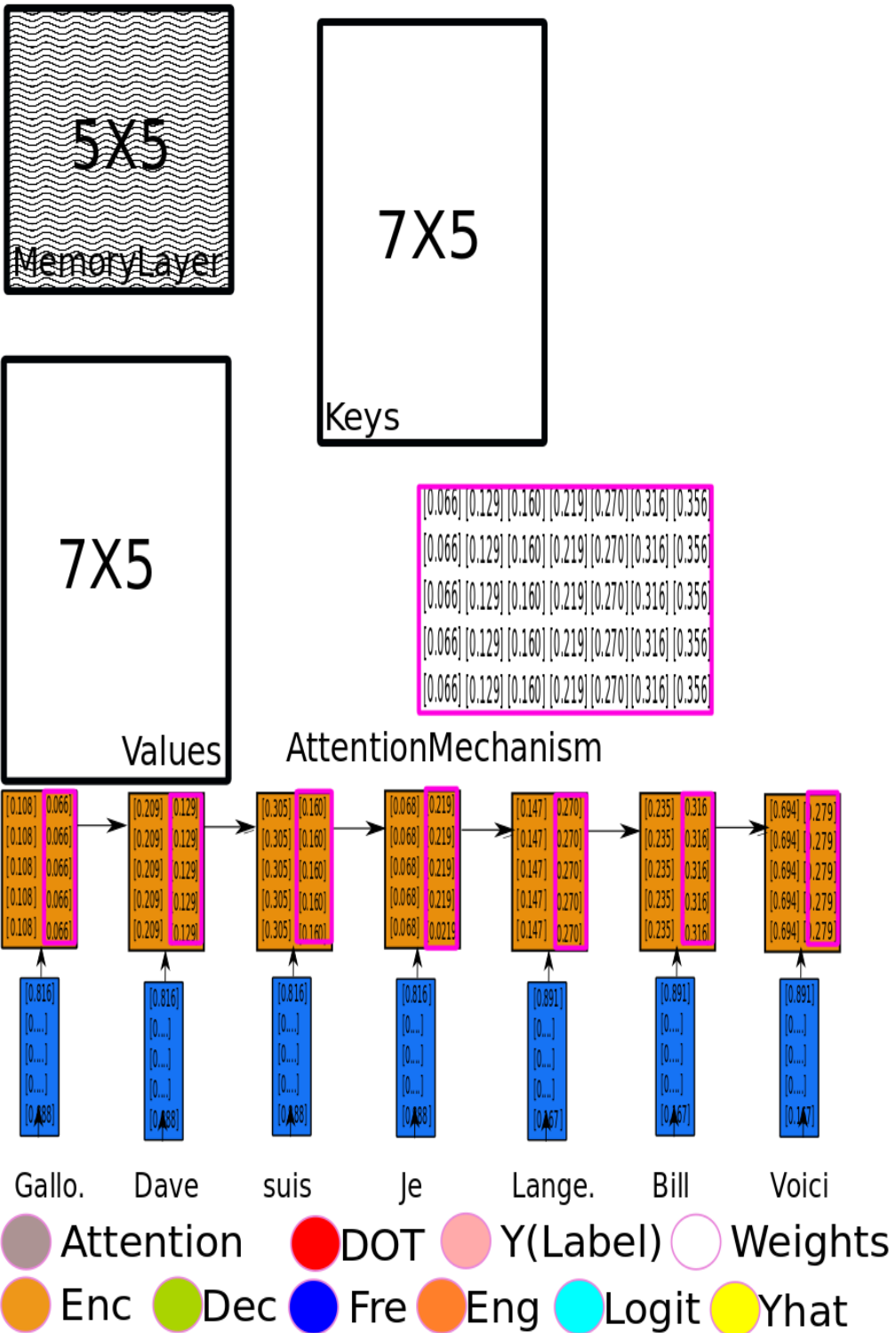
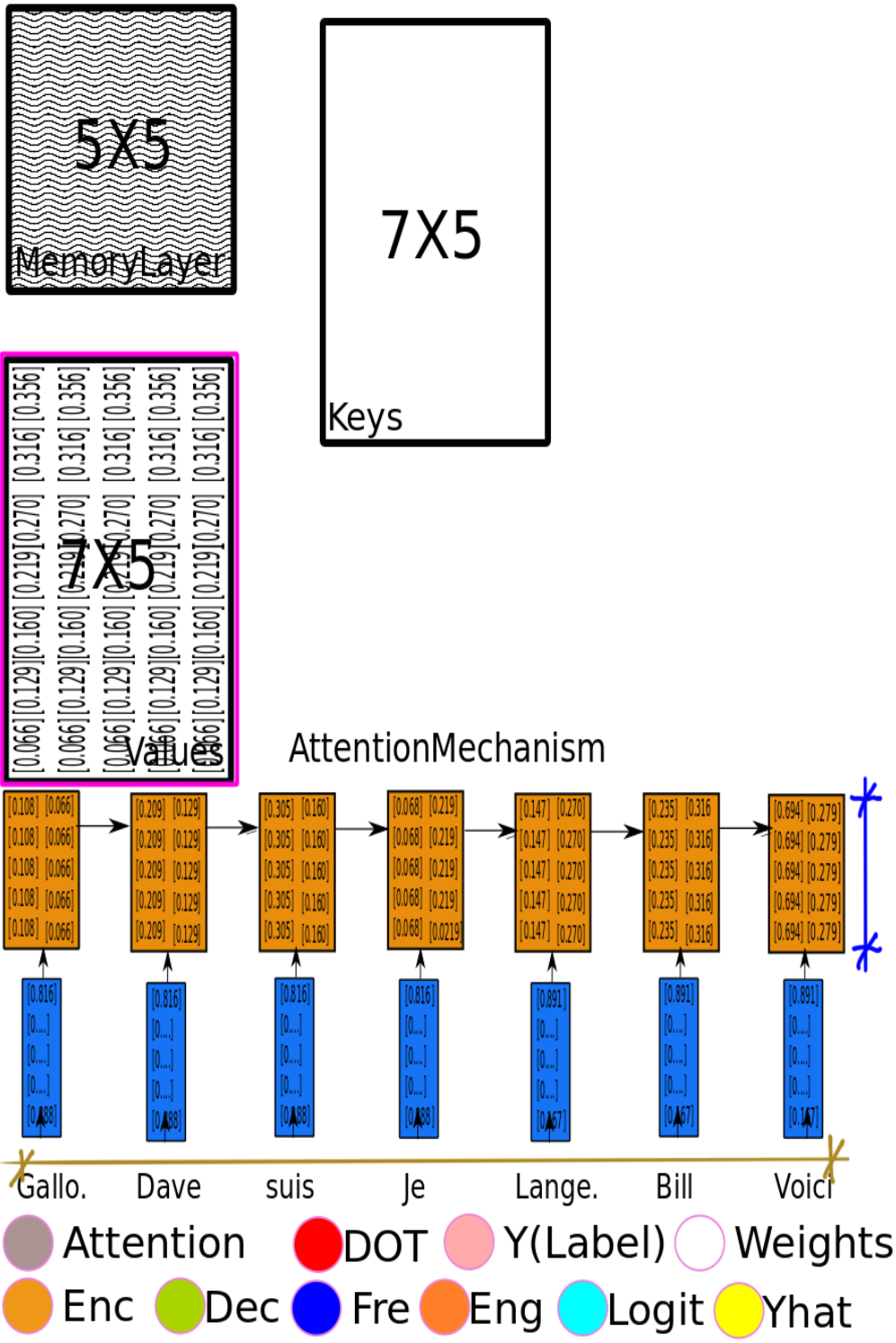
Attention Architecture Internals Sample Slides
By Mohit Kumar

NMT:Advanced:Better Performance:Attention(Detailed)



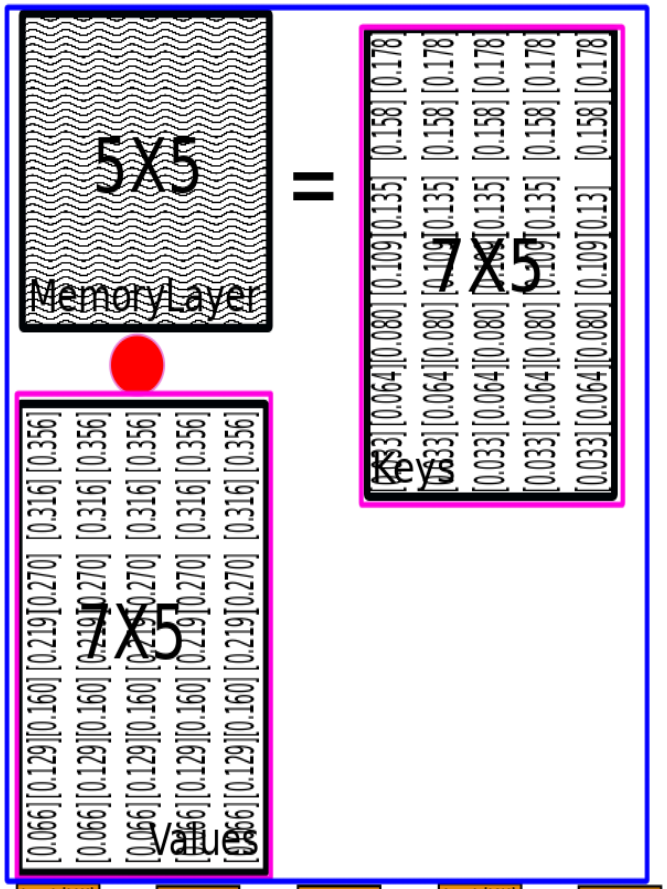
- All the h-states of the encoder sequence.

