```
import nltk
import re
import math
from collections import Counter
import numpy as np
```

```
corpus = """
Natural language processing helps computers understand language.
Language models are widely used in artificial intelligence.
Machine learning improves healthcare and technology.
Bigram and trigram models capture context.
Perplexity evaluates language models.
""" * 300   # repeated to exceed 1500 words
print(corpus[:500])
```

```
Natural language processing helps computers understand language.
Language models are widely used in artificial intelligence.
Machine learning improves healthcare and technology.
Bigram and trigram models capture context.
Perplexity evaluates language models.

Natural language processing helps computers understand language.
Language models are widely used in artificial intelligence.
Machine learning improves healthcare and technology.
Bigram and trigram models capture context.
Perplexity evaluat
```

```
def preprocess_text(text):
    text = text.lower()
    text = re.sub(r'[^a-z\s]', '', text)
    tokens = text.split()
    return tokens
tokens = preprocess_text(corpus)
print(tokens[:20])
```

```
['natural', 'language', 'processing', 'helps', 'computers', 'understand', 'language', 'language', 'models', 'are', 'widely',
```

```
split_index = int(0.8 * len(tokens))
train_tokens = tokens[:split_index]
test_tokens = tokens[split_index:]
```

```
unigram_counts = Counter(train_tokens)
total_words = sum(unigram_counts.values())
vocab_size = len(unigram_counts)
```

```
bigrams = list(zip(train_tokens[:-1], train_tokens[1:]))
bigram_counts = Counter(bigrams)
```

```
trigrams = list(zip(train_tokens[:-2], train_tokens[1:-1], train_tokens[2:]))
trigram_counts = Counter(trigrams)
```

```
def unigram_prob(word):
    return (unigram_counts[word] + 1) / (total_words + vocab_size)

def bigram_prob(w1, w2):
    return (bigram_counts[(w1, w2)] + 1) / (unigram_counts[w1] + vocab_size)

def trigram_prob(w1, w2, w3):
    return (trigram_counts[(w1, w2, w3)] + 1) / (bigram_counts[(w1, w2)] + vocab_size)
```

```
def sentence_probability(sentence, model):
    words = preprocess_text(sentence)
    prob = 1.0

    if model == "unigram":
        for w in words:
```

```
            prob *= unigram_prob(w)

    elif model == "bigram":
        for i in range(len(words)-1):
            prob *= bigram_prob(words[i], words[i+1])

    elif model == "trigram":
        for i in range(len(words)-2):
            prob *= trigram_prob(words[i], words[i+1], words[i+2])

    return prob
```

```
sentences = [
    "language models are important",
    "machine learning improves technology",
    "perplexity evaluates models",
    "artificial intelligence is powerful",
    "healthcare uses language processing"
]

for s in sentences:
    print("Sentence:", s)
    print("Unigram:", sentence_probability(s, "unigram"))
    print("Bigram :", sentence_probability(s, "bigram"))
    print("Trigram:", sentence_probability(s, "trigram"))
    print()
```

```
Sentence: language models are important
Unigram: 5.3772031790399636e-08
Bigram : 0.0005961063412244835
Trigram: 0.0018008593312161404

Sentence: machine learning improves technology
Unigram: 1.086297855185503e-06
Bigram : 0.003121019364979144
Trigram: 0.0034318262726949094

Sentence: perplexity evaluates models
Unigram: 0.00010066532817169129
Bigram : 0.0034318262726949094
Trigram: 0.0037735849056603774

Sentence: artificial intelligence is powerful
Unigram: 1.8703153444078146e-11
Bigram : 0.00013727305090779638
Trigram: 0.0001509433962264151

Sentence: healthcare uses language processing
Unigram: 1.79737304597591e-08
Bigram : 3.693132841681831e-05
Trigram: 0.0016
```

```
def perplexity(test_tokens, model):
    N = len(test_tokens)
    log_prob = 0.0

    if model == "unigram":
        for w in test_tokens:
            log_prob += math.log(unigram_prob(w))

    elif model == "bigram":
        for i in range(len(test_tokens)-1):
            log_prob += math.log(bigram_prob(test_tokens[i], test_tokens[i+1]))

    elif model == "trigram":
        for i in range(len(test_tokens)-2):
            log_prob += math.log(trigram_prob(test_tokens[i], test_tokens[i+1], test_tokens[i+2]))

    return math.exp(-log_prob / N)
```

```
print("Unigram Perplexity:", perplexity(test_tokens, "unigram"))
print("Bigram Perplexity :", perplexity(test_tokens, "bigram"))
print("Trigram Perplexity:", perplexity(test_tokens, "trigram"))
```

```
Unigram Perplexity: 22.288588810918228
Bigram Perplexity : 1.434639685958874
Trigram Perplexity: 1.1459452462746273
```

```
import matplotlib.pyplot as plt

# Example perplexity values (use your computed values if already available)
models = ['Unigram', 'Bigram', 'Trigram']
perplexities = [
    perplexity(test_tokens, "unigram"),
    perplexity(test_tokens, "bigram"),
    perplexity(test_tokens, "trigram")
]

# Plot
plt.figure()
plt.bar(models, perplexities)
plt.xlabel("N-gram Models")
plt.ylabel("Perplexity")
plt.title("Perplexity Comparison of N-gram Models")
plt.show()
```