence in the compiler’s environment. This will be a particularly strong step in the integration of artificial intelligence and traditional compiler methodology.

By utilizing the LLMs with the XML for the CFG grammar representation, this research has shown how there is a new paradigm for the development of compilers. The integration heralds a new era of compiler construction, characterized by enhanced efficiency, adaptability, and a deeper synergy between computational linguistics and compiler theory. As we look toward the future, it becomes increasingly clear that the fusion of LLM advancements and XML structures will play a pivotal role in overcoming traditional constraints and unlocking new possibilities in compiler development.

It can be seen from Figure 6 that the insertion of grammar into this application is possible. This can be achieved by inserting the production rules of the grammar by clicking on (1) **insert Production**, and afterward it is saved into the database. The grammar can be recalled and tested from the database by clicking on (2) **GetData**. Left-Recursive can be gotten rid of, and the grammar can be converted to non-Left recursive by clicking on (3) **Convert**. Lastly, when the user clicks on (4) **GO**, the grammar moves to test towards LL(1) which is performed in the second stage.



Fig. 6. WLeftRec.aspx Page

When **insert Production** which is shown in figure (6), is clicked, the page below in Figure (7) will run, thereby allowing the insertion of production. Initially, the roots must be inserted by addressing features (1 to 6), and afterward, the bodies should be inserted by addressing the features (7 to 12) as shown below. There is no limit to the number of grammars that can be inserted and tested by this application. Figure (7) shows how this page which contains several features can be used.

Fig. 7. WFInsertSchema.aspx Page

Following the description of how the application works, below is the testing of different grammars:
 Experiment (1): an example of left-recursive can be seen in the grammar below, and the results are presented in Figures (8,9, and 10):

- E → E+T | T
- T → T * F | F
- F → (E) | id

ID	Parent	Production
1	E	E,+T, ,T
2	T	T,*F, ,F
3	F	(E), ,id

Fig. 8. Result of Insert Productions

E---->E+T | T
 T---->T * F | F
 F---->(E) | id
 This CFG grammar is left-recursive and cannot be LL(1)

Fig. 9. Result of testing

E---->TE'
 E'---->+TE'\$
 T---->FT'
 T'---->*FT'\$
 F---->(E) | id

Fig. 10. Result of Conversion to non Left-recursive

Test (2): the grammar presented below is classified as an LL(1) type of grammar, and it is also non-left recursive. The result is shown in Figures (11,12,13,14, and 15):

- E → T E'
- E' → + T E' | ε
- T → F T'
- T' → * F T' | ε
- F → (E) | id

ID	Root	ORoot	Schema	MainRoot
1	E	-	1	<input checked="" type="checkbox"/>
2	E'	\$t	1	<input type="checkbox"/>
3	T	-	1	<input type="checkbox"/>
4	T'	\$t	1	<input type="checkbox"/>
5	F	idt	1	<input type="checkbox"/>

Fig. 11. Result of insert roots of grammar

ID	NodeName	ParentNode	RootNode	Terminal	OrNode
35	T	1	3	<input type="checkbox"/>	
36	E'	1	2	<input type="checkbox"/>	
37	+	2		<input checked="" type="checkbox"/>	
38	T	2	3	<input type="checkbox"/>	
39	E'	2	2	<input type="checkbox"/>	
40	F	3	5	<input type="checkbox"/>	
41	T'	3	4	<input type="checkbox"/>	
42	#	4		<input checked="" type="checkbox"/>	
43	F	4	5	<input type="checkbox"/>	
44	T'	4	3	<input type="checkbox"/>	
45	(5		<input checked="" type="checkbox"/>	
46	E	5	1	<input type="checkbox"/>	

Fig. 12. Result of inserting terminal and nonterminal grammar

As seen in Figure 18, the (Next) button which is circled with black color can be used to generate the grammar tree which is presented in Figure 19. There are several features possessed by the tree, and the details are presented below.