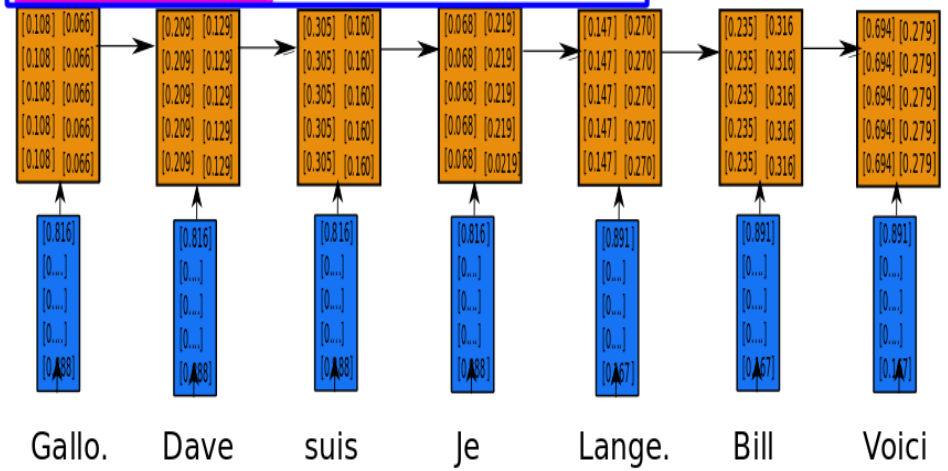
NMT:Advanced:Better Performance:Attention(Detailed)



- Values initialized with all the h-states of the encoder sequence.
- Assuming state-size of LSTM/RNN cell is 5 for the example and the sequence size being 7. The shape of values is 7X5.

NMT:Advanced:Better Performance:Attention(Detailed)



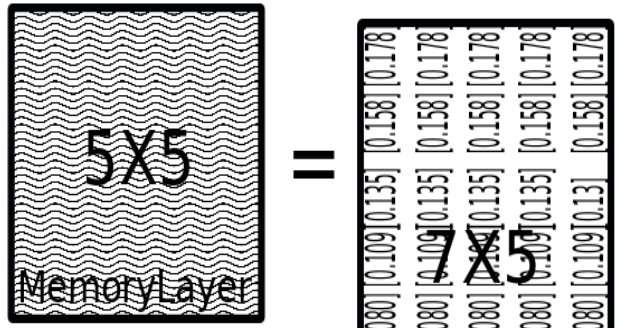


Attention DOT Y(Label) Weights

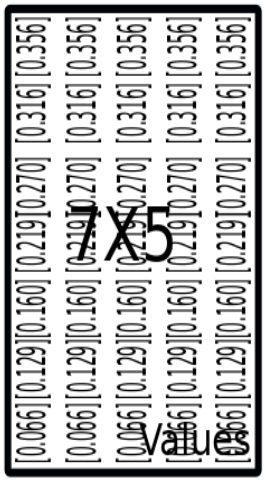
Enc Dec Fre Eng Logit Yhat

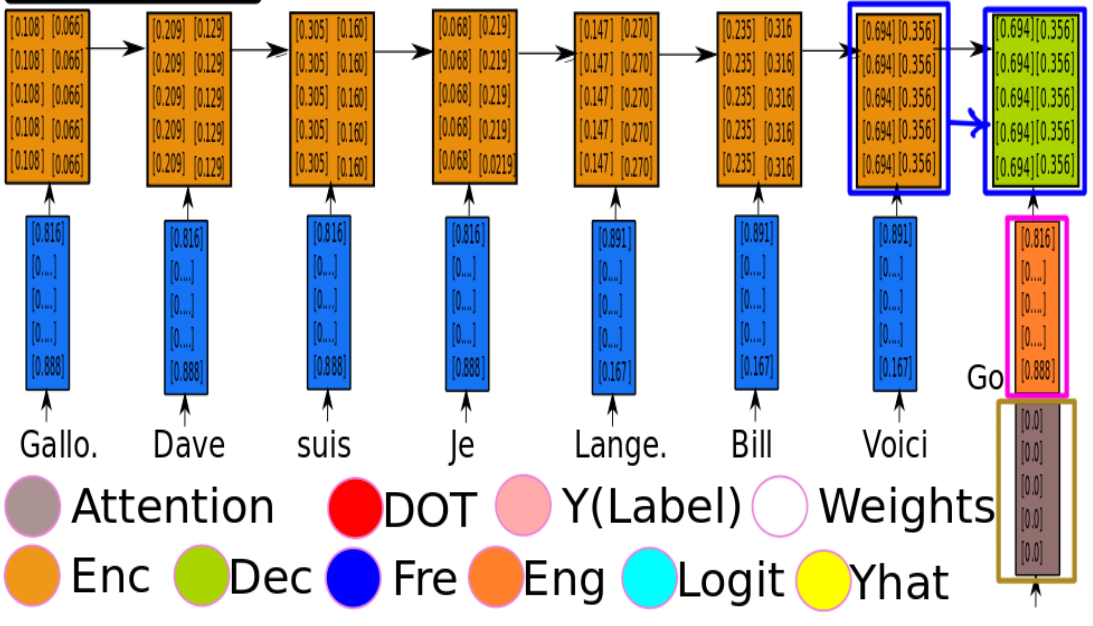
- Idea here is to use all encoder's h-state to calculate the next $\hat{y}$.
- This is a part of Attention mechanism initialization

NMT:Advanced:Better Performance:Attention(Detailed)



