In [15]:

```
# 1. Load Necessary Modules

import numpy as np
import nltk
import networkx as nx
from collections import Counter, defaultdict
from itertools import combinations
from nltk.corpus import stopwords as nltk_stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
import matplotlib.pyplot as plt
import math

# Download necessary NLTK resources
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('punkt_tab')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
```

Out[15]:

```
True
```

In [16]:

```
# 2. Initialize the Document

text = """
I am learning natural language processing
Natural language processing is the important module of subject artificial intellige
This domain has seen many recent advancements in terms of its execution
"""
```

In [17]:

```
# 3. Tokenize the Document

tokens = word_tokenize(text.lower())

print("Tokens :- ")

for token in tokens :
  print(token)
```

```
Tokens :-
i
am
learning
natural
language
processing
natural
language
processing
is
the
important
module
of
subject
artificial
intelligence
this
domain
has
seen
many
recent
advancements
in
terms
of
its
execution
```

In [18]:

```python
# 4.  Preprocessing (Stopword Removal and Lemmatization)

stop_words = set(nltk_stopwords.words('english'))
lemmatizer = WordNetLemmatizer()

# Remove stopwords and Lemmatize tokens
filtered_tokens = [lemmatizer.lemmatize(word) for word in tokens if word.isalnum()
print("Filtered Tokens:", filtered_tokens)
```

```
Filtered Tokens: ['learning', 'natural', 'language', 'processing', 'natural', 'language', 'processing', 'important', 'module', 'subject', 'artificial', 'intelligence', 'domain', 'seen', 'many', 'recent', 'advancement', 'term', 'execution']
```

In [19]:

```python
# 5. Generate N-Grams

def generate_n_grams(tokens, n):
    """Generates N-grams from tokenized words."""
    n_grams = [tuple(tokens[i:i + n]) for i in range(len(tokens) - n + 1)]
    return n_grams

# Generate bigrams
n = 2
n_grams = generate_n_grams(filtered_tokens, n)
print(f"{n}-grams:", n_grams)
```

```
2-grams: [('learning', 'natural'), ('natural', 'language'), ('language', 'processin
g'), ('processing', 'natural'), ('natural', 'language'), ('language', 'processing'),
('processing', 'important'), ('important', 'module'), ('module', 'subject'), ('subje
ct', 'artificial'), ('artificial', 'intelligence'), ('intelligence', 'domain'), ('do
main', 'seen'), ('seen', 'many'), ('many', 'recent'), ('recent', 'advancement'), ('a
dvancement', 'term'), ('term', 'execution')]
```

In [20]:

```python
# 6. Train the N-Gram Model

def train_grams(n_grams):
    """Trains the N-gram model by counting occurrences."""
    model = defaultdict(Counter)

    for ngram in n_grams:
        prefix = ngram[:-1]
        next_gram = ngram[-1]
        model[prefix][next_gram] += 1

    return model

# Train the model
model = train_grams(n_grams)
```

In [21]:

```python
# 7. Predict the Next Word

def predict_next_word(model, prefix_words):
    """Predicts the next word given the prefix."""
    if isinstance(prefix_words, str):
        prefix_words = prefix_words.split(" ")

    prefix = tuple(prefix_words)

    if prefix in model:
        return model[prefix].most_common(1)
    else:
        return "No Prediction"
```

In [22]:

```python
print("Prediction:", predict_next_word(model, ("natural",)))
print("Prediction:", predict_next_word(model, ("language",)))
print("Prediction:", predict_next_word(model, ("artificial",)))
```

```
Prediction: [('language', 2)]
Prediction: [('processing', 2)]
Prediction: [('intelligence', 1)]
```