โครงงานเลขที่ วศ.คพ. S020-2/2565

เรื่อง

วิซิวิกสำหรับตรวจการใช้ไวยากรณ์ภาษาไทย

โดย

นายอภิสิทธิ์ ฤทธิ์เรืองโรจน์ รหัส 620610820

โครงงานนี้
เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต
ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ มหาวิทยาลัยเชียงใหม่
ปีการศึกษา 2565

PROJECT No. CPE S020-2/2565

WYSIWYG typing assistant and Thai grammar checker

Apisit Ritreungroj 620610820

A Project Submitted in Partial Fulfillment of Requirements
for the Degree of Bachelor of Engineering
Department of Computer Engineering
Faculty of Engineering
Chiang Mai University
2022

หัวข้อโครงงาน	: วิซิวิกสำหรับตรวจการใช้ไวยากรณ์ภาษาไทย
	: WYSIWYG typing assistant and Thai grammar checker
โดย	: นายอภิสิทธิ์ ฤทธิ์เรืองโรจน์ รหัส 620610820
ภาควิชา	: วิศวกรรมคอมพิวเตอร์
อาจารย์ที่ปรึกษา	: อ.ดร. ชินวัตร อิศราดิสัยกุล
ปริญญา	: วิศวกรรมศาสตรบัณฑิต
สาขา	: วิศวกรรมคอมพิวเตอร์
ปีการศึกษา	: 2565

ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ มหาวิทยาลัยเชียงใหม่ ได้อนุมัติให้โครงงานนี้เป็นส่วน-หนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต (สาขาวิศวกรรมคอมพิวเตอร์)

. . . . หัวหน้าภาควิชาวิศวกรรมคอมพิวเตอร์
(รศ.ดร. สันติ พิทักษ์กิจนุกูร)

คณะกรรมการสอบโครงงาน

. . . . ประธานกรรมการ
(อ.ดร. ชินวัตร อิศราดิสัยกุล)

. . . . กรรมการ
(ผศ.ดร. กานต์ ปทานุคม)

. . . . กรรมการ
(อ.ดร. พฤษภ์ บุญมา)

หัวข้อโครงงาน	: วิซิวิกสำหรับตรวจการใช้ไวยากรณ์ภาษาไทย
	: WYSIWYG typing assistant and Thai grammar checker
โดย	: นายอภิสิทธิ์ ฤทธิ์เรืองโรจน์ รหัส 620610820
ภาควิชา	: วิศวกรรมคอมพิวเตอร์
อาจารย์ที่ปรึกษา	: อ.ดร. ชินวัตร อิศราดิสัยกุล
ปริญญา	: วิศวกรรมศาสตรบัณฑิต
สาขา	: วิศวกรรมคอมพิวเตอร์
ปีการศึกษา	: 2565

บทคัดย่อ

ในปัจจุบัน งานเขียนภาษาไทยหลายประเภท เช่น ในงานวิชาการ ธุรกิจ และงานราชการ ต้องใช้ความเป็นทางการสูง แม้ว่านักเขียนอาจเขียนในลักษณะนี้ทุกวัน แต่ก็ยังมีแนวโน้มที่จะทำผิดพลาดและใช้เวลามากในการพิสูจน์อักษรก่อนตีพิมพ์

จากปัญหาดังกล่าว ผู้จัดทำจึงได้คิดค้นเครื่องมือแก้ไขคำขณะพิมพ์จะช่วยให้ผู้เขียนเหล่านี้ประหยัดเวลาในการพิสูจน์อักษรและส่งเสริมการใช้ภาษาไทยให้มีคุณภาพดีขึ้นและให้คนไทยทั่วไปได้มาใช้งานกัน โดยจะสร้างตัวตรวจสอบไวยากรณ์ภาษาไทยให้เป็นส่วนขยายเบราว์เซอร์ WYSIWYG ซึ่งผู้ใช้สามารถพิมพ์ได้อย่างอิสระ ในขณะที่ตัวตรวจสอบทำงานในพื้นหลังและวิเคราะห์คำ วลี และประโยคของผู้ใช้งาน ส่วนขยายทำหน้าที่เชื่อมต่อกับเซิร์ฟเวอร์ และส่งคืนการแก้ไขและคำแนะนำทางไวยากรณ์ ซึ่งผู้ใช้สามารถเลือกสิ่งที่เข้ากับบริบทที่สุดได้ นอกจากนี้ เครื่องมือนี้ยังสามารถให้ผลลัพธ์ของการวิเคราะห์วากยสัมพันธ์ในรูปแบบโครงสร้างแผนผังไวยากรณ์สำหรับนักเขียนมืออาชีพเพื่อตรวจสอบและทำความเข้าใจประโยคในระดับที่ลึกขึ้น และเพื่อให้ครูสามารถช่วยให้นักเรียนเข้าใจไวยากรณ์ไทยได้ดีขึ้นจากการแสดงภาพประโยค

Project Title	: WYSIWYG typing assistant and Thai grammar checker
Name	: Apisit Ritreungroj 620610820
Department	: Computer Engineering
Project Advisor	: Chinawat Isradisaikul, Ph.D.
Degree	: Bachelor of Engineering
Program	: Computer Engineering
Academic Year	: 2022

ABSTRACT

Many types of Thai writing, such as in academic, business, and official communications, require high formality. Although writers might write in such a way every day, they still tend to commit mistakes and spend a substantial amount of time proofreading before publication. A tool that corrects words as they type will help these authors save time for proofreading and promote the use of the Thai language in better quality for Thai people in general.

This project aims to construct a Thai grammar checker as a WYSIWYG browser extension where users can type freely while the checker works in the background and analyzes their words, phrases, and sentences. The browser extension connects to a service that returns corrections and grammatical suggestions from which the users can choose what's most appropriate.

Acknowledgments

I would like to express my deep appreciation for Chinawat Isradisaikul, Ph.D. for his exceptional dedication, expertise, and detailed guidance throughout the project, which served as a constant source of inspiration and helped ensure its successful completion.

Apisit Ritreungroj
1 April 2023

Contents

List of Figures

List of Tables

Chapter 1
Introduction

1.1 Project rationale

Nowadays, there are many advancements in computational linguistics. in English, there are so many grammar checker vendors out there, e.g., Grammarly [10]. But there are some differences between Thai and English as word segmentation remains a fundamental challenge in the Thai language since it does not require spaces to separate words. And the process of segmentation can cause ambiguity if the context provided is not enough, such as 'กอดอก' is segmented into either 'กอ|ดอก' or 'กอด|อก'. This is, so far, challenging to create a grammar checker for Thai.

This project dedicates to constructing a basic grammar checker for the Thai language. It is planned for normal people to freely use and access this service publicly. In this report, all the concepts and implementation details are explained. The core language model is implemented in highly optimized algorithms and partially integrated with PyThaiNLP modules [17]. The report also includes details about the process of collecting dictionary data from the Royal Institute Dictionary [32] through the process of serving suggestions to users.

1.2 Objectives

- To create a Thai grammar checker service for people to use publicly.

1.3 Project scope

The project is a web-based application capable of performing the following tasks:

- autocorrection [27]
- grammatical error detection
- character arrangement, e.g., the upper vowel and the lower vowel cannot sit at the same consonant or position
- wrong or outdated character usage, e.g., usage of 'เเ' (double 'เ') to act as 'แ' is incorrect
- sentence, word, and conjunction spacing
- punctuation usage.

There are also tasks we do not consider to be in the scope, which is mostly about machine learning, as the project primarily focuses on the foundation of the language grammar. This does not mean that they will not be implemented in further development. These tasks include:

- English compatibility
- paraphrasing [31] and clarifying the sentence
- reference ambiguity and unclear detection
- monotonous sentences detection
- positive or negative sentences detection
- sarcastic sentences detection
- overall tone of the sentences detection
- named-entity recognition and Wikipedia linking
- plagiarism detection in writings
- copywriting [29]
- parsing unstructured text
- summarization [28]

1.3.1 Hardware scope

All PCs, tablets, and smartphones that can use a modern browser and access the internet. A modern browser is a browser which at least compatible with HTTP2 connections and TLS1.2–1.3 encryption, and generally supports modern JavaScript syntax (ES6) and modern media compression, e.g., `.webp` (image) and `.woff2` (font).

1.3.2 Software scope

The project is a web application, available as a Chrome-based browser extension.

1.4 Expected outcomes

- The service is readily available online to the general public.
- Minimizing the duration individuals allocate to proofread their writing.
- People write Thai in the official context more correctly.

1.5 Technology and tools

A variety of tools are required in this project for building complex application services. Since the services in this project are cloud-based, the technology mainly involves web services and practices.

1.5.1 Hardware technology

Two DigitalOcean's cloud-based IaaS instances (Droplets) for back-end services deployment. One is for the main backend service, and one is for hosting the database. Both instances use a shared Intel 1vCPU, 1GiB / 25GiB SSD disk.

1.5.2 Software technology

Programming languages

- Rust [19]
- TypeScript [25]

Tools

- Docker (container and deployment) [7]
- Debian (Linux distro for deployment) [4]
- DigitalOcean (cloud platform) [6]
- Solid (front-end framework) [22]
- SCSS (style sheet language, compiled to CSS) [8]
- Vite (front-end build tool) [26]
- Actix (back-end framework) [1]
- SeaORM (back-end ORM) [21]
- PostgreSQL (RDBMS) [16]
- DBML (DSL for define database schemas) [3]
- Protocol Buffers (payload serialization language) [9]

Network and architecture

- HTTP2 [30] with TLS1.3 [11]
- token-based authentication [14]
- client-server architecture [24]
- automated deployment pipeline [15]

1.6 Project plan

Task	Apr 2022	May 2022	Jun 2022	Jul 2022	Aug 2022	Sep 2022	Oct 2022	Nov 2022	Dec 2022	Jan 2023	Feb 2023
Planning and research	■	■	■								
Dictionary and language module implementation			■	■	■	■	■	■	■		
Back-end implementation							■	■	■		
Front-end implementation								■	■		
Testing and evaluation										■	■

1.7 Roles and responsibilities

Apisit Ritreungroj holds the role and responsibility for all aspects of the project, encompassing planning, research, design, management, and execution.