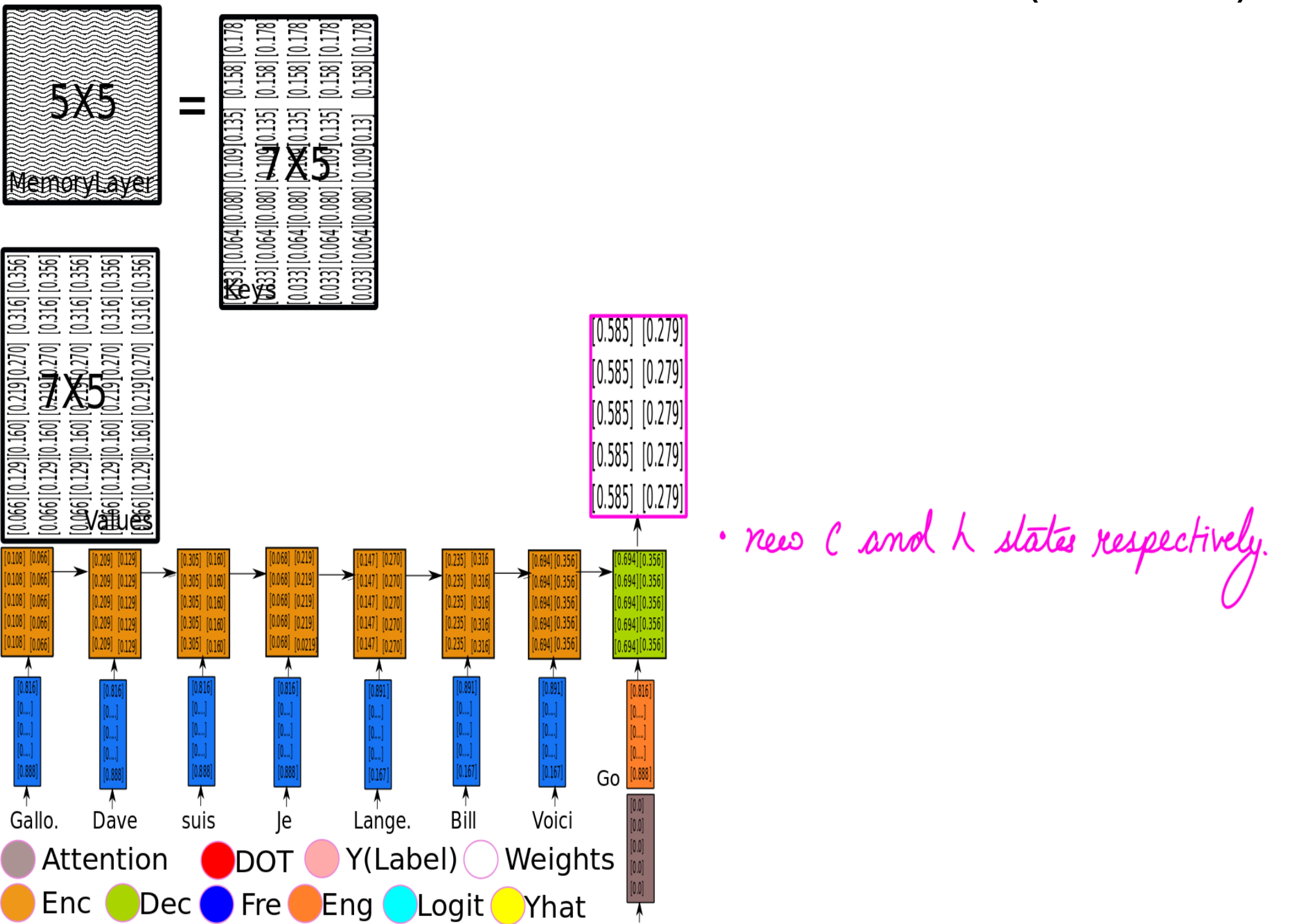
=





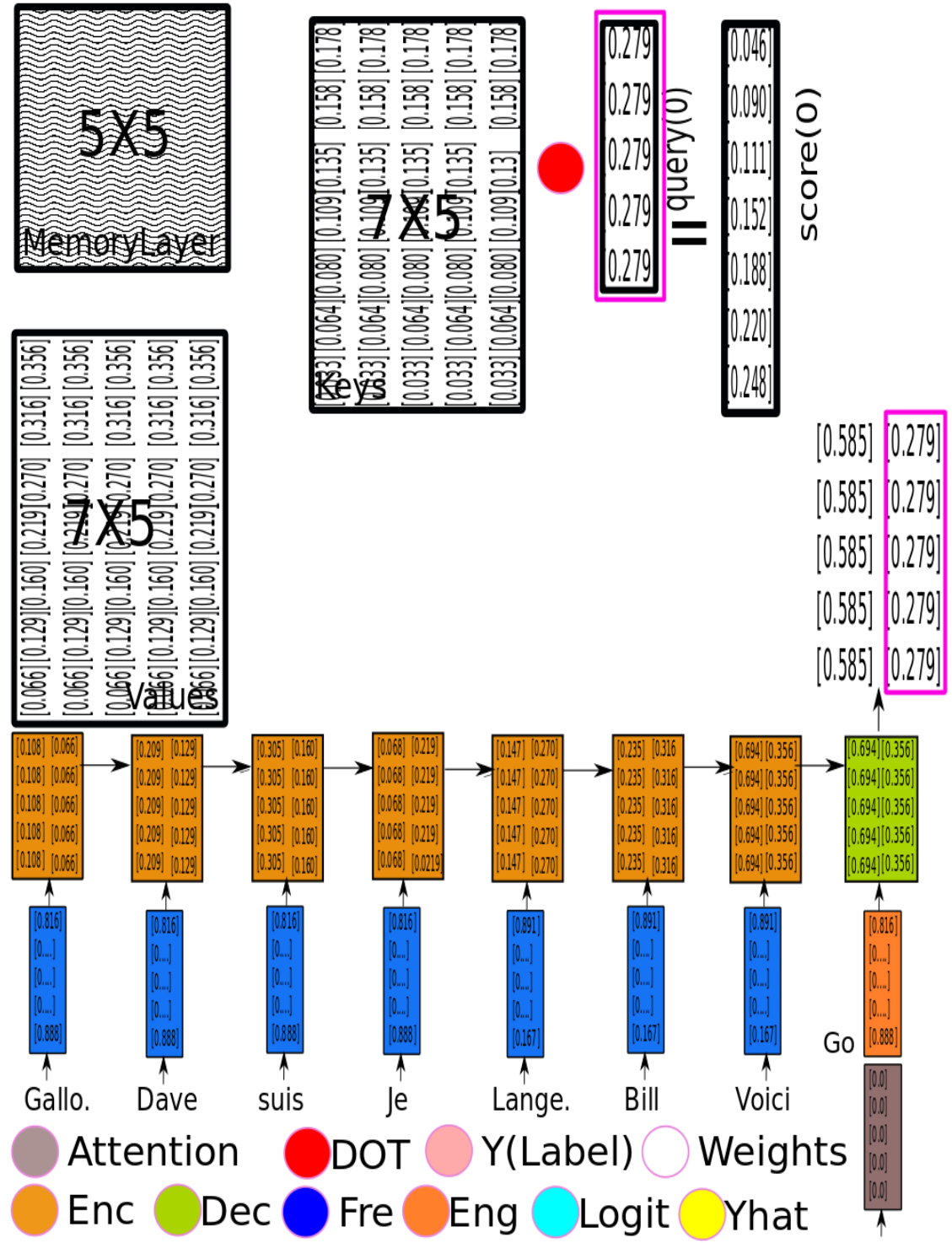
- The decoder LSTM is initialized with the last encoder state.
- Pick up the embedding of the GO token from the English embedding.
- Concatenate the embedding with attention state at time (t_0).

NMT:Advanced:Better Performance:Attention(Detailed)



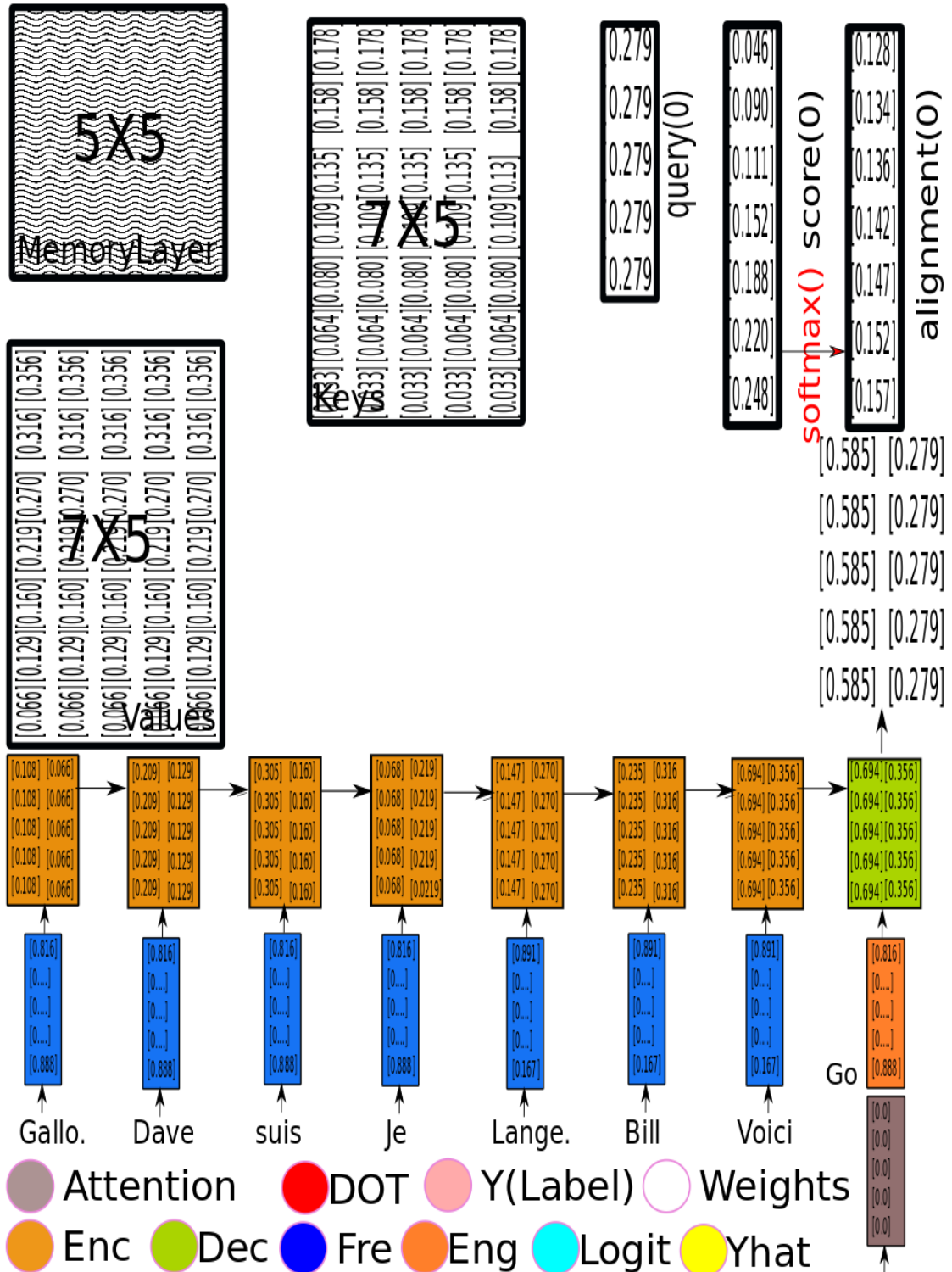
• new C and h states respectively.

NMT:Advanced:Better Performance:Attention(Detailed)

