Interpretable Natural Language Segmentation Based on Link Grammar

Vignav Ramesh
SingularityNET, Saratoga H.S.
Saratoga, CA, USA

Anton Kolonin
Novosibirsk State University
Novosibirsk, Russia

November 15, 2020

Overview

- Introduction
  - Natural Language Segmentation
  - Sources of Unsegmented Text
  - Motivations
  - Link Grammar
- Methodology
  - Loader
  - Segment
- Results
- Conclusion
  - Applications
  - Future Work

Introduction

Natural Language Segmentation

- Process of dividing written text into meaningful units
- Focus on the sub-problem of sentence segmentation
  - Dividing a string of natural language (NL) text into its component sentences that can then be used to build a parse tree
  - Proposed architecture **solely relies on grammatical relationships between tokens**
    - Generalizes to languages other than English or Russian

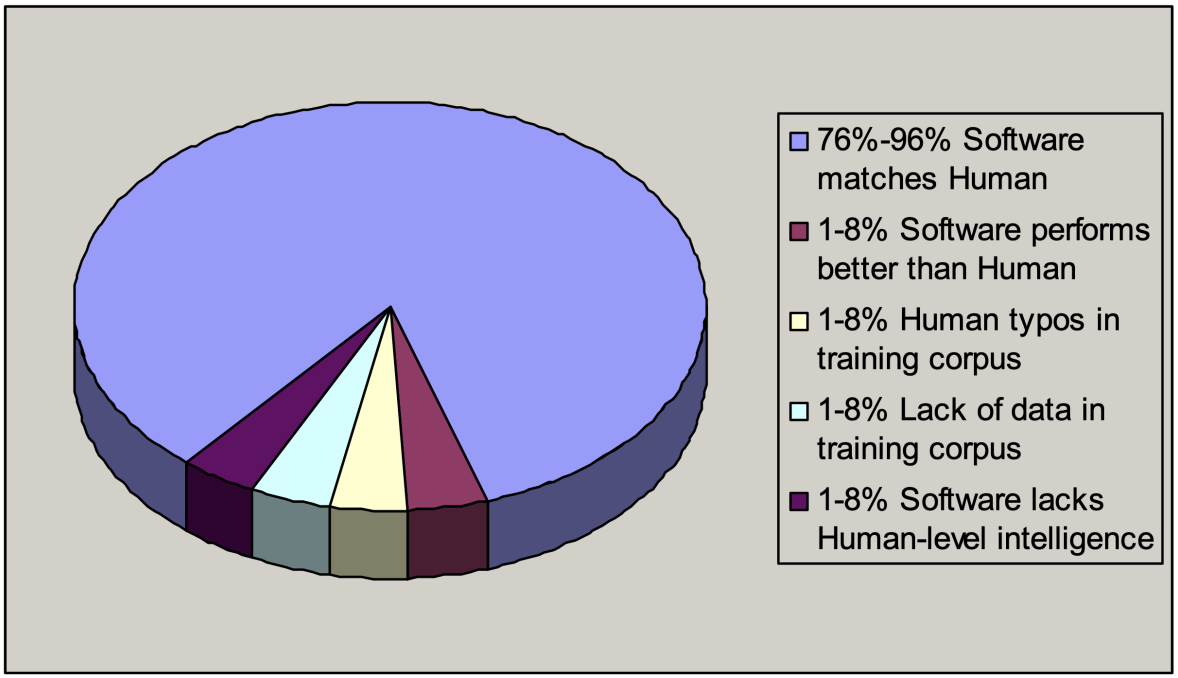
Sources of Unsegmented Text

1. Text output from speech-to-text (STT) recognition engines
   - STT engines transcribe audio content and return sequences of tokens **without delimeters**
   - Often used to preprocess the inputs of voice-activated question answering or chatbot pipelines (among various other scenarios involving the transcription of audio without recorded punctuation)
2. Crawled web pages
   - Returns raw HTML and CSS output that often **does not contain proper punctuation**
   - HTML and CSS tags are **neither accurate nor consistent** sentence boundaries

