Unified Attention Template

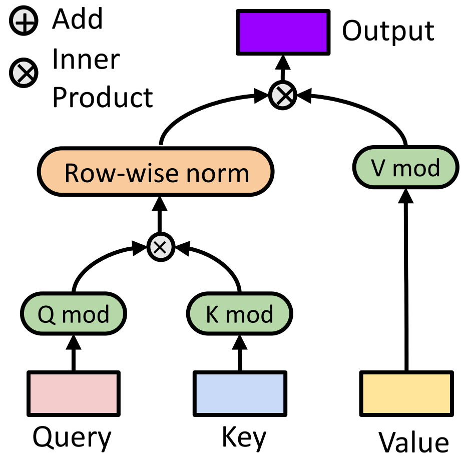
```
state = initialize_state()  # Initialize the state
output = []  # To store final outputs for each token
Q = Qmod(Q)
K = Kmod(K)
V = Vmod(V)

for i in range(sequence_length):
   # Compute relevance for the current position
   relevance = relevance_scoring(Q[i], K, state)
   # Apply normalization or weighting
   weights = customizable_function(relevance)
   # Update state results for the current position
   state = aggregate(weights, V, state)
   # Store the output for the current token
   output.append(customizable_function(state))
```

Instantiation

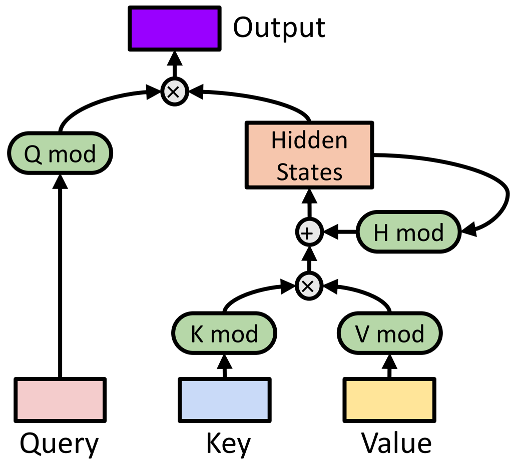
a. Parallel Pattern

```
def Parallel(Query, Key, Value):
    Q = Qmod(Query)
    K = Kmod(Key)
    scores = Q @ K
    scores = customizeable_function(scores)
    V = Vmod(Value)
    state = scores @ V
    output = customizeable_function(state)
    return output
```



b. Recurrent Pattern

```
def Recurrent(Query, Key, Value):
    K = Kmod(Key)
    V = Vmod(Value)
    for i in range(sequence_length):
        h[i] = K[i] @ V[i]
        + customizeable_function(h[i-1])
    output = Qmod(Query) @ h
    return output
```


Output
Q mod
Hidden States
H mod
K mod
V mod
Query
Key
Value