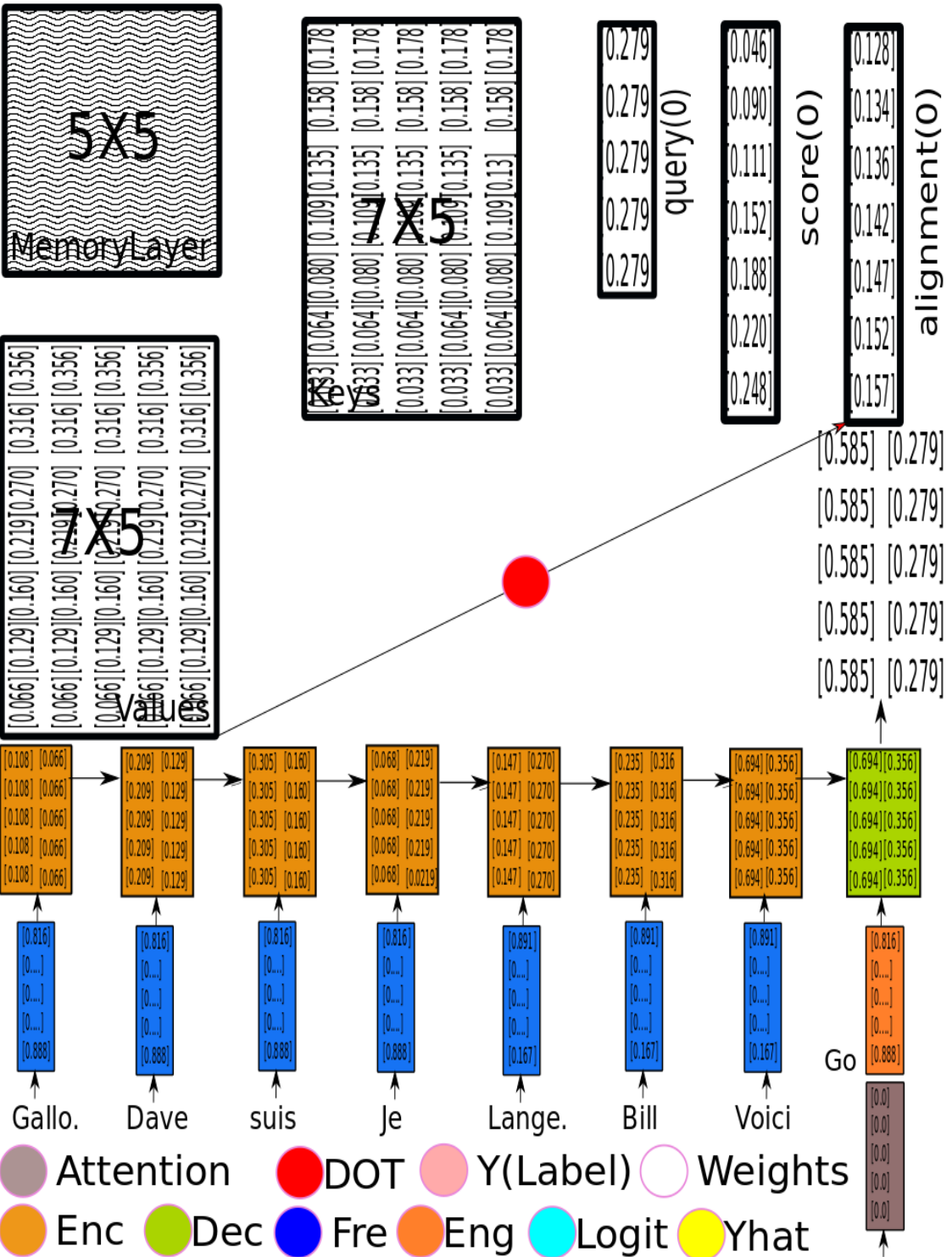


- multiply keys with the current h-State of the decoder.

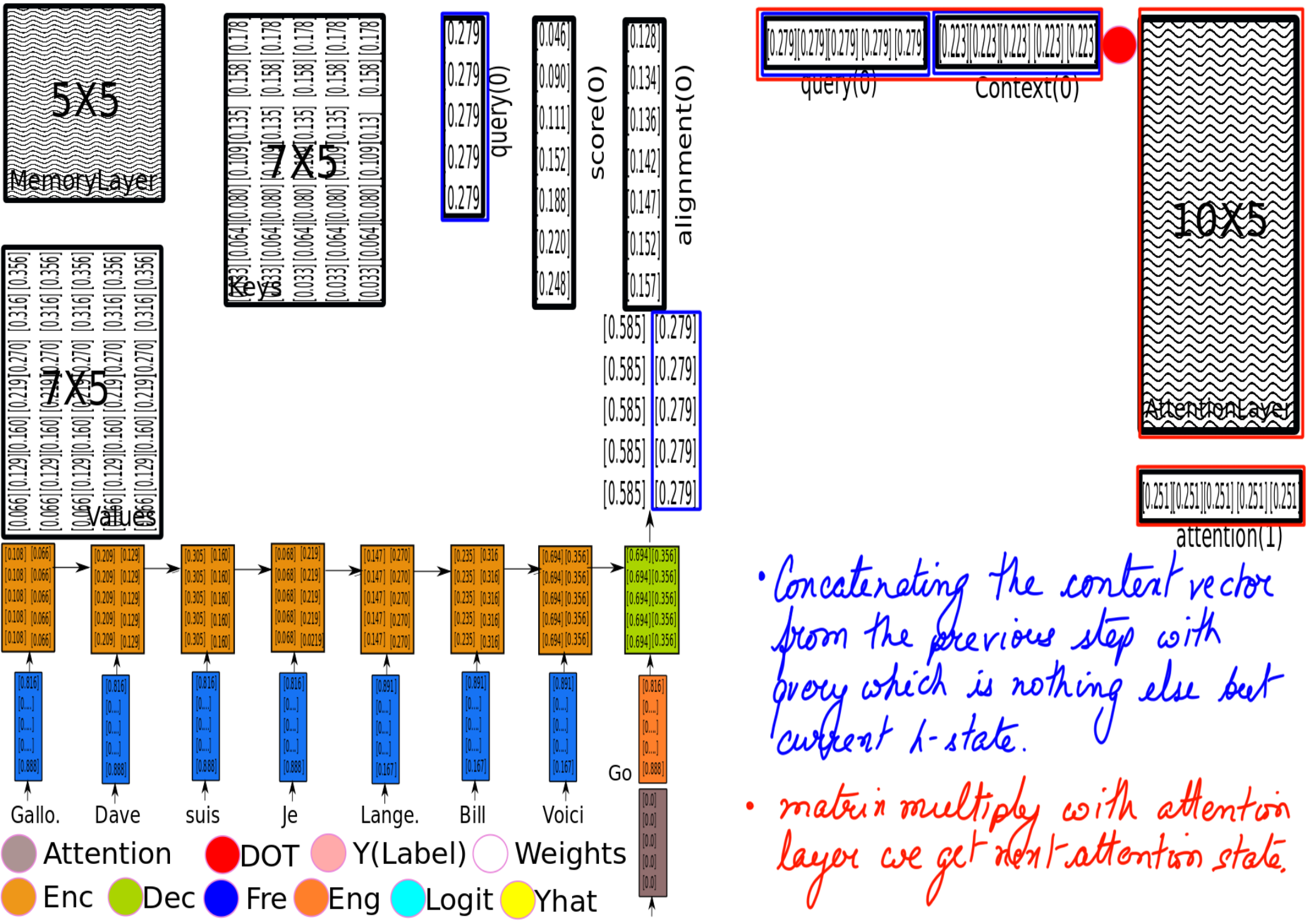
NMT:Advanced:Better Performance:Attention(Detailed)



NMT:Advanced:Better Performance:Attention(Detailed)

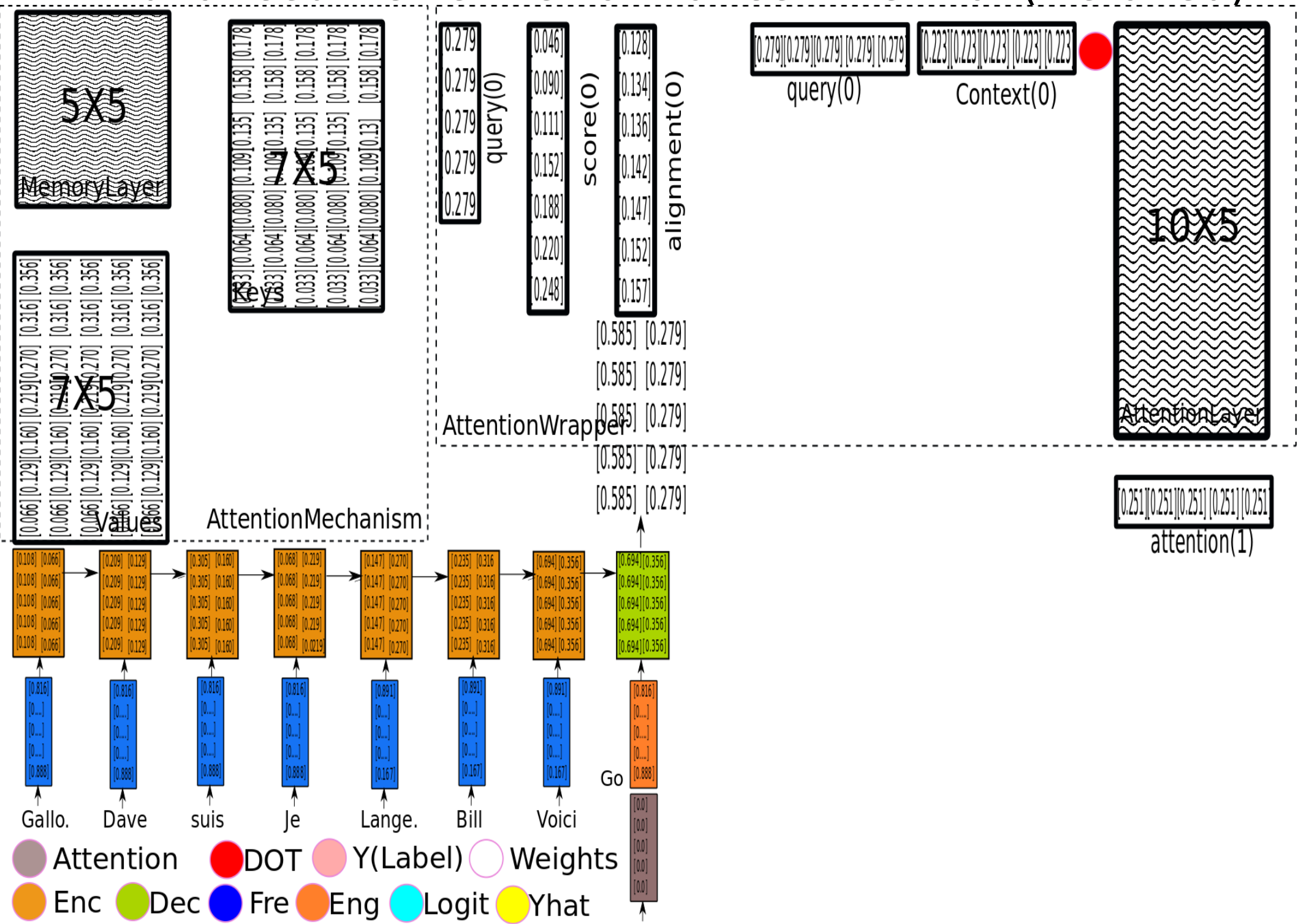


NMT:Advanced:Better Performance:Attention(Detailed)

