Enhancing the English natural language processing dictionary using natural language processing++

Jayanth Chikkarangaiah[1], Adarsh Uday[2], David De Hilster[3], Shobha Gangadhar[2], Jyoti Shetty[2]
[1]Department of Information Science and Engineering, R. V. College of Engineering, Bangalore, India
[2]Department of Computer Science and Engineering, R. V. College of Engineering, Bangalore, India
[3]Supercomputing Consultant, Lexis Nexis Risk Solution, Boca Raton, United States

Article Info

Article history:

Received Sep 24, 2023
Revised Nov 7, 2023
Accepted Nov 16, 2023

Keywords:

Analyzers
English dictionary
Hpcc systems
Knowledge base
Linguistics
Natural language processing
Natural language processing ++

ABSTRACT

Every natural language-based project requires the use of an English dictionary. But the current English dictionaries are not updated as the English language is constantly evolving. The English dictionary used for natural language processing (NLP) projects needs to be enhanced by adding more words and phrases. This helps in improving the accuracy of NLP applications such as machine translation, performance of text analysis, recognition, and part of speech (POS) tagging. Several approaches are proposed in this direction, this paper develops and demonstrates enhancement of the English dictionary using a more versatile and robust programming language known as NLP++, a plugin to distributed big data analytics platforms such as HPCC systems. The unique features of NLP++ language is the enabler for realization of the proposed approach. This paper also discusses key NLP techniques, dictionary refinements analysis using NLP and NLP++. The results show that the proposed approach using NLP++ has significantly improved the accuracy and comprehensiveness of the English dictionary.

Corresponding Author:

Jayanth Chikkarangaiah
Department of Information Science and Engineering, R. V. College of Engineering
Bangalore, Karnataka, India
Email: jayanthc84@gmail.com

1. INTRODUCTION

In this section introduction to natural language processing (NLP) and the state-of-the-art NLP++ programming language is discussed along with a few NLP concepts and NLP++ features. NLP technologies enable computers to understand, interpret, and generate human language [1]. The arrival of the computer led to attempts to restate Chomsky's principles [1] into rules for computers. Therefore, were born the areas of computational linguistics and NLP. In the early 1990s, neural networks and machine learning were introduced, which offered a new approach to NLP.

Almost no NLP project can function without a lexicon [2]. There is a growing relationship between computational NLP and dictionaries. Therefore, is an attempt to increase the size of the lexicon for computational systems from the online resources and make it into a machine-readable dictionary which serves the NLP. The English dictionary used for NLP needs to be enhanced [3] to add more words and phrases since the English language is constantly evolving. This further helps in improving the accuracy of machine translation, performance of text analysis and recognition, part of speech (POS) tagging and many more applications. Overall, an enhanced dictionary would be valuable to the NLP systems. We explore the use of NLP++ programming language to improve the English dictionary. We also propose a new approach to enhancing the dictionary, which involves the use of corpus linguistics, analysers and knowledge base (KB) to identify new words and phrases.

NLP++ [4] is a C++ like programming language specially designed for building deep text analyzers. NLP++ is available as an extension in high-performance computing cluster (HPCC) systems (HPCC-NLP++ plugin). It allows programmers to capture and apply linguistic and world knowledge, emulating processes by which humans read and understand text. NLP++ combines bottom up, recursive grammar, and other methods in a multi-pass architecture [5] that operates on one parse tree. It works with a hierarchical KB, called conceptual grammar (CG), to dynamically build and use stored knowledge in analyzing text. Applications range from simple syntactic processing to full natural language understanding. VisualText is a developer's environment that exploits NLP++ and CG to rapidly elaborate text analyzers. Passes and KBs from one analyzer can be utilized to efficiently develop and customize new text analyzers. NLP++ programming language is integrated and compiled in VisualText IDE [6] (an extension of it in VisualStudio code).

The English NLP dictionary is a valuable resource for NLP tasks. In recent years, there have been a number of efforts to enhance the English NLP dictionary. One approach is to use crowdsourcing to collect new words and phrases. Since the so-called "statistical revolution" [7], important NLP exploration has relied heavily on machine literacy. Using statistical conclusions, machines automatically learn similar rules through the analysis of a large corpora of typical real-world examples. A major disadvantage of statistical styles is that they require intricate fine-tuning. The field has therefore largely abandoned statistical styles and shifted to neural networks for NLP.

There are several technologies available for English dictionary creation and enhancement for NLP. In corpus-based approaches [8], for the production and improvement of dictionaries, large text corpora can be analyzed to extract words and the data that goes with them. Word meanings, usages, and relationships can be derived from the corpus data using statistical techniques like frequency analysis and collocation extraction. Machine learning algorithms also help in the creation and enhancement of NLP dictionaries. For instance, word embeddings [9] can record the semantic and syntactic features of words, enabling the inference of word meanings and similarities.

Goldhahn *et al.* [10] presented their methodology for dictionary construction, which involves preprocessing and analyzing the vast collection of textual data available at the Leipzig Corpora Collection, using the techniques such as tokenization, morphological analysis, and frequency-based filtering to extract lexical information and create the monolingual dictionaries. Here the paper proposes a method for creating a common syntactic dictionary of English [11]. The authors propose the process for creating the dictionary by extracting syntactic information from existing dictionaries and merging the extracted information to create a unified representation.

WordNet [12], a ubiquitous tool for NLP, has been used for application specific dictionary creation [13] for NLP purposes [14]. There are examples of WordNet in other languages [15]–[17]. However, it suffers from a diversity of words [18]. There is a lack or nonexistence of extensive collections of texts or linguistic data [19]. Enriching WordNet with subject-specific terms sourced from wikidata, thereby enhancing its effectiveness in domain-specific NLP tasks, as discussed in [20].