Motivations

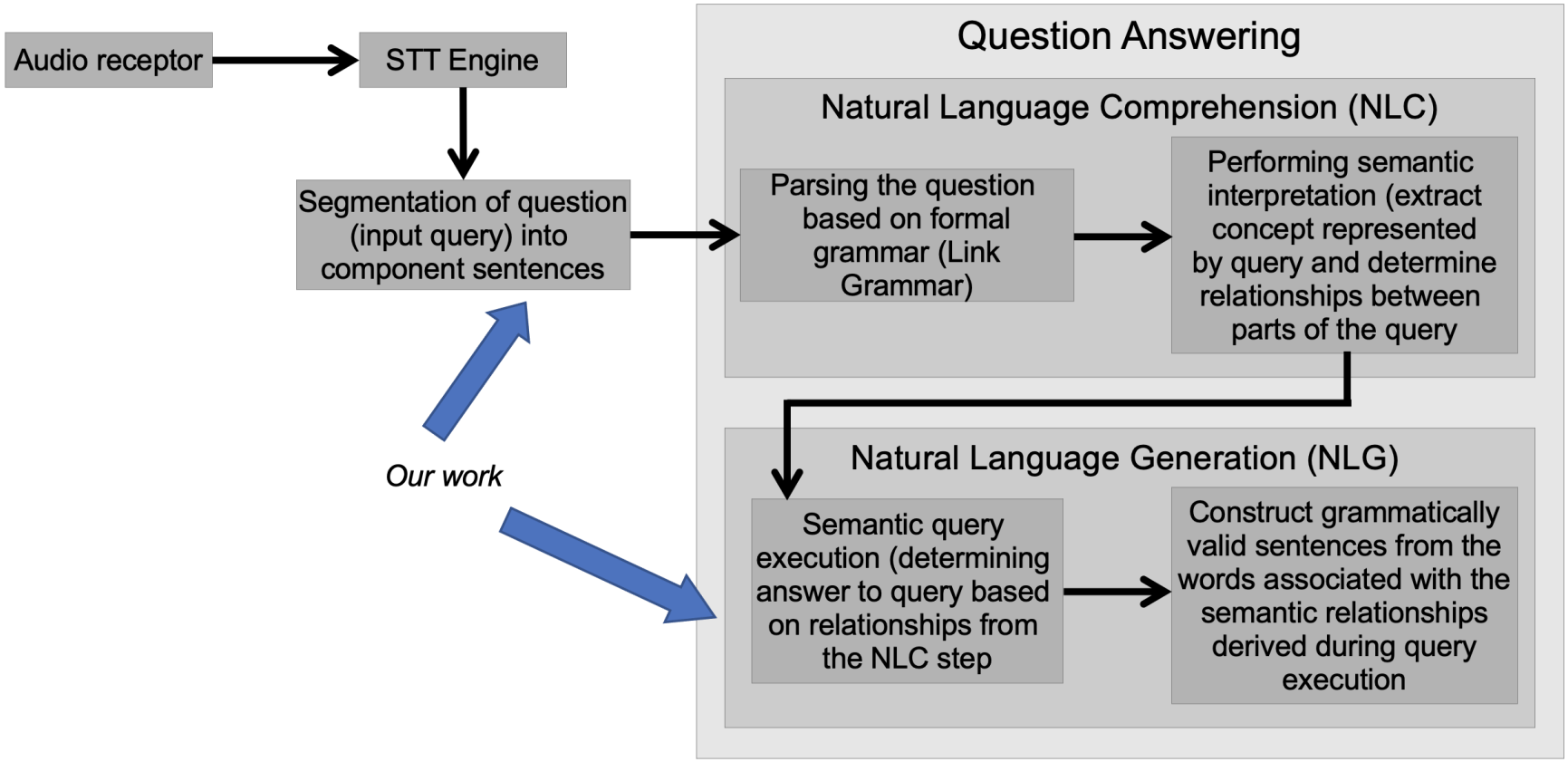
Interpretable Natural Language Processing

- Application of the XAI concept to NLP
- Allows for the learning of NL material, comprehension of bodies of text, and generation of textual material via logical and transparent methods that rely on interpretable models of formal grammars
- Mitigates blind spots in corpora, allows model errors to be located



http://webstructor.net/papers/Kolonin-HP-ACA-IC-text.pdf

Question Answering



Question Answering (cont.)