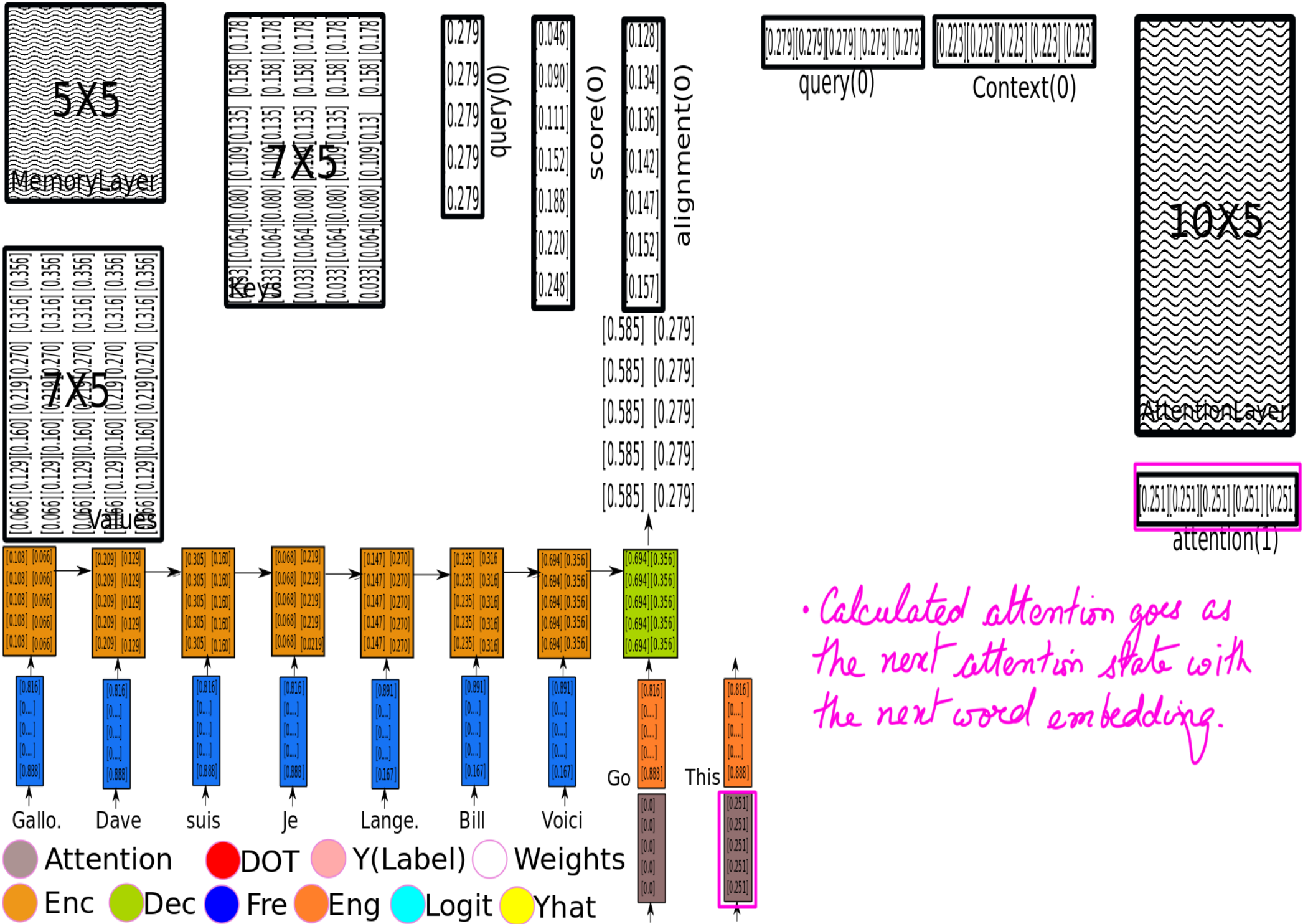


- Concatenating the content vector from the previous step with query which is nothing else but current t-state.
- matrix multiply with attention layer we get next-attention state.

NMT:Advanced:Better Performance:Attention(Detailed)

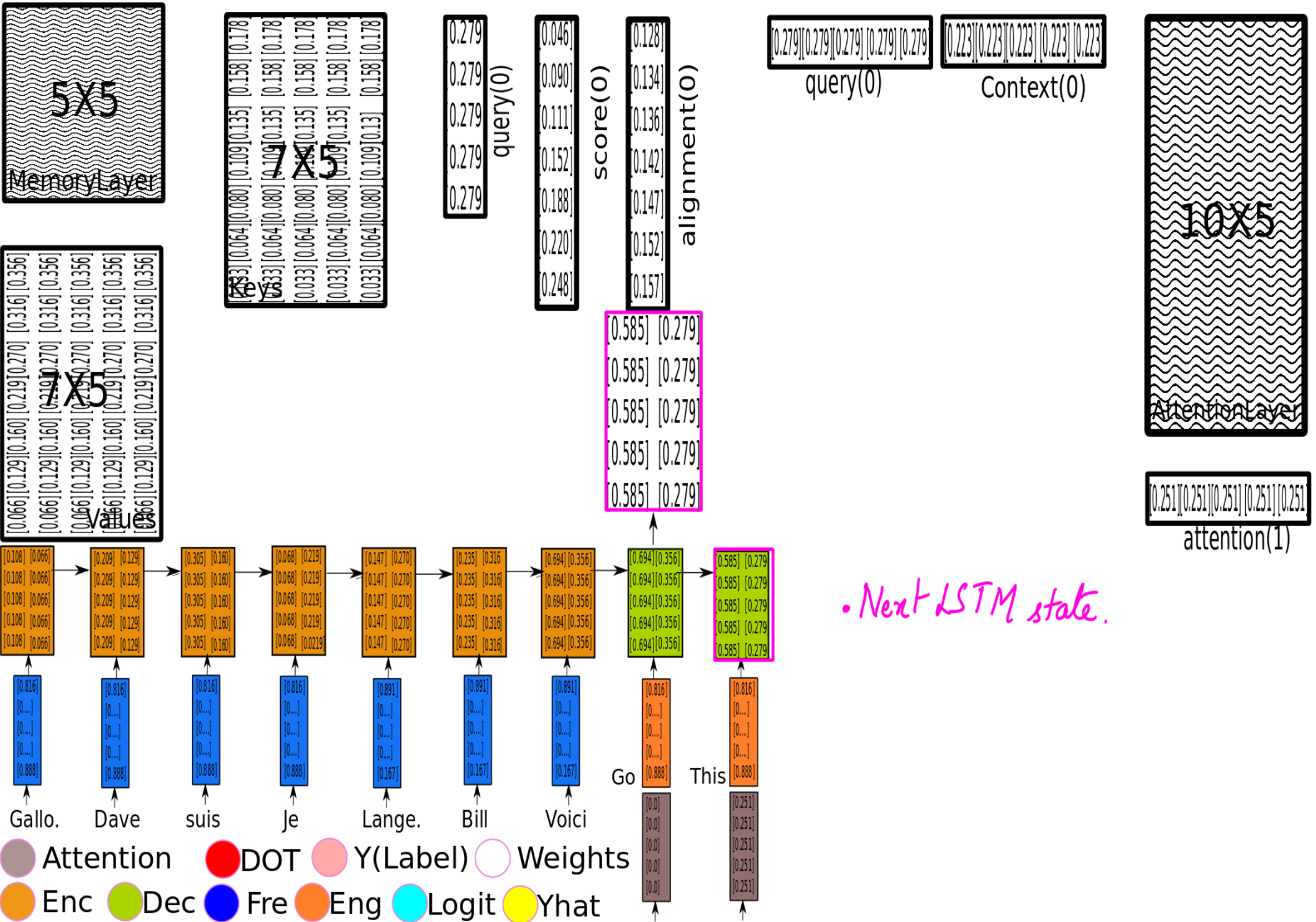


NMT:Advanced:Better Performance:Attention(Detailed)



