```
import nltk
from nltk import word_tokenize, bigrams
from collections import defaultdict, Counter

nltk.download('punkt')
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 punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
True
```

```
# Step 1: Training corpus
corpus = [
    "I love NLP",
    "I love Python",
    "Python is great",
    "NLP is fun"
]

# Step 2: Tokenize corpus
tokenized_sentences = []
for sentence in corpus:
    tokens = word_tokenize(sentence.lower())  # Lowercase and tokenize
    tokenized_sentences.append(tokens + ['<END>'])  # Add end token for each sentence

# Step 3: Build bigram frequency counts
bigram_model = defaultdict(Counter)

for tokens in tokenized_sentences:
    for w1, w2 in bigrams(tokens, pad_right=True, pad_left=False):
        bigram_model[w1][w2] += 1

# Step 4: Function to compute conditional probabilities
def get_bigram_probabilities(word):
    word = word.lower()
    next_words = bigram_model[word]
    total = sum(next_words.values())
    return {w2: count / total for w2, count in next_words.items()}

# Step 5: Test the model
query_word1 = 'i'
probs = get_bigram_probabilities(query_word1)

print(f"Bigram probabilities for '{query_word1}':")
for word, prob in probs.items():
    print(f"P({word} | {query_word1}) = {prob:.2f}")

query_word2 = 'love'
probs = get_bigram_probabilities(query_word2)

print(f"\nBigram probabilities for '{query_word2}':")
for word, prob in probs.items():
    print(f"P({word} | {query_word2}) = {prob:.2f}")
```

```
Bigram probabilities for 'i':
P(love | i) = 1.00

Bigram probabilities for 'love':
P(nlp | love) = 0.50
P(python | love) = 0.50
```