A framework for building the old English WordNet (OldEWN), employing established dictionaries and the Naisc system suggests an enhanced schema to capture etymological intricacies, facilitating comprehensive lexical representation and cross-resource compatibility [21]. An approach for managing evolving ontologies and maintaining alignment validity in knowledge management is introduced in [22], while Hnatkowska *et al.* [23] presents a method for automatically designating attribute semantics in ontologies using Word2Vec similarity and WordNet lexical database. Additionally, an approach to creating a comprehensive Singlish dictionary using a combination of an AI language model (ChatGPT) and manual annotation by native speakers, demonstrates the effectiveness of using large language models (LLMs) in creating language resources for low-resource varieties and highlights the potential for further expansion and refinement [24].

Therefore, with the help of English Wiktionary [25], we propose to create a more linguistic dictionary which further aids in NLP. Based on the study discussed here, some limitations observed are: it is seen that lexical variations for comprehensive and up-to-date coverage in the dictionaries are lacking and not delving into lexical semantics or contextual meaning, which are crucial for a comprehensive NLP dictionary. Many papers do not provide a detailed evaluation or analysis of the practical implementation of the proposed dictionary. They lack empirical evidence to support the effectiveness and accuracy of the dictionary in real-world NLP applications.

Our approach to English dictionary creation has diverse applications in the field of domain-specific text analysis for dictionary creation [26]. It is particularly valuable for information extraction, serving as a foundational resource for a wide range of text analytics tasks focused on the COVID-19 crisis [27]. Additionally, our methodology is suitable for the development of open-source NLP dictionaries and inference engines designed for event identification, enabling the processing of extensive textual datasets [28].

This paper proposes an approach to improve the accuracy and comprehensiveness of the English dictionary using NLP++ programming language. NLP++ is effective in building and improving NLP dictionaries as it is a powerful and adaptable language used to describe a variety of linguistic concepts. The following paragraph discusses the architecture of HPCC-NLP++ plugin at first, followed by the sections- method, implementation and recursive relation and parse tree representation of the proposed approach.

HPCC systems is an open source, data-intensive platform developed by LexisNexis risk solutions [29], [30]. It is a software architecture implemented on computing clusters to provide high-performance, data-parallel processing for big data applications. Several noteworthy applications have been developed in collaboration with HPCC systems, including the generalised neural networks (GNN) bundle [31], density-based spatial clustering of applications with noise (DBSCAN) [32] for density-based clustering, and the creation of a SQL-like language [33] for data analysis within distributed platforms, such as HPCC systems.

The HPCC-NLP++ plugin as shown in Figure 1 is an open source, high-performance NLP plugin for HPCC systems. It is based on the NLP++ language extension, which provides a powerful and flexible way to define NLP tasks. The plugin is designed to be scalable and efficient, making it suitable for large-scale NLP tasks.

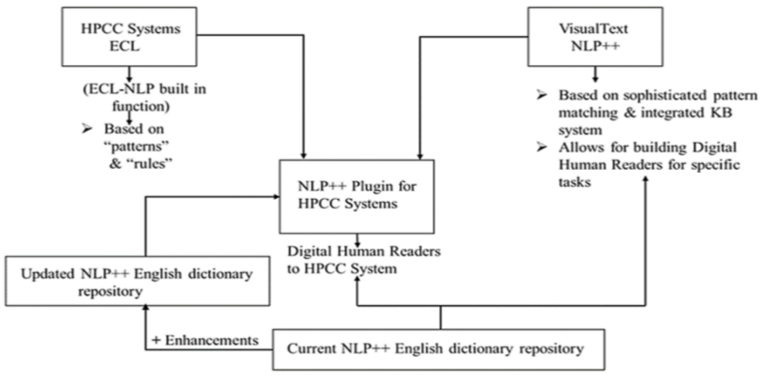

HPCC Systems ECL
(ECL-NLP built in function)
➢ Based on "patterns" & "rules"
VisualText NLP++
➢ Based on sophisticated pattern matching & integrated KB system
➢ Allows for building Digital Human Readers for specific tasks
NLP++ Plugin for HPCC Systems
Digital Human Readers to HPCC System
Updated NLP++ English dictionary repository
+ Enhancements
Current NLP++ English dictionary repository

Figure 1. System architecture of HPCC systems-NLP++ plugin

2. METHOD

In this section, we provide a detailed description of the methodology followed during the course of the implementation of the proposed approach methodology for English dictionary with explanations of each module. The steps shown in Figure 2 of extracting the xml files by web scraping the English Wiktionary and processing the text data files, then creation of the analyzers sequence with rules and functions using NLP++ in VisualText with KB development lead to the creation of the English NLP dictionary. Experimental setup- using VisualText in Figure 3 and its extension in visual studio code to deploy the methodology following the steps in Figure 2. The system provides a text input area, a user-friendly step-by-step analyzer sequence for text processing, and a KB for reference. It offers an output section to visualize results and an analyzer section for configuring analyzers. The logging section tracks the system's status and helps identify errors during analysis.