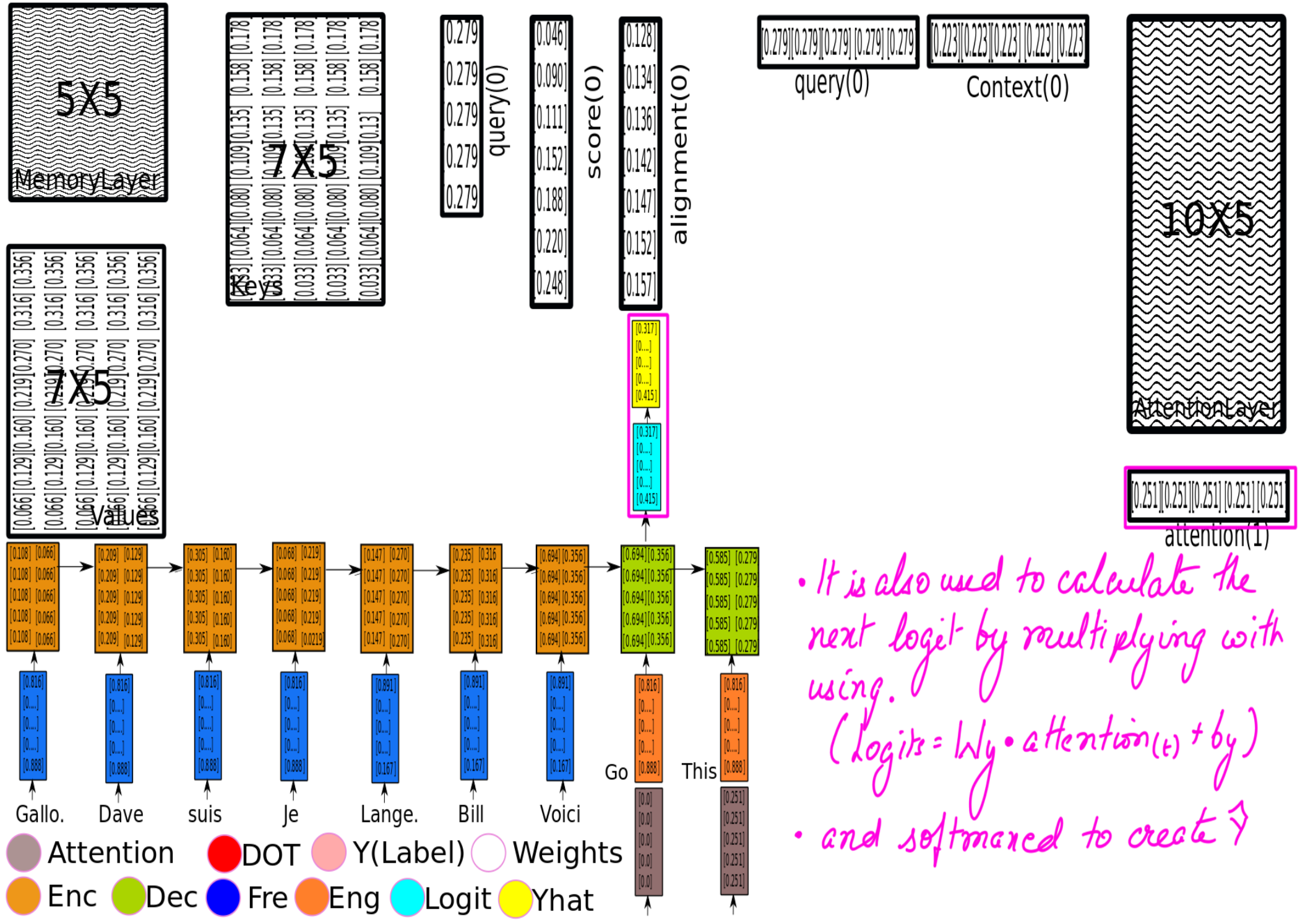
- Calculated attention goes as the next attention state with the next word embedding.

NMT:Advanced:Better Performance:Attention(Detailed)



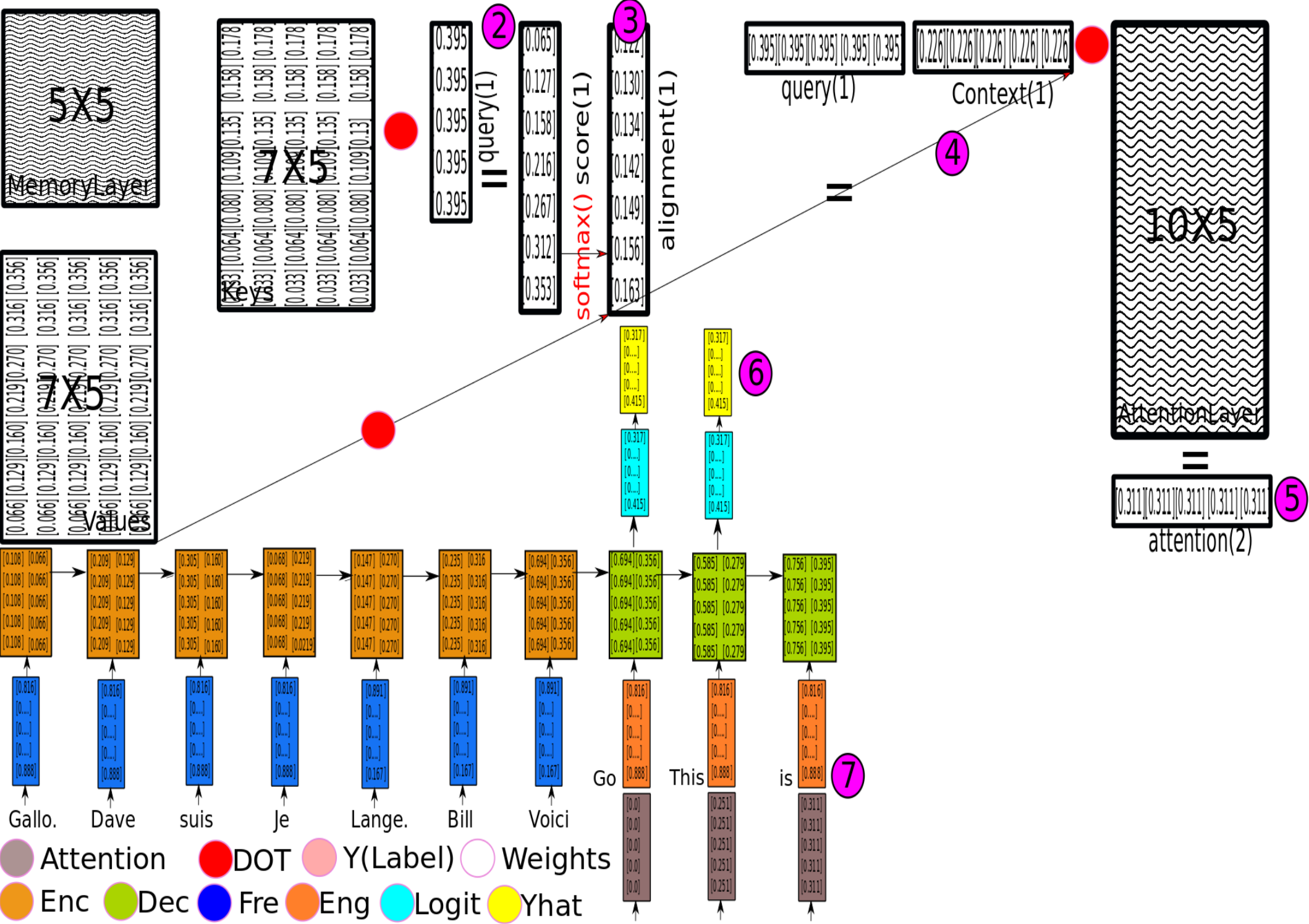
• Next LSTM state.

NMT:Advanced:Better Performance:Attention(Detailed)

