Saint Petersburg State University

Department of System Programming

Group 22.M07-мм

Hurmatullah Karimi

Deep Learning-powered grammar correction tool

Internship report
in a «Solution» form

Scientific supervisor:
Associate Professor.PhD D. V. Lutsiv

Saint Petersburg
2024

Contents

Introduction

Effective communications in both personal and professional are quite essential. Writing clear and concise can help us to convey ideas, information between individuals or organizations. Despite that, still writers have been struggling with various issues while producing flawless texts due to various reasons including lack of time, lack of advance knowledge of that language, etc. All these reasons cause writers to make mistake while writing a text. To address these challenges, many popular tools created in order to assist writers in refining their works. One of those popular tools is LanguageTool[1]. LanguageTool is an open-source software that aims to help users in rectifying the grammar issues, spelling, and text styles issues in various languages. LanguageTool is a multilingual spelling and grammar checker. It helps us to rectify the grammar issues and refining our works. LanguageTool, in order to fix spelling and grammatical issues, uses rule-based approach and artificial intelligence for more advanced issues and for suggesting the corrected version of text or professional one [1]. In addition, rule-based approach works based on human made rules to store, manipulate, and sort the data. A rule-based approach needs a set of data in order to be used for the system, and a set of rules in order to manipulate that data based on the requirements. These rules are usually referred to "IF" statements. As they tend to follow the "IF X" then Y should be happened.[2] Moreover, to make it clear and understandable, this point should be dealt with that, set of rules are dependent on the set of actions that will be considered in the state of system. Assume the state of the system requires 100 various actions, then 100 different rules should be created. Meanwhile, rule-based approach have too many pros points, but still there are cons points that make it stop to be used in complex and large systems. One of those are limited scopes, which doesn't have capability to learn, and it is restricted by programming conditions. In addition, the rule-based approach is unscalable and doesn't change, and changing it causes time-consuming and expensive complication. At present, in order to refine and make grammar corrector tool better, deep

[1]LanguageTool: https://languagetool.org/

learning approach models are considered. Deep learning provide flexibility and adaptability, it means that deep learning models can learn complex patterns without relying on predefined rules. This flexibility allows deep learning to adapt in a variety of grammatical structures without need for predefined rules. Moreover, deep learning provide better contextual understanding than rule-based approach. Currently, in order to make the trained model usable for end-users, one approach has been identified: LanguageTool offers an option for professional users known as "other servers". To facilitate this, the development of an API is necessary. Therefore, FastAPI[2] is being considered for API development. Through this API, data from LanguageTool will be sent to the API via the plugin. The developed API will then send the incorrect texts to the trained deep learning model. After rectification, the corrected text will be sent back to the interface as a suggestion. Users can rectify the incorrect text by clicking on the suggestion.

[2]FastAPI: https://fastAPI.tiangolo.com/

1 Problem statement

The goal is to develop an API to obtain data through LanguageTool browser plugin and generate rectified version of incorrect text through trained model.

1. Analyze the technologies that can be used to create stable and secure APIs.

2. Develop API that obtain the incorrect data from LanguageTool browser plugin and sends back rectified version as suggestion.

3. Ensure the reliability and efficiency of API using testing library.

2 Background of study

2.1 FastAPI

FastAPI[3] is a high-performing web framework has been using for building API. FastAPI combines the simplicity of Python and performance from asynchronous programming that make the best choice for applications that require high performance and high speed and in the meantime efficient.

2.2 LanguageTool

LanguageTool is a multilingual spelling and grammar checking tool that assists authors in writing text by detecting and correcting typos and grammatical errors in various writing formats, including blogs, websites, books, and more. Similarly, several text editors employ a rule-based approach to error correction, including LanguageTool. Therefore, the following discussion will focus on the rule-based model used by LanguageTool[4].

2.2.1 Rule-based approach

The rule-based approach in the field of artificial intelligence (Ai) relies on a set of predefined rules to determine the next step. This approach involves using a set of inputs and a set of rules to generate output. The system first identifies an applicable rule based on the input data. If the rule matches the input data, the system executes corresponding steps to produce the output. Otherwise, the system may generate a default response or request additional information from the user.