1.8 Impacts of this project on society, health, safety, legal, and cultural issues

The project's primary objective is to provide a practical solution for proofreading various types of writing in the Thai language, thereby minimizing the time spent on proofreading each publication and writing. However, there is apprehension that the implementation of this solution may negatively impact those who rely on proofreading as their profession [18].

Chapter 2
Background Knowledge and Theory

Our grammar-checking service needs to have modules that can process the Thai language, e.g., tokenization, named-entity recognition (NER), and dependency parsing. Those modules are contributed to PyThaiNLP [17], an open-source NLP toolkit for Thai. The PyThaiNLP module is well-equipped with many tools, covering all the needs of the project. The drawback is that the module is implemented in Python, which is resource-consuming and slow [2] and not suitable for low-end servers for cost-saving purposes. To avoid this drawback, the project will use a compiled programming language, Rust [19]. Therefore, the project reimplements some parts of the PyThaiNLP module, e.g., the TCC tokenization engine, into the Rust implementation as it has solid grammar rules to work with and also gains a performance boost.

The language validation modules in the project are created from scratch with Rust. The part of tokenization, whether the grapheme tokenization or TCC tokenization process, can break into grammar rules, and the engine needs to validate each of them in smaller parts, which PyThaiNLP does not have this validation process. The project also uses its own parser, which means implementing the Thai grammar parser from scratch using GLR parser to parse Thai phrases and to handle the language's ambiguity.

2.1 Comprehensive Thai grammar knowledge

The fundamental Thai grammar knowledge uses in this project, e.g., the study of word spelling, word usage, tonal marks usage on a consonant, and punctuation usage. In order to correct the misspelled words in writing, the study of correcting words is applied. A misspelled word such as 'กระเพรา' needs to be corrected as 'กะเพรา'. There are many words out there that people usually misspelled [34]. In a formal context, punctuation usage is also very important, for example, the repetition character 'ๆ' needs to have a single space before and after the character. Furthermore, the formation of sentences is studied in the project as words need to form together into sentences.

2.2 Natural language processing (NLP)

The practices of NLP are involved generally in the project. The process of text cleaning and preprocessing is applied in the part of fetching words from the Royal Institute Dictionary. The fetched words are embedded in various HTML formats. Hence those words need to be processed and stored in the structured form before the service can use them. Word segmentation is also applied in the selection of techniques used in this project, e.g., the maximum-matching algorithm.

2.3 Compiler design

Compiler design knowledge is crucial in this project as it has to implement its very own parser to handle the ambiguity of Thai language grammar. The type of parser that is used in the project is the GLR parser as it can generate multiple parse trees, allowing for a more accurate and comprehensive analysis of the sentence structure. By using a GLR parser, we can achieve better parsing results and improve the accuracy of natural language processing tasks.

2.4 Cloud engineering and web development

Web development is used to create an interactive interface for users to interact with the service. The project used a lot of web development practices, e.g. framework usage, built tools usage, styling and layout, accessibility, and responsive design.

Cloud knowledge is involved in the process of the web client and back-end service deployment, and also in the security configuration of the containers and the virtual machines.

2.5 UX/UI Design

The knowledge applies to the practices of designing an easy-to-use and appealing to attract users. The design knowledge mainly applied in this project includes color theory, usability heuristics, and UX design psychology.

2.6 CPE knowledge used, applied, or integrated in this project

- 261207 – CPE Lab: applied in web service development
- 261217 – Data Structures: applied in creating performant data flow and dictionary datastore design
- 261218 – Algorithms: applied in backend process optimization
- 261342 – Database: applied in database schemas design
- 261361 – Software Engineering: applied in practices during the development process
- 261335 – Networks: applied in designing the secure service over the internet
- 269462 – Human-Computer Interaction: applied in usability heuristics of the application design

2.7 Extracurricular knowledge used, applied, or integrated in this project

The obvious one that cannot lack is *linguistics* knowledge. Huge linguistics, especially Thai linguistics, was applied in this project to create programming that can analyze a whole sentence and even a whole article. The second is *data science* knowledge to deal with huge amounts of data, e.g., linguistics corpus and dictionary manipulation of over 80,000 entries. The last is the knowledge of *natural language processing* (NLP), used during the development process of the project, e.g., text preprocessing after fetching words from the Royal Institute Dictionary and other websites, i.e., Royal Institute's transliteration website, Wiktionary, PyThaiNLP's main repository, etc. Other minor knowledge applied in the project includes *UI Design*; knowledge about the programming language used in the project, e.g., *Rust*; and the practice of *DevOps*.

Chapter 3
Project Structure and Methodology

Many components and modules comprise the entire grammar-checking service. Each is a standalone service communicating with one another to form a seamless service.

The service consists of 4 main modules: the dictionary management system, the language core, the back-end service, and the front-end service. The dictionary management system manages the corpus for the language core module. The language core is where all the language validation processes are. The back-end is a part that delivers the language core to the internet via API endpoints. Finally, the front-end is where the user interacts with the service.

3.1 Dictionary management system

This module enables other services to communicate with the dictionary. It performs querying words and adding new words on the request. There is only a single dictionary management system per back-end service. When any changes occur to the dictionary Protobuf words and definitions file, the whole service needs to be redeployed.

Dictionary words and their definitions are saved in the Protobuf format. Thanks to Protobuf the program can start faster over 100 times using JSON format, which means, we don't have to wait in the initialization for a long period for each run.

Words and their definitions are saved in the below Protobuf format:

```
syntax = "proto3";

package docs.dictionary;

message Dictionary {
  message Entry {
    string word = 1;
    repeated WordDef defs = 2;
  }

  message WordDef {
    optional string pron = 1;
    optional string role = 2;
    DefPOS pos = 3;
    string meaning = 4;
```

