```
pos_tagged_sentences = [
    [('The', 'DT'), ('study', 'NN'), ('investigated', 'VBD'), ('the', 'DT'
    [('Results', 'NNS'), ('showed', 'VBD'), ('a', 'DT'), ('significant', '
    [('Further', 'JJ'), ('analysis', 'NN'), ('will', 'MD'), ('be', 'VB'),
    [('The', 'DT'), ('proposed', 'VBN'), ('methodology', 'NN'), ('provides
    [('Experimental', 'JJ'), ('data', 'NNS'), ('was', 'VBD'), ('collected'
    [('This', 'DT'), ('paper', 'NN'), ('introduces', 'VBZ'), ('a', 'DT'),
    [('The', 'DT'), ('model', 'NN'), ('demonstrated', 'VBD'), ('high', 'JJ
    [('Future', 'JJ'), ('work', 'NN'), ('includes', 'VBZ'), ('expanding',
    [('Understanding', 'VBG'), ('the', 'DT'), ('mechanisms', 'NNS'), ('is'
    [('A', 'DT'), ('detailed', 'JJ'), ('comparison', 'NN'), ('with', 'IN')
    [('The', 'DT'), ('primary', 'JJ'), ('objective', 'NN'), ('was', 'VBD')
    [('This', 'DT'), ('framework', 'NN'), ('can', 'MD'), ('be', 'VB'), ('a
    [('Data', 'NNS'), ('preprocessing', 'NN'), ('steps', 'NNS'), ('were',
    [('The', 'DT'), ('algorithm', 'NN'), ('outperformed', 'VBD'), ('previo
    [('Limitations', 'NNS'), ('of', 'IN'), ('the', 'DT'), ('current', 'JJ'
    [('This', 'DT'), ('research', 'NN'), ('contributes', 'VBZ'), ('to', 'T
    [('Reproducibility', 'NN'), ('of', 'IN'), ('results', 'NNS'), ('is', '
    [('The', 'DT'), ('system', 'NN'), ('architecture', 'NN'), ('is', 'VBZ'
    [('Further', 'JJ'), ('refinements', 'NNS'), ('are', 'VBP'), ('needed',
    [('The', 'DT'), ('findings', 'NNS'), ('have', 'VBP'), ('implications',
    [('An', 'DT'), ('extensive', 'JJ'), ('literature', 'NN'), ('review', 'I
    [('This', 'DT'), ('work', 'NN'), ('aims', 'VBZ'), ('to', 'TO'), ('brid
    [('The', 'DT'), ('proposed', 'VBN'), ('algorithm', 'NN'), ('achieves',
    [('Ethical', 'JJ'), ('considerations', 'NNS'), ('were', 'VBD'), ('take
    [('The', 'DT'), ('source', 'NN'), ('code', 'NN'), ('is', 'VBZ'), ('ava
]

print(f"Generated {len(pos_tagged_sentences)} sample POS-tagged sentences.'
print("First sentence:")
print(pos_tagged_sentences[0])
```

```
Generated 25 sample POS-tagged sentences.
First sentence:
[('The', 'DT'), ('study', 'NN'), ('investigated', 'VBD'), ('the', 'DT'), ('effec
```

```
observations = []
states = []

for sentence in pos_tagged_sentences:
    current_words = []
    current_tags = []
    for word, tag in sentence:
        current_words.append(word)
        current_tags.append(tag)
    observations.append(current_words)
    states.append(current_tags)
```

```
print("First 3 observation sequences:")
for i in range(min(3, len(observations))):
    print(observations[i])

print("\nFirst 3 state sequences:")
for i in range(min(3, len(states))):
    print(states[i])
```

```
First 3 observation sequences:
['The', 'study', 'investigated', 'the', 'effects', 'of', 'temperature', 'on', 'm
['Results', 'showed', 'a', 'significant', 'increase', 'in', 'tensile', 'strength
['Further', 'analysis', 'will', 'be', 'conducted', 'to', 'optimize', 'the', 'pro

First 3 state sequences:
['DT', 'NN', 'VBD', 'DT', 'NNS', 'IN', 'NN', 'IN', 'NN', 'NNS', '.']