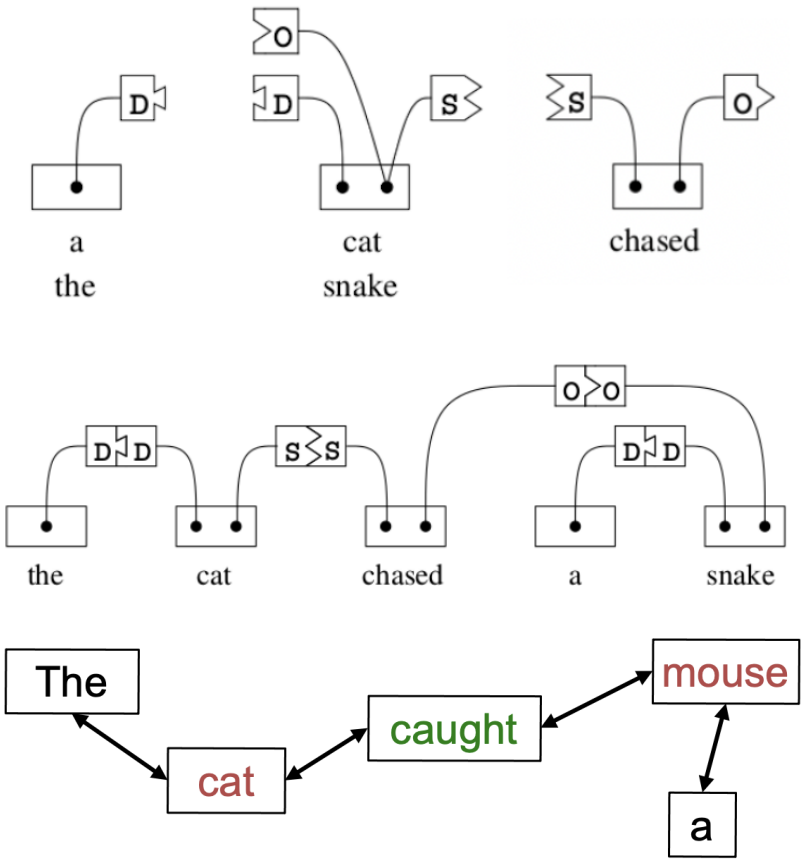
- Parsing the input query: separate the token sequence that represents the question into component sentences
- Semantic query execution: Segment documents matched with input query into component sentences
  - Most relevant subset passed on to NLG component of pipeline

Link Grammar

- Each rule describes a list of words corresponding to that rule and a set of defining disjuncts
- Disjuncts correspond to sets of typed connectors that define the valid use of a given word
- A sentence refers to a set of tokens that can be connected by matching up connectors between certain pairs of tokens
- Two directionally opposite connectors of the same type form a link in a parsed sentence
- The Link Grammar database contains dictionaries across more than 10 languages
    - English database contains approximately 1,430 distinct word clusters and 86,863 word forms

Link Grammar (cont.)

```
a the: D+;
cat snake: D- & (S+ or O-);
chased: S- & O+;
```