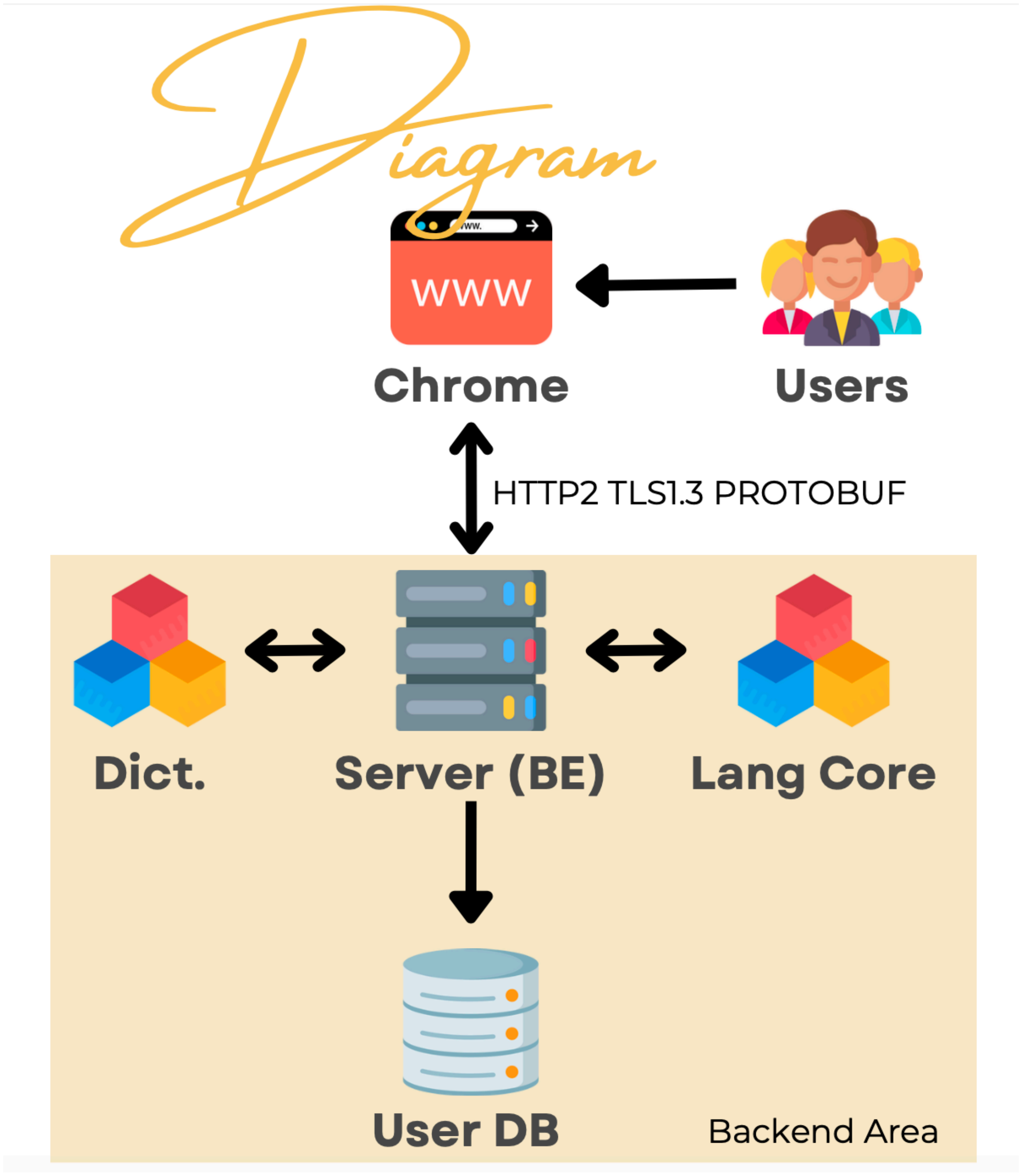


Figure 3.1: The overview architecture of the service

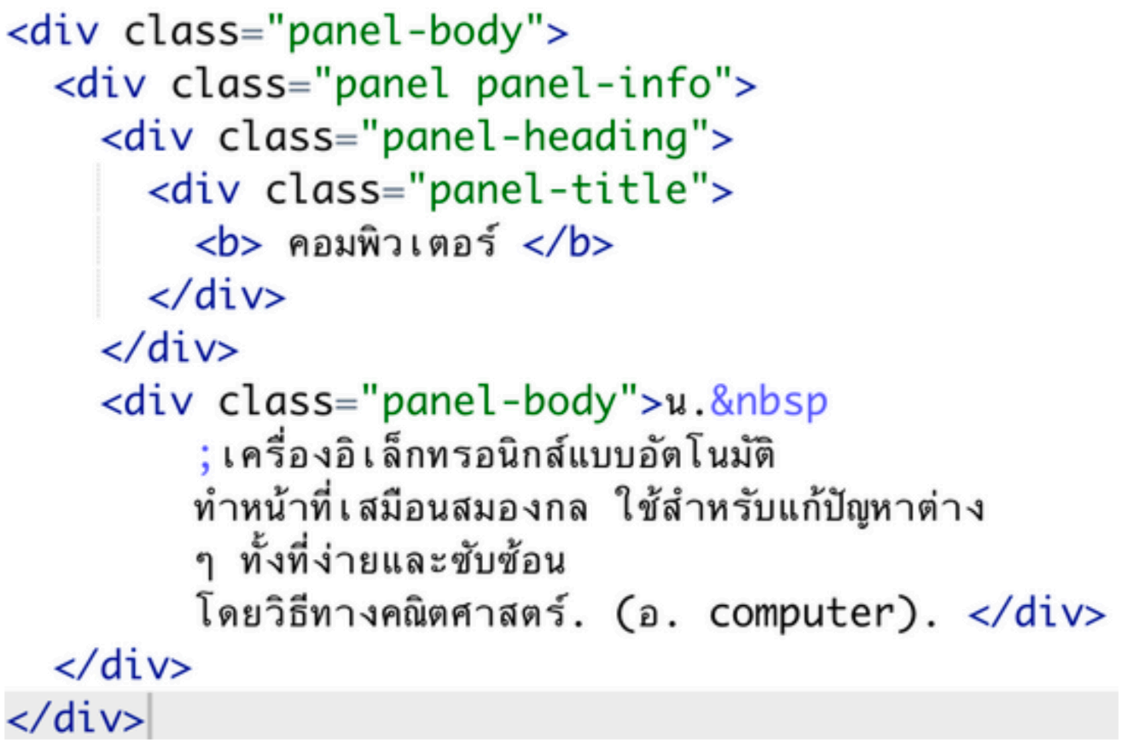
```
    optional string origin = 5;
  }

  enum DefPOS {
    V = 0;
    N = 1;
    PREP = 2;
    ADV = 3;
    PRON = 4;
    CONJ = 5;
    INTERJ = 6;
  }

  repeated Entry entries = 1;
}
```

3.1.1 Words gathering

This is the process of gathering words and their meaning from various websites

Here is the list of websites that we brought from:

1. `https://dictionary.orst.go.th` — The main website that hosts all Thai words and their definitions. In the fetching process, raw words embedded inside HTML are returned.
2. `https://transliteration.orst.go.th` — The website hosts Thai transliteration words from various languages not limited to English but the distribution of English words is three-quarters of them. JSON data are returned from the process of fetching.
3. `https://th.wiktionary.org/wiki/หมวดหมู่:การย่อภาษาไทย` — The website hosts abbreviations of a word, e.g., ‘`อาจารย์`’ can be abbreviated as ‘`อ.`’ or ‘`อจ.`’. We do fetch the whole page of the website, hence HTML embedded data are returned.
4. `https://github.com/rathpanyowat/Thai-zip-code-latitude-and-longitude` — Name entities of location in Thailand.
5. `https://github.com/PyThaiNLP/pythainlp` — Name entities of men, women, and families. And we also gather some Thai transliteration words.
6. `http://legacy.orst.go.th/?page_id=641` — Noun classifiers are gathered from this website. There is no way to fetch the whole page since changing from the category



```
<div class="panel-body">
  <div class="panel panel-info">
    <div class="panel-heading">
      <div class="panel-title">
        <b> คอมพิวเตอร์ </b>
      </div>
    </div>
    <div class="panel-body">น.&nbsp
        ;เครื่องอิเล็กทรอนิกส์แบบอัตโนมัติ
        ทำหน้าที่เสมือนสมองกล ใช้สำหรับแก้ปัญหาต่าง
        ๆ ทั้งที่ง่ายและซับซ้อน
        โดยวิธีทางคณิตศาสตร์. (อ. computer). </div>
  </div>
</div>
```

Figure 3.2: Embedded word and its information are embedded inside HTML

'ก' to 'ฃ' is still at the same URL. So we have to copy HTML data manually for all alphabetical categories.

3.1.2 Text preprocessing

Node.js is used in text processing and also fetching data from websites. The HTML parser is used to parse raw HTML-embedded words, then the words and their information can be extracted. The script has to locate which part needs to be extracted as different websites use different formats. After text preprocessing, the extracted words and information will be saved into the Protobuf format according to the specified structure.

3.1.3 Words validation

After text preprocessing and extracting words, We validate words with defined RegEx rules, and end up encountering 2 words that are malformatted from Royal Institute Dictionary, i.e., 'กระแสน้ำ' and 'ย่ำอยู่กับที่'. The word 'กระแสน้ำ' doesn't use '-ำ' correctly, and the word 'ย่ำอยู่กับที่' places the tonal mark twice at the grapheme 'ที่'. Hence we need a special case to handle these words.

3.2 Language core

The language core is the core service, which validates users' text and returns the result as correction suggestions. To achieve real-time results and high concurrency in low-end servers, it is implemented in the Rust programming language as it is performant [35], memory-safe [20], and low in memory footprint [23]. The process is defined in the form of pipeline flow split into stages.

3.2.1 Validation pipeline

The validation pipeline analyzes text in stages to validate its correctness. The pipeline has three stages: tokenization, parsing, and contextual analysis.

Tokenization

The first stage is tokenization, where every single word is tokenized. Tokenization itself is split further into three steps: Grapheme tokenization, TCC tokenization, and word tokenization.

1. Grapheme tokenization is the first step of the validation process. It splits text into *graphemes*, i.e., vertical tokens; for example, the word 'น้ำท่า' will be split into น้ำ|ท่|า. This step validates arrangements of vertical vowels and consonant characters; for example, 'จุ๊' would be incorrect, since the upper vowel and the lower vowel cannot appear simultaneously in a grapheme.

   Here is the RegEx that is used to tokenize text into graphemes:

```
lexer! {
      let low_consonant = [ 'พ' 'ภ' 'ค' 'ฅ' 'ฆ' 'ฟ' 'ฑ' 'ฒ' 'ท' 'ธ' 'ซ' 'ช' '
      let middle_consonant = [ 'ก' 'จ' 'ฎ' 'ด' 'ฏ' 'ต' 'บ' 'ป' 'อ' ];
      let high_consonant = ['ผ' 'ฝ' 'ฐ' 'ถ' 'ข' 'ฃ' 'ศ' 'ษ' 'ส' 'ห' 'ฉ'];

    let consonant = $low_consonant | $middle_consonant | $high_consonant;

    let independent_vowel = [ 'ฤ' 'ฦ' ];

      let leading_vowel = [ 'เ' 'แ' 'โ' 'ใ' 'ไ' ];
      let trailing_vowel = [ 'ะ' 'า' 'ๅ' ];
      let upper_vowel = [ '\u{e31}' '\u{e33}'-'\u{e37}' ];
      let lower_vowel = [ '\u{e38}'-'\u{e39}' ];
```

```
  let dependent_vowel = $leading_vowel | $trailing_vowel | $upper_vowel | $lowe

  let vowel = $independent_vowel | $dependent_vowel;

    let tonal_mark = [ '\u{e48}'-'\u{e4b}' ];
    let special_mark = [ '\u{e3a}' '\u{e4d}' '\u{e4e}' ];
    let taiku = [ '\u{e47}' ];
    let thanthakhaat = [ '\u{e4c}' ];

  let mark = $tonal_mark | $special_mark | $taiku | $thanthakhaat;

  let numeral = [ '๐'-'๙' ];

  let symbol = [ '฿' ];

  let punctuation = [ 'ฯ' 'ๆ' '๏' '๚' '๛' ];

  let space = [' ' '\t' '\n']+ | "\r\n";

let thai_char = $consonant | $vowel | $mark | $numeral | $symbol | $punctuation

let other_char = _ # $thai_char;

// composit vars
let thanthakhaat_accent = [ '\u{e34}' '\u{e38}' ];
let tmp_k = $consonant $thanthakhaat_accent? $thanthakhaat;
let tmp_k_alt = $consonant $thanthakhaat $thanthakhaat_accent?;

let karan = $tmp_k | $tmp_k_alt;

rule Init {
  $space,
  $leading_vowel | $trailing_vowel | $numeral | $symbol | $punctuation | $indep
  $karan,
  $consonant $taiku,
  $consonant ($upper_vowel | $lower_vowel)? $tonal_mark?,
  $consonant $tonal_mark? ($upper_vowel | $lower_vowel)?,
  }
}
```

2. TCC (*Thai character cluster*) tokenization follows. It joins grapheme tokens into TCC tokens. TCC is the combination of horizontally independent vowels, such as 'า' and 'แ', and one or more consonant characters together. These vowels cannot appear alone and must connect to a consonant character. Examples of a TCC token: 'เพลิง' is เพ|ลิ|ง, 'น้ำท่า' is น้ำ|ท่า. This step also validates the character arrangements in the horizontal axis; for example, 'โโ' or 'เา' would be definitely wrong grammatically.