['NNS', 'VBD', 'DT', 'JJ', 'NN', 'IN', 'JJ', 'NN', '.']
['JJ', 'NN', 'MD', 'VB', 'VBN', 'TO', 'VB', 'DT', 'NN', '.']
```

```
import nltk
from nltk.tag.hmm import HiddenMarkovModelTagger

# Example observations and states (must already exist)
# observations = [['I', 'love', 'NLP'], ['NLTK', 'is', 'useful']]
# states = [['PRP', 'VBP', 'NNP'], ['NNP', 'VBZ', 'JJ']]

# 1. Prepare tagged sentences for HMM training
tagged_sentences_for_hmm = []

for i in range(len(observations)):
    tagged_sentence = list(zip(observations[i], states[i]))
    tagged_sentences_for_hmm.append(tagged_sentence)

print(f"Prepared {len(tagged_sentences_for_hmm)} tagged sentences for HMM trair
print("First prepared tagged sentence:")
print(tagged_sentences_for_hmm[0])

# 2. Train HMM Tagger
hmm_tagger = HiddenMarkovModelTagger.train(tagged_sentences_for_hmm)

print("\nHMM Tagger trained successfully.")

# 3. Correct way to get states and symbols
print(f"Number of states (POS tags): {len(hmm_tagger._states)}")
print(f"Number of symbols (words): {len(hmm_tagger._symbols)}")
```

```
Prepared 25 tagged sentences for HMM training.
First prepared tagged sentence:
[('The', 'DT'), ('study', 'NN'), ('investigated', 'VBD'), ('the', 'DT'), ('effec

HMM Tagger trained successfully.
```

```
Number of states (POS tags): 16
Number of symbols (words): 163
```

```
import nltk
from nltk.tag.hmm import HiddenMarkovModelTagger

# 2. Create a list of tagged sentences suitable for HMM training
tagged_sentences_for_hmm = []
for i in range(len(observations)):
    tagged_sentence = list(zip(observations[i], states[i]))
    tagged_sentences_for_hmm.append(tagged_sentence)

print(f"Prepared {len(tagged_sentences_for_hmm)} tagged sentences for HMM trair
print("First prepared tagged sentence:")
print(tagged_sentences_for_hmm[0])

# 3. Initialize an instance of HiddenMarkovModelTagger
hmm_tagger = HiddenMarkovModelTagger.train(tagged_sentences_for_hmm)

print("\nHMM Tagger trained successfully.")

# Calculate the number of unique states (POS tags) and symbols (words)
all_tags = set()
all_words = set()
for sentence in tagged_sentences_for_hmm:
    for word, tag in sentence:
        all_words.add(word)
        all_tags.add(tag)

print(f"Number of states (POS tags): {len(all_tags)}")
print(f"Number of symbols (words): {len(all_words)}")
```

```
Prepared 25 tagged sentences for HMM training.
First prepared tagged sentence:
[('The', 'DT'), ('study', 'NN'), ('investigated', 'VBD'), ('the', 'DT'), ('effec

HMM Tagger trained successfully.
Number of states (POS tags): 16
Number of symbols (words): 163
```

```
import pandas as pd

# 1. Get all unique POS tags (states) and convert to a sorted list
unique_pos_tags = sorted(list(all_tags))

# 2. Create a pandas DataFrame for the transition matrix
transition_matrix = pd.DataFrame(0.0, index=unique_pos_tags, columns=unique_pos

print("Initialized an empty transition matrix DataFrame.")
print(f"Dimensions: {transition_matrix.shape}")
```

```
Initialized an empty transition matrix DataFrame.
