```
def install_and_import_stanza():
    """Install and import Stanza library."""
    import os
    try:
        import stanza
    except ImportError:
        os.system('pip install stanza')
        import stanza
    return stanza



def download_tamil_model(stanza):
    """Download the Tamil language model for Stanza."""
    stanza.download('ta')  # Download if not already done


def initialize_tamil_pipeline(stanza, use_gpu=False):

    return stanza.Pipeline('ta', processors='tokenize,pos', use_gpu=use_gpu)


def process_tamil_text(pipeline, text):

    doc = pipeline(text)
    pos_tags = [(word.text, word.upos) for sentence in doc.sentences for word in sentence.words]
    return pos_tags



# Define the grammar correction function
def correct_grammar(pos_tags):

    words, tags = zip(*pos_tags)
    words = list(words)  # Convert tuple to list for modifications
    errors = []

    # Rule 1: Subject-Object-Verb (SOV) Order
    if "PRON" in tags and "NOUN" in tags and "VERB" in tags:
        pron_index = tags.index("PRON")
        noun_index = tags.index("NOUN")
        verb_index = tags.index("VERB")
        if not (pron_index < noun_index < verb_index):
            errors.append("Error: Subject-Object-Verb (SOV) order is incorrect.")
            # Reorder words to follow SOV
            reordered_words = [words[pron_index], words[noun_index], words[verb_index]]
            remaining_words = [w for i, w in enumerate(words) if i not in (pron_index, noun_index, verb_index)]
            words = reordered_words + remaining_words

    # Rule 2: Adjective-Noun Order
    if "ADJ" in tags and "NOUN" in tags:
        for i, tag in enumerate(tags):
            if tag == "ADJ":
                adj_index = i
                # Look for a noun after the adjective
                for j in range(adj_index + 1, len(tags)):
                    if tags[j] == "NOUN" and adj_index > j:
                        errors.append("Error: Adjective should precede the noun.")
                        # Swap adjective and noun
                        words[adj_index], words[j] = words[j], words[adj_index]

    # Rule 3: Plural Agreement
    if "PRON" in tags and "VERB" in tags:
        pron_index = tags.index("PRON")
        verb_index = tags.index("VERB")
        pron_word = words[pron_index]
        verb_word = words[verb_index]
        if pron_word.endswith("ள்") and not verb_word.endswith("ோம்"):
            errors.append("Error: Plural pronoun does not match plural verb.")
            # Correct the verb to plural form
            if "ேன்" in verb_word:
                words[verb_index] = verb_word.replace("ேன்", "ோம்")
            else:
                words[verb_index] += "ோம்"

    corrected_sentence = " ".join(words)
    return corrected_sentence, errors
```

```
# Extend process_tamil_text to include correction
def process_tamil_text_with_correction(pipeline, tamil_text):
    """
    Process Tamil text for POS tagging and correct grammatical errors.

    Parameters:
    - pipeline: Stanza NLP pipeline for Tamil.
    - tamil_text: str, the input Tamil text.

    Returns:
    - corrected_text: str, the corrected Tamil text.
    - errors: List of detected grammatical errors.
    """
    doc = pipeline(tamil_text)
    pos_tags = [(word.text, word.upos) for sent in doc.sentences for word in sent.words]

    corrected_text, errors = correct_grammar(pos_tags)
    return corrected_text, errors



if __name__ == "__main__":
    # Step 1: Install and Import Stanza
    stanza = install_and_import_stanza()

    # Step 2: Download Tamil Model
    download_tamil_model(stanza)

    # Step 3: Initialize Pipeline
    pipeline = initialize_tamil_pipeline(stanza, use_gpu=False)
```

```
Downloading https://raw.githubusercontent.com/stanfordnlp/stanza-                                     421k/? [00:00<00:00, 12.6MB/s]
resources/main/resources_1.10.0.json:
INFO:stanza:Downloaded file to /root/stanza_resources/resources.json
INFO:stanza:Downloading default packages for language: ta (Tamil) ...