To get FIRST and FOLLOW sets and results for testing towards LL(1)

Return back to (Fig 17 and Fig 18) in case of try another inputs

Get First And Follow

Get XML DOC

Open Table

Back

Transfers and save the production rules as XML document(.xml)

To Building predictive parsing table

Fig. 13. Tree of Grammar

```

First of E:(,id
First of E":+,S
First of T:(,id
First of T":#,S
First of F:(,id
follow of E = ),S
follow of E" = ),S
follow of T = +,),S
follow of T" = +,),S
follow of F = #,+,),S
These Production Rules is LL(1)
    
```

Fig. 14. FIRST and FOLLOW sets of grammar with result of the test

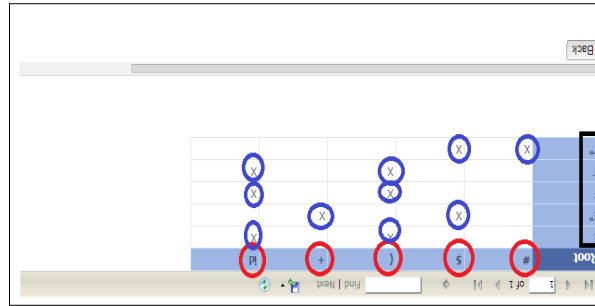


Fig. 15. Predictive parsing table of the grammar

The red-colored circles shown in Figure 15 are the terminals, whereas the nonterminal are denoted by the black rectangle, the blue-colored circles represent the intersection between terminal and nonterminal based on the grammar’s production rules.

Test (3):it can be observed that the grammar below cannot be classified as LL(1) because it is not; the results are presented in Figures (16,17,18, and 19). The red-colored circles show that the new sequence (2) has been allocated to the grammar contained in Figure (22) below.

- S → i E t S S' | a
- S' → e S | ε
- E → B

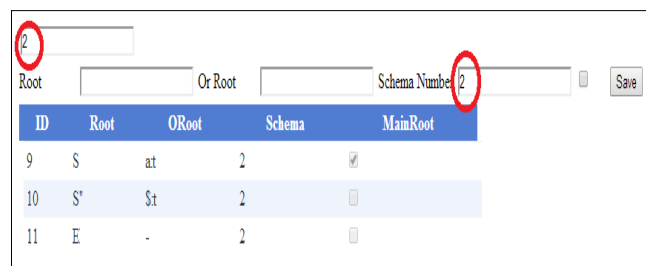


Fig. 16. Result of insertion of grammar roots

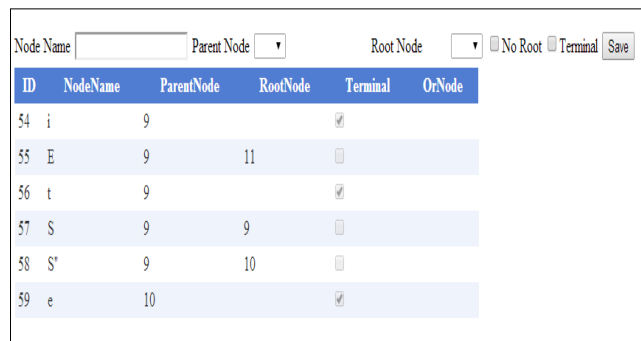


Fig. 17. Insertion of other terminal and nonterminal grammar (3.2)