   Here is the RegEx that is used to tokenize graphemes into TCCs:

```
lexer! {
  rule Init {
    $space,
    $new_line,
    ($$alphabetic # $thai_char)+,
    $$ascii_digit+
    $$ascii_punctuation,
    $symbol | $punctuation,
    $other_char,
    $independent_vowel,
    $numeral+,
    $independent_vowel 'ๅ',
    $karan,
    $consonant $taiku,
    $consonant $taiku 'อ' $consonant,
    'เ' $consonant? $consonant $taiku $consonant,
    'แ' $consonant? $consonant $taiku $consonant,
    'เ' $consonant '\u{e35}' $tonal_mark? 'ย' 'ะ'?,
    'เ' $consonant '\u{e37}' $tonal_mark? 'อ' 'ะ'?,
    'เ' $consonant '\u{e34}' $tonal_mark? $karan? $consonant,
    'เ' $consonant? $consonant $tonal_mark? 'อ' 'ะ',
    $consonant '\u{e37}' $tonal_mark? $consonant,
    'เ' $consonant $tonal_mark? 'า'? 'ะ'?,
    'เ' $consonant? $consonant $tonal_mark? 'า' 'ะ'?,
    ('แ' | 'โ') $consonant $tonal_mark? 'ะ'?,
    $consonant '\u{e31}' $tonal_mark? 'ว' 'ะ'?,
    $consonant '\u{e31}' $tonal_mark? $consonant ($thanthakhaat_accent | 'า'),
    $consonant $tonal_mark? 'ะ'?,
    $consonant '\u{e31}' $tonal_mark? $consonant},
    ('ไ' | 'ใ') $consonant $tonal_mark?,
    $consonant $tonal_mark? ('า' | 'ำ'),
    $consonant ($lower_vowel | $upper_vowel)? $tonal_mark?,
    $consonant $tonal_mark? ($lower_vowel | $upper_vowel)?,
  }
}
```

3. Word tokenization is the last step of tokenization. It forms TCCs into words by using a maximum-matching algorithm against the words in the dictionary, which can produce all possible words that can be matched. Also, This step involves autocorrections, which attempt to correct words if they do not match any word in the dictionary. Finally, after matching all words, the topological ordering is used to find the shortest sequence from the starting to the ending point, and it is possible to have possible starting or ending points.

Parsing

The parser that is used to parse the Thai grammar is the GLR parser as it can handle ambiguous grammar and can generate all possible parse trees. If the parsing words have zero possible parse tree to continue, the inputted text is grammatically incorrect. The parser is implemented from scratch upon the advisor's supervision. In this stage, sentences are also formed. Currently, the parser is not being integrated into the pipeline as it is not stabilized in terms of performance and does not support incremental parsing.

The grammar utilized by the parser has been sourced exclusively from a paper titled "Building a Hierarchical Annotated Corpus of Thai Using Phrase Structure Grammar." [12] While this represents the sole source utilized in the development of the parser, some modifications have been made to the grammar to ensure alignment with our specific project structure. The final grammar employed by the parser is presented below:

```
sentence -> sub_sentence con_sub_sentence*
con_sub_sentence -> Conj sub_sentence | clause_separator sub_sentence
sub_sentence -> phrase_structure+
phrase_structure -> noun_phrase? verb_phrase
noun_phrase -> noun_bar_specifier_relclause
noun_bar_specifier_relclause -> noun_bar relative_clause?
```

```
noun_bar -> noun_words
noun_words -> serial_noun | Pron post_noun*
post_noun -> adjective_phrase
adjective_phrase -> Adverb prepositional_phrase
serial_noun -> Noun+ post_noun*
nominal_prefix -> 'การ' verb_phrase | 'ความ' adjective_phrase
verb_phrase -> verb_bar_specifier_relclause noun_phrase? prepositional_phrase?
verb_bar_specifier_relclause -> verb_bar relative_clause?
verb_bar -> pre_verb? verb_words post_verb?
pre_verb -> negator | series_preverb_auxiliary
verb_words -> Verb
post_verb -> XVAE | FIXV? Adverb
negator -> 'ไม่' | 'มิ'
series_preverb_auxiliary -> XVBM negator? | negator XVAM? | XVAM | XVMM negator? | nega
FIXV -> 'อย่าง'
XVBM -> 'จะ' | 'เคย'
XVAM -> 'เกิด' | 'เกือบ' | 'กำลัง'
XVMM -> 'ค่อย' | 'น่า' | 'ได้'
XVMM -> 'ควร' | 'เคย' | 'ต้อง'
XVAE -> 'ไป' | 'มา' | 'ขึ้น' | 'แล้ว'
EAFF -> 'นะจ๊ะ' | 'จ้ะ' | 'ครับ' | 'นะ' | 'น่า' | 'เถอะ' | 'นะคะ' | 'ค่ะ' | 'ว่ะ' | '
EITT -> 'นะจ๊ะ' | 'จ๊ะ' | 'หรือ' | 'รึ' | 'หรอ' | 'เหรอ' | 'ไหม' | 'มั้ย' | 'เปล่า' |
prepositional_phrase -> Prep noun_phrase?
relative_clause -> Prep sentence
```

Contextual analysis

This is the final stage of the pipeline. And it isn't readily implemented yet as the parser needs to be stabilized first. The goal of this stage is to define the types of a sentence, e.g., simple, compound, complex-compound, only performing the basic level of analysis, and suggesting spacing between sentences. No validation process involves at this stage.

After the final stage is completed (currently at word tokenization), suggestions and errors collected during those three stages in the pipeline will be sent as output. Note that one of the stages could interrupt the whole process early, e.g., grapheme tokenization error, and send results immediately. Some errors need to be corrected before proceeding to the next pipeline stage.

3.3 Web back-end

The back-end service provides the language core and dictionary to the public internet, and interacts with front-end clients through API endpoints, receiving user input and returning compressed results from the language service. The service is implemented using Actix, a web framework in Rust programming language, chosen for its support of multithreading and concurrency. To minimize payload size, the request-response payload is encoded in a compact binary format using Protocol Buffers. The back-end service is also responsible for the business logic, processing the majority of user data.

Table 3.1: Language core API endpoints

Endpoint	Method	Description
/nlp/suggestions	POST	Gets any suggestions for the message.
/nlp/words	POST	Retrieves word's definitions.
/nlp/clfs	POST	Retrieves word's classifiers.
/nlp/abbrs	POST	Retrieves word's abbreviations.
/nlp/transll	POST	Retrieves word's transliteration.

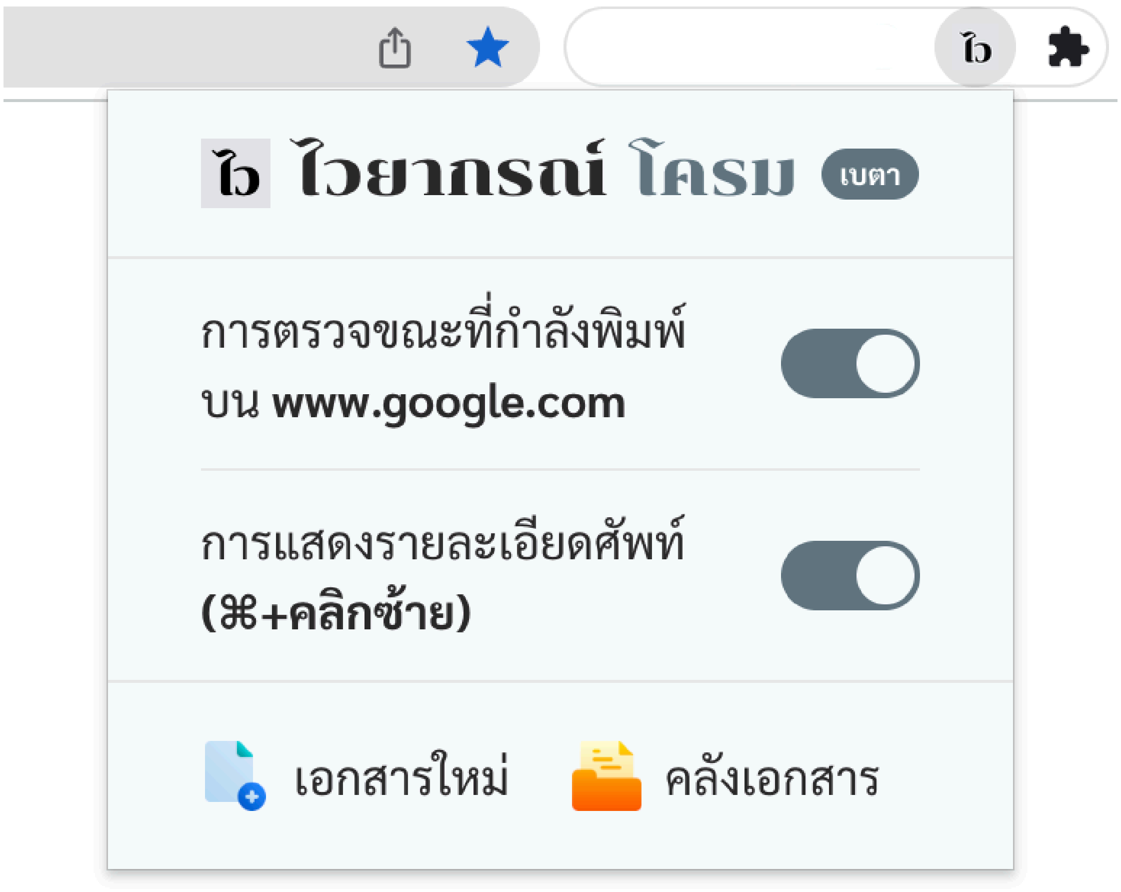
To communicate with these routes, you will need a different Protobuf contract for a different endpoint.