Downloading https://huggingface.co/stanfordnlp/stanza-                                             377M/377M [00:02<00:00, 158MB/s]
ta/resolve/v1.10.0/models/default.zip: 100%
INFO:stanza:Downloaded file to /root/stanza_resources/ta/default.zip
INFO:stanza:Finished downloading models and saved to /root/stanza_resources
INFO:stanza:Checking for updates to resources.json in case models have been updated.  Note: this behavior can be turned off with downloa
Downloading https://raw.githubusercontent.com/stanfordnlp/stanza-                                     421k/? [00:00<00:00, 12.9MB/s]
resources/main/resources_1.10.0.json:
INFO:stanza:Downloaded file to /root/stanza_resources/resources.json
WARNING:stanza:Language ta package default expects mwt, which has been added
INFO:stanza:Loading these models for language: ta (Tamil):
===========================
| Processor | Package      |
---------------------------
| tokenize  | ttb          |
| mwt       | ttb          |
| pos       | ttb_nocharlm |
===========================

INFO:stanza:Using device: cpu
INFO:stanza:Loading: tokenize
```

```
!pip install indic-nlp-library
```

```
Collecting indic-nlp-library
  Downloading indic_nlp_library-0.92-py3-none-any.whl.metadata (5.7 kB)
Collecting sphinx-argparse (from indic-nlp-library)
  Downloading sphinx_argparse-0.5.2-py3-none-any.whl.metadata (3.7 kB)
Collecting sphinx-rtd-theme (from indic-nlp-library)
  Downloading sphinx_rtd_theme-3.0.2-py2.py3-none-any.whl.metadata (4.4 kB)
Collecting morfessor (from indic-nlp-library)
  Downloading Morfessor-2.0.6-py3-none-any.whl.metadata (628 bytes)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (from indic-nlp-library) (2.2.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from indic-nlp-library) (1.26.4)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas->indic-nlp-library) (2.8.2
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas->indic-nlp-library) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from pandas->indic-nlp-library) (2024.2)
Requirement already satisfied: sphinx>=5.1.0 in /usr/local/lib/python3.10/dist-packages (from sphinx-argparse->indic-nlp-library) (8.1.3
Requirement already satisfied: docutils>=0.19 in /usr/local/lib/python3.10/dist-packages (from sphinx-argparse->indic-nlp-library) (0.21
Collecting sphinxcontrib-jquery<5,>=4 (from sphinx-rtd-theme->indic-nlp-library)
  Downloading sphinxcontrib_jquery-4.1-py2.py3-none-any.whl.metadata (2.6 kB)
```

```
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2->pandas->indic-nlp-libra
Requirement already satisfied: sphinxcontrib-applehelp>=1.0.7 in /usr/local/lib/python3.10/dist-packages (from sphinx>=5.1.0->sphinx-arg
Requirement already satisfied: sphinxcontrib-devhelp>=1.0.6 in /usr/local/lib/python3.10/dist-packages (from sphinx>=5.1.0->sphinx-argpa
Requirement already satisfied: sphinxcontrib-htmlhelp>=2.0.6 in /usr/local/lib/python3.10/dist-packages (from sphinx>=5.1.0->sphinx-argp
Requirement already satisfied: sphinxcontrib-jsmath>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from sphinx>=5.1.0->sphinx-argpar
Requirement already satisfied: sphinxcontrib-qthelp>=1.0.6 in /usr/local/lib/python3.10/dist-packages (from sphinx>=5.1.0->sphinx-argpar
Requirement already satisfied: sphinxcontrib-serializinghtml>=1.1.9 in /usr/local/lib/python3.10/dist-packages (from sphinx>=5.1.0->sphi
Requirement already satisfied: Jinja2>=3.1 in /usr/local/lib/python3.10/dist-packages (from sphinx>=5.1.0->sphinx-argparse->indic-nlp-li
Requirement already satisfied: Pygments>=2.17 in /usr/local/lib/python3.10/dist-packages (from sphinx>=5.1.0->sphinx-argparse->indic-nlp
Requirement already satisfied: snowballstemmer>=2.2 in /usr/local/lib/python3.10/dist-packages (from sphinx>=5.1.0->sphinx-argparse->ind
