```
# DeltaPattern Adaptive Interface  Prototype Implementation

import spacy
from collections import Counter
from sentence_transformers import SentenceTransformer, util

# Load models
nlp = spacy.load('en_core_web_sm')
embedder = SentenceTransformer('all-MiniLM-L6-v2')

# Step 1: Pattern Detection
def extract_patterns(text):
    doc = nlp(text)
    tokens = [token.lemma_.lower() for token in doc if token.is_alpha]
    return Counter(tokens)

# Step 2: ATM - Associative Tokenized Memory
class ATM:
    def __init__(self):
        self.token_memory = Counter()

    def update_memory(self, tokens):
        self.token_memory.update(tokens)

    def get_signature_vector(self):
        tokens = list(self.token_memory.elements())
        joined = ' '.join(tokens[:100])
```

```
        return embedder.encode(joined, convert_to_tensor=True)


# Step 3: Behavior Modulation
def modulate_response(user_input, atm_vector):
    prompt = f"User style detected. Emulate signature context. Input: {user_input}"
    return prompt  # this would be fed into an LLM in a real implementation


# Usage Example
atm = ATM()
for message in ["Hello, I like theoretical systems.", "My thinking is layered like logic."]:
    patterns = extract_patterns(message)
    atm.update_memory(patterns)

signature = atm.get_signature_vector()
response = modulate_response("Tell me about quantum fields.", signature)
print(response)
```