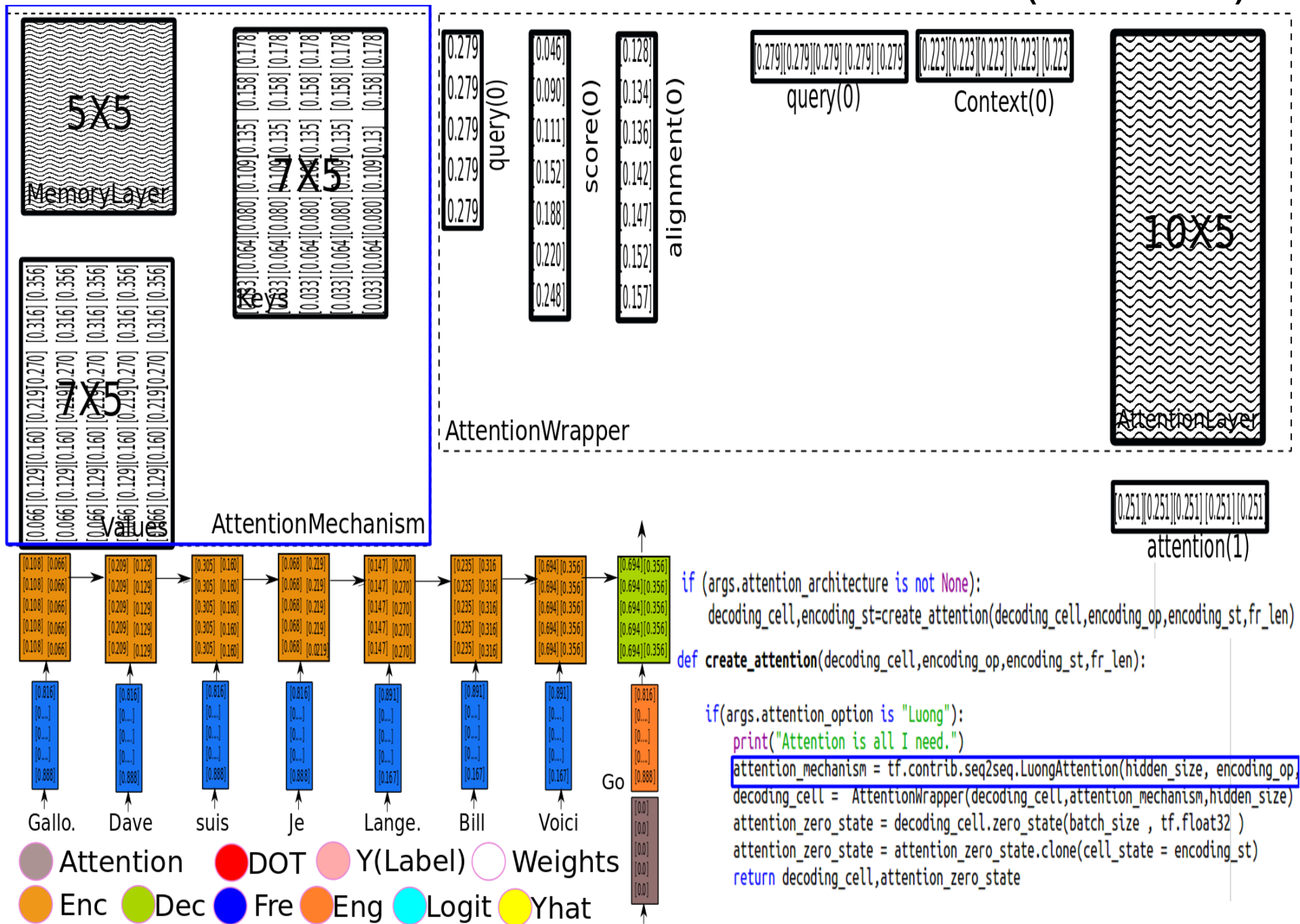


- It is also used to calculate the next logit by multiplying with using.
  (logits = $W_y \cdot attention_{(t)} + b_y$)
- and softmaxed to create ŷ

NMT:Advanced:Better Performance:Attention(Detailed)

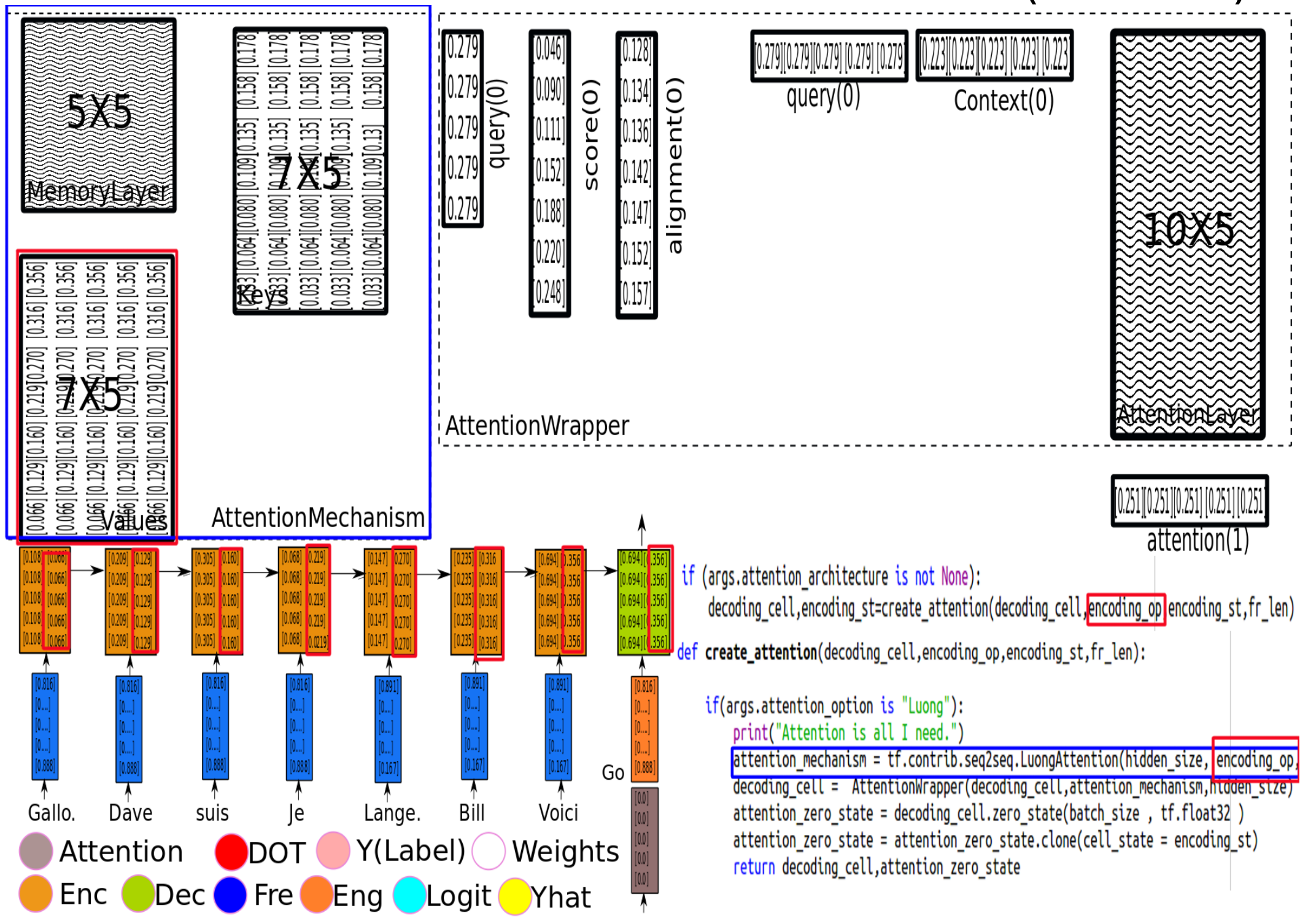


NMT:Advanced:Better Performance:Attention(Detailed)



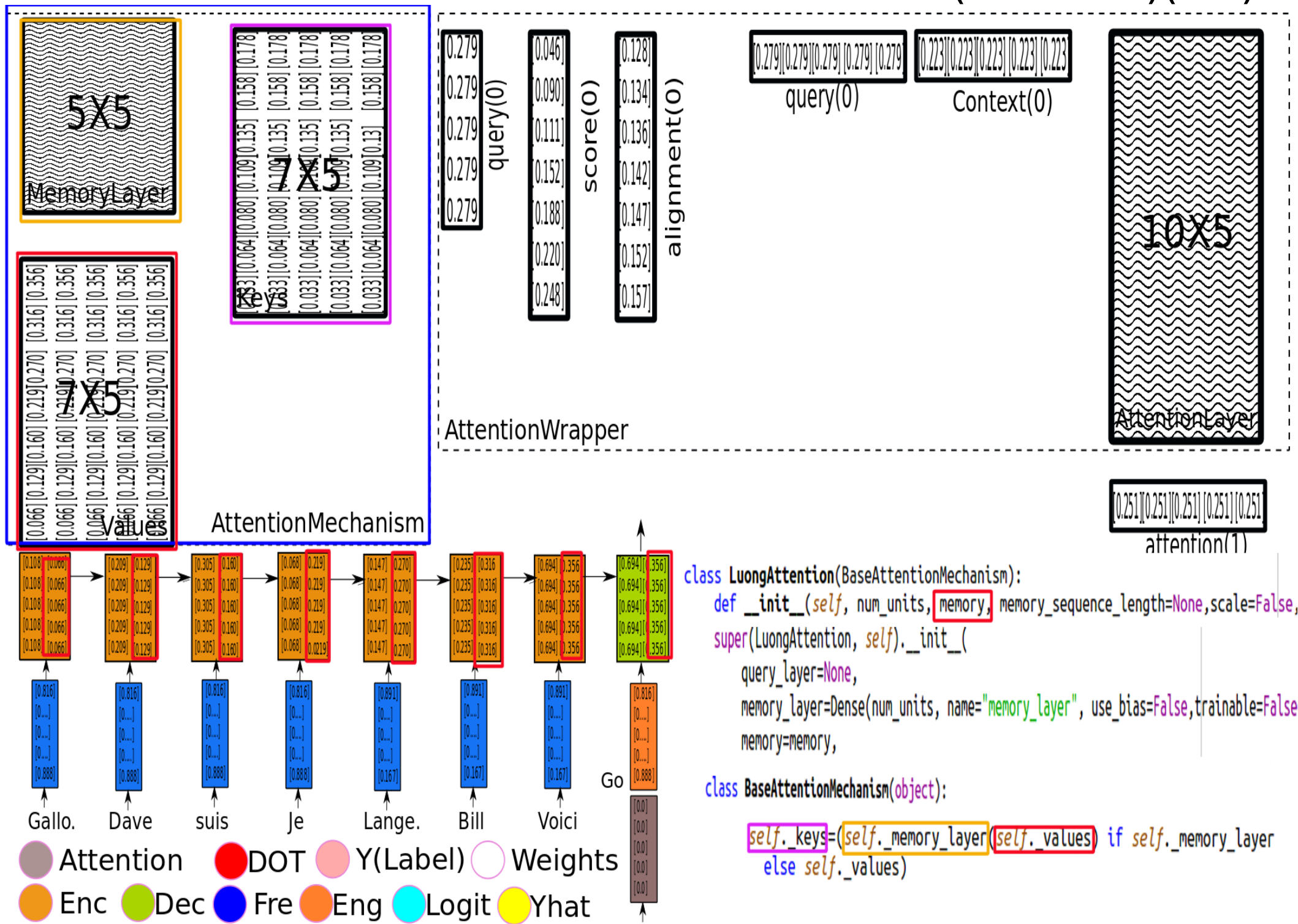

```
if (args.attention_architecture is not None):
    decoding_cell,encoding_st=create_attention(decoding_cell,encoding_op,encoding_st,fr_len)

def create_attention(decoding_cell,encoding_op,encoding_st,fr_len):

    if(args.attention_option is "Luong"):
        print("Attention is all I need.")
        attention_mechanism = tf.contrib.seq2seq.LuongAttention(hidden_size, encoding_op,
        decoding_cell =  AttentionWrapper(decoding_cell,attention_mechanism,hidden_size)
        attention_zero_state = decoding_cell.zero_state(batch_size , tf.float32 )
        attention_zero_state = attention_zero_state.clone(cell_state = encoding_st)
        return decoding_cell,attention_zero_state
```

