Deep Science Publishing
https://doi.org/10.70593/978-93-49910-92-8_3

Chapter 3: Context free grammar and its application in natural language processing

Ashish Kumar, Sudhanshu Shekhar Dubey

Abstract: The Context-Free Grammar (CFG) is a fundamental concept in computational linguistics, particularly within the realm of Natural Language Processing (NLP). This paper examines how CFG is utilized in essential NLP domains such as question answering, machine translation, speech recognition, and dialogue systems, including chatbots. By employing recursive rules that define valid sentence structures, CFG enables computers to interpret and analyze natural language more effectively. For instance, in question answering, it assists in mapping user queries to structured formats. Regarding chatbots, CFG contributes to identifying user intent and managing conversations. While CFG excels in rule-based systems, it does have limitations concerning ambiguity and context, which are prevalent in natural language. To mitigate this, modern systems often integrate CFG with statistical or neural methods. This paper underscores how CFG remains relevant in these hybrid NLP models.

Keywords: Context Free Grammar, Context Free Language, Grammar, Natural Language Processing

Ashish Kumar
Department of Mathematics, Chandigarh University, Punjab, India.

Sudhanshu Shekhar Dubey
Department of Mathematics, Chandigarh University, Punjab, India.

1 Introduction

Language is one of the most fascinating and intricate aspects of human intelligence. For centuries, scholars have sought to understand and formalize the structure of language, searching for rules that dictate how words combine into meaningful sentences. (Jurafsky, D., & Martin, J. H., 2021). A significant breakthrough occurred in the 1950s when linguist Noam Chomsky introduced Context-Free Grammar (CFG) — a formal system capable of capturing the syntactic structure of natural languages in a mathematically precise manner. Originally developed as a tool to describe the syntax of both natural and programming languages, CFGs quickly established themselves at the core of computational linguistics. Chomsky's work not only revolutionized theoretical linguistics but also laid the foundation for many of the natural language processing (NLP) techniques we utilize today. From parsing sentences to translating languages, and even powering voice assistants, CFG has quietly become a fundamental pillar in how machines comprehend human language. (Harabagiu, S., Pasca, M., & Maiorano, S., 2001)

1.1 Grammar: It is a set of rules which checks whether a string belongs to a particular language or not. A program consists of various strings of characters. But every string is not a proper or meaningful string. So, to identify valid strings in a language, some rules should be specified to check whether the string is valid or not. These rules are nothing but make Grammar. Grammar is basically composed of two basic elements: (Chomsky, N., 1956)

- Terminal Symbols: Terminal symbols are those that are the components of the sentences generated using grammar and are represented using small case letters like a, b, c, etc.
- Non-Terminal Symbols: Non-terminal symbols are those symbols that take part in the generation of the sentence but are not the component of the sentence. Non-Terminal Symbols are also called Auxiliary Symbols and Variables. These symbols are represented using a capital letter like A, B, C, etc.

Representation of Grammar:

Any Grammar can be represented by 4 tuples – <N, T, P, S>

- N – Finite Non-Empty Set of Non-Terminal Symbols.
- T – Finite Set of Terminal Symbols.
- P – Finite Non-Empty Set of Production Rules.

- S – Start Symbol (Symbol from where we start producing our sentences or strings).

In this we will discuss about the context free grammar which is type-2 grammar and which is subset of type-0 and type-1 grammar.

1.2 Context Free Grammar (CFG): It is a notation used to specify the syntax of the language. Context-free Grammar is used to design parsers. Lexical Analyzer generates a string of tokens which are given to the parser to construct a parse tree. But, before constructing the parse tree, these tokens will be grouped so that the results of grouping will be a valid construct of a language. (Harabagiu, S., Pasca, M., & Maiorano, S. 2001) So, to specify constructs of language, a suitable notation is used, which will be precise & easy to understand.

Or

CFG is a formal grammar which is used to generate all possible patterns in a given finite language. Context Free Grammar is defined as a 4 tuple G = (V, Σ, R, S) where:

1. V is a finite set of elements known as variables.
2. Σ is a finite set of elements known as terminals
3. V ∩ Σ = Null (empty set)
4. S is an element of V and is known as the start variable.
5. R is a fine set of elements known as rule. Each rule is like A -> w where A belongs to V and w belongs to union of (V and Σ)*. The * denotes repetition of elements.