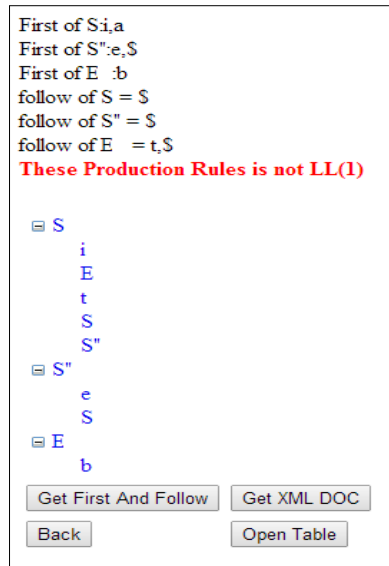


Fig. 18. FIRST and FOLLOW sets of grammar with results of test and tree

Root	\$	a	b	e	i
E			X		
S		X			X
S''	X			X	

Fig. 19. Predictive parsing table of grammar

5. CONCLUSION

In this article, we introduced a novel approach to schema extraction from XML structures specifically tailored for Context-Free Grammars (CFGs) that adhere to the LL(1) classification, propelling forward the efficiency and adaptability of compiler construction. The integration of Large Language Models (LLMs) with XML-based compiler tools has emerged as a pivotal development, automating and optimizing grammar analysis, thereby streamlining the parsing process and semantic analysis. This synergy between LLMs and XML not only simplifies the representation of LL(1) grammars, making parsing both easy and rapid, but also enhances the compiler's internal syntax analysis phase. We have shown that such integration has a significant effect on the compiler's speed, and this is still true even if we include the complexity of inserting various grammars. Nonetheless, understanding production rules and CFG constraints by the users is the key to unlocking the true value of this application.

With a look to the future, I see the development of LLMs' abilities, particularly their success in the area of programming language understanding and generation, as a significant factor in the advancement of compiler technology. The possibility of exploiting higher-end LLM methods for grammar optimization, error detection, and even the computerization of compiler tool development contributes to the breadth of the research and development. By applying the use of LLM integration for more programming paradigms and languages, we would like to increase the scope and impact of compiler tools meaningfully.

The findings of this study represent a giant leap toward the development of more intelligent, flexible and helpful compiler tools. The development of an XML schema for LL(1) grammar representation in combination with the modern features of LLMs marks the advent of a major breakthrough in compiler technology. By this trailblazing implementation, not only the existing problems of compiler design but also the groundwork for future innovations are being laid down. With this in

view, the machine learning technology is poised to inspire the development of global schemas capable of handling the whole range of CFGs, thereby ushering the era of flexibility in compiler construction. Free and open-source software can be used to further extend their reach into different areas by using an integrated approach; this could eventually lead to the development of web-based applications that will be able to handle a variety of grammars, such as lexical analysis, employed by various compiler stages.

This is the essence of the fact that the collaboration of LLMs with XLM-based compiler builders reflects the transformative character of the compiler development field. The pioneering approach to grammar analysis and optimization of the compiler is what heralds the advent of a new era in the evolution of compiler technology, which sets the stage for a future when compilers will be more efficient, intelligent, and accessible to a wider audience of developers and educators.

Conflicts of Interest

The authors declare no conflicts of interest.

Funding

No funding is provided and no financial support is received to carry out the research presented in this paper.

Acknowledgment

The author would like to express gratitude to the institution for their invaluable support throughout this research project.