Requirement already satisfied: babel>=2.13 in /usr/local/lib/python3.10/dist-packages (from sphinx>=5.1.0->sphinx-argparse->indic-nlp-li
Requirement already satisfied: alabaster>=0.7.14 in /usr/local/lib/python3.10/dist-packages (from sphinx>=5.1.0->sphinx-argparse->indic-
Requirement already satisfied: imagesize>=1.3 in /usr/local/lib/python3.10/dist-packages (from sphinx>=5.1.0->sphinx-argparse->indic-nlp
Requirement already satisfied: requests>=2.30.0 in /usr/local/lib/python3.10/dist-packages (from sphinx>=5.1.0->sphinx-argparse->indic-n
Requirement already satisfied: packaging>=23.0 in /usr/local/lib/python3.10/dist-packages (from sphinx>=5.1.0->sphinx-argparse->indic-nl
Requirement already satisfied: tomli>=2 in /usr/local/lib/python3.10/dist-packages (from sphinx>=5.1.0->sphinx-argparse->indic-nlp-libra
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from Jinja2>=3.1->sphinx>=5.1.0->sphinx-argpa
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests>=2.30.0->sphinx>=5.1.0
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests>=2.30.0->sphinx>=5.1.0->sphinx-arg
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests>=2.30.0->sphinx>=5.1.0->sphi
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests>=2.30.0->sphinx>=5.1.0->sphi
Downloading indic_nlp_library-0.92-py3-none-any.whl (40 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 40.3/40.3 kB 3.0 MB/s eta 0:00:00
Downloading Morfessor-2.0.6-py3-none-any.whl (35 kB)
Downloading sphinx_argparse-0.5.2-py3-none-any.whl (12 kB)
Downloading sphinx_rtd_theme-3.0.2-py2.py3-none-any.whl (7.7 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7.7/7.7 MB 63.8 MB/s eta 0:00:00
Downloading sphinxcontrib_jquery-4.1-py2.py3-none-any.whl (121 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 121.1/121.1 kB 8.4 MB/s eta 0:00:00
Installing collected packages: morfessor, sphinxcontrib-jquery, sphinx-argparse, sphinx-rtd-theme, indic-nlp-library
Successfully installed indic-nlp-library-0.92 morfessor-2.0.6 sphinx-argparse-0.5.2 sphinx-rtd-theme-3.0.2 sphinxcontrib-jquery-4.1
```

```python
import re
from indicnlp.tokenize.indic_tokenize import trivial_tokenize # indicnlp is available now
from collections import defaultdict

# ... (Rest of the code remains the same)

class RuleBasedChecker:
    def __init__(self):
        # Load Tamil dictionary
        self.tamil_words = self._load_tamil_dictionary()
        self.grammar_rules = {
            'subject_verb_agreement': [
                (r'நான்.*கிறார்கள்', 'First person singular with plural verb'),
                (r'நான்.*கிறார்', 'First person singular with formal verb'),
                (r'நான்.*கிறது', 'First person with neuter verb'),
                (r'நாங்கள்.*கிறான்', 'First person plural with singular verb'),
                (r'நாங்கள்.*கிறாள்', 'First person plural with feminine singular'),
                (r'நீ.*கிறார்கள்', 'Second person singular with plural verb'),
                (r'நீங்கள்.*கிறான்', 'Second person plural with singular verb'),
                (r'ஆசிரியர்.*கிறான்', 'Honorific subject with informal verb')
            ],
            'spelling_patterns': [
                (r'சல்', 'Possible misspelling of செல்'),
                (r'பதில்', 'Possible misspelling of பதிவு'),
                (r'எங்க', 'Possible misspelling of எங்கே')
            ],
            'word_spacing': [
                (r'\w+க்கு\w+', 'Missing space before க்கு'),
                (r'\w+யில்\w+', 'Missing space before யில்'),
                (r'\w+உடன்\w+', 'Missing space before உடன்')
            ]
        }

    def _load_tamil_dictionary(self) -> dict:
        """
        Load Tamil words from a dictionary file or use a basic predefined dictionary.