3.4 Web front-end

The front-end user interface is provided as a Chrome extension service that enables users to interact with the back-end service. This interface is built using the Solid framework [22] and TypeScript programming language. The Solid framework has been chosen for its ability to render the bare minimum and reduce memory overhead, resulting in faster processing compared to popular frameworks such as React [13]. The front-end service is responsible for a range of tasks, including sentence chunking, preprocessing, indexing, caching of back-end responses, and presentation of user-friendly results. This extension allows users to validate messages and articles on any website where a WYSIWYG editor exists, such as a plain text editor like Google Forms text input.

The extension has the capability to perform several tasks, including:

- Highlighting of tokenized words
- Replacement of corrections
- Toggling of information related to a specific word, such as definitions, classifiers, abbreviations, and so on.


ใวยากรณ์ โครม เบตา

การตรวจขณะที่กำลังพิมพ์ บน www.google.com

การแสดงรายละเอียดศัพท์ (⌘+คลิกซ้าย)

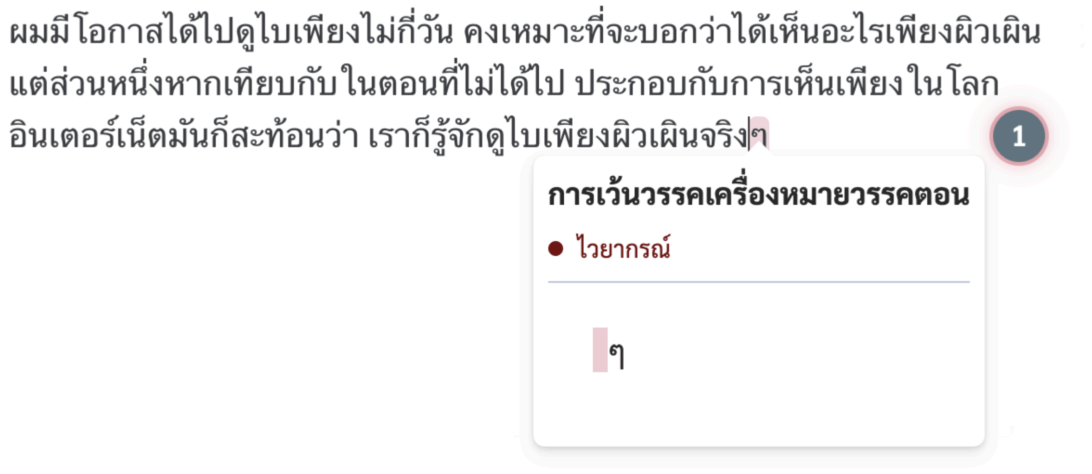
เอกสารใหม่ คลังเอกสาร

Figure 3.3: Extension controller panel shown in Chrome

It is worth noting that due to project time constraints, the extension has been developed for Chrome only, as it is the most widely used browser [33]. However, in future development phases, the extension will be made available for other browsers such as Safari, Firefox, and Edge to enhance its accessibility across a variety of platforms.

3.5 Deployment

Upon completion of the back-end service, it is deployed via a Bash script. The script initiates the Docker compilation process within an isolated environment, and upon successful completion, the Docker is exported and transferred to the server through the 'scp' command. The Docker engine on the server is then instructed by the script to load the file, thereby making the service operational on the server. The service is positioned behind Nginx, which serves as a front-end proxy and enables message encryption between clients through TLS1.3, ensuring that the service remains inaccessible to external parties.



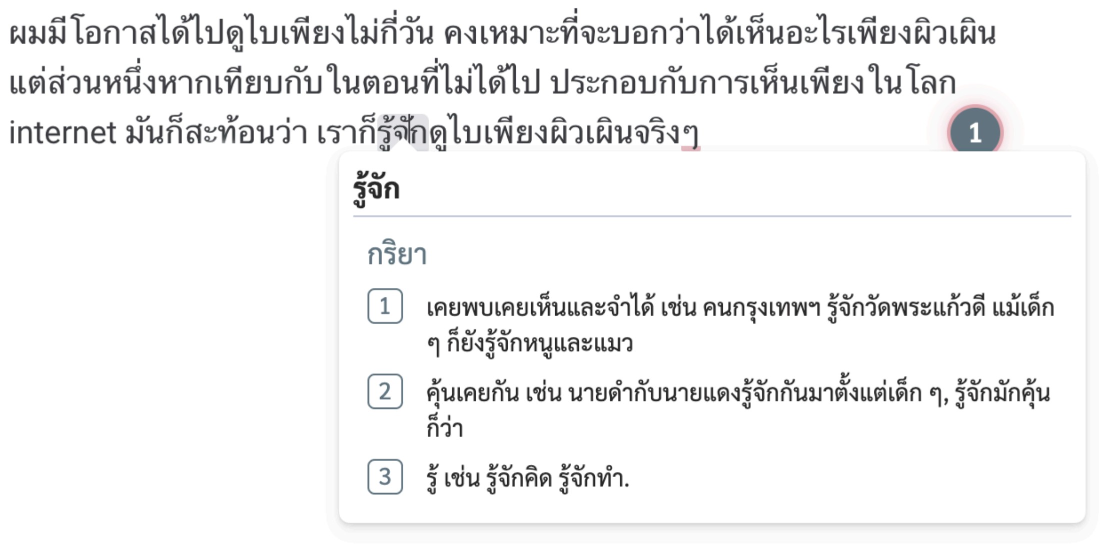
ผมมีโอกาสได้ไปดูไบเพียงไม่กี่วัน คงเหมาะที่จะบอกว่าได้เห็นอะไรเพียงผิวเผิน แต่ส่วนหนึ่งหากเทียบกับในตอนที่ไม่ได้ไป ประกอบกับการเห็นเพียงในโลกอินเตอร์เน็ตมันก็สะท้อนว่า เราก็รู้จักดูไบเพียงผิวเผินจริงๆ

การเว้นวรรคเครื่องหมายวรรคตอน

● ไวยากรณ์

ๆ

Figure 3.4: Toggling suggestion panel



ผมมีโอกาสได้ไปดูไบเพียงไม่กี่วัน คงเหมาะที่จะบอกว่าได้เห็นอะไรเพียงผิวเผิน แต่ส่วนหนึ่งหากเทียบกับในตอนที่ไม่ได้ไป ประกอบกับการเห็นเพียงในโลก internet มันก็สะท้อนว่า เราก็รู้จักดูไบเพียงผิวเผินจริงๆ

รู้จัก

กริยา

1 เคยพบเคยเห็นและจำได้ เช่น คนกรุงเทพฯ รู้จักวัดพระแก้วดี แม้เด็ก ๆ ก็ยังรู้จักหนูและแมว

2 คุ้นเคยกัน เช่น นายดำกับนายแดงรู้จักกันมาตั้งแต่เด็ก ๆ, รู้จักมักคุ้น ก็ว่า

3 รู้ เช่น รู้จักคิด รู้จักทำ.

Figure 3.5: Toggling word info panel

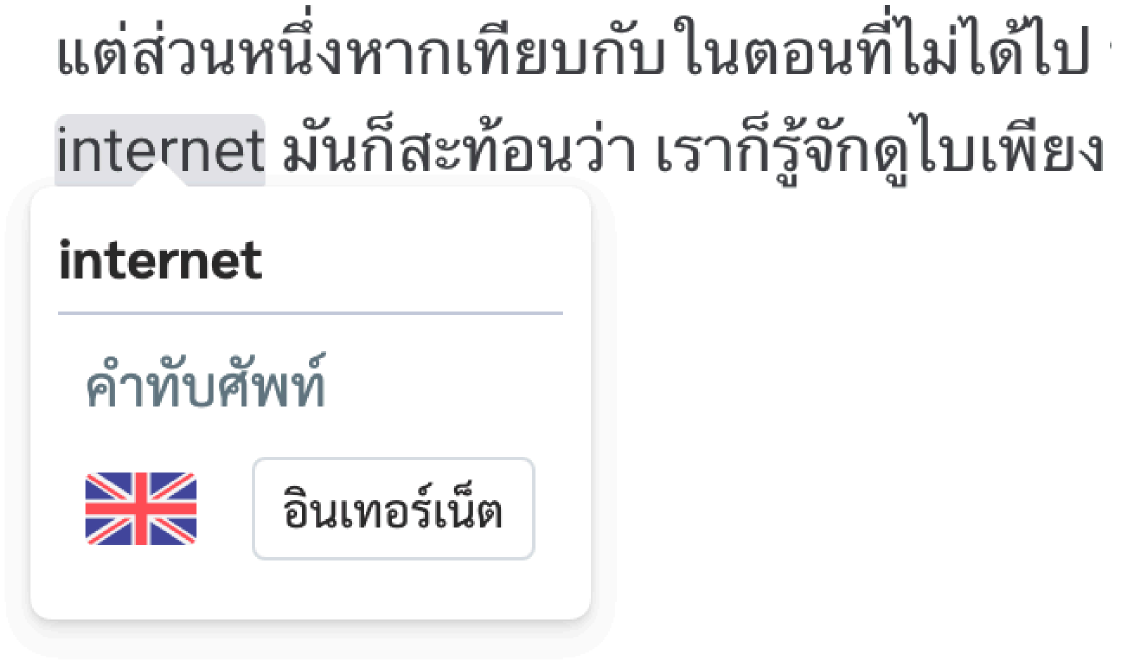

แต่ส่วนหนึ่งหากเทียบกับในตอนที่ไม่ได้ไป
internet มันก็สะท้อนว่า เราก็รู้จักดูไบเพียง

Figure 3.6: Toggling transliteration panel

Chapter 4
Experimentation and Results

The evaluation process consists of three parts: the dictionary evaluation, the language core evaluation, and the back-end evaluation.