Dimensions: (16, 16)
```

```
# Fill transition matrix correctly
for from_tag in unique_pos_tags:
    for to_tag in unique_pos_tags:

        if from_tag in hmm_tagger._transitions:
            prob_dist = hmm_tagger._transitions[from_tag]
            prob = prob_dist.prob(to_tag)
        else:
            prob = 0.0

        transition_matrix.loc[from_tag, to_tag] = prob

print("Transition Matrix (first 5 rows and columns):")
print(transition_matrix.iloc[:5, :5].round(3))

print("\nFull Transition Matrix:\n")

# Format floats for readability
def format_float(x):
    return f"{x:.3f}"

display(transition_matrix.applymap(format_float))
```

```
Transition Matrix (first 5 rows and columns):
        .     CC     DT     IN     JJ
.   0.000  0.000  0.000  0.000  0.000
CC  0.015  0.015  0.015  0.015  0.167
DT  0.003  0.003  0.003  0.003  0.297
IN  0.004  0.004  0.093  0.004  0.358
JJ  0.003  0.003  0.003  0.061  0.061

Full Transition Matrix:

/tmp/ipython-input-3167171786.py:22: FutureWarning: DataFrame.applymap has been
  display(transition_matrix.applymap(format_float))
```

	.	CC	DT	IN	JJ	MD	NN	NNS	RB	TO	VB	VI
.	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.0(
CC	0.015	0.015	0.015	0.015	0.167	0.015	0.470	0.015	0.015	0.015	0.015	0.0
DT	0.003	0.003	0.003	0.003	0.297	0.003	0.493	0.101	0.003	0.003	0.003	0.0(
IN	0.004	0.004	0.093	0.004	0.358	0.004	0.314	0.093	0.004	0.004	0.004	0.0(
JJ	0.003	0.003	0.003	0.061	0.061	0.003	0.552	0.292	0.003	0.003	0.003	0.0(
MD	0.028	0.028	0.028	0.028	0.028	0.028	0.028	0.028	0.028	0.028	0.583	0.0
NN	0.210	0.089	0.002	0.210	0.002	0.036	0.089	0.089	0.002	0.019	0.002	0.0
NNS	0.402	0.004	0.004	0.149	0.004	0.004	0.076	0.004	0.004	0.004	0.004	0.1
RB	0.028	0.028	0.028	0.028	0.028	0.028	0.028	0.028	0.028	0.028	0.028	0.0
TO	0.010	0.010	0.115	0.010	0.010	0.010	0.010	0.010	0.010	0.010	0.740	0.0
VB	0.009	0.009	0.292	0.009	0.104	0.009	0.198	0.104	0.009	0.009	0.009	0.0(
VBD	0.009	0.009	0.198	0.009	0.198	0.009	0.009	0.009	0.009	0.104	0.009	0.0(
VBG	0.013	0.013	0.408	0.013	0.276	0.013	0.013	0.145	0.013	0.013	0.013	0.0
VBN	0.154	0.007	0.007	0.228	0.007	0.007	0.154	0.007	0.007	0.301	0.007	0.0(
VBP	0.022	0.022	0.022	0.022	0.022	0.022	0.022	0.239	0.239	0.022	0.022	0.0
VBZ	0.008	0.008	0.246	0.008	0.246	0.008	0.008	0.008	0.087	0.167	0.008	0.0(

```
for from_tag in unique_pos_tags:
    for to_tag in unique_pos_tags:
        prob = 0.0
        if from_tag in hmm_tagger._transitions:
            prob_dist_for_from_tag = hmm_tagger._transitions[from_tag]
            # The prob_dist object has a .prob() method to get the probability
            prob = prob_dist_for_from_tag.prob(to_tag)
        transition_matrix.loc[from_tag, to_tag] = prob
```

```
print("Transition Matrix (first 5 rows and columns):")