References

- [1] M. Měchura, “Better than XML: Towards a lexicographic markup language,” *Data Knowl. Eng.*, vol. 146, p. 102196, Jul. 2023, doi: 10.1016/J.DATAK.2023.102196.
- [2] E. Flondor, “Exploring AndroidManifest.xml for Automated Android Apps Classification,” *Proc. - 2023 IEEE Int. Conf. Big Data, BigData 2023*, pp. 6145–6147, 2023, doi: 10.1109/BIGDATA59044.2023.10386962.
- [3] J. Zhang, X. Qiao, and B. Lin, “VTD-XML Parsing Performance Optimization based on Helper Thread Sampling Prefetching,” *Int. Conf. Ubiquitous Futur. Networks, ICUFN*, vol. 2023-July, pp. 740–745, 2023, doi: 10.1109/ICUFN57995.2023.10200816.
- [4] E. Nuyts, M. Bonduel, and R. Verstraeten, “Comparative analysis of approaches for automated compliance checking of construction data,” *Adv. Eng. Informatics*, vol. 60, p. 102443, Apr. 2024, doi: 10.1016/J.AEI.2024.102443.
- [5] P. Singh and S. Sachdeva, “A Landscape of XML Data from Analytics Perspective,” *Procedia Comput. Sci.*, vol. 173, pp. 392–402, Jan. 2020, doi: 10.1016/J.PROCS.2020.06.046.
- [6] J. Chen *et al.*, “A Survey of Compiler Testing,” *ACM Comput. Surv.*, vol. 53, no. 1, Feb. 2020, doi: 10.1145/3363562.
- [7] O. S. Albahri *et al.*, “Helping doctors hasten COVID-19 treatment: Towards a rescue framework for the transfusion of best convalescent plasma to the most critical patients based on biological requirements via ml and novel MCDM methods,” *Comput. Methods Programs Biomed.*, vol. 196, p. 105617, Nov. 2020, doi: 10.1016/J.CMPB.2020.105617.
- [8] I. A. Zahid and S. S. Joudar, “Does Lack of Knowledge and Hardship of Information Access Signify Powerful AI? A Large Language Model Perspective,” *Appl. Data Sci. Anal.*, vol. 2023, pp. 150–154, Dec. 2023, doi: 10.58496/ADSA/2023/014.
- [9] Q. Dong, X. Chen, and M. Satyanarayanan, “Creating Edge AI from Cloud-based LLMs,” *Proc. 25th Int. Work. Mob. Comput. Syst. Appl.*, pp. 8–13, Feb. 2024, doi: 10.1145/3638550.3641126.
- [10] H. Leather and C. Cummins, “Machine Learning in Compilers: Past, Present and Future,” *Forum Specif. Des. Lang.*, vol. 2020-September, Sep. 2020, doi: 10.1109/FDL50818.2020.9232934.
- [11] M. Li *et al.*, “The Deep Learning Compiler: A Comprehensive Survey,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 3, pp. 708–727, Mar. 2021, doi: 10.1109/TPDS.2020.3030548.
- [12] R. Albertoni, D. Browning, S. Cox, A. N. Gonzalez-Beltran, A. Perego, and P. Winstanley, “The W3C Data Catalog Vocabulary, Version 2: Rationale, Design Principles, and Uptake,” *Data Intell.*, pp. 1–37, Dec. 2023, doi: 10.1162/DINT_A_00241.
- [13] M. Jodłowiec, M. Krótkiewicz, and P. Zabawa, “The analysis of data metamodels’ extensional layer via extended generalized graph,” *Appl. Intell.*, vol. 53, no. 8, pp. 8510–8535, Apr. 2023, doi: 10.1007/S10489-022-04440-