2 Basic terms used in CFG:

2.1 Terminals: These are the characters that make up the actual content of the final sentence. These can include words or letters depending on which of these is used as the basic building block of a sentence. Non-Terminals: These are also called variables. These act as a sub language within the language defined by the grammar. Non terminals are placeholders for the terminals. We can use non terminals to generate different patterns of terminal symbols.

Start Symbol: a start symbol is a special non terminal that represents the initial string that will be generated by the grammar.

2.2 CFL: CFL stands for Context free language, and CFL is a language which is generated by CFG

Production Rules: It is also known as rewrite rule, defines how non-terminals can be replaced by other symbols to generate strings in language defined by a formal grammar (provides a structured way to define the syntax or structure of a language using rules and symbols).

2.3 Simplification of CFG: Simplification of CFG means production of Grammar by removing useless symbols, null and unit production. The property of reduced grammar is:

1. Each variable(non-terminal) and each terminal of Grammar G appears in the derivation of L(language)
2. There should not be any production as X $\rightarrow Y$, where X $\rightarrow \epsilon$
3. If ϵ is not in the language L, then their need not to be production of the type X $\rightarrow \epsilon$

Reduced Grammar means

- Removal of useless symbol
- Removal of null production
- Removal of unit production

2.3.1 Removal of useless symbol: A symbol is said to be useless if it doesn't appear on RHS of the grammar and doesn't take part in the derivation of any string. (Jelinek, F. 1997) Steps to follow:

1. Remove non-generating symbols
2. Remove unreachable symbols.

2.3.2 Removal of null production: The production of the type $X \rightarrow \in$ are called null or ∈ - production. To remove null production, follow following steps:

1. Find out all nullable non-terminal that derives ∈
2. For each production of the type A $\rightarrow a$ construct all productions A $\rightarrow X$ where X is obtained from a by removing one or more non-terminal.

3. Combine the result of 2 with the original production and remove null production.

2.3.3 Removal of unit production: Unit production means one non-terminal generating another non-terminal. To remove unit production, follow following steps:

1. To remove $X \to Y$ add productions as $X \to a$ to the grammar rule, whenever Y → a occurs in grammar.

2. Now delete $X \to Y$ from the grammar

3. Repeat step 1 and 2 until all unit productions are removed

2.4 Forms of Context Free Grammar:

Table1.1 Forms of CFG

2.4.1. Chomsky Normal Form (CNF)	**2.4.2 Greibach Normal Form (GNF)**
Definition: Every production is of the form: A → BC (two non-terminals) A → a (a terminal) or S → ε (only if ε is in the language and S is the start symbol only)	Definition: Every production is of the form: A → aBC…. (a terminal followed by any number of non-terminals) A → a (a terminal) or S → ε (only if ε is in the language and S is the start symbol only)
Example: S → AB S → c S → a	Example: S → ABC S → cB S → a

2.4.1 Conversion of CFG to CNF:

Step 0: Ensure Start Symbol Doesn't Appear on RHS

If the start symbol S appears on the right-hand side of any production, introduce a new start symbol:

- Add: $S_0 \rightarrow S$

Step 1: Remove ε-productions:

Step 2: Remove unit Production

Step 3: Remove useless symbol

Step 4: Convert Productions to CNF format by Eliminating terminals symbol from the RHS if they exist with other non-terminal or terminal and Eliminating RHS with more than two terminals.

Example: $S \rightarrow a|aA|B$

$A \rightarrow aBB|\epsilon$

$B \rightarrow Aa|b$

2.4.2 Conversion of CFG to GNF:

Step 1. If the given grammar is not in CNF, convert it to CNF. You can refer following article to convert CFG to CNF: Converting Context Free Grammar to Chomsky Normal Form.

Step 2. Change the names of non-terminal symbols to A1 till AN in same sequence.

Step 3. Check for every production rule if RHS has first symbol as non-terminal say Aj for the production of Ai, it is mandatory that i should be less than j. Not great and not even equal.

If i> j then replaces the production rule of Aj at its place in Ai.

If i=j, it is the left recursion. Create a new state Z which has the symbols of the left recursive production, once followed by Z and once without Z, and change that production rule by removing that particular production and adding all other production once followed by Z.

Step 4. Replace very first non-terminal symbol in any production rule with its production until production rule satisfies the above conditions.

For converting a CNF to GNF always move left to right for renaming the variables.