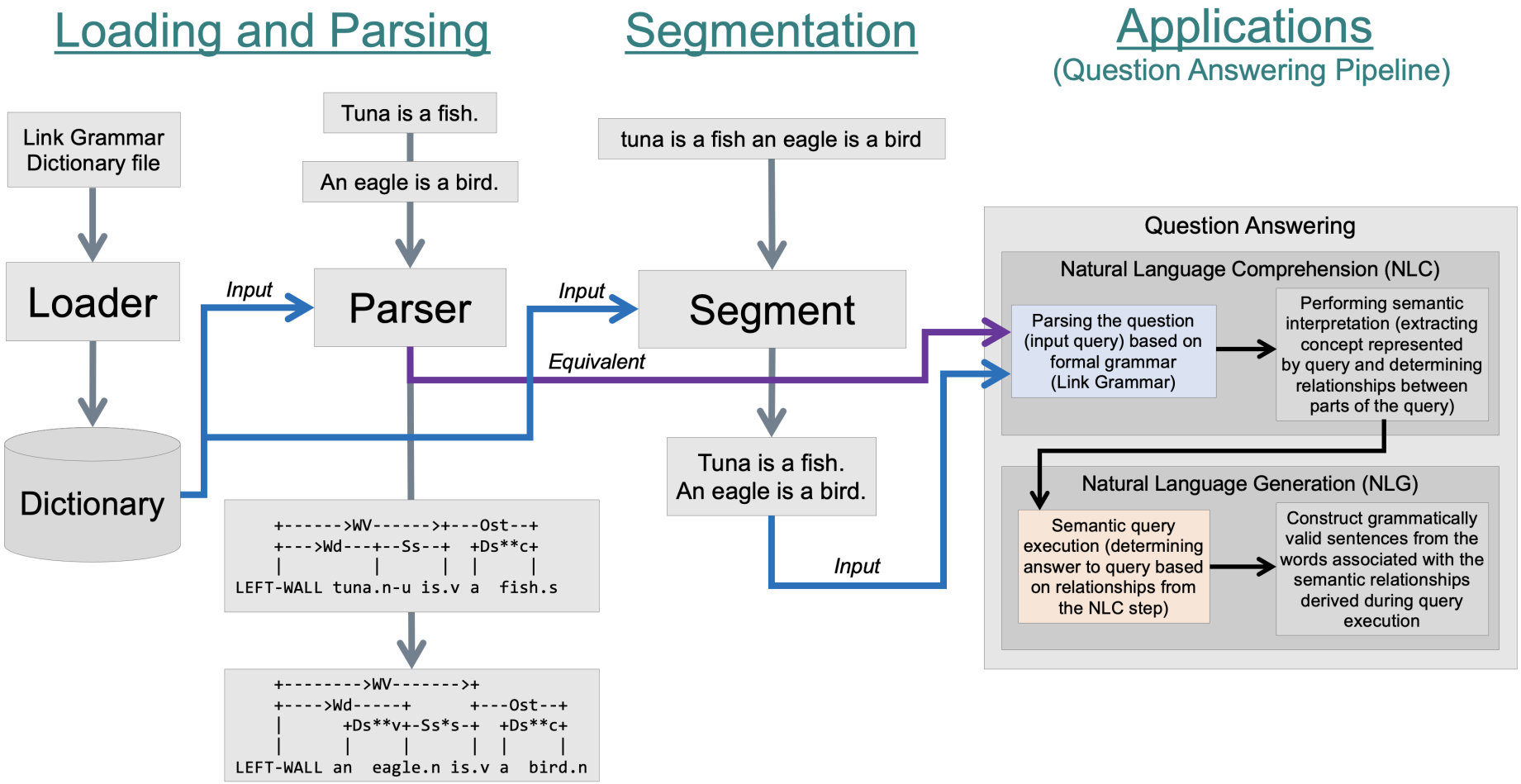


Why Link Grammar?