- 0/TABLES/16.
- [14] D. Van Assche *et al.*, “Leveraging Web of Things W3C Recommendations for Knowledge Graphs Generation,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 12706 LNCS, pp. 337–352, 2021, doi: 10.1007/978-3-030-74296-6_26/COVER.
- [15] P. Späth and J. Friesen, “Working with XML and JSON Documents,” *Learn Java Android Dev.*, pp. 641–697, 2020, doi: 10.1007/978-1-4842-5943-6_16.
- [16] Z. Brahmia, H. Hamrouni, and R. Bouaziz, “TempoX: A disciplined approach for data management in multi-temporal and multi-schema-version XML databases,” *J. King Saud Univ. - Comput. Inf. Sci.*, vol. 34, no. 1, pp. 1472–1488, Jan. 2022, doi: 10.1016/J.JKSUCI.2019.08.009.
- [17] X. Zuo, H. Li, and L. Zhou, “Design and Research of Remote Sensing Process Language based on XML,” *Proc. 2021 IEEE 2nd Int. Conf. Inf. Technol. Big Data Artif. Intell. ICIBA 2021*, pp. 1048–1053, 2021, doi: 10.1109/ICIBA52610.2021.9687912.
- [18] Z. Brahmia, H. Hamrouni, and R. Bouaziz, “XML data manipulation in conventional and temporal XML databases: A survey,” *Comput. Sci. Rev.*, vol. 36, p. 100231, May 2020, doi: 10.1016/J.COSREV.2020.100231.
- [19] T. Kasampalis, D. Park, Z. Lin, V. S. Adve, and G. Rosu, “Language-parametric compiler validation with application to LLVM,” *Int. Conf. Archit. Support Program. Lang. Oper. Syst. - ASPLOS*, pp. 1004–1019, Apr. 2021, doi: 10.1145/3445814.3446751.
- [20] H. Xu, S. Fan, Y. Wang, Z. Huang, H. Xu, and P. Xie, “Tree2tree Structural Language Modeling for Compiler Fuzzing,” *Lect. Notes Comput. Sci.*, vol. 12452 LNCS, pp. 563–578, 2020, doi: 10.1007/978-3-030-60245-1_38/COVER.
- [21] H. Jiang, Z. Zhou, Z. Ren, J. Zhang, and X. Li, “CTOS: Compiler Testing for Optimization Sequences of LLVM,” *IEEE Trans. Softw. Eng.*, vol. 48, no. 7, pp. 2339–2358, Jul. 2022, doi: 10.1109/TSE.2021.3058671.
- [22] S. A. Hussein and I. A. Zahid, “Improved Naked Mole-Rat Algorithm Based on Variable Neighborhood Search for the N-Queens Problem,” *Iraqi J. Sci.*, vol. 65, no. 1, pp. 528–545, Jan. 2024, doi: 10.24996/IJS.2024.65.1.41.
- [23] J. L. Hoover, M. Sonderegger, S. T. Piantadosi, and T. J. O’donnell, “The Plausibility of Sampling as an Algorithmic Theory of Sentence Processing,” *Open Mind*, vol. 7, pp. 350–391, Jul. 2023, doi: 10.1162/OPMI_A_00086/116522/THE-PLAUSIBILITY-OF-SAMPLING-AS-AN-ALGORITHMIC.
- [24] P. A. Martínez, G. Bernabé, and J. M. García, “Code Detection for Hardware Acceleration Using Large Language Models,” *IEEE Access*, vol. 12, pp. 35271–35281, 2024, doi: 10.1109/ACCESS.2024.3372853.
- [25] N. Chomsky, “Three factors in language design,” *Linguist. Inq.*, vol. 36, no. 1, pp. 1–22, Dec. 2005, doi: 10.1162/0024389052993655.
- [26] N. Chomsky, “On certain formal properties of grammars,” *Inf. Control*, vol. 2, no. 2, pp. 137–167, Jun. 1959, doi: 10.1016/S0019-9958(59)90362-6.
- [27] N. Chomsky, “Rules and representations,” *Behav. Brain Sci.*, vol. 3, no. 1, pp. 1–15, 1980, doi: 10.1017/S0140525X00001515.
- [28] K. M. A. Hasan, Al-Mahmud, A. Mondal, and A. Saha, “Recognizing Bangla Grammar using Predictive Parser,” *Int. J. Comput. Sci. Inf. Technol.*, vol. 3, no. 6, pp. 61–73, Jan. 2012, doi: 10.5121/ijcsit.2011.3605.
- [29] M. Rohrmeier, Q. Fu, and Z. Dienes, “Implicit Learning of Recursive Context-Free Grammars,” *PLoS One*, vol. 7, no. 10, p. e45885, Oct. 2012, doi: 10.1371/JOURNAL.PONE.0045885.
- [30] N. A. Zafar, S. A. Khan, F. Alhumaidan, and B. Kamran, “Formal Modeling towards the Context Free Grammar,” *Life Sci. J.*, vol. 9, no. 4, 2012.
- [31] S. Tharun Reddy, R. Mothe, S. Ghate, and G. Sunil, “Evaluation and scrutiny of XML files, XML empowered records and native XML records,” *IOP Conf. Ser. Mater. Sci. Eng.*, vol. 981, no. 2, p. 022074, Dec. 2020, doi: 10.1088/1757-899X/981/2/022074.
- [32] C. M. Sperberg-McQueen, “Balisage Paper: An XML infrastructure for spell checking with custom dictionaries,” *Balisage Ser. Markup Technol.*, vol. 25, 2020, doi: 10.4242/BALISAGEVOL25.SPERBERG-MCQUEEN01.
- [33] I. Dongo, R. Ticona-Herrera, Y. Cadinale, and R. Guzmán, “Semantic similarity of XML documents based on structural and content analysis,” *ACM Int. Conf. Proceeding Ser.*, Nov. 2020, doi: 10.1145/3440084.3441185.
- [34] A. S. Albahri *et al.*, “Explainable Artificial Intelligence Multimodal of Autism Triage Levels Using Fuzzy Approach-Based Multi-criteria Decision-Making and LIME,” *Int. J. Fuzzy Syst.*, pp. 1–30, Nov. 2023, doi: 10.1007/s40815-023-01597-9.
- [35] A. H. Alamoodi *et al.*, “Sentiment analysis and its applications in fighting COVID-19 and infectious diseases: A systematic review,” *Expert Syst. Appl.*, vol. 167, p. 114155, Apr. 2021, doi: 10.1016/J.ESWA.2020.114155.
- [36] H. H. M. Al-Ghabawi, M. M. Khatlab, I. A. Zahid, and B. Al-Oubaidi, “The prediction of the ultimate base shear of BRB frames under push-over using ensemble methods and artificial neural networks,” *Asian J. Civ. Eng.*, vol. 25, no. 2, pp. 1467–1485, 2024, doi: 10.1007/s42107-023-00855-3.