2.5 CYK Algorithm: CYK stands for Cocke–Younger–Kasami. It is a parsing algorithm for Context-Free Grammars (CFGs) in Chomsky Normal Form

(CNF). This algorithm is used to determine whether a given string belongs to the language L generated by a CFG G. To convert CFG to CNF. For a string of length N, construct a table T of size N x N. Each cell in the table T[i, j] contains the set of all constituents that can produce the substring spanning from position i to j. The process involves filling the table with the solutions to the subproblems encountered in the bottom-up parsing process. Therefore, cells will be filled from left to right and bottom to top. In T[i, j], the row number i denotes the start index, while the column number j denotes the end index. A ∈ T[i,j] iff B ∈ T[i,k], C ∈ T[k,j], and A → BC is a rule of G. The algorithm considers every possible subsequence of letters and adds K to T[i, j] if the sequence of letters from i to j can be generated from the non-terminal K. For subsequences of length 2 and greater, it examines every possible partition of the subsequence into two parts and checks if there is a rule of the form A → BC in the grammar, where B and C can generate the two parts respectively, based on already existing entries in T. The sentence can be produced by the grammar only if the entire string is matched by the start symbol, i.e., if S is a member of T[1, n].

2.6. **Derivation in CFG:** Derivation is a sequence of production rules. It is used to get the input string through the given production rules during parsing, we have to take two decisions

1. Which non-terminal would be replaced
2. By which production rule our selected non-terminal would be replaced.

Derivation are of two types:

1. Left most Derivation: In this input is scanned and replaced by the production rules from left to right.
2. Right most Derivation: In this input is scanned and replaced by the production rules from right to left.

2.6.1 Derivation tree in CFG: Derivation tree is a graphical representation for the derivation of given production rules for a CFG. It has the following properties:

1. The start symbols are always indicated by the root node.
2. The leaf node will be the terminal node.
3. The derivation is read from left to right.

The interior nodes are always non-terminal nodes

2.7 Ambiguity in Context Free Grammar: A context free grammar is called ambiguous if there exists more than one LMD or more than one RMD for a string which is generated by grammar. There will also be more than one derivation tree for a string in ambiguous grammar. If the grammar has Ambiguity, then it will not look for compiler construction.

2.8 Pumping Lemma for CFG: Pumping lemma is used to show that a language is not a context free.

Statement: Let L be a context free language and n be an integer constant, select a string z from L such that $z \geq n$, now divide the string z into 5 parts i.e. uvwxy such that $|vwx| \leq n$ & $|vx| \geq 1$,

$for\ i \geq 0,\ uv^iwx^iy$ is in L

2.9 NLP: Natural Language Processing (NLP) is a field of artificial intelligence dedicated to enabling computers to understand and interpret human language. It encompasses various tasks such as sentiment analysis, speech recognition, and language translation. Parsing is a crucial aspect of NLP that aids in grasping the grammatical structure of sentences. NLP signifies a computer system's capability to comprehend and interpret human language meaningfully. It involves analysing text to extract pertinent information and derive insights. NLP techniques are extensively utilized in applications like chatbots, recommendation systems, and language translation. (Moldovan, D. I., & Mihalcea, R. 2003)

3 CFG in NLP:

3.1 Syntax Parsing: The most direct application of CFG is in syntax parsing, where a sentence is analyzed to determine its grammatical structure. A parse tree, also known as a syntax tree, is generated to illustrate how a sentence adheres to grammatical rules. This tree representation shows how a sentence is constructed using the production rules of a grammar. Each node in the tree signifies a grammatical category (non-terminal), while each leaf node represents a terminal (word). (Young et. al. 2006)

- Constituency Parsing: CFG serves as the foundation for constituency parsers, which decompose sentences into sub-phrases such as noun phrases (NP) and verb phrases (VP).
- Ambiguity Resolution: Although CFG cannot resolve ambiguity on its own, it aids in identifying multiple potential parses for ambiguous

sentences, which can then be further clarified using probabilities (e.g., in probabilistic CFGs).

3.1.1 Importance of CFG in Parsing: CFG outlines the rules for forming valid sentences. By applying these rules, a parser can determine whether a sentence is grammatically correct and, if so, how it can be deconstructed into its components. This process is essential in:

- Grammar checking: Identifying syntactic errors in writing.
- Language understanding: Analysing sentence structure to extract meaning.
- Text-to-speech: Establishing pauses and intonation based on grammatical structure.