4.1 Dictionary evaluation

4.1.1 Words validation

In the part of dictionary evaluation, we validate words from websites whether their correct according to our grammar or not. As shown earlier in Chapter 3, two words from Royal Institute Dictionary are malformatted to our grapheme and TCC RegEx tokenization rules, i.e., ‘กระแสน้ํา’ and ‘ย่ําอยู่กับที่’. The word ‘กระแสน้ํา’ doesn’t use ‘-ำ’ correctly, and the word ‘ย่ําอยู่กับที่’ places the tonal mark twice at the grapheme ‘ที่’. The rest words are valid.

4.1.2 Words distribution

There is an interesting insight from the preprocessed words, hence shown in the form of distribution figures:

4.2 Language core evaluation

4.2.1 Grapheme and TCC tokenization

Validation was performed on the grapheme and TCC tokenization rules against the Thai National Corpus [5] using a randomly selected sample of 1,000 entries. The results revealed that all phrases were able to satisfy the established rules.

Moreover, the runtime complexity of the tokenization rules was observed to be linear, as depicted in the accompanying figure:

4.2.2 Word tokenization

The same dataset from the Thai National Corpus was utilized to assess the validity and efficacy of the word tokenization process. The tokenization process is capable of handling up to 4,000 characters before encountering a crash, and the runtime complexity is evidently quadratic. All the tokenization process outputs can perform finding the shortest path from the starting points to the ending points.

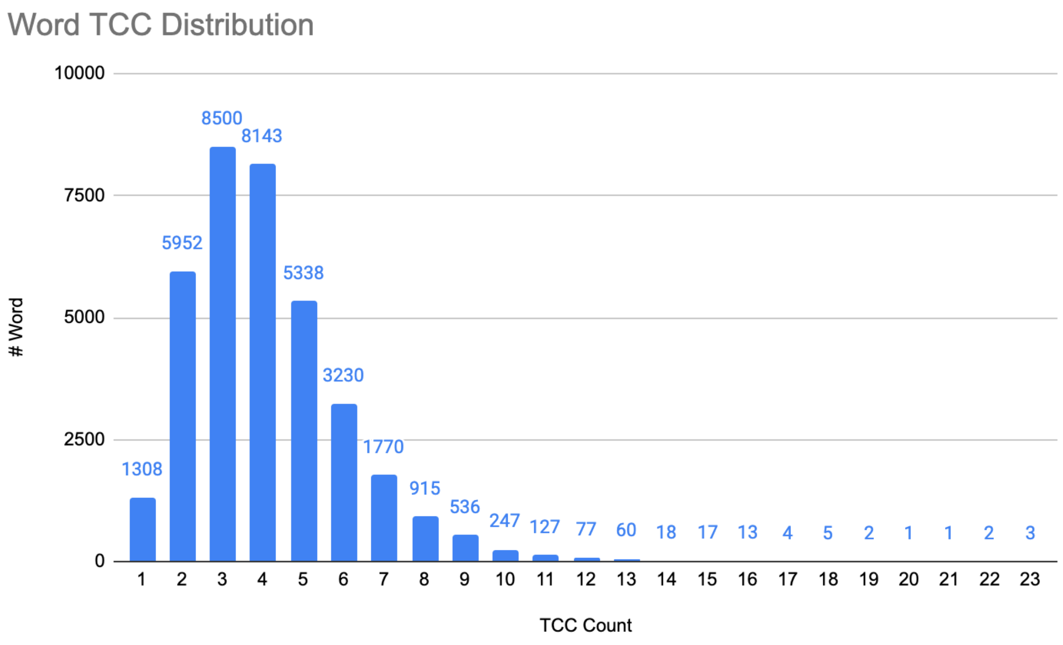


Figure 4.1: Word by TCC distribution (after TCC tokenization). The maximum length is 23 TCCs from Royal Institute Dictionary

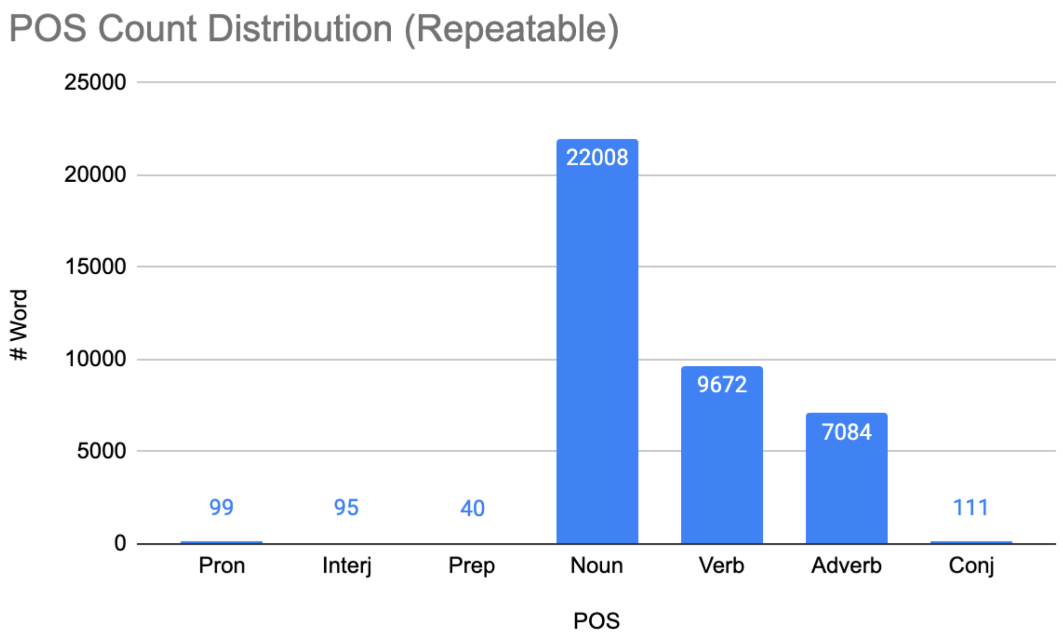


Figure 4.2: Total part-of-speech count distribution (repeatable)