- Besides Link Grammar, other grammar rule dictionaries and APIs exist, such as spaCy and Universal Dependencies
- spaCy and UD rely on or are structured as a dependency grammar
  - Requires a head-dependent relationship, i.e. links must be directional; the English Link Grammar database does not require links to indicate direction, and thus can be applied to a greater variety of sentential forms
- No grammatical knowledge hardcoded into software programs, allowing the Link Grammar database to continually be updated and enhanced via any manual or automated input (such as a ULL system) without having to modify the client code
  - **Our proposed Loader architecture creates extensive value in providing the first native Java support for Link Grammar**
  - Human-readable and editable nature of Link Grammar allows our grammar induction algorithm to better serve as an XAI when integrated into the rest of our NLS architecture

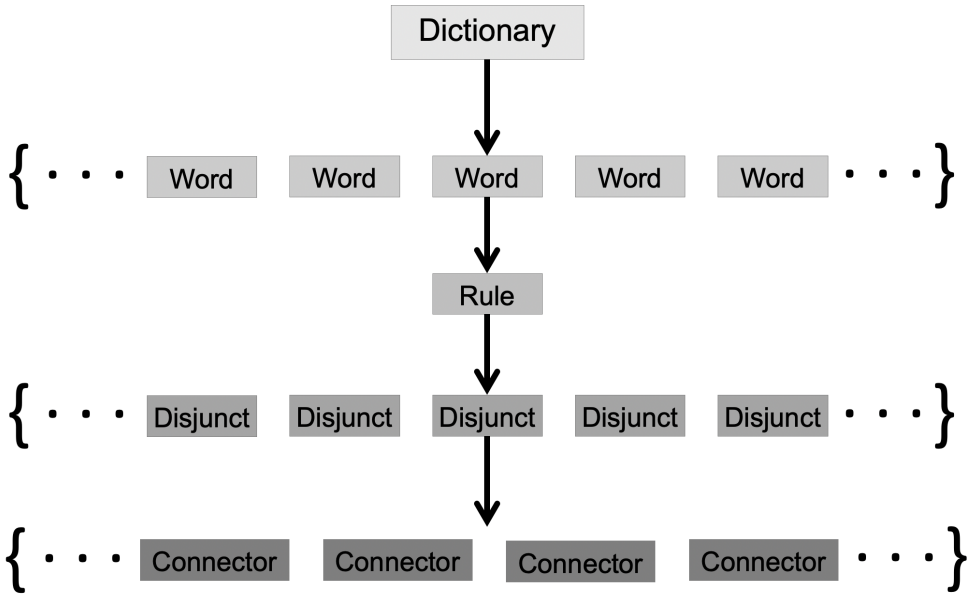
Methodology

Overall Architecture



Loader

- Loads and stores the Link Grammar database in various classes for future usage
- Currently, the Loader architecture only supports the English language and does not handle more complex morphological structures, including those needed to support languages such as Russian that require extensive morphology usage



Loader (cont.)

- MAKEDICT utilizes the information in array of lines obtained from the Link Grammar database to create an array of Dictionary objects
  - Extracts a Link Grammar rule from each element of *lines*, replacing each macro in the rule with its expanded expression and assigning the modified rule to the word it defines

Algorithm 1: MAKEDICT

Input : An array *lines* of all lines in the Link Grammar Database
Output: An array [*dict*, *hyphenated*] of Dictionary objects, one for common words and one for common phrases with words separated by underscores

Initialize *dict* and *hyphenated*
Initialize *macros*, which maps single links to the large connector expressions they define

define ASSIGN(w, r)**:**
if *w is a hyphenated phrase* **then**
  Add (w, r) to *hyphenated*
else
  Add (w, r) to *dict*
end
end

for *line in lines* **do**
  if *line starts with a macro* **then**
    Split the single link, *macro*, from its definition, *rule*
    Add $(macro, rule)$ to *macros*
  else
    if *line contains a filename f* **then**
      Parse f to obtain the list of words it contains
      Replace all instances of macros in the rule *rule* specified in the following lines of the Link Grammar database with their expanded definitions as contained in *macros*
      Store *rule* in a Rule object r
      for *word w in f* **do**
        ASSIGN(w, r)
      end
    else
      Split the word, w, from its definition, *rule*
      Process *rule* and store it in a Rule object r
      Replace all instances of macros in *rule* with their expanded definitions as contained in *macros*
      ASSIGN(w, r)
    end
  end
end
return [*dict*, *hyphenated*]