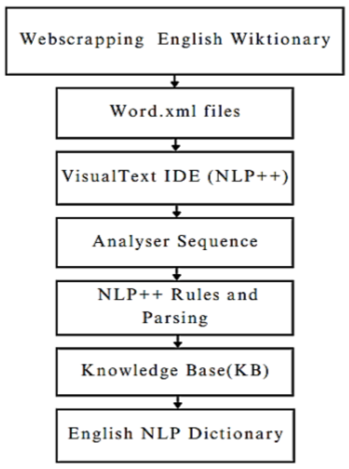
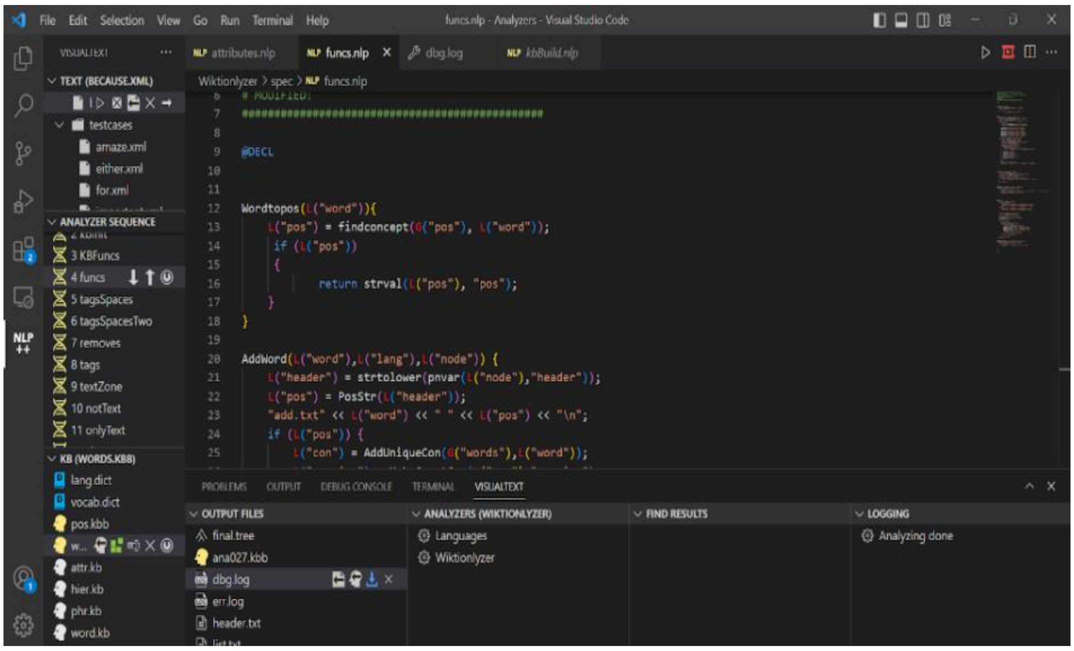


Figure 2. NLP++ approach methodology



```
# MODIFIED:
##################################################

@DECL


Wordtopos(L("word")){
    L("pos") = findconcept(G("pos"), L("word"));
     if (L("pos"))
     {
            return strval(L("pos"), "pos");
     }
}

AddWord(L("word"),L("lang"),L("node")) {
    L("header") = strtolower(pnvar(L("node"),"header"));
    L("pos") = PosStr(L("header"));
    "add.txt" << L("word") << " " << L("pos") << "\n";
    if (L("pos")) {
        L("con") = AddUniqueCon(G("words"),L("word"));
```

Figure 3. Architecture of VisualText IDE extension visual studio code for NLP++

3. IMPLEMENTATION

Following subsections provide the description of procedural implementation of the steps mentioned in Figure 2. The first step is collection of input data files by web scraping the English Wiktionary. Feeding these files as input to NLP++ parser and building an analyzer sequence in VisualText. Once an analyzer is created, it is tokenized in the analyzer sequence. Then after running the file in analyzer sequence, understanding the input file with the help of the final parse tree generated and looking for patterns. Based on this, accordingly, write rules to parse the words. This will further help in the creation of a KB. Following are the implemented steps.

3.1. Web scraping English Wiktionary

After research and analysis, the English Wiktionary [25], [34] was best suited for the project since it is community-contributed with detailed entries and frequent upgrades, providing information on a wide range of topics on every word. It is also free and open-source. Listed is a wikitext entry for the word “this” where the word is in the:

```
<title></title > tag
<page>
<title>this</title>
<ns>0</ns>
<id>3974</id>
<revision>
<id>69637328</id>
<parentid>69425033</parentid>
<timestamp>2022-11-01T20:59:43Z</timestamp>
<contributor>
<username>Ri132</username>
<id>3977588</id>
</contributor>
<comment>t+bho:[[ई]] ([[WT:EDIT|Assisted]])</comment>
<model>wikitext</model>
<format>text/x-wiki</format>
<text bytes="26312" xml:space="preserve">{{also|This|thîs|þis}}
==English==
{{wikipedia}}
===Etymology===
```

The next step was to download the web pages from the English Wiktionary and convert them into word.xml files. This was done using web scraping techniques in Python (3.9). The wikitext entries of the xml files were analyzed in order to write parsers in NLP++.

3.2. Building the analyzers

The heart of the text analyzer lies in the pass files, which contain NLP++ code and rules. The @PATH is a selector that defines the context in which rules will be tried. The @RULES marks the start of a rule region. An NLP++ rule consists of a pattern (called a phrase) that is matched against the parse tree. When a rule matches, NLP++ code in the post region (@POST) tells the rule matcher what actions to take. NLP++ code has three regions specifically. They are the code, declaration and rules regions. NLP++ pass file includes a CODE region that is independent of any rule matching.

The multiple passes of such a methodology must communicate with each other. The output of one pass in the analyzer sequence should serve as the input to the next. A parse tree is universally recognized as an ideal data structure for holding at least a subset of such information. Further, the multiple passes should be able to post to and utilize information from a shared KB.

VisualText favors a bottom-up approach to text processing, where small constructs are processed and grouped (or reduced) first, followed by identification of successively larger components of a text. Its extension is .nlp. Out of the 24 analyzers we have written in NLP++ to carry out the enhancement, the three main analyzers are discussed in following section.

3.2.1. HeaderZones.nlp

The headerZones.nlp analyzer extracts the header and restricts the textZone in the xml to specific header groups. Basically, it is confining the text Zone into multiple header groups based on the Wikitext entries. Here the NLP++ parsers a text string and extracts the header, language, and any other words in the text. The @POST section defines the actions that will be taken after the text is parsed. In this case, the code will create a new variable called "header" and assign it the value of the first word in the text that matches the expression "_header". If the text also contains the word "pofs", the code will create a context variable called "pofs" and assign it the suggested word "pofs". Similar rule is followed for lang variable too.