2.2.2 Rule-based approach components

The main components of the rule-based system:

1. Knowledge base: This is a repository of rules, facts, and domain-specific information used by the rule-based system to make deci-

[3]FastAPI: https://FastAPI.tiangolo.com/

[4]LanguageTool: https://languagetool.org/

sions, providing the necessary information for logical analysis and rule matching [2].

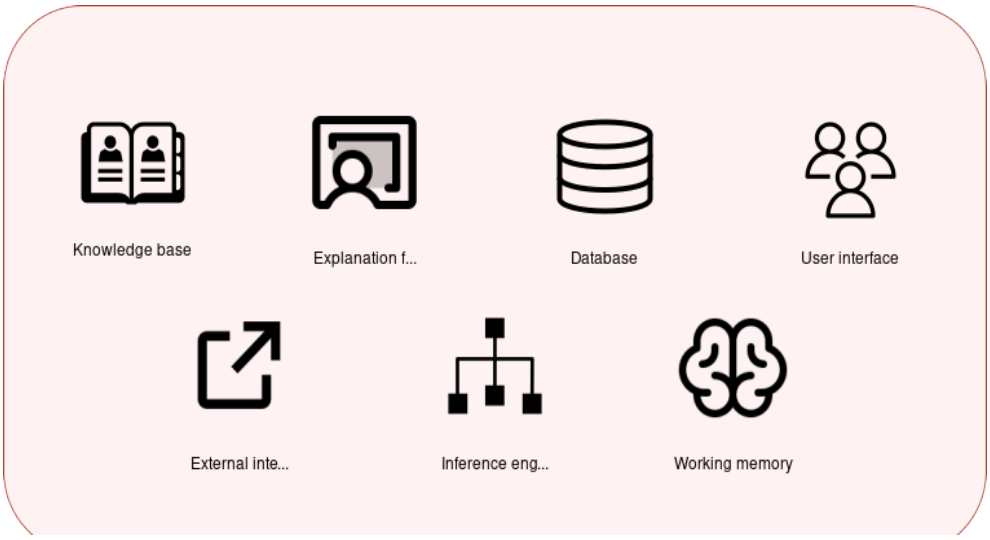

Figure 1: Main components

2. Explanation facilities: It generates justifications or explanations of the system's decisions, increasing transparency and helping users understand the logic underlying the system's outputs, which enhances trust and interpretability. [2]

3. Database: It stores relevant information used by the rule-based system, such as input data or historical records, providing a data source for the inference process and enabling decisions based on data. [2]

4. User interface (Ui): It allows users to interact with the rule-based system, providing the ability to input data, modify rules, and receive results or recommendations, thus facilitating user interaction and system usability. [2]

5. External interface: It provides the capability to communicate and integrate with external systems or services, allowing for data exchange, interaction with other software components, or integration with external sources for input retrieval or output delivery. [2]

6. Inference engine: It processes rules and data from the knowledge base, applying logical reasoning and rule matching to determine appropriate actions or outputs based on input data. [2]

7. Working memory: It temporarily stores the current state of the system during the inference process, retaining input data, intermediate results, and generated outputs, providing the necessary context for rule matching and facilitating the decision-making process. [2]

3 API Implementation

This section focuses on the process of working LanguageTool with trained deep learning model instead of their own LanguageTool[5] model.

3.1 Integration process architecture

In order to make it clear and understandable, that how trained deep learning model works with LanguageTool instead of default model, sequence diagram has designed to make it obvious.