NMT:Advanced:Better Performance:Attention(Detailed)



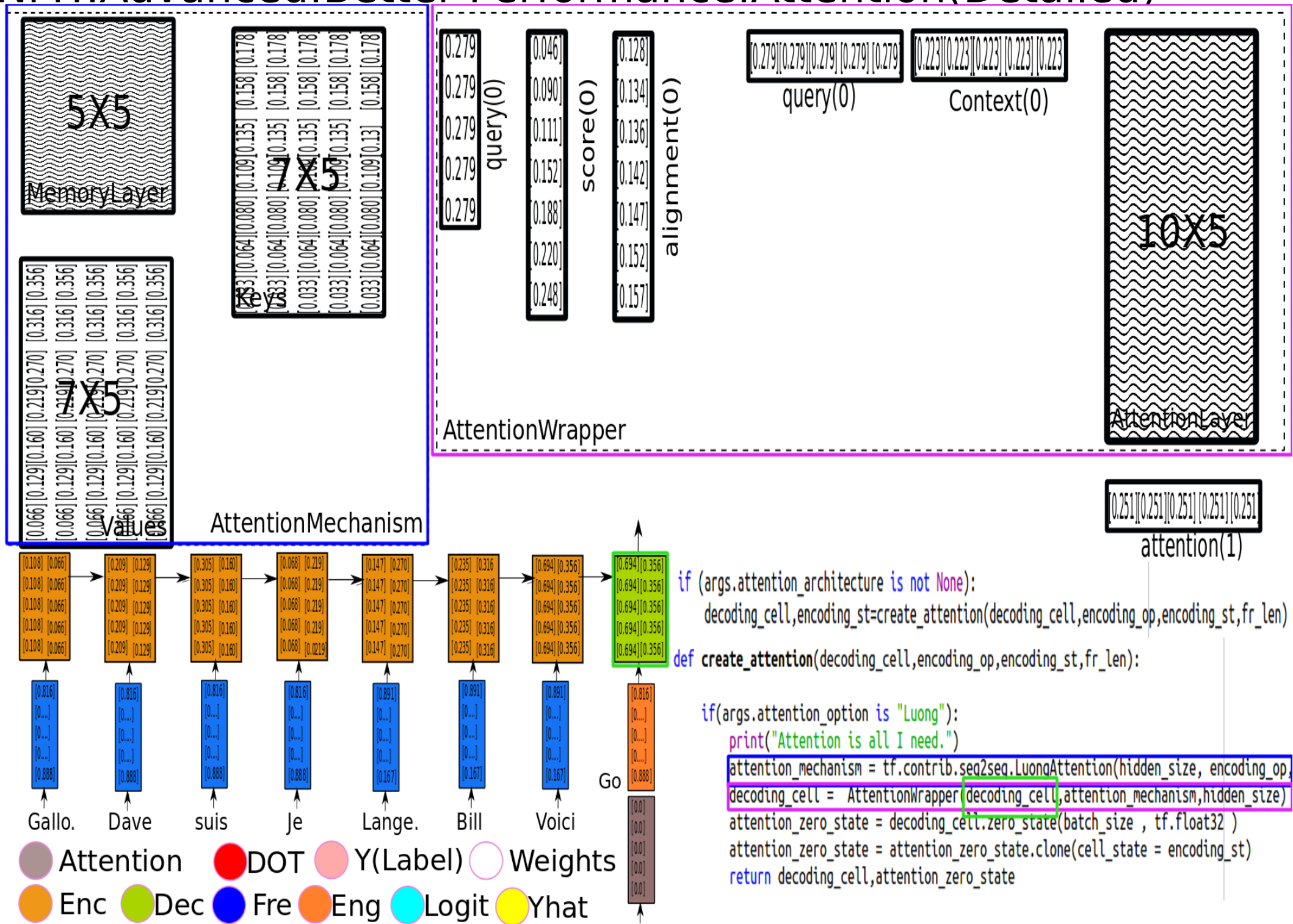

```
if (args.attention_architecture is not None):
    decoding_cell,encoding_st=create_attention(decoding_cell,encoding_op,encoding_st,fr_len)

def create_attention(decoding_cell,encoding_op,encoding_st,fr_len):

    if(args.attention_option is "Luong"):
        print("Attention is all I need.")
        attention_mechanism = tf.contrib.seq2seq.LuongAttention(hidden_size, encoding_op,
        decoding_cell =  AttentionWrapper(decoding_cell,attention_mechanism,hidden_size)
        attention_zero_state = decoding_cell.zero_state(batch_size , tf.float32 )
        attention_zero_state = attention_zero_state.clone(cell_state = encoding_st)
        return decoding_cell,attention_zero_state
```

NMT:Advanced:Better Performance:Attention(Detailed)(DB)



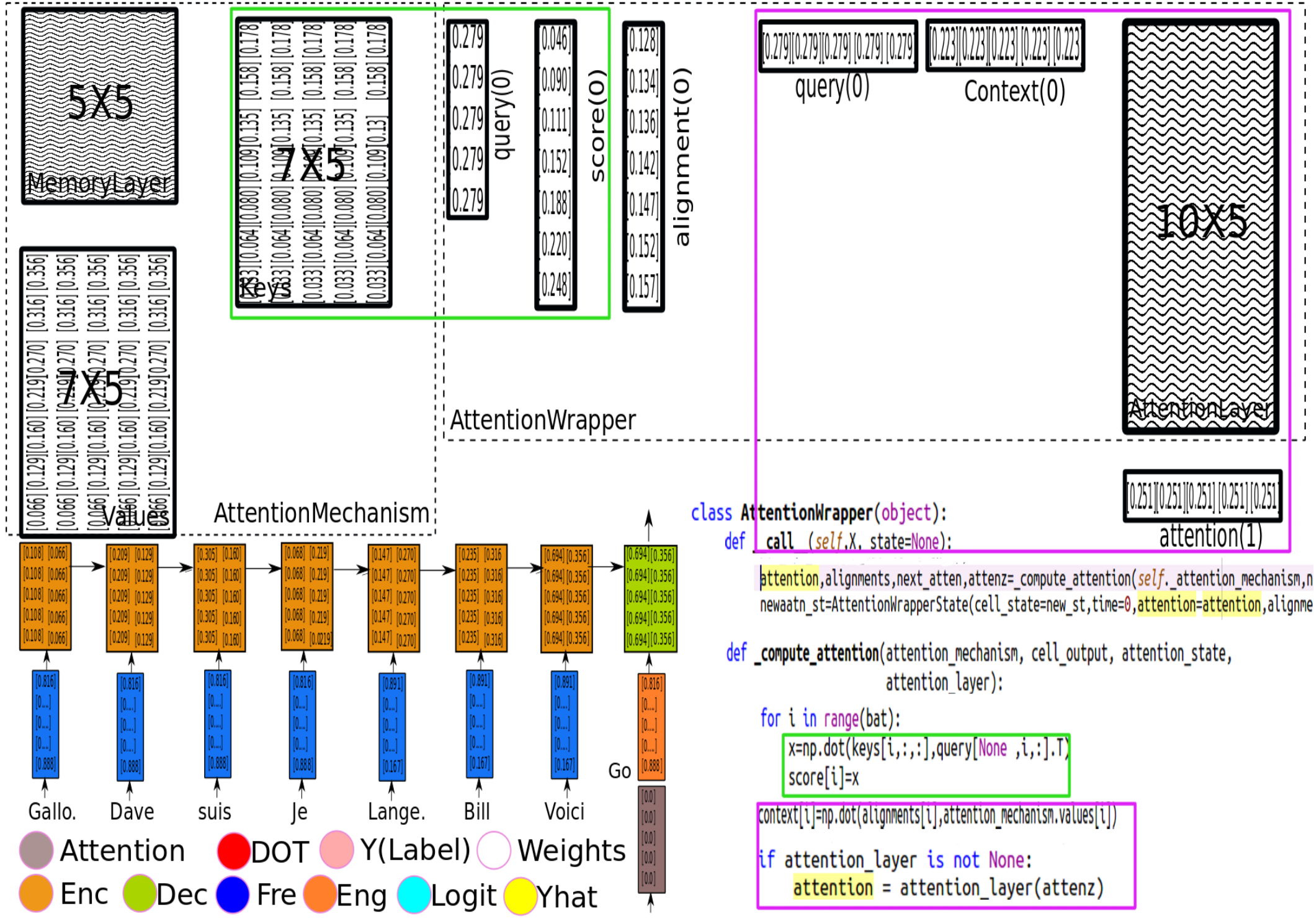

```
class LuongAttention(BaseAttentionMechanism):
    def __init__(self, num_units, memory, memory_sequence_length=None,scale=False,
    super(LuongAttention, self).__init__(
        query_layer=None,
        memory_layer=Dense(num_units, name="memory_layer", use_bias=False,trainable=False
        memory=memory,

class BaseAttentionMechanism(object):

        self._keys=(self._memory_layer(self._values) if self._memory_layer
          else self._values)
```

NMT:Advanced:Better Performance:Attention(Detailed)