The @RULES section defines the rules that will be used to parse the text. In this case, the code defines two rules. Rule-(##1): the header is the first word in the text that matches the regular expression "_header". Rule-(##2): any word in the text that does not match rule 1 or rule 2 is considered to be a "wild" word. The code then uses these rules to parse the text and extract the header, language, and any other words in the text. Listed is the code snippet of headerZones analyzer in NLP++,

```
@PATH _ROOT _textZone
@POST
S("header") = N("header",1);
if(N("pofs",1))
 S("pofs") = N("pofs",1);
S("lang") = N("lang",1);
single();
@RULES
_headerZone <-
 _header ### (1)
 _xWILD [fail= (_header _language _xEND)] ### (2)
 @@
```

3.2.2. PofZone.nlp

The pofZone.nlp is key analyzer pass in this process where it compares and checks with every Wikitext entry whether it belongs to the set of the English parts of speech that is any one among adjective, verb, noun, pronoun, interjection, conjunction, adverb and marks the respective word to the respective identified POS. The function AddUniqueStr() in the above code, adds the POS tag selected from the third suggested variable[##(3)] from the rules section text to a list of unique POS tags. It takes three arguments: the word, the part-of-speech tag and the text string. The single() - tells the parser to end the parsing process.

_pos <- starts the @RULES section of the code. The fifth line defines a rule that says that the POS tag for the English word is "en". The last _xWILD defines a rule that says that the POS tag for any other word is the part-of-speech tag that is associated with the word in the list of POS tags. The NLP++ code snippet of pofZone analyzer is as follows:

```
@POST
AddUniqueStr(G("word"),"pos",N("$text",3));
single();
@RULES
_pos <-
 en ### (1)
 _xWILD [ one match = (\- \|)] ### (2)
 _xWILD [ one match = (adj adjective intj interj interjection noun art article prep preposition con conj
conjunction pron pronoun det determiner adv adverb cont contraction verb proper noun)] ### (3)
@@
```

3.2.3. Output.nlp

The snippet of NLP++ code is similar to knowledge representation used in artificial intelligence. The first part of the code is used to check if the current file is the last file in the directory, or if the script is not being run from a directory. If either of these conditions is true, then the code will save the KB to a file called "words.kbb". The code then closes the debug file and creates a new buffer called "out". The next part of the code iterates through the list of words in the KB, and it prints out the word name and its POS. The code snippet of the output analyzer in NLP++ is as follows:

```
if (G("$islastfile") || !G("$isdirrun")) {
 L("file") = G("$kbpath") +"words.kbb";
 L("output") = openfile(L("file"));
 SaveKB(L("output"),G("words"),2);
 closefile(L("output"));
 G("debug") << "Conjugations: " << str(G("conju")) << "\n";
}
closefile(G("debug"));
L("out") = cbuf();
L("word") = down(G("words"));

while (L("word")) {
 L("pos") = findvals(L("word"),"pos");
 if(L("pos"))
 {
 L("out") << "<word>" << conceptname(L("word")) << "<\word>" << "\n";
 }
 L("word") = next(L("word"));
}
```

3.3. Knowledge base

The CG knowledge base management system (KBMS) in NLP++ provides a permanent store for linguistic, conceptual, and domain knowledge. The KBMS serves as a flexible "vanilla" framework in which users may implement arbitrary representation schemes. Users can manage the KB manually, while text analyzers can automatically access and update information in the KB. The KB can be used to collect information across multiple texts. The KB stores the parts of speech of each word for the analysers to be displayed.

4. RECURSIVE RELATION AND PARSE TREE REPRESENTATION

In NLP++, the recursive relation supports the arrangement of linguistic and conceptual elements in a hierarchical manner, which aids in the representation of information in alignment with the grammar rules of the language. The parse tree represents the tiered structure of the code, showing how statements and expressions are structured according to the language's grammar rules. CG refers to the hierarchical KB used in NLP++. It contains concepts, attributes, and phrases that store linguistic, conceptual, and domain knowledge. Analyzers can use CG to access and update information during text analysis.

4.1. Recursive relation for conceptual grammar

The recursive relation enables the inclusion of concepts, attributes, and phrases within CG, demonstrating the hierarchical knowledge organization employed in NLP++. This hierarchy is utilized to represent linguistic and conceptual information. This structured approach enhances the organization and retrieval of knowledge.

Conceptual Grammar <- (Concept) | (Attribute) | (Phrase)

4.2. Parse tree representation of conceptual grammar

The provided parse tree in Figure 4 and the diagram represents the hierarchical structure of CG in NLP++, where CG consists of concepts, attributes, and phrases each with its own specific components and purposes for organizing linguistic and conceptual knowledge. This structured approach contributes significantly to the field of NLP by advancing the capacity to model and analyze the complex relationships found within language.

```
Conceptual Grammar
├── Concept (A knowledge unit)
│   ├── Concept Name
│   └── Attributes
├── Attribute (Defines specific properties)
│   ├── Attribute Name
│   └── Attribute Value
├── Phrase (Captures linguistic or conceptual expressions)
├── Phrase Name
└── Phrase Value
```

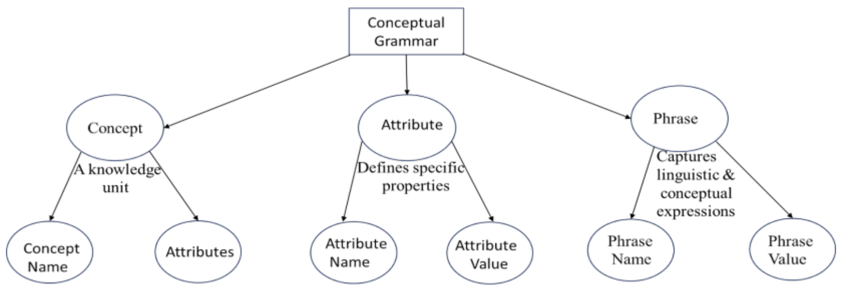


Figure 4. CG parse tree

4.3. Parse tree representation of the above three analyzers in section 3.2.

The parse tree given in this subsection outlines the composition of three analyzers with name, description, and functionality. Each analyzer is designed to perform distinct tasks, such as extracting header and language information, identifying parts of speech, and managing KB data and output generation. This organized breakdown of analyzers within the parse tree serves as a valuable reference, facilitating a clear understanding of the NLP++ text processing capabilities and enabling efficient configuration and customization.

```
Analyzers
├── Analyzer1
│   ├── Analyzer Name (headerZones.nlp)
│   ├── Analyzer Description (Extracts header and language)
│   └── Analyzer Functionality
│   ├── @POST Actions
│   └── @RULES
├── Analyzer2
│   ├── Analyzer Name (pofZone.nlp)
│   ├── Analyzer Description (Matches and extracts parts of speech)
│   └── Analyzer Functionality
│   ├── @POST Actions
│   └── @RULES
├── Analyzer3
├── Analyzer Name (output.nlp)
├── Analyzer Description (Saves KB and generates output)
└── Analyzer Functionality
└── @CODE
```