There are two primary approaches to syntax parsing using CFG: (Wang, Y., & Acero, A., 2006)

(a)Top-Down Parsing:
• Begins with the start symbol (S) and attempts to rewrite it to correspond with the input sentence.
• Aims to construct the parse tree from the root to the leaves.
• Example algorithm: Recursive Descent Parser

(b) Bottom-Up Parsing:
• Starts with the input words (terminals) and works up to the start symbol.
• Seeks to reduce the sentence to the root (S) by applying grammar rules in reverse.
• Example algorithm: Shift-Reduce Parser

3.1.2 Structural Ambiguity in CFG Parsing: One challenge in syntax parsing is ambiguity, where a sentence can be interpreted in multiple ways. This phenomenon is known as structural ambiguity.
Example:
Sentence: "I saw the man with the telescope."
Possible interpretations:

1. I used a telescope to see the man.
2. I saw a man who had a telescope.

A CFG parser may produce multiple parse trees, each representing a valid grammatical interpretation. Resolving this ambiguity often requires additional information, such as context or probabilities, leading to Probabilistic CFGs (PCFGs).

Practical Use Cases of CFG-Based Parsing: (McTear, M., 2020)

- **Question Answering Systems:** Identify sentence structure to comprehend what is being asked.
- **Machine Translation:** Understand the syntactic structure of a source language to generate grammatically correct output in a target language.
- **Voice Assistants:** Parse spoken commands to extract meaning and perform actions.
- **Linguistic Analysis Tools:** Assist linguists in analysing sentence patterns, phrase structures, and syntactic phenomena

3.2 Machine Translation: Machine Translation (MT) is one of the most significant applications of Natural Language Processing (NLP), where text is automatically translated from one language (source) to another (target). While modern MT systems frequently utilize neural networks and large language models, Context-Free Grammar (CFG) has historically played — and continues to play in many systems — a foundational role in rule-based and hybrid translation systems. CFG facilitates the syntactic analysis of sentences, which helps ensure the grammatical structure of the output in the target language. (Radziwill, N. M., & Benton, M. C., 2017)

3.2.1 Role of CFG in Machine Translation: CFG is utilized to analyze and transform the grammatical structure of the source language text prior to generating a syntactically correct sentence in the target language. This is particularly crucial for preserving syntactic coherence and grammatical accuracy, which are vital for producing high-quality translations. (Bohus, D., & Rudnicky, A. I., 2009)The general process includes:

1. Parsing the source sentence with CFG to identify its syntactic structure.
2. Mapping the parse tree or structure to a corresponding format in the target language.
3. Generating the translated sentence using the grammar rules of the target language.
   This method is predominantly employed in Rule-Based Machine Translation (RBMT) and Example-Based Machine Translation (EBMT), as well as in certain hybrid systems that integrate rule-based and statistical techniques.

3.2 Information Extraction: Information Extraction (IE) is the process of automatically identifying and extracting structured information from unstructured or semi-structured text data. The goal is to convert raw text into meaningful data that can be utilized for tasks such as question answering, knowledge base construction, or decision-making. Common types of information extracted include named entities, relationships between entities, events, and facts. In NLP, Context-Free Grammar (CFG) contributes to IE by providing a syntactic framework that aids in identifying patterns, relationships, and linguistic structures in text, thereby enhancing the accuracy and structure of relevant information extraction.

3.3 The Role of CFG in Information Extraction: CFG enables the system to analyze sentence structure and detect specific syntactic patterns where useful information is likely to appear. By applying CFG rules, an IE system can:

- Identify noun phrases (NP) and verb phrases (VP) where entities and relationships typically reside. (Kumar, S., & Kour, G., 2025)
- Recognize clausal structures that indicate actions or events.
- Match patterns

3.3.1 Pattern-Based Extraction with CFG: In many real-world IE systems, grammars are used to define extraction patterns. These patterns are often based on linguistic observations and can be formalized using CFG-style rules. (Kumar, S., 2024 and Sharma, S., Rana, S., & Dubey, S. S. 2024)

3.4 Speech Recognition: Speech Recognition, also referred to as Automatic Speech Recognition (ASR), is the process of converting spoken language into written text. It serves as a foundational technology for voice assistants (like Siri or Alexa), transcription services, voice-controlled systems, and more. The objective of ASR is to accurately identify and transcribe the spoken input into a readable and meaningful sequence of words. A successful speech recognition system involves multiple stages, including:

- Acoustic modeling – interpreting audio signals
- Language modeling – predicting likely word sequences
- Decoding – converting sound patterns into text

In the language modeling and decoding stages, Context-Free Grammar (CFG) plays a crucial role, especially in grammar-based ASR systems, which are used in constrained environments like voice menus, command-and-control systems, and embedded applications.