print(transition_matrix.iloc[:5, :5].round(3))

print("\nFull Transition Matrix:\n")
# Optional: Display the full matrix with formatting
# Using a custom format function for better readability for all values
def format_float(x):
    return f"{x:.3f}"

from IPython.display import display
display(transition_matrix.applymap(format_float))
```

```
Transition Matrix (first 5 rows and columns):
        .     CC     DT     IN     JJ
.   0.000  0.000  0.000  0.000  0.000
CC  0.015  0.015  0.015  0.015  0.167
DT  0.003  0.003  0.003  0.003  0.297
IN  0.004  0.004  0.093  0.004  0.358
JJ  0.003  0.003  0.003  0.061  0.061

Full Transition Matrix:

/tmp/ipython-input-4287236083.py:20: FutureWarning: DataFrame.applymap has been
  display(transition_matrix.applymap(format_float))
```

	.	CC	DT	IN	JJ	MD	NN	NNS	RB	TO	VB	VI
.	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.0(
CC	0.015	0.015	0.015	0.015	0.167	0.015	0.470	0.015	0.015	0.015	0.015	0.0
DT	0.003	0.003	0.003	0.003	0.297	0.003	0.493	0.101	0.003	0.003	0.003	0.0(
IN	0.004	0.004	0.093	0.004	0.358	0.004	0.314	0.093	0.004	0.004	0.004	0.0(
JJ	0.003	0.003	0.003	0.061	0.061	0.003	0.552	0.292	0.003	0.003	0.003	0.0(
MD	0.028	0.028	0.028	0.028	0.028	0.028	0.028	0.028	0.028	0.028	0.583	0.0
NN	0.210	0.089	0.002	0.210	0.002	0.036	0.089	0.089	0.002	0.019	0.002	0.0
NNS	0.402	0.004	0.004	0.149	0.004	0.004	0.076	0.004	0.004	0.004	0.004	0.1
RB	0.028	0.028	0.028	0.028	0.028	0.028	0.028	0.028	0.028	0.028	0.028	0.0
TO	0.010	0.010	0.115	0.010	0.010	0.010	0.010	0.010	0.010	0.010	0.740	0.0
VB	0.009	0.009	0.292	0.009	0.104	0.009	0.198	0.104	0.009	0.009	0.009	0.0(
VBD	0.009	0.009	0.198	0.009	0.198	0.009	0.009	0.009	0.009	0.104	0.009	0.0(
VBG	0.013	0.013	0.408	0.013	0.276	0.013	0.013	0.145	0.013	0.013	0.013	0.0
VBN	0.154	0.007	0.007	0.228	0.007	0.007	0.154	0.007	0.007	0.301	0.007	0.0(
VBP	0.022	0.022	0.022	0.022	0.022	0.022	0.022	0.239	0.239	0.022	0.022	0.0
VBZ	0.008	0.008	0.246	0.008	0.246	0.008	0.008	0.008	0.087	0.167	0.008	0.0(

```
for from_tag in unique_pos_tags:
    for to_tag in unique_pos_tags:
        prob = 0.0
        if from_tag in hmm_tagger._transitions:
            prob_dist_for_from_tag = hmm_tagger._transitions[from_tag]
            # The prob_dist object has a .prob() method to get the probability
            prob = prob_dist_for_from_tag.prob(to_tag)
        transition_matrix.loc[from_tag, to_tag] = prob
```

```
print("Transition Matrix (first 5 rows and columns):")
print(transition_matrix.iloc[:5, :5].round(3))

print("\nFull Transition Matrix:\n")
# Optional: Display the full matrix with formatting
# Using a custom format function for better readability for all values
def format_float(x):
    return f"{x:.3f}"

from IPython.display import display
display(transition_matrix.map(format_float))