The overall recursive relationship present in analyzers involves the five main components that make up an NLP++ analyzer. Each of these components contributes to the functionality and structure of the analyzer. Understanding this recursive relationship is pivotal for both the development and optimization of NLP++ analyzers, as it forms the core framework upon which its capabilities are built.

NLP++ Analysers <- (Variables) | (Regions) | (Rules) | (Functions) | (ConceptualGrammar)

5. EXPERIMENTAL ANALYSIS

After looking at the Wikipedia pages that were to be processed, it was determined that NLP++ was not only suited for such a project, but virtually impossible without it. Wikipedia pages are somewhat regular, their variations required us to use NLP++ given that it was specifically created to handle unstructured text. To do the same task with another language would require recreating text trees, rules, and a KB that is automatically provided by NLP++. While technically it was feasible, it would be a tedious task.

In conventional machine learning NLP requires a massive amount of human generated examples to train, while on the other hand NLP++ needs a small text to develop. As mentioned earlier, NLP++ makes use of the exclusive KB which helps in the parsing of natural texts unlike the conventional machine learning approaches. NLP++ encodes human thinking. In any given example, first the NLP++ analyzers perform syntactic parse on the input. A KB may be previously built for the specific task or can be developed from the parsed text. This later serves as linguistic and world KB for further processing. This is exactly the same way a human would approach analyzing a language related task.

In these regards, NLP++ is by far the most advanced programming language for NLP. NLP++ is an evolving general programming language that "talks" parse tree and KB, as well as supporting a range of actions that are customized for rule and pattern matching. Parse tree nodes and KB objects are built-in NLP++ data types, enabling simple programmatic manipulation of these objects. NLP++ also enables dynamically decorating the parse tree and KB with semantic representations. A key feature of NLP++ is that its multi-pass framework communicates via a single parse tree. Further, constraining the analyzer to deal with a single parse tree helps guide the development of a fast and efficient text analyzer.

6. RESULTS AND DISCUSSION

The proposed approach of enhancing the English dictionary using NLP++ has shown promising results in improving the accuracy and comprehensiveness of the dictionary. By utilizing the NLP++ programming language and techniques, we were able to add new words and phrases to the dictionary, which were previously missing. This enhances the dictionary's coverage and ensures that it keeps up with the evolving English language.

The output of the proposed methodology is the Words.kbb file which consists of extracted parts of speech for each test case word. This file serves as a crucial component of the KB and holds valuable linguistic information. A snapshot of contents of Words.kbb under the KB is as follows:

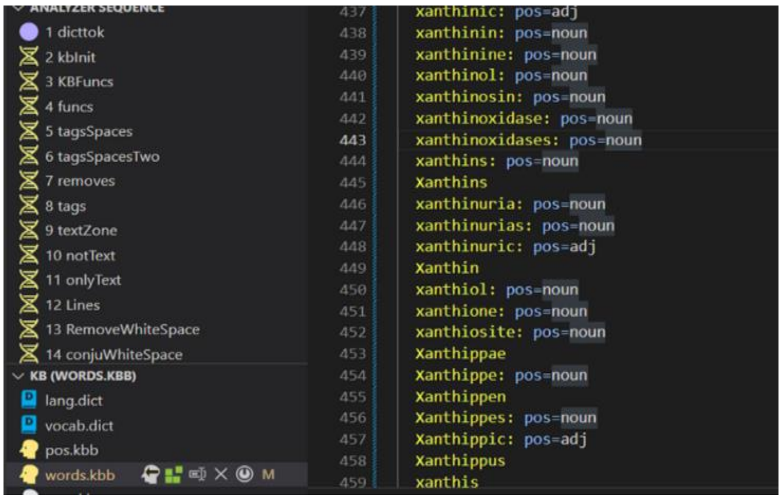
```
would: pos=[verb,noun]
amaze: pos=[verb,noun]
because: pos=[adv,con,prep]
either: pos=[det,pron,adv,con]
for: pos=[con,prep]
important: pos=adj
jump: pos=[verb,conj,noun,adv,adj]
large: pos=[adj,noun,adv]
since: pos=[adv,prep,con]
some: pos=[pron,det,adv]
this: pos=[adv,pron,noun,interj,determiner]
```

The debug log file contains detailed information about the various stages and components of the pipeline, allowing developers and users to track the execution flow, identify any errors or issues, and gather debugging information. We can observe that the total time taken to parse 24 analyzer passes is 0.077 seconds for the input- beacuse.xml file. A snapshot of the dbg.log file generated during the execution of the NLP++ pipeline is as follows:

```
[Pass 1 time: 0.009 sec     dicttok]
[Pass 2 time: 0.001 sec     kbInit]
```

```
[Pass 3 time: 0 sec          KBFuncs]
[Pass 4 time: 0 sec          funcs]
[Pass 5 time: 0.004 sec      tagsSpaces]
[Pass 6 time: 0.003 sec      tagsSpacesTwo]
[Pass 7 time: 0.011 sec      removes]
[Pass 8 time: 0.005 sec      tags]
[Pass 9 time: 0.004 sec      textZone]
[Pass 10 time: 0 sec         notText]
[Pass 11 time: 0 sec         onlyText]
[Pass 12 time: 0.006 sec     Lines]
[Pass 13 time: 0.022 sec     RemoveWhiteSpace]
[Pass 14 time: 0 sec         conjuWhiteSpace]
[Pass 15 time: 0.003 sec     headers]
[Pass 16 time: 0 sec         ups]
[Pass 17 time: 0.001 sec     headerZones]
[Pass 18 time: 0.002 sec     posZone]
[Pass 19 time: 0 sec         lineHeader]
[Pass 20 time: 0 sec         langZone]
[Pass 21 time: 0.002 sec     attributes]
[Pass 22 time: 0 sec         languageZone]
[Pass 23 time: 0 sec         kbBuild]
[Pass 24 time: 0.004 sec     output]
[Input file: c:\NLP++\dict-en-us\Analyzers\Wiktionlyzer\input\because.xml]
[Total passes: 24]
[Total time: 0.077 sec]
[Time per 1000 chars (0.077/18646): 0.00412957 sec]
[Time per pass per 1000 chars: 0.000172066]
infile passes tottime length tot/1000 tot/pass/1000
c:\NLP++\dict-en-us\Analyzers\Wiktionlyzer\input\because.xml 24 0.077 18646 0.00412957 0.000172066
[Clean time: 0.002 sec]
```