3.5 Role of CFG in Speech Recognition: In speech recognition, CFGs are used to define the set of allowable phrases or sentence structures that a user might say. This dramatically improves recognition accuracy by narrowing down the search space of possible word sequences during decoding. Instead of trying to recognize any sentence from an entire language, the system uses a grammar to recognize only those sentences that follow certain rules. This is especially useful in applications with a limited domain or vocabulary. (Singh, M. K., & Kumar, S., 2024)

3.5.1 CFG vs Statistical Language Models in ASR: There are two common types of language models in ASR:

1. Statistical Language Models (SLMs) – such as N-grams or neural models (e.g., RNNs, Transformers), which predict word sequences based on training data.
2. Grammar-Based Models – where CFG defines all acceptable sentence structures manually.

CFG is particularly advantageous in:

- Command-and-control systems
- Interactive voice response (IVR) systems
- Small-vocabulary applications
- Applications with strict structure

3.5.2 Practical Use in Speech Recognition Systems: CFGs are often compiled into Finite State Transducers (FSTs) or Parsing Graphs, which are integrated into ASR engines. When speech is processed, the recognition engine uses the grammar to match spoken phrases against valid sentence patterns. (Tiwari, K. K., Singh, A., & Kumar, S., 2025)

3.5.3 Dialogue Systems and Chatbots: Dialogue systems and chatbots are interactive software systems designed to converse with human users using natural language. These systems can operate via text (as in chatbots) or speech (as in virtual assistants like Siri, Alexa, or Google Assistant). They are used in a wide range of applications, such as customer service, personal assistants, education, healthcare, and entertainment. At the core of effective dialogue systems is the ability to understand user input, maintain context, and generate appropriate responses. To support these functionalities, Context-Free Grammar (CFG) is often used, particularly in rule-based and hybrid dialogue systems, to parse user inputs, guide dialogue flow, and manage intent recognition.

Role of CFG in Dialogue Systems: CFG is mainly used in dialogue systems to:

- Parse user utterances into structured representations
- Define the set of valid inputs a system can understand
- Enable intent recognition by mapping grammatical structures to specific commands or actions
- Guide conversation flow using rule-based patterns

By using predefined grammatical rules, a dialogue system can precisely determine the user's intent from a limited set of expressions. This is especially useful in task-oriented or domain-specific chatbots.

CFG in Intent Recognition and Slot Filling: In many dialogue systems, understanding what the user wants involves two key tasks:

- Intent Recognition – What is the user trying to do?
- Slot Filling – What specific information is provided (date, time, location, etc.)?

CFG enables both by parsing user input according to predefined rules and identifying relevant constituents.

Use of CFG in Dialogue Flow Management: CFG is also used to control the structure and flow of conversation, especially in rule-based dialogue systems. The system uses grammar rules to determine:

- What the user might say next
- How to respond appropriately
- When to ask for clarification

4 Applications of CFG:

1. Compiler Design: Used in syntax analysis (parsing) to define programming language syntax. CFGs are used to build parse trees which validate the structure of source code.
2. Natural Language Processing (NLP): CFG helps in parsing and understanding human languages. It models grammatical rules in languages like English for tasks such as sentence generation and structure checking.
3. Artificial Intelligence: In AI, CFG can model structured knowledge, especially for planning, decision-making, and interpreting commands in natural language.
4. Formal Verification: Used in model checking and program verification to describe and verify the properties of systems.
5. Automata Theory: CFG corresponds to Pushdown Automata (PDA), which is more powerful than finite automata and useful in recognizing context-free languages.
6. Education and Research: Used to teach and analyze theoretical foundations of computer science and formal language theory.

5 Challenges & Future Directions of CFG:

5.1 Challenges:

- Ambiguity: CFGs can be ambiguous, making it hard to parse languages deterministically.
- Limited Expressiveness: CFGs can’t describe certain language constructs (like cross-serial dependencies in natural languages).
- Parser Efficiency: While powerful, parsing some CFGs (e.g., using Earley or CYK parsers) is computationally intensive.
- Error Handling: Difficult to integrate robust and user-friendly error handling in CFG-based parsers.

5.2 Future Directions:

- Extended CFGs: Research into grammars beyond CFG (like PEGs or context-sensitive grammars).