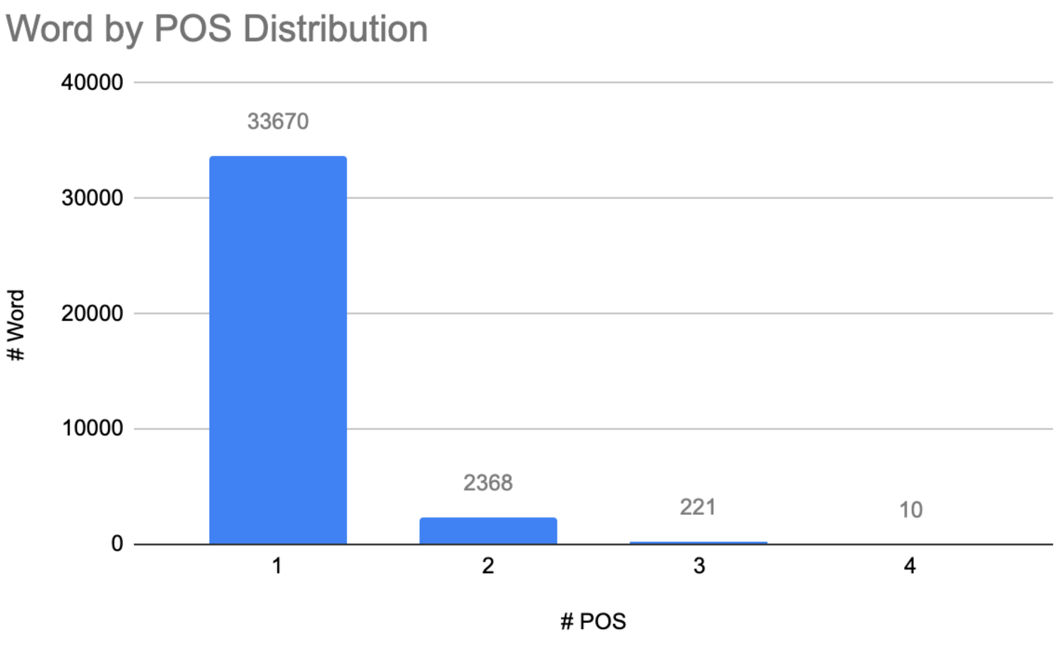


Figure 4.3: A word and its part-of-speech count distribution

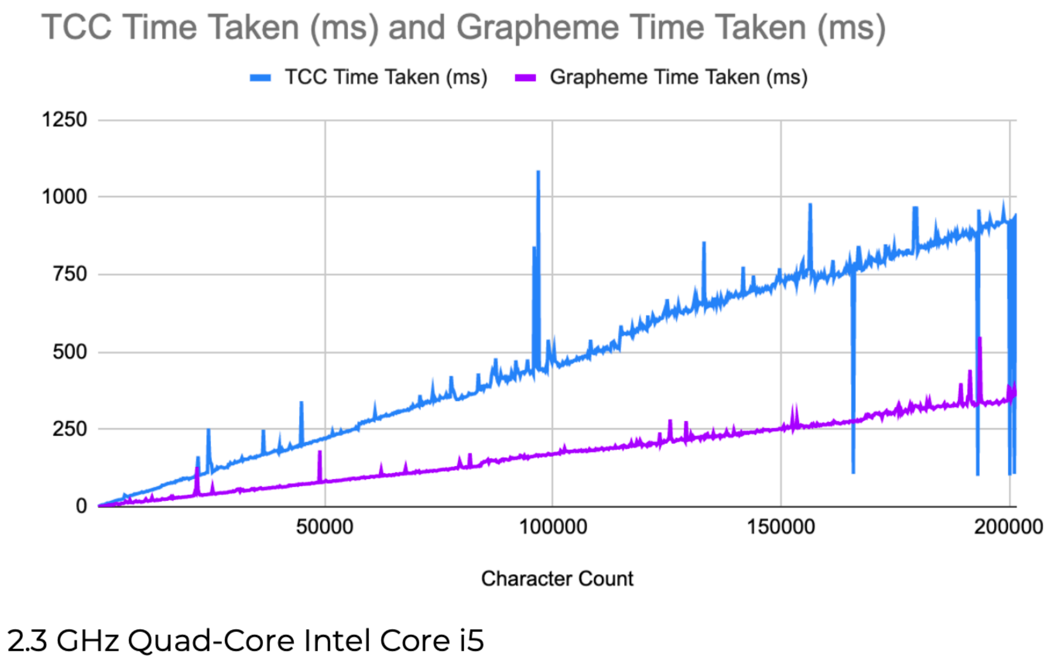


Figure 4.4: Grapheme and TCC tokenization performance

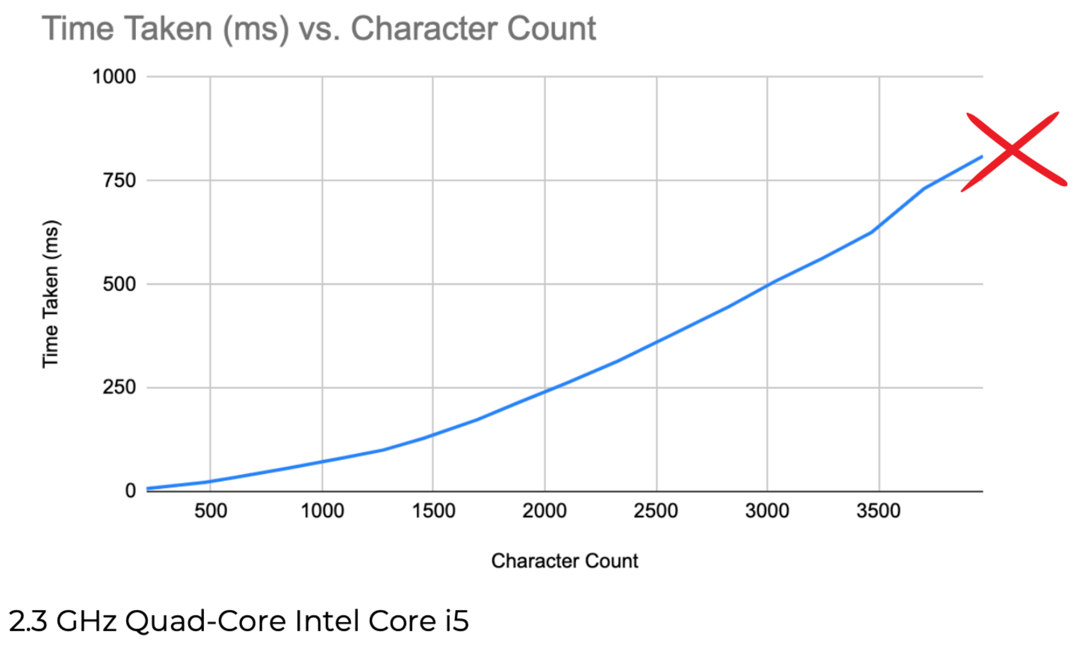


Figure 4.5: Word tokenization performance

4.2.3 Parsing

As the parser is currently in an unstable state, only preliminary testing has been conducted on the tool. Specifically, the phrase ‘วัน|นี้|เป็น|วันที่|ดี’ was tested, resulting in the successful parsing of 12 trees generated within 25ms on the same machine used for Grapheme and TCC tokenization.

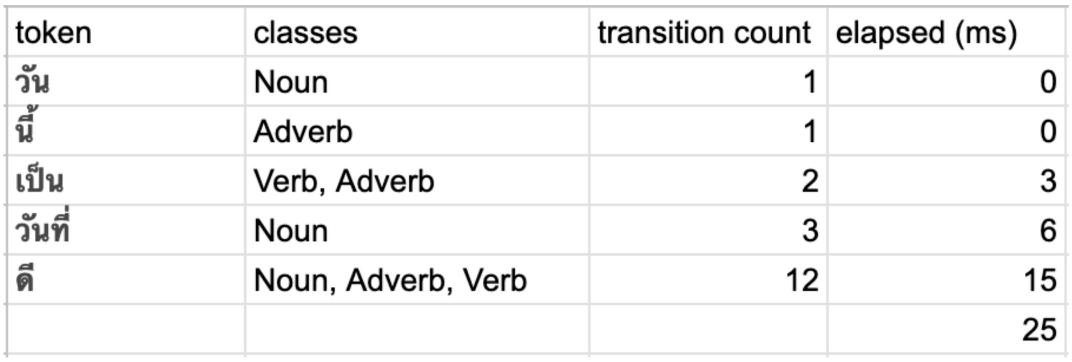
However, the phrase ‘เขา|เป็น|นัก|สู้|ผู้|มี|อุดมการณ์|ใน|ประชาธิปไตย|ทั้ง|ด้าน|การเมือง
produced over 7,078 parse trees and took 19 seconds to execute, resulting in a relatively poor outcome despite being parsed successfully.

This observation suggests that the current version of the parser is inadequate for parsing long-word inputs. Further development is required to address the performance issues and improve the parser’s functionality.

4.3 Back-end evaluation

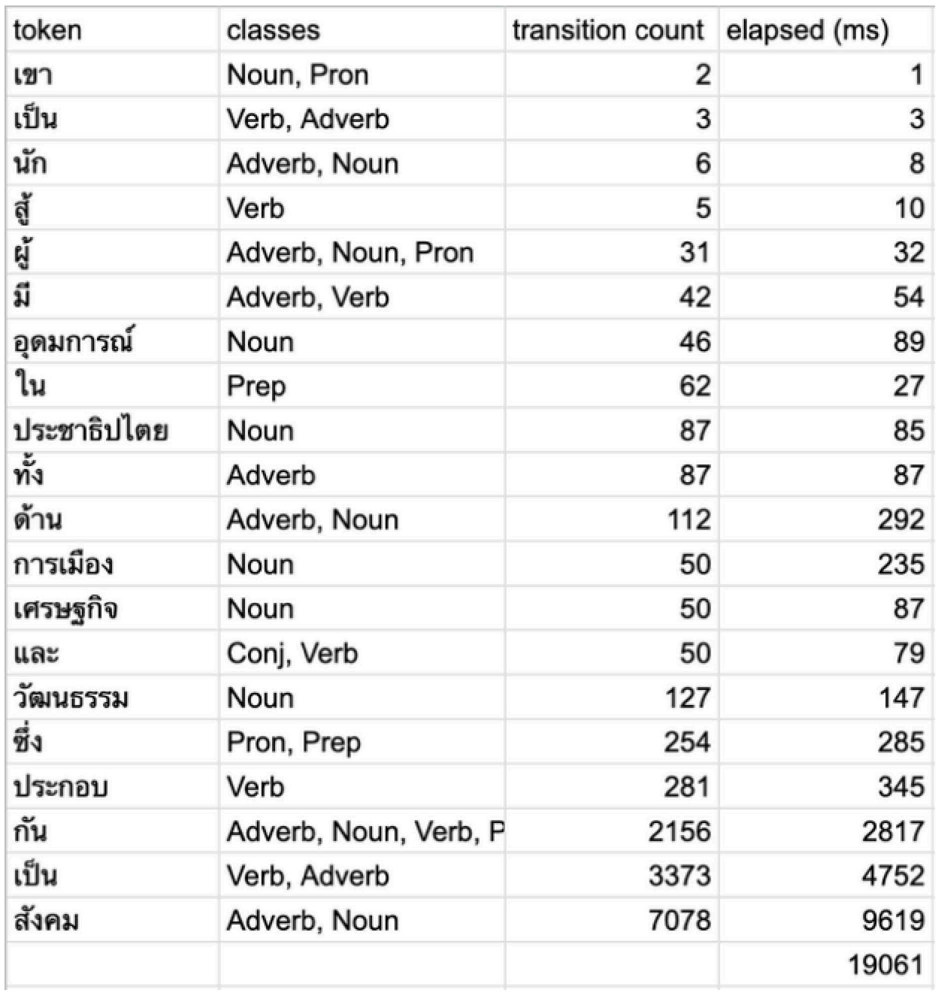
4.3.1 Back-end request-per-second capabilities

Evaluations were conducted on the major endpoints, namely `/nlp/suggestions` and `/nlp/words`, to measure their throughput in terms of requests-per-second. Requests were sent from a Node.js client to the server. Two metrics were considered: sequence and parallel. The sequence metric represents the maximum number of requests a user can perform sequentially within a second, while the parallel metric represents the maximum number of concurrent



token	classes	transition count	elapsed (ms)
วัน	Noun	1	0
นี้	Adverb	1	0
เป็น	Verb, Adverb	2	3
วันที่	Noun	3	6
ดี	Noun, Adverb, Verb	12	15
			25

Figure 4.6: Parsing result on a short phrase



token	classes	transition count	elapsed (ms)
เขา	Noun, Pron	2	1
เป็น	Verb, Adverb	3	3
นัก	Adverb, Noun	6	8
สู้	Verb	5	10
ผู้	Adverb, Noun, Pron	31	32
มี	Adverb, Verb	42	54
อุดมการณ์	Noun	46	89
ใน	Prep	62	27
ประชาธิปไตย	Noun	87	85
ทั้ง	Adverb	87	87
ด้าน	Adverb, Noun	112	292
การเมือง	Noun	50	235
เศรษฐกิจ	Noun	50	87
และ	Conj, Verb	50	79
วัฒนธรรม	Noun	127	147
ซึ่ง	Pron, Prep	254	285
ประกอบ	Verb	281	345
กัน	Adverb, Noun, Verb, P	2156	2817
เป็น	Verb, Adverb	3373	4752
สังคม	Adverb, Noun	7078	9619
			19061

Figure 4.7: Parsing result on a long phrase

requests the server can handle from multiple users using the service simultaneously.