Segment

- Computes a segmentation for a string of text
  1. Extracts the tokens (words and punctuation) from that text via PROCESSSENTENCES
  2. Clusters those tokens into valid sentences as per the SEGMENT function
- SEGMENT loops through *tokens* and determines if certain subsets of *tokens* can form valid sentences via ISVALID and CHECK

Algorithm 2: SEGMENT

Input : An array $tokens$ of words and commas extracted from the input text by PROCESSSENTENCES

Output: A list of sentences obtained by segmenting $tokens$ into grammatically and morphologically valid arrays of tokens

Start a counter idx, representing the index of the current token in $tokens$
Initialize an empty list ret, which will eventually contain the sentences that SEGMENT will return

while $idx < length(tokens)$ **do**
  for i *in* $[idx, length(tokens)]$ **do**
    Create array arr containing the subset of $tokens$ from indices idx to i
    if ISVALID(arr) *and* CHECK($tokens[i+1], tokens[i+2]$) **then**
      $threshold = n$ (default value of 2)
      Add $tokens[i+1], tokens[i+2]$ ... $tokens[i+n]$ to arr
      if ISVALID(arr) *and* CHECK($tokens[i+n+1], tokens[i+n+2]$) **then**
        Construct a sentence from arr, i.e. create a string with the tokens in arr separated by spaces and add appropriate punctuation
        Add the sentence to ret
        $idx \leftarrow i+n+1$
      else
        Construct a sentence from the original value of arr
        Add the sentence to ret
        $idx \leftarrow i+1$
      end
    end
  end
end
return ret

Segment (cont.)

- isValid:
  - Determines if *arr* can form a valid sentence via Link Grammar rules by ensuring that every pair of consecutive words in *arr* can be connected by links in the Dictionary
  - Uses the CONNECTS function, which returns a boolean value indicating whether its two parameters, the tokens *left* and *right*, can be linked together

Algorithm 3: CONNECTS

Input : A pair of strings $left$ and $right$, representing the two words to potentially be connected

Output: An boolean value indicating whether $left$ and $right$ can be connected via valid Link Grammar rules

Obtain $leftList$, the list of rules corresponding with $left$ (i.e. the rule when $left$ is a verb, the rule when $left$ is a gerund, etc.), from the global Dictionary variables $dict$ and $hyphenated$

Obtain $rightList$ in a similar manner

```
for leftRule in leftList do
    for rightRule in rightList do
        Split leftRule and rightRule into lists of Disjuncts ld and rd
        for l in ld do
            for r in rd do
                Replace all instances of '−' in l with '+' and vice versa
                if l = r then
                    return true
                else
                    continue
                end
            end
        end
    end
end
return false
```

Segment (cont.)

- CHECK:
  - Determines if the first two tokens following arr satisfy initial checks of the planarity and connectivity metarules (e.g. first two tokens after *arr* can form links to the right and left, respectively)
- An example segmentation query is as follows: SEGMENT("tuna is a fish eagle is a bird dog is a mammal") = ["Tuna is a fish.", "Eagle is a bird.", "Dog is a mammal."]

Results

Evaluation

TABLE I. "SMALL WORLD" CORPUS NLS RESULTS

Metric	Result
Ground Truth (POC-English Corpus)	
Total number of sentences	88
Average sentence length	5.51136
NLS Algorithm Results	
Total number of sentences	87
Average sentence length	5.57471
Overall Statistics	
Runtime	57 sec
Number of sentences matching exactly	78
Number of sentence boundaries accurately identified	85/87
Accuracy of boundary identification	0.97701

TABLE II. GUTENBERG CORPUS NLS RESULTS

Metric	Result
Ground Truth (Gutenberg Corpus)	
Total number of sentences	10
Average sentence length	13.2
NLS Algorithm Results	
Total number of sentences	11
Average sentence length	13.2
Overall Statistics	
Runtime	14 sec
Number of sentences matching exactly	7
Number of sentence boundaries accurately identified	7/9
Accuracy of boundary identification	0.77778