The screenshot in Figure 5 of the output is shown for X alphabetical words with their respective POS. The use of corpus linguistics, analyzers, and KB played a crucial role in identifying and extracting new words and their linguistic features. The integration of NLP++ with VisualText IDE and its extension in visual studio code facilitated the implementation of the enhanced dictionary. The analyzer sequence in VisualText allowed for the parsing and analysis of text, while the KBMS stored and managed the linguistic, conceptual, and domain knowledge. The multiple passes in NLP++ communicated with each other, and the output of one pass served as the input to the next, ensuring a cohesive and efficient process.



```
xanthinic: pos=adj
xanthinin: pos=noun
xanthinine: pos=noun
xanthinol: pos=noun
xanthinosin: pos=noun
xanthinoxidase: pos=noun
xanthinoxidases: pos=noun
xanthins: pos=noun
Xanthins
xanthinuria: pos=noun
xanthinurias: pos=noun
xanthinuric: pos=adj
Xanthin
xanthiol: pos=noun
xanthione: pos=noun
xanthiosite: pos=noun
Xanthippae
Xanthippe: pos=noun
Xanthippen
Xanthippes: pos=noun
Xanthippic: pos=adj
Xanthippus
xanthis
```

Figure 5. Words.kbb (the final KB) which has successfully been updated for all the parsed files

There are several technologies available for English Dictionary creation and enhancement for NLP. In corpus-based approaches [8], [26], [35] these methods involve in the analysis of extensive text corpora to extract words, their meanings, usages, and relationships using statistical techniques such as frequency analysis and collocation extraction. It relies on statistical patterns and relationships observed within the corpus and it is more data-driven, examining word patterns and usage frequencies within the corpus. With NLP++, it enables direct and explicit encoding of linguistic rules and concepts. It diverges from relying solely on statistical patterns in the corpus, allowing for a structured and rule-based approach to define linguistic concepts and relationships.

According to Bosc and Vincent [9], they leverage machine learning algorithms particularly focusing on word embeddings to capture the semantic and syntactic features of words, enabling the inference of meanings and similarities. Word embeddings capture semantic and syntactic features of words by representing them as vectors in a continuous vector space. They focus on learning word representations based on their contextual usage in large text corpora. NLP++ programming language may not directly focus on creating vector representations like word embeddings do. Instead, it primarily concentrates on encapsulating linguistic rules and concepts in a structured and clearly defined manner, fostering a more rule-based approach to language processing.

Utilization of WORDNET in [12], [13], [18], [19] for creating application-specific dictionaries. WORDNET organizes words into synsets and showcases relationships between them, offering a thesaurus-like structure, providing a structured framework to understand word relationships through hierarchical associations. It encountered challenges due to word diversity and lack of comprehensive linguistic data. Unlike WORDNET, NLP++ based approach is more flexible and adaptable, allowing for the creation of rules and relationships based on linguistic concepts and not solely on predefined hierarchical relationships. Our approach emphasizes the utilization of the NLP++ programming language, highlighting its effectiveness in building and improving NLP dictionaries. NLP++ is regarded as powerful and adaptable, catering to various linguistic concepts.

The enhanced English dictionary has several implications for NLP systems. It improves the accuracy of machine translation by incorporating new words and their meanings. It enhances the performance of text analysis and recognition by providing comprehensive and up-to-date lexical information. The results obtained from the enhancement process, as reflected in the updated KB, demonstrated the successful addition of new words with their respective parts of speech. Inorder to analyse the the vast volume of wiktionary data, we propose to use an open-source distributed computing platform HPCC systems and enterprise control language (ECL) [36] programming language. Overall, the enhanced English dictionary using NLP++ proves to be valuable in improving the accuracy and performance of NLP systems. It addresses the challenge of keeping up with the evolving English language and provides a solid foundation for various NLP applications.

7. CONCLUSION

In this paper, we presented an approach to enhance the English dictionary used in NLP systems using NLP++ programming language. By leveraging the power of NLP++, corpus linguistics, analyzers, and KB, we successfully added new words and phrases to the dictionary, improving its comprehensiveness and accuracy. The results demonstrated that the enhanced dictionary significantly improved the coverage and accuracy of lexical information. NLP++ proved to be an effective programming language for building and improving NLP dictionary, also with its extension in HPCC systems it is highly scalable. Its flexibility, adaptability, and support for linguistic concepts make it a powerful tool in the field of natural language. Therefore, NLP++ is a good choice for researchers and developers who want to work on specific NLP tasks.

8. FUTURE ENHANCEMENTS

The analyzers written in NLP++ effectively parse down the English Wiktionary files and generate the KB and the dictionary files. Owing to the enormous amount of wiktionary data, the processing using a standalone system becomes a performance bottleneck. Hence it is proposed to use HPCC systems and ECL programming language for parallelized operations, increasing the speed and efficiency of processing.

