```
import logging
import sys
import traceback
import re
import nltk

nltk.download('punkt_tab')
nltk.download('averaged_perceptron_tagger_eng')
nltk.download('universal_tagset')
nltk.download('brown')
```

```
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger_eng to
[nltk_data]     /root/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger_eng is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package universal_tagset to /root/nltk_data...
[nltk_data]   Package universal_tagset is already up-to-date!
[nltk_data] Downloading package brown to /root/nltk_data...
[nltk_data]   Package brown is already up-to-date!
True
```

```
# Configure comprehensive logging
logging.basicConfig(
    level=logging.DEBUG,
    format='%(asctime)s - %(levelname)s: %(message)s',
    handlers=[
        logging.FileHandler('pos_tagger_log.txt'),
        logging.StreamHandler(sys.stdout)
    ]
)

class AdvancedPOSTagger:
    def __init__(self):
        """
        Initialize POS Tagger with comprehensive logging and error handling
        """
        self.logger = logging.getLogger(self.__class__.__name__)
        self.setup_nltk_resources()

    def setup_nltk_resources(self):
        """
        Attempt to download and setup NLTK resources with detailed error handling
        """
        try:
            self.logger.info("Downloading NLTK Resources...")

            # List of required NLTK downloads
            nltk_resources = [
                'punkt',
                'averaged_perceptron_tagger',
                'universal_tagset',
                'brown'
            ]

            for resource in nltk_resources:
                try:
                    nltk.download(resource, quiet=True)
                    self.logger.info(f"Successfully downloaded {resource}")
```

```
                self.logger.info(f"Successfully downloaded {resource}")
            except Exception as download_error:
                self.logger.error(f"Failed to download {resource}: {download_error}")

        self.logger.info("NLTK Resource Setup Complete")

    except Exception as e:
        self.logger.critical(f"Critical Error in NLTK Setup: {e}")
        traceback.print_exc()
        sys.exit(1)

def rule_based_pos_tagger(self, sentence):
    """
    Advanced Rule-Based POS Tagging with detailed logging
    """
    try:
        self.logger.info(f"Applying Rule-Based POS Tagging to: {sentence}")

        # Enhanced Rule Sets
        rules = {
            # Suffix-based Rules
            'Noun Suffixes': {
                'ness': 'NOUN',
                'ment': 'NOUN',
                'ship': 'NOUN'
            },
            'Verb Suffixes': {
                'ize': 'VERB',
                'ate': 'VERB',
                'ify': 'VERB'
            },
            'Adjective Suffixes': {
                'able': 'ADJ',
                'ible': 'ADJ',
                'ous': 'ADJ'
            },
            'Adverb Suffixes': {
                'ly': 'ADV'
            }
        }

        # Tokenize sentence
        tokens = nltk.word_tokenize(sentence)
        tagged_tokens = []

        for token in tokens:
            # Default to NOUN
            tag = 'NOUN'

            # Check suffix rules
            for category, suffix_rules in rules.items():
                for suffix, pos_tag in suffix_rules.items():
                    if token.lower().endswith(suffix):
                        tag = pos_tag
                        self.logger.debug(f"Applied {category} rule: {token} -> {tag}")
                        break

            # Context-based rules
            if token.istitle():
                tag = 'PROPN'  # Proper Noun
            elif token.lower() in ['the', 'a', 'an']:
                tag = 'DET'  # Determiner
```

```
                    tag = 'DET'  # Determiner
                elif token.isdigit():
                    tag = 'NUM'  # Number

                tagged_tokens.append((token, tag))

            self.logger.info("Rule-Based Tagging Completed")
            return tagged_tokens

        except Exception as e:
            self.logger.error(f"Error in Rule-Based Tagging: {e}")
            traceback.print_exc()
            return []

    def stochastic_pos_tagger(self, sentence):
        """
        Stochastic POS Tagging using NLTK's Advanced Tagger
        """
        try:
            self.logger.info(f"Applying Stochastic POS Tagging to: {sentence}")

            # Tokenize sentence
            tokens = nltk.word_tokenize(sentence)

            # Use NLTK's default tagger
            tagged_tokens = nltk.pos_tag(tokens)

            self.logger.info("Stochastic Tagging Completed")
            return tagged_tokens

        except Exception as e:
            self.logger.error(f"Error in Stochastic Tagging: {e}")
            traceback.print_exc()
            return []

    def compare_taggers(self, sentence):
        """
        Compare Rule-Based and Stochastic Taggers
        """
        try:
            self.logger.info(f"Comparing Taggers for: {sentence}")

            # Rule-Based Tagging
            rule_based_tags = self.rule_based_pos_tagger(sentence)

            # Stochastic Tagging
            stochastic_tags = self.stochastic_pos_tagger(sentence)

            # Print Comparison
            print("\n=== Tagging Comparison ===")
            print("Rule-Based Tagging:")
            for word, tag in rule_based_tags:
                print(f"{word}: {tag}")

            print("\nStochastic Tagging:")
            for word, tag in stochastic_tags:
                print(f"{word}: {tag}")

        except Exception as e:
            self.logger.error(f"Comparison Error: {e}")
            traceback.print_exc()
```

```
# Test Sentences
test_sentences = [
    "The quick brown fox jumps over the lazy dog.",
    "Natural language processing is an exciting field of artificial intelligence.",
    "She quickly ran towards the beautiful sunset."
]

# Create POS Tagger Instance
pos_tagger = AdvancedPOSTagger()
```

```
# Compare Taggers for Each Sentence
print(test_sentences[0])
pos_tagger.compare_taggers(test_sentences[0])
```

```
The quick brown fox jumps over the lazy dog.

=== Tagging Comparison ===
Rule-Based Tagging:
The: PROPN
quick: NOUN
brown: NOUN
fox: NOUN
jumps: NOUN
over: NOUN
the: DET
lazy: NOUN
dog: NOUN
.: NOUN

Stochastic Tagging:
The: DT
quick: JJ
brown: NN
fox: NN
jumps: VBZ
over: IN
the: DT
lazy: JJ
dog: NN
.: .
```

```
# Compare Taggers for Each Sentence
print(test_sentences[1])
pos_tagger.compare_taggers(test_sentences[1])
```

```
Natural language processing is an exciting field of artificial intelligence.

=== Tagging Comparison ===
Rule-Based Tagging:
Natural: PROPN
language: NOUN
processing: NOUN
is: NOUN
an: DET
exciting: NOUN
field: NOUN
of: NOUN
artificial: NOUN
intelligence: NOUN
.: NOUN
```

```
Stochastic Tagging:
Natural: JJ
language: NN
processing: NN
is: VBZ
an: DT
exciting: JJ
field: NN
of: IN
artificial: JJ
intelligence: NN
.: .
```

```
# Compare Taggers for Each Sentence
print(test_sentences[2])
pos_tagger.compare_taggers(test_sentences[2])
```

```
She quickly ran towards the beautiful sunset.

=== Tagging Comparison ===
Rule-Based Tagging:
She: PROPN
quickly: ADV
ran: NOUN
towards: NOUN
the: DET
beautiful: NOUN
sunset: NOUN
.: NOUN

Stochastic Tagging:
She: PRP
quickly: RB
ran: VBD
towards: IN
the: DT
beautiful: JJ
sunset: NN
.: .
```