        Returns:
            dict: A dictionary of Tamil words with their parts of speech.
        """
        basic_dictionary = {
            'நான்': 'pronoun',
            'நீ': 'pronoun',
            'நாங்கள்': 'pronoun',
            'பள்ளி': 'noun',
            'பள்ளிக்கு': 'noun',
            'செல்கிறேன்': 'verb',
            'செல்கிறான்': 'verb',
            'செல்கிறாள்': 'verb',
            'செல்கிறது': 'verb',
            'படிக்கிறோம்': 'verb',
            'பாடல்': 'noun',
            'பாடுகிறேன்': 'verb',
            'ஆசிரியர்': 'noun',
            'நல்ல': 'adjective',
            'பாடங்களை': 'noun',
            'கற்றுக்': 'verb',
            'கொடுக்கிறார்': 'verb'
        }

        try:
            # Load dictionary from file if available
            with open("data/tamil_dictionary.txt", "r", encoding="utf-8") as file:
                for line in file:
                    if line.strip():
                        parts = line.strip().split(',')
                        if len(parts) >= 2:
                            basic_dictionary[parts[0]] = parts[1]
        except FileNotFoundError:
            pass  # Use the basic dictionary if file is not found

        return basic_dictionary

    def split_sentences(self, text: str) -> list:

        sentences = re.split(r'[.!?।]', text)
        return [s.strip() for s in sentences if s.strip()]

    def check_spelling(self, text: str) -> list:

        errors = []
        words = trivial_tokenize(text)

        for word in words:
            # Check spelling patterns
            for pattern, msg in self.grammar_rules['spelling_patterns']:
                if re.match(pattern, word):
                    errors.append(('spelling', msg, word))

            # Check against dictionary
            if word not in self.tamil_words and not any(char.isdigit() for char in word):
                errors.append(('spelling', f'Unknown word: {word}', word))

            # Check word spacing
            for pattern, msg in self.grammar_rules['word_spacing']:
                if re.match(pattern, word):
                    errors.append(('spelling', msg, word))

        return errors

    def check_grammar(self, text: str) -> list:
        """
        Check for grammatical errors in the text.

        Args:
            text (str): The text to check.

        Returns:
            list: A list of grammatical errors.
        """
        errors = []
        sentences = self.split_sentences(text)

        for sentence in sentences:
            # Check subject-verb agreement
            for pattern, error_msg in self.grammar_rules['subject_verb_agreement']:
                if re.search(pattern, sentence):
                    errors.append(('grammar', error_msg, sentence))

        return errors

    def check_text(self, text: str) -> list:
        """
        Check the text for both spelling and grammar errors.

        Args:
            text (str): The text to check.

        Returns:
            list: A list of errors found in the text.