ACKNOWLEDGEMENTS

We would like to thank HPCC Systems, LexisNexis and R.V. College of Engineering for their active support, encouragement and assistance in the project enhance the English NLP Dictionary.

REFERENCES

[1] B. Goel, "Developments in the field of natural language processing," *International Journal of Advanced Research in Computer*

Science, vol. 8, no. 3, pp. 23–28, 2017.

[2] L. Guthrie, J. Pusteljovsky, Y. Wilks, and B. M. Slator, “The role of lexicons in natural language processing,” *Communications of the ACM*, vol. 39, no. 1, pp. 63–72, 1996, doi: 10.1145/234173.234204.

[3] J. P. McCrae, A. Rademaker, F. Bond, E. Rudnicka, and C. Fellbaum, “English wordnet 2019 - An open-source wordNet for English,” *Proceedings of the 10th Global WordNet Conference*, pp. 245–252, 2020.

[4] P. Deane, D. D. Hilster, and A. Meyers, “Text processing in an integrated development environment (IDE): integrating natural language processing (NLP) techniques,” *PC AI*, vol. 15, no. 5, 2001.

[5] A. Meyers, “Multi-pass multi-strategy NLP,” *Text Analysis International*, pp. 1-5, 2023.

[6] Text Analysis International, “Integrated development environments for natural language,” *Text Analysis International*, Sunnyvale, California, 2001.

[7] D. Pinto, J. Civera, A. B. -Cedeño, A. Juan, and P. Rosso, “A statistical approach to crosslingual natural language tasks,” *Journal of Algorithms*, vol. 64, no. 1, pp. 51–60, 2009, doi: 10.1016/j.jalgor.2009.02.005.

[8] A. C. Junior and S. M. Aluisio, “Building a corpus-based historical Portuguese Dictionary: challenges and opportunities,” *Traitement Automatique Des Langues*, vol. 50, no. 2, pp. 73–102, 2009.

[9] T. Bosc and P. Vincent, “Auto-encoding dictionary definitions into consistent word embeddings,” *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, EMNLP 2018*, pp. 1522–1532, 2018, doi: 10.18653/v1/d18-1181.

[10] D. Goldhahn, T. Eckart, and U. Quasthoff, “Building large monolingual dictionaries at the leipzig corpora collection: From 100 to 200 languages,” *Proceedings of the 8th International Conference on Language Resources and Evaluation, LREC 2012*, pp. 759–765, 2012.

[11] C. Macleod, R. Grishman, and A. Meyers, “Creating a common syntactic dictionary of English,” *SNLR: International Workshop on Sharable Natural Language Resources*, 1994.

[12] G. D. Melo and G. Weikum, “Towards universal multilingual knowledge bases,” *Global Wordnet Conference, GWC 2010*, pp. 149–156, 2010.

[13] D. B. Bracewell, “Semi-automatic creation of an emotion dictionary using WordNet and its evaluation,” *2008 IEEE International Conference on Cybernetics and Intelligent Systems, CIS 2008*, pp. 1385–1389, 2008, doi: 10.1109/ICCIS.2008.4670735.

[14] and G. R. E. Zotova, M. Cuadros, “Towards the integration of WordNet into ClinIDMap,” *Proceedings of the 12th Global Wordnet Conference*, pp. 352–362, 2023.

[15] Ö. Bakay *et al.*, “Turkish WordNet KeNet,” *GWC 2021 - Proceedings of the 11th Global Wordnet Conference*, pp. 166–174, 2021.

[16] F. Bond and T. Kuribayashi, “The Japanese Wordnet 2.0,” *Proceedings of the 12th Global Wordnet Conference*, pp. 179–186, 2023.

[17] M. Jain, R. Jindal, and A. Jain, “Automatic construction of interval-valued fuzzy Hindi wordnet using lexico-syntactic patterns and word embeddings,” *ACM Transactions on Asian and Low-Resource Language Information Processing*, 2024, doi: 10.1145/3643132.

[18] J. B. -Graber, C. Fellbaum, D. Osherson, and R. Schapire, “Adding dense, weighted connections to WordNet,” *GWC 2006: 3rd International Global WordNet Conference, Proceedings*, pp. 29–35, 2005.

[19] D. Beneventano, S. Bergamaschi, and S. Sorrentino, “Extending WordNet with compound nouns for semi-automatic annotation in data integration systems,” *2009 International Conference on Natural Language Processing and Knowledge Engineering, NLP-KE*, 2009, doi: 10.1109/NLPKE.2009.5313842.

[20] Kanika, S. Chakraverty, P. Chakraborty, A. Aggarwal, M. Madan, and G. Gupta, “Enriching WordNet with subject specific out of vocabulary terms using existing ontology,” *Lecture Notes in Networks and Systems*, vol. 238, pp. 205–212, 2022, doi: 10.1007/978-981-16-2641-8_19.

[21] A. F. Khan *et al.*, “Towards the construction of a WordNet for old english background, methods, infrastructure,” *2022 Language Resources and Evaluation Conference, LREC 2022*, pp. 3934–3941, 2022.

[22] M. Pietranik and A. Kozierkiewicz, “Methods of managing the evolution of ontologies and their alignments,” *Applied Intelligence*, vol. 53, no. 17, pp. 20382–20401, 2023, doi: 10.1007/s10489-023-04545-0.

[23] B. Hnatkowska, A. Kozierkiewicz, M. Pietranik, and H. B. Truong, “Hybrid approach to designating ontology attribute semantics,” *International Conference on Computational Collective Intelligence*, vol. 13501, pp. 351–363, 2022, doi: 10.1007/978-3-031-16014-1_28.

[24] S. Y. Chow, C. U. Shin, and F. Bond, “This word mean what: constructing a singlish dictionary with ChatGPT,” *Proceedings of the 2nd Workshop on Resources and Technologies for Indigenous, Endangered and Lesser-resourced Languages in Eurasia (EURALI) @ LREC-COLING*, pp. 41–50, 2024.

[25] T. Zesch, C. Müller, and I. Gurevych, “Extracting lexical semantic knowledge from Wikipedia and Wiktionary,” *Proceedings of the 6th International Conference on Language Resources and Evaluation, LREC 2008*, pp. 1646–1652, 2008.