```

```
Transition Matrix (first 5 rows and columns):
        .     CC     DT     IN     JJ
.   0.000  0.000  0.000  0.000  0.000
CC  0.015  0.015  0.015  0.015  0.167
DT  0.003  0.003  0.003  0.003  0.297
IN  0.004  0.004  0.093  0.004  0.358
JJ  0.003  0.003  0.003  0.061  0.061

Full Transition Matrix:
```

	.	CC	DT	IN	JJ	MD	NN	NNS	RB	TO	VB	VI
.	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.0(
CC	0.015	0.015	0.015	0.015	0.167	0.015	0.470	0.015	0.015	0.015	0.015	0.0
DT	0.003	0.003	0.003	0.003	0.297	0.003	0.493	0.101	0.003	0.003	0.003	0.0(
IN	0.004	0.004	0.093	0.004	0.358	0.004	0.314	0.093	0.004	0.004	0.004	0.0(
JJ	0.003	0.003	0.003	0.061	0.061	0.003	0.552	0.292	0.003	0.003	0.003	0.0(
MD	0.028	0.028	0.028	0.028	0.028	0.028	0.028	0.028	0.028	0.028	0.583	0.0
NN	0.210	0.089	0.002	0.210	0.002	0.036	0.089	0.089	0.002	0.019	0.002	0.0
NNS	0.402	0.004	0.004	0.149	0.004	0.004	0.076	0.004	0.004	0.004	0.004	0.1
RB	0.028	0.028	0.028	0.028	0.028	0.028	0.028	0.028	0.028	0.028	0.028	0.0
TO	0.010	0.010	0.115	0.010	0.010	0.010	0.010	0.010	0.010	0.010	0.740	0.0
VB	0.009	0.009	0.292	0.009	0.104	0.009	0.198	0.104	0.009	0.009	0.009	0.0(
VBD	0.009	0.009	0.198	0.009	0.198	0.009	0.009	0.009	0.009	0.104	0.009	0.0(
VBG	0.013	0.013	0.408	0.013	0.276	0.013	0.013	0.145	0.013	0.013	0.013	0.0
VBN	0.154	0.007	0.007	0.228	0.007	0.007	0.154	0.007	0.007	0.301	0.007	0.0(
VBP	0.022	0.022	0.022	0.022	0.022	0.022	0.022	0.239	0.239	0.022	0.022	0.0
VBZ	0.008	0.008	0.246	0.008	0.246	0.008	0.008	0.008	0.087	0.167	0.008	0.0(

```
import operator

# 1. Define a list of specific POS tags
selected_pos_tags = ['NN', 'VBD', 'JJ', 'DT', 'IN']
# Noun, Verb Past Tense, Adjective, Determiner, Preposition

print("\n--- Emission Probability Examples ---")

# Ensure all_words is a list for consistent iteration
all_words_list = list(all_words)

for tag in selected_pos_tags:
    print(f"\nPOS Tag: {tag}")

    # ✅ Correct attribute: _outputs (emission distributions)
    if tag in hmm_tagger._outputs:
        prob_dist_for_tag = hmm_tagger._outputs[tag]

        # Collect probabilities for all words for the current tag
        word_probabilities = {}
        for word in all_words_list:
            word_probabilities[word] = prob_dist_for_tag.prob(word)

        # Sort words by probability (descending) and get top 5
        sorted_words = sorted(
            word_probabilities.items(),
            key=operator.itemgetter(1),
            reverse=True
        )

        top_5_words = sorted_words[:5]

        # Print top 5 words with non-zero probability
        for word, prob in top_5_words:
            if prob > 0:
                print(f"  Word: '{word}', Probability: {prob:.4f}")
    else:
        print(f"  No emission distribution found for tag '{tag}'.")