The results are shown below:

Table 4.1: API endpoints throughput

Endpoint	Sequence (rps)	Parallel (rps)
`/nlp/suggestions`	5.2	143.1
`/nlp/words`	5.2	133.3

Chapter 5
Conclusions and Discussions

5.1 Conclusions

The extension is designed to identify and correct misaligned Thai characters based on the rules of the language's grammar, as well as detect punctuation misuse and provide explanations for why a correction is necessary. Users have the option to select suggested corrections, which will replace the incorrect part of input. The extension also offers the ability to toggle for additional information about a given word, such as its meaning and classifier.

However, the extension currently faces some limitations, including difficulty in achieving consistent layout across multiple websites, inability to parse grammar in input text, and incapability of autocorrecting misspelled words. These limitations will require further development to address.

5.2 Challenges

This involves several challenges in natural language processing for the Thai language. These challenges include achieving consistent layout across multiple websites, developing a custom parser to handle Thai grammar, performing text preprocessing for data sourced from the Royal Institute Dictionary, and optimizing the parsing algorithm to handle long text inputs. Additionally, there is a lack of readily available resources and suitable frameworks for certain aspects of the project, such as tokenization and building extensions. Overcoming these challenges requires careful consideration and customized solutions, often developed from scratch under the supervision of a project advisor.

5.2.1 Text preprocessing and data preparation

Text preprocessing of data sourced from the Royal Institute Dictionary presents challenges due to inconsistencies in word formatting. Consequently, a carefully crafted script was necessary to extract words and associated information and transfer them into the designated data structure.

5.2.2 Tokenization

Tokenization represents a significant challenge, as there are no readily available resources to guide the process. As a result, the rules for tokenization must be developed from scratch under the guidance of the project advisor. While tokenization rules for TCC are present in the PyThaiNLP repository, these rules do not encompass words from the Royal Institute Dictionary. Furthermore, the maximum-matching algorithm was implemented from scratch in Rust, the programming language of choice for the project, despite the availability of a Python-based algorithm in PyThaiNLP.

5.2.3 Parser

Developing a parser capable of parsing Thai grammar constitutes a particularly challenging aspect of the project. Due to the potential for multiple word forms for a given word, it was necessary to develop a custom parser to accommodate this feature of the language. Given the inherent ambiguity of the grammar, a GLR parser was deemed most appropriate for this task. Unfortunately, there is a dearth of GLR parsers available in the Rust package registry, Cargo. Moreover, the available GLR parsers in Rust do not support a single token with multiple word forms. Consequently, it was necessary to develop a custom GLR parser from scratch, tailored to the specific demands of the Thai language and input stream, under the guidance of the project advisor.

5.2.4 Front-end layout

Achieving consistent layout across multiple websites poses a challenge, as a solution that works for one website, such as Google Translate, may not be suitable for others. This task demands significant effort and patience. Additionally, a hindrance in front-end development is the absence of a specialized framework for constructing extensions, necessitating the creation of extensions from the ground up, with the exception of utilizing UI frameworks for building controller panels.

5.3 Suggestions and further improvements

The project will undergo ongoing updates to enhance its capacity for correcting Thai grammar. One potential avenue for advancement involves leveraging machine learning techniques for tasks such as sentiment analysis, enabling the program to comprehend the context and content of user input. Another crucial objective is to ensure the extension functions seamlessly across multiple web browsers and supports rich text editing capabilities through a WYSIWYG interface. Additionally, optimizing the parsing algorithm is imperative to facilitate processing of long text inputs without inducing crashes or delays in response time.

References

[1] Actix developers. What is an orm, 2022. [Online; accessed 12-October-2022].

[2] Nadav Altshuler. Why is python so slow? - tackling python's performance issues, 2019. [Online; accessed 12-October-2022].

[3] DBML developers. Dbml - database markup language, 2022. [Online; accessed 12-October-2022].

[4] Debian developers. Debian – about debian, 2022. [Online; accessed 12-October-2022].

[5] Department of Linguistics, Faculty of Arts, Chulalongkorn University. Tnc: Thai national corpus (third edition), 2023. [Online; accessed 05-October-2023].

[6] DigitalOcean developers. Digitalocean – the developer cloud, 2022. [Online; accessed 12-October-2022].

[7] Docker developers. Docker overview | docker documentation, 2022. [Online; accessed 12-October-2022].

[8] GeeksforGeeks authors. What is the difference between css and scss?, 2022. [Online; accessed 12-October-2022].

[9] Google developers. Overview | protocol buffers | google developers', 2022. [Online; accessed 12-October-2022].

[10] Grammarly developers. Free gramamr checker by grammarly, 2022. [Online; accessed 12-October-2022].

[11] Brian Jackson. An overview of tls 1.3 – faster and more secure, 2022. [Online; accessed 12-October-2022].

[12] Prasert Luekhong and Rattasit Sukhahuta. Building a hierarchical annotated corpus of thai using phrase structure grammar. *International Journal of Artificial Intelligence*, 5:56–59, 07 2013.

[13] Iniubong Obonguko. Solidjs vs. react: Comparing declarative ui libraries, 2022. [Online; accessed 14-October-2022].

[14] okta authors. What is token-based authentication?, 2022. [Online; accessed 12-October-2022].

[15] PagerDuty authors. What is a deployment pipeline?, 2022. [Online; accessed 12-October-2022].

[16] PostgreSQL developers. Postgresql: The world's most advanced open source relational database, 2022. [Online; accessed 12-October-2022].

[17] PyThaiNLP authors. Pythainlp, 2022. [Online; accessed 12-October-2022].

[18] replacedbyrobot authors. Will "proofreaders and copy markers" be replaced by robots?, 2022. [Online; accessed 12-October-2022].

[19] Rust developers. Rust programming language, 2022. [Online; accessed 12-October-2022].

[20] Deepu K Sasidharan. Why safe programming matters and why a language like rust matters, 2022. [Online; accessed 12-October-2022].

[21] SeaORM developers. Seaorm an async & dynamic orm for rust, 2022. [Online; accessed 12-October-2022].

[22] Solid developers. Solidjs · reactive javascript library, 2022. [Online; accessed 12-October-2022].

[23] Michael Spencer. What are the greenest programing languages?, 2022. [Online; accessed 12-October-2022].

[24] The Editors of Encyclopaedia Britannica. client-server architecture, 2022. [Online; accessed 12-October-2022].

[25] TypeScript developers. Typescript: Javascript with syntax for types., 2022. [Online; accessed 12-October-2022].

[26] Vite developers. Overview | vite, 2022. [Online; accessed 12-October-2022].

[27] Wikipedia contributors. Autocorrection - wikipedia, 2022. [Online; accessed 12-October-2022].

[28] Wikipedia contributors. Automatic summarization - wikipedia, 2022. [Online; accessed 12-October-2022].

[29] Wikipedia contributors. Copywriting - wikipedia, 2022. [Online; accessed 12-October-2022].

[30] Wikipedia contributors. Http/2 - wikipedia, 2022. [Online; accessed 12-October-2022].

[31] Wikipedia contributors. Paraphrase - wikipedia, 2022. [Online; accessed 12-October-2022].

[32] Wikipedia contributors. Royal institute dictionary - wikipedia, 2022. [Online; accessed 14-October-2022].

[33] Wikipedia contributors. Usage share of web browsers - wikipedia, 2022. [Online; accessed 14-October-2022].

[34] Wiktionary contributors. ภาคผนวก:รายชื่อคำในภาษาไทยที่มักเขียนผิด, 2022. [Online; accessed 14-October-2022].

[35] Serdar Yegulalp. What is rust? safe, fast, and easy software development, 2021. [Online; accessed 12-October-2022].

Appendix A
Manual

To install the extension, please navigate to the URL provided: https://chrome.google.com/webstore/detail/jkbklnignckalhgncccnlhpoiokjidja. Once you have accessed this URL, proceed to install the extension onto your Chrome browser.

After successful installation, the extension will become active when typing is available on any website.

Biographical Sketch

Apisit Ritreungroj, born on 10 September 2000, is a computer engineering student (B.E.) at Chiang Mai University, having previously attended Yupparaj Wittayalai. He readily equips with various programming capabilities, the most prominent being his web development skill. He can do both front-end and back-end development professionally and proficiently.

His professional experience includes a three-month internship as a front-end developer at LINE MAN Wongnai, as well as working on projects for TrueMove H and two start-ups. He has received recognition for his work during his time at the university, including NSC 2021 finalist (education) round ”Alpha Aids” and 1st Runner-Up HUAWEI CLOUD DEVELOPER CONTEST 2021.

In addition to his technical expertise, Apisit is passionate about sharing his knowledge and insights with fellow engineers and enthusiasts.