[26] S. Häffner, M. Hofer, M. Nagl, and J. Walterskirchen, “Introducing an interpretable deep learning approach to domain-specific dictionary creation: a use case for conflict prediction,” *Political Analysis*, vol. 24, no. 1, 2023, doi: 10.1017/pan.2023.7.

[27] S. K. Rashed, J. Frid, and S. Aits, “English dictionaries, gold and silver standard corpora for biomedical natural language processing related to SARS-CoV-2 and COVID-19,” *Arxiv-Quantitative Biology*, pp. 1-8, 2020.

[28] T. Burley *et al.*, “NLP workflows for computational social science: understanding triggers of state-led mass killings,” *ACM International Conference Proceeding Series*, pp. 152–159, 2020, doi: 10.1145/3311790.3397343.

[29] A. M. Middleton, D. A. Bayliss, G. Halliday, A. Chala, and B. Furht, “The HPCC/ECL platform for big data,” *Big Data Technologies and Applications*, pp. 159–183, 2016, doi: 10.1007/978-3-319-44550-2_6.

[30] “HPCC systems,” *HPCC Systems*, 2011, [Online]. Available: https://hpccsystems.com/.

[31] A. Karthik, J. Shetty, G. Shobha, and R. Dev, “Implementation of generative adversarial networks in HPCC systems using GNN bundle,” *IAES International Journal of Artificial Intelligence*, vol. 10, no. 2, pp. 374–381, 2021, doi: 10.11591/ijai.v10.i2.pp374-381.

[32] H. R. Yatish *et al.*, “Massively scalable density based clustering (DBSCAN) on the HPCC systems big data platform,” *IAES International Journal of Artificial Intelligence*, vol. 10, no. 1, pp. 207–214, 2021, doi: 10.11591/ijai.v10.i1.pp207-214.

[33] A. S. Bhadauria, A. Bain, J. Shetty, G. Shobha, A. Chala, and J. Clements, “Design and Implementation of HSQL: A SQL-like language for Data Analysis in Distributed Systems,” *International Journal of Advanced Computer Science and Applications*, vol. 12, no. 11, pp. 796–803, 2021, doi: 10.14569/IJACSA.2021.0121190.

[34] C. M. Meyer, “Wiktionary the metalexicographic and the natural,” P.hD. Thesis, Department of Computer Science, Technischen Universität Darmstadt, Darmstadt, Germany, 2013.

[35] E. Riloff and J. Shepherd, “A corpus-based bootstrapping algorithm for Semi-Automated semantic lexicon construction,” *Natural Language Engineering*, vol. 5, no. 2, pp. 147–156, 1999, doi: 10.1017/S1351324999002235.

[36] A. M. Middleton, D. A. Bayliss, and G. Halliday, “ECL/HPCC: a unified approach to big data,” *Handbook of Data Intensive Computing*, pp. 59–107, 2011, doi: 10.1007/978-1-4614-1415-5_3.

BIOGRAPHIES OF AUTHORS

Jayanth Chikkarangaiah currently pursuing Bachelor of Engineering in Information Science and Engineering from R. V. College of Engineering, Bengaluru, India. His interests include software development and machine learning. He is also Joint Secretary of IEEE RVCE Computer Society and Student Executive Member of IEEE SIGHT Bangalore Section. He is also part of the coding club and astra robotics of RVCE. He can be contacted at email: jayanthc84@gmail.com.

Adarsh Uday currently pursuing Bachelor of Engineering in Computer Science and Engineering from R. V. College of Engineering, Bengaluru, India. He is a highly motivated tech enthusiast with a strong track record in user-centric development, specializing in Solidity for blockchain solutions and Android app development. Notable contributions to NLP, machine learning, and computer vision, including research and competition wins. He thrives in collaborative environments, poised to drive innovation and contribute to organizational success. He can be contacted at email: adarshu4321@gmail.com.

David de Hilster has over 30 years experience in practical NLP and is a consulting software engineer for LexisNexis and is currently part of the HPCC systems super computing group. Before joining the company, he worked as a senior engineer or scientist for text analysis international, I-Search, TRW, McDonnell Douglas, and Battelle Memorial Institute where he developed practical NLP tool kits and systems. He holds a Bachelor's degree in Mathematics and a Masters degree in Linguistics from Ohio State University. He is the architect of VisualText and co-architect of the language NLP++ and is responsible for creating the VisualText open-source project, the port of VisualText to VS Code, and creating the NLP engine which runs on Windows, Mac, and Linux. He can be contacted at email: David.Dehilster@lexisnexisrisk.com.

Dr. Shobha Gangadhar is Professor at Department of Computer Science, and Engineering, R.V College of Engineering, Bengaluru, India. She has teaching experience of 28 years, her specialization includes data mining, machine learning, and image processing. She has published more than 150 papers in reputed journals or conferences. She has also executed sponsored projects worth INR 200 lakhs funded by various agencies nationally and internationally. She is a recipient of various awards such as the career award for young teachers 2007-08 constituted by the All India Council of Technical Education, Best Researcher award from Cognizant 2017, GHC Faculty Scholar for Women in Computing in 2018, IBM Shared University Research Award in 2019, HPCC Systems community recognition award 2020. She can be contacted at email: shobhag@rvce.edu.in.

Dr. Jyoti Shetty is Assistant Professor at Department of Computer Science and Engineering, R. V. College of Engineering, Bengaluru, India. She has 16 years of teaching and 2 years of industry experience. Her specialization includes data mining, machine learning, and cloud computing. She has published research papers in reputed journals and conferences. She has also executed sponsored projects funded by various agencies nationally and internationally. She was the recipient of awards such as the SAP Award of excellence from IIT Bombay for demonstrating ICT in education in 2016 and the HPCC systems mentor badge award in 2021 for providing guidance and direction towards the successful completion of intern open-source projects. She can be contacted at email: jyothis@rvce.edu.in.