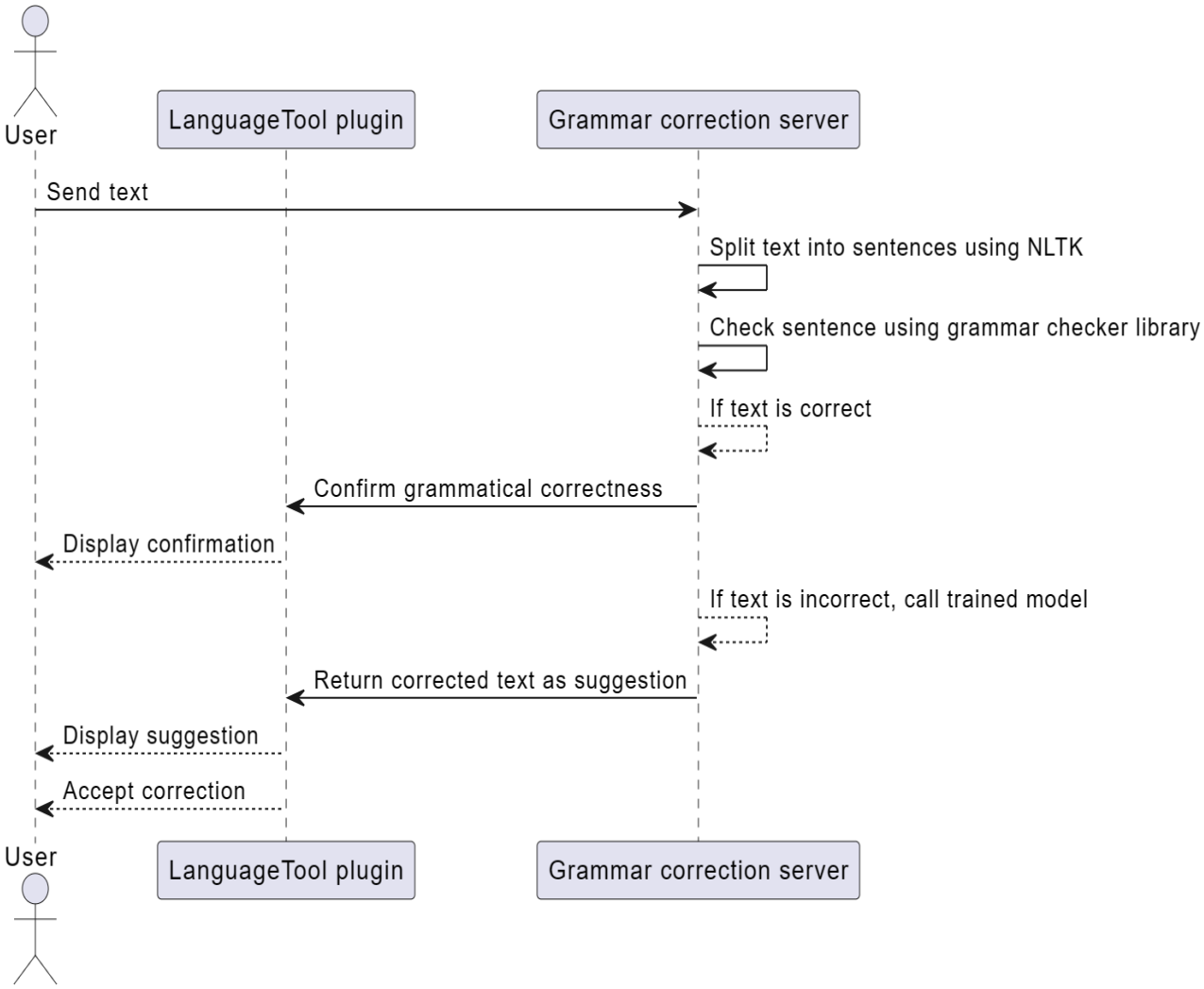

Figure 2: Integration process architecture

[5]LanguageTool - https://languagetool.org/

So, based on the above sequence diagram, the user has the role of actor will write a text in any text editor connected to LanguageTool, and in order to get text from LanguageTool plugin, API developed using FastAPI[6]. Afterward, within API, text will be separated to sentences using Natural language toolkit (NLTK)[7]. When the process of splitting text is completed, each sentence will be checked by LanguageTool grammar checker library in order to identify whether a grammar error existed in the separated sentence. In the meantime, if the grammar error is detected, then those incorrect sentences will be iterated in trained deep learning model to be rectified until whole separated sentences are recognized as correct sentences, and get back to the user with rectified version of those sentences. But, if the separated sentences were identified as correct sentences, then the confirmation will be shown on interface (UI) for user.

3.2 API integration process

As per above explained the whole integration process of architecture, Now it is time to discuss, what are main API and sub API.