Comparison with Prior Work

- Compared our NLS architecture to three widely used open-source sentence segmentation frameworks: Syntok, PragmaticNet, and DeepSegment
- Syntok and PragmaticNet
  - Syntok computes segmentations by recognizing "terminal markers"
  - PragmaticNet is an unsupervised, opinionated, and conservative segmentation framework that segments text into sentences based on punctuation, quotations, and parentheticals
  - Both Syntok and PragmaticNet identified zero boundaries in the text from both the POC-English and Gutenberg corpora
- DeepSegment
  - Utilizes bidirectional long short-term memory networks (BiLSTMs) in a CRF based supervised segmentation model aimed at segmenting unpunctuated bodies of text into sentences
  - Identified only 1 segmentation boundary in the POC-English corpus and 2 boundaries in the Gutenberg corpus (all of which were accurate), yielding accuracies of 1.15% and 22.2%

Grammatical Ambiguity

- Situations in which the same word may take on different parts of speech (such as the word "saw" in its verb and noun forms)
- "dad has a hammer mom has a hammer"
  - Appropriate to segment this text into the two sentences "Dad has a hammer" and "Mom has a hammer" because the words "hammer" and "Mom" cannot be connected
  - Only true for the noun form of "hammer"; the verb form of "hammer" can technically be linked to the object "Mom"
- **Solution: Semantic (word sense) disambiguation**
  - Determines which "sense" or definition of a word is activated by that word's use in a particular context
  - Current unsupervised semantic disambiguation methods include dictionary-based algorithms that utilize knowledge encoded in lexical resources to learn the senses of words

Conclusion

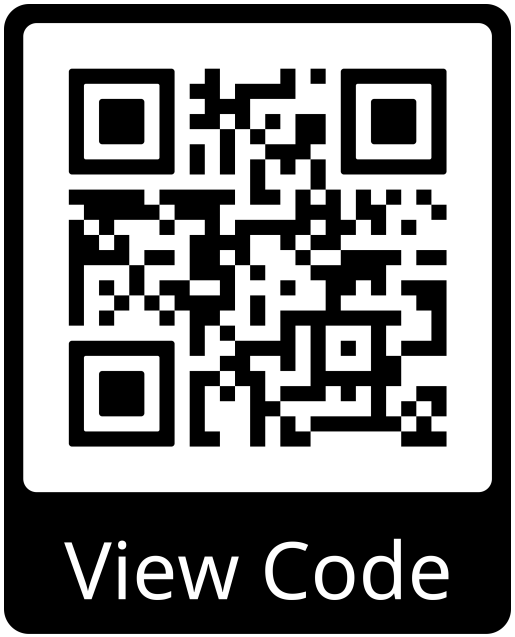
Applications

1. Semantic query execution component of the question answering pipeline
   - Aigents Social Media Intelligence Platform
   - Currently only handles oversimplified "pidgin" English
   - Can enable Aigents to support question answering from spoken audio input transcribed by STT as well as extract information from crawled web pages to answer such questions
2. Text simplification
   - Enhance a corpus of human-readable text as to simplify the grammar and structure of the text while maintaining the original meaning
   - "Mom saw Dad, who saw Mom sawing." $\longrightarrow$ ["Mom saw Dad," "Dad saw Mom," "Mom was sawing."]
   - Improve the quality of corpora that would otherwise contain vocabulary and complex sentence constructions not easily processed via computational means
3. Any NLP algorithms that operate at the sentential level
   - Automatic summarization, entity extraction, sentiment identification

Future Work

1. Implement grammatical and semantic disambiguation
2. Extend the algorithm's segmentation capabilities to languages other than English

Thank you!



- Code & Data: https://github.com/aigents/aigents-java-nlp
- Contact: rvignav@gmail.com