        """
        try:
            # Perform spelling checks
            spelling_errors = self.check_spelling(text)

            # Perform grammar checks
            grammar_errors = self.check_grammar(text)

            # Combine all errors
            all_errors = spelling_errors + grammar_errors

            return all_errors if all_errors else []

        except Exception as e:
            return [('error', f'Error in text analysis: {str(e)}', text)]


import re
from typing import List, Tuple

class RuleBasedChecker:
    def __init__(self):
        # Define spelling and grammar rules
        self.grammar_rules = [
            {
                "pattern": r"^(?:நான்|நீ|அவர்|அவள்|அது|அவர்கள்)\s+\S+\s+(?:செய்கிறேன்|செய்கிறாய்|செய்கிறார்|செய்கிறாள்|செய்கி
                "message": "This sentence structure is correct for simple present tense.",
            },
            {
                "pattern": r"நாங்கள்.*செல்கிறான்",
                "message": "Subject-verb agreement error. Use 'செல்கிறோம்' for 'நாங்கள்'.",
            }
        ]

        # Common Tamil spelling errors and their corrections
        self.spelling_corrections = {
            "சல்கிறேன்": "செல்கிறேன்",
            "செல்கிறது": "செல்கிறேன்",
            "செல்கிறான்": "செல்கிறார்"
        }

    def check_text(self, text: str) -> List[Tuple[str, str, str]]:
        """
        Check the text for errors based on predefined rules.

        Returns a list of tuples containing:
        (error_type, message, context)
        """
        errors = []

        # Check for spelling errors
        words = text.split()
        for word in words:
            if word in self.spelling_corrections:
                errors.append(("spelling", f"Incorrect spelling: '{word}'", f"Suggested: '{self.spelling_corrections[word]}'"))

        # Check for grammar errors
        for rule in self.grammar_rules:
            if re.search(rule["pattern"], text):
                errors.append(("grammar", rule["message"], text))

        return errors

    def correct_spelling(self, text: str) -> str:
        """
        Correct spelling errors in the text.
        """
        corrected_text = text
        for incorrect, correct in self.spelling_corrections.items():
            corrected_text = corrected_text.replace(incorrect, correct)
        return corrected_text


if __name__ == "__main__":
    rule_checker = RuleBasedChecker()

    # Example sentences
    text = "நான் பள்ளிக்கு சல்கிறேன்"

    # Check for errors
    errors = rule_checker.check_text(text)
    print("Errors Found:")
    for error in errors:
        print(f"Type: {error[0]}, Message: {error[1]}, Context: {error[2]}")

    # Correct the text
    corrected_text = rule_checker.correct_spelling(text)
    print("\nCorrected Text:")
    print(corrected_text)
```

```
Errors Found:
Type: spelling, Message: Incorrect spelling: 'சல்கிறேன்', Context: Suggested: 'செல்கிறேன்'

Corrected Text:
நான் பள்ளிக்கு செல்கிறேன்
```

```
    tamil_text = "நாங்கள் பாடம் படிக்கிறேன்"  # Example with grammatical errors


    corrected_text, errors = process_tamil_text_with_correction(pipeline, tamil_text)


    print("Original Text:", tamil_text)
    if errors:
        print("\nDetected Grammar Errors:")
        for error in errors:
            print(f"- {error}")
        print("\nCorrected Sentence:", corrected_text)
    else:
        print("\nThe sentence is grammatically correct!")
```

```
Original Text: நாங்கள் பாடம் படிக்கிறேன்

Detected Grammar Errors:
- Error: Plural pronoun does not match plural verb.

Corrected Sentence: நாங்கள் பாடம் படிக்கிறோம்
```

```
    tamil_text = "பாடம் நாங்கள் படிக்கிறோம்"  # Example with grammatical errors


    corrected_text, errors = process_tamil_text_with_correction(pipeline, tamil_text)


    print("Original Text:", tamil_text)
    if errors:
        print("\nDetected Grammar Errors:")
        for error in errors:
            print(f"- {error}")
        print("\nCorrected Sentence:", corrected_text)
    else:
        print("\nThe sentence is grammatically correct!")
```

```
Original Text: பாடம் நாங்கள் படிக்கிறோம்

Detected Grammar Errors:
- Error: Subject-Object-Verb (SOV) order is incorrect.