- [37] N. Muennighoff *et al.*, “Crosslingual Generalization through Multitask Finetuning,” Nov. 2022, Accessed: Jul. 22, 2023. [Online]. Available: <https://arxiv.org/abs/2211.01786v2>.
- [38] P. Deligiannis, A. Lal, N. Mehrotra, and A. Rastogi, “Fixing Rust Compilation Errors using LLMs,” *Proc. ACM Conf.*, vol. 1, Aug. 2023.
- [39] S. Yin *et al.*, “A Survey on Multimodal Large Language Models,” Jun. 2023, Accessed: Dec. 20, 2023. [Online]. Available: <http://arxiv.org/abs/2306.13549>.
- [40] A. Mumford and D. Wijesekera, “Toward the Discovery and Extraction of Money Laundering Evidence from Arbitrary Data Formats using Combinatory Reductions,” *STIDS*, pp. 32–39, 2014.
- [41] ParrTerence, HarwellSam, and FisherKathleen, “Adaptive LL(*) parsing: the power of dynamic analysis,” *ACM SIGPLAN Not.*, vol. 49, no. 10, pp. 579–598, Oct. 2014, doi: 10.1145/2714064.2660202.
- [42] S. F. Shetu, M. Saifuzzaman, M. Parvin, N. N. Moon, R. Yousuf, and S. Sultana, “Identifying the Writing Style of Bangla Language Using Natural Language Processing,” *2020 11th Int. Conf. Comput. Commun. Netw. Technol. ICCCNT 2020*, Jul. 2020, doi: 10.1109/ICCCNT49239.2020.9225670.
- [43] C. M. Medeiros, M. A. Musicante, and U. S. Costa, “LL-based query answering over RDF databases,” *J. Comput. Lang.*, vol. 51, pp. 75–87, Apr. 2019, doi: 10.1016/J.COLA.2019.02.002.
- [44] M. A. Exusia, “A Novel Approach to Version XML Data Warehouse,” *Artic. Int. J. Comput. Sci. Eng.*, 2021, doi: 10.14445/23488387/IJCSE-V8I9P102.
- [45] A. Koren, M. Jurcevic, and R. Prasad, “Semantic Constraints Specification and Schematron-Based Validation for Internet of Medical Things- Data,” *IEEE Access*, vol. 10, pp. 65658–65670, 2022, doi: 10.1109/ACCESS.2022.3182486.