3.2.1 Main API

The main API encompasses the core functionality of GecTool (trained deep learning model). This implies that within the main API, data will be received from the LanguageTool browser plugin. Subsequently, the received data will undergo an initial extraction process, followed by a check for correctness or incorrectness by the LanguageTool checker library. Once it has been validated whether the text is correct or not, the API will proceed with its additional functionalities. If the text is deemed correct, an empty response will be sent to the LanguageTool browser, displaying a prompt indicating that the text is correct. Conversely, if the data is found to be incorrect, it will be rectified by a trained deep learning model. The corrected version of the mentioned text will then be suggested to end-users via the

[6]FastAPI - https://fastAPI.tiangolo.com/

[7]NLTK - https://www.nltk.org/

LanguageTool interface.

```
async def check(request: Request):
    indices = []
    original_texts = []
    corrected_text = []
    body = await request.body()
    decoded_body = urllib.parse.unquote(body.decode())
    # Try to split data and catch final text for the model
    try:
        form_data = urllib.parse.parse_qs(decoded_body)
        data_value = form_data.get("data", [""])[0]
        data_dict = json.loads(data_value)
        texts = data_dict.get("text", "")
        sentences = sent_tokenize(texts)

        for sentence in sentences:
            ...

        rectified_body = RectifySentence(
            ...
        )

        return rectified_body

    except Exception as e:
        print("Error extracting text:", e)
```

3.2.2 Sub API

The sub API extends the functionality of the main API by providing additional endpoints for end-users. The concept of separating APIs was to minimize conflicts between them. Within the sub API, there are endpoints for adding words, removing words, listing words, and listing available lan-

guages. Currently, the sentence rectifier operates based on a trained model and is accessible to users, while the remaining APIs remain static. However, they are planned to become dynamic through increased data and the inclusion of additional features in the tool.

```
def rectify_sentence(request: Grammar):
    return t5_correct(request.text)

def support_language():
    return {"name": "English", "code": "en"}

def list_word():
    return "..."

def add_word(request: Word):
    return "..."

def remove_word(request: Word):
    return "..."
```

4 Experiment

In this section, the comprehensive testing process conducted for the developed API using the Pytest[8] framework is outlined, ensuring functionality and reliability.

4.1 Pytest Setup

Pytest was utilized for automated testing of the developed API endpoints. The Pytest framework was installed via pip, and no additional configuration was required.

4.2 Test Cases

4.2.1 Test Language route

Description: This test case validates the functionality of the "/language" endpoint, ensuring that it returns the correct language information. **Expected behavior:** The endpoint should return a 200 OK status code along with the expected language details in json format.

4.2.2 Test Sentence Rectifier Endpoint

Description: This test case validates the "/rectifier" endpoint, which corrects input sentences.
Expected Behavior: The endpoint should return a 200 OK status code and the corrected sentence based on the provided input.

4.2.3 Test Word List Endpoint

Description: This test case verifies the functionality of the "/word" endpoint, which retrieves a list of words.
Expected Behavior: The endpoint should return a 200 OK status code along with the list of words in json format.

[8]Pytest: https://docs.pytest.org/en/8.2.x/

4.2.4 Test Add Word Endpoint:

Description: This test case validates the "/word/add" endpoint, which adds a new word to the dictionary.
Expected Behavior: The endpoint should return a 200 OK status code upon successfully adding the word to the dictionary.

4.2.5 Test Remove Word Endpoint

Description: This test case verifies the functionality of the "/word/remove" endpoint, which removes a word from the dictionary.
Expected Behavior: The endpoint should return a 200 OK status code upon successfully removing the word from the dictionary.

4.3 Test Results

All test cases were executed successfully, with each endpoint returning the expected status codes and responses. No failures were encountered during the testing phase

Conclusion

The following tasks have been completed at the moment:

1. Analyzed the technologies that can be used to create stable and secure API.

2. Developed API that obtain the incorrect data from LanguageTool browser plugin and sends back rectified version as suggestion.

3. Ensured the reliability and efficiency of API using testing library

4. Source code is available in: https://github.com/Hurmatullah/English-Grammar-Corrector.git

References

[1] Gina. LanguageTool. — Access url - https://languagetool.org/insights/post/artificial-intelligence/, Online accessed - 2023.

[2] Noor Ammara. What are rule-based systems in AI? — Access url -https://www.educative.io/answers/what-are-rule-based-systems-in-ai, Online accessed - 2023.