Corrected Sentence: நாங்கள் பாடம் படிக்கிறோம்
```

```
    tamil_text = "நான் வாசிக்கிறேன் கதை"  # Example with grammatical errors
    corrected_text, errors = process_tamil_text_with_correction(pipeline, tamil_text)


    print("Original Text:", tamil_text)
    if errors:
        print("\nDetected Grammar Errors:")
        for error in errors:
            print(f"- {error}")
        print("\nCorrected Sentence:", corrected_text)
    else:
        print("\nThe sentence is grammatically correct!")
```

```
Original Text: நான் வாசிக்கிறேன் கதை

Detected Grammar Errors:
- Error: Subject-Object-Verb (SOV) order is incorrect.

Corrected Sentence: நான் கதை வாசிக்கிறேன்
```

```
tamil_text = "நான் நேற்று பாடம் படித்தேன்"  # Example with grammatical errors

corrected_text, errors = process_tamil_text_with_correction(pipeline, tamil_text)

print("Original Text:", tamil_text)
if errors:
    print("\nDetected Grammar Errors:")
    for error in errors:
        print(f"- {error}")
    print("\nCorrected Sentence:", corrected_text)
else:
    print("\nThe sentence is grammatically correct!")
```

```
Original Text: நான் நேற்று பாடம் படித்தேன்

The sentence is grammatically correct!
```

```
from typing import List, Dict, Tuple

def calculate_accuracy(
    checker,
    dataset: List[Dict[str, str]]
) -> Tuple[float, float]:
    """
    Calculate accuracy for the rule-based checker.

    Parameters:
        checker: The RuleBasedChecker instance.
        dataset: A list of dictionaries with 'text', 'expected_errors', and 'expected_correction'.

    Returns:
        Tuple containing:
            - error_detection_accuracy: Accuracy of detecting errors.
            - correction_accuracy: Accuracy of providing correct suggestions.
    """
    total_sentences = len(dataset)
    correct_error_detections = 0
    correct_corrections = 0

    for data in dataset:
        text = data['text']
        expected_errors = data['expected_errors']  # List of expected errors [(type, message, context), ...]
        expected_correction = data['expected_correction']

        # Model predictions
        detected_errors = checker.check_text(text)
        corrected_text = checker.correct_spelling(text)

        # Check error detection accuracy
        detected_set = set((e[0], e[1], e[2]) for e in detected_errors)
        expected_set = set((e[0], e[1], e[2]) for e in expected_errors)

        if detected_set == expected_set:
            correct_error_detections += 1

        # Check correction accuracy
        if corrected_text == expected_correction:
            correct_corrections += 1

    error_detection_accuracy = correct_error_detections / total_sentences
    correction_accuracy = correct_corrections / total_sentences

    return error_detection_accuracy, correction_accuracy


# Example usage
if __name__ == "__main__":
    # Example dataset for testing
    test_dataset = [
        {
            "text": "நாங்கள் பள்ளிக்கு சல்கிறேன்",
            "expected_errors": [
                ("spelling", "Incorrect spelling: 'சல்கிறேன்'", "Suggested: 'செல்கிறேன்'"),
                ("grammar", "Subject-verb agreement error. Use 'செல்கிறோம்' for 'நாங்கள்'",
                 "நாங்கள் பள்ளிக்கு செல்கிறான் சல்கிறேன்")
            ],
            "expected_correction": "நாங்கள் பள்ளிக்கு செல்கிறோம்"
        },
        {
            "text": "நான் பள்ளிக்கு செல்கிறேன்",
            "expected_errors": [],
            "expected_correction": "நான் பள்ளிக்கு செல்கிறேன்"
        }
    ]

    # Instantiate the rule-based checker
    rule_checker = RuleBasedChecker()

    # Calculate accuracy
    error_acc, correction_acc = calculate_accuracy(rule_checker, test_dataset)

    # Print the results
    print(f"Error Detection Accuracy: {error_acc:.2%}")
    print(f"Correction Accuracy: {correction_acc:.2%}")
```

```
Error Detection Accuracy: 50.00%
Correction Accuracy: 50.00%
```