- Probabilistic CFGs (PCFGs): Useful in NLP for handling ambiguity using statistical methods.
- Neural Parsing: Leveraging deep learning models to approximate or augment traditional CFG parsing.
- Hybrid Models: Combining symbolic CFG rules with data-driven ML/NLP models.

Conclusion: Context-Free Grammar (CFG) remains a vital component in the field of Natural Language Processing, particularly in applications where syntactic structure and rule-based understanding are essential. This paper has examined the multifaceted role of CFG across key NLP areas—question answering, machine translation, speech recognition, and dialogue systems—highlighting its strengths in formal language modeling, controlled input interpretation, and domain-specific task execution. CFG enables systems to interpret complex linguistic constructs with precision and to generate well-formed outputs, making it especially valuable in environments with limited vocabulary or predefined user interaction patterns. Its use in speech recognition enhances accuracy through constrained grammars, while in dialogue systems, it contributes to reliable intent detection and conversation flow control. Despite its advantages, CFG also has inherent limitations, such as a lack of contextual awareness and difficulty scaling to open-domain or highly variable inputs. However, the integration of CFG with machine learning and neural network-based models has led to hybrid systems that leverage the structural rigor of grammar rules alongside the adaptability of data-driven approaches. As NLP technologies continue to evolve, CFG remains relevant—not as a standalone solution, but as a complementary technique that strengthens system performance, interpretability, and reliability in structured language understanding tasks.

References:

Jurafsky, D., & Martin, J. H. (2021). *Speech and Language Processing* (3rd ed. draft). Stanford University.

Chomsky, N. (1956). Three models for the description of language. *IRE Transactions on Information Theory, 2*(3), 113–124.

Harabagiu, S., Pasca, M., & Maiorano, S. (2001). Experiments with open-domain textual question answering. *Proceedings of the ACL 2001 Conference on Empirical Methods in Natural Language Processing*, 292–302

Moldovan, D. I., & Mihalcea, R. (2003). Pattern matching for answer extraction. *Computers and the Humanities, 37*(1), 97–120.

Jelinek, F. (1997). *Statistical methods for speech recognition*. MIT Press.

Young, S., Evermann, G., Gales, M., Kershaw, D., Liu, X., Moore, G., ... & Woodland, P. (2006). *The HTK Book (for HTK Version 3.4)*. Cambridge University Engineering Department.

Wang, Y., & Acero, A. (2006). Grammar learning for spoken language understanding. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 1, I–1041

McTear, M. (2020). *Conversational AI: Dialogue systems, conversational agents, and chatbots*. Springer

Radziwill, N. M., & Benton, M. C. (2017). Evaluating quality of chatbots and intelligent conversational agents. *Journal of Systems and Software*, *137*, 216–227.

Bohus, D., & Rudnicky, A. I. (2009). The RavenClaw dialog management framework: Architecture and systems. *Computer Speech & Language*, *23*(3), 332–361.

Kumar, S., & Kour, G. (2025, March). Advanced Machine Learning Approaches for Fastag Fraud Detection. In *2025 International Conference on Automation and Computation (AUTOCOM)* (pp. 149-154). IEEE.

Kumar, S. (2024, May). Advancements in meta-learning paradigms: a comprehensive exploration of techniques for few-shot learning in computer vision. In *2024 International conference on intelligent systems for cybersecurity (ISCS)* (pp. 1-8). IEEE.

Sharma, S., Rana, S., & Dubey, S. S. (2024). ESAF: An Enhanced and Secure Authenticated Framework for Wireless Sensor Networks. *Wireless Personal Communications*, *136*(3), 1651-1673.

Singh, M. K., & Kumar, S. (2024, April). Stress Detection During Social Interactions with Natural Language Processing and Machine Learning. In *2024 International Conference on Expert Clouds and Applications (ICOECA)* (pp. 297-301). IEEE.

Tiwari, K. K., Singh, A., & Kumar, S. (2025, February). A Comprehensive Analysis of CNN-Based Deep Learning Models: Evaluating the Impact of Transfer Learning on Model Accuracy. In *2025 2nd International Conference on Computational Intelligence, Communication Technology and Networking (CICTN)* (pp. 62-67). IEEE.

Kumar, S., Rampal, S., Gaur, M., & Gaur, M. (2024, March). Advanced ensemble learning approach for asthma prediction: Optimization and evaluation. In *2024 International Conference on Automation and Computation (AUTOCOM)* (pp. 283-288). IEEE.