```

```
--- Emission Probability Examples ---

POS Tag: NN
  Word: 'process', Probability: 0.0290
  Word: 'work', Probability: 0.0290
  Word: 'system', Probability: 0.0290
  Word: 'algorithm', Probability: 0.0290
  Word: 'performance', Probability: 0.0290

POS Tag: VBD
  Word: 'was', Probability: 0.1225
```

```
  Word: 'were', Probability: 0.0830
  Word: 'outperformed', Probability: 0.0435
  Word: 'demonstrated', Probability: 0.0435
  Word: 'investigated', Probability: 0.0435

POS Tag: JJ
  Word: 'various', Probability: 0.0629
  Word: 'state-of-the-art', Probability: 0.0426
  Word: 'Further', Probability: 0.0426
  Word: 'available', Probability: 0.0223
  Word: 'Ethical', Probability: 0.0223

POS Tag: DT
  Word: 'The', Probability: 0.2009
  Word: 'the', Probability: 0.2009
  Word: 'This', Probability: 0.0905
  Word: 'a', Probability: 0.0905
  Word: 'A', Probability: 0.0243

POS Tag: IN
  Word: 'for', Probability: 0.1635
  Word: 'of', Probability: 0.1099
  Word: 'in', Probability: 0.1099
  Word: 'on', Probability: 0.0563
  Word: 'throughout', Probability: 0.0295
```

```
import operator

# 1. Define a list of specific POS tags
selected_pos_tags = ['NN', 'VBD', 'JJ', 'DT', 'IN'] # Noun, Verb Past Tense, Ad

print("\n--- Emission Probability Examples ---")

# Ensure all_words is a list for consistent iteration
all_words_list = list(all_words)

for tag in selected_pos_tags:
    print(f"\nPOS Tag: {tag}")

    # Check if the tag exists in the HMM's emission distributions (use _outputs
    if tag in hmm_tagger._outputs:
        prob_dist_for_tag = hmm_tagger._outputs[tag]

        # Collect probabilities for all words for the current tag
        word_probabilities = {}
        for word in all_words_list:
            # The prob_dist object has a .prob() method to get the probability
            word_probabilities[word] = prob_dist_for_tag.prob(word)

        # Sort words by probability in descending order and get the top 5
        sorted_words = sorted(word_probabilities.items(), key=operator.itemgett
        top_5_words = sorted_words[:5]
```

```
            # Print the top 5 words and their probabilities
            for word, prob in top_5_words:
                if prob > 0: # Only print words with non-zero probability
                    print(f"  Word: '{word}', Probability: {prob:.4f}")
        else:
            print(f"  No emission distribution found for tag '{tag}'.")
```

```
--- Emission Probability Examples ---

POS Tag: NN
  Word: 'process', Probability: 0.0290
  Word: 'work', Probability: 0.0290
  Word: 'system', Probability: 0.0290
  Word: 'algorithm', Probability: 0.0290
  Word: 'performance', Probability: 0.0290

POS Tag: VBD
  Word: 'was', Probability: 0.1225
  Word: 'were', Probability: 0.0830
  Word: 'outperformed', Probability: 0.0435
  Word: 'demonstrated', Probability: 0.0435
  Word: 'investigated', Probability: 0.0435

POS Tag: JJ
  Word: 'various', Probability: 0.0629
  Word: 'state-of-the-art', Probability: 0.0426
  Word: 'Further', Probability: 0.0426
  Word: 'available', Probability: 0.0223
  Word: 'Ethical', Probability: 0.0223

POS Tag: DT
  Word: 'The', Probability: 0.2009
  Word: 'the', Probability: 0.2009
  Word: 'This', Probability: 0.0905
  Word: 'a', Probability: 0.0905
  Word: 'A', Probability: 0.0243

POS Tag: IN
  Word: 'for', Probability: 0.1635
  Word: 'of', Probability: 0.1099
  Word: 'in', Probability: 0.1099
  Word: 'on', Probability: 0.0563
  Word: 'throughout', Probability: 0.0295
```

Reasoning: To identify the most frequent tag transitions, I will convert the transition matrix into a list of (from_tag, to_tag, probability) tuples, sort them by probability in descending order, and then print the top 10.

```
import operator

# 1. Convert the transition_matrix DataFrame into a series of (from_tag, to_t
```

```
    transition_tuples = []
    for from_tag in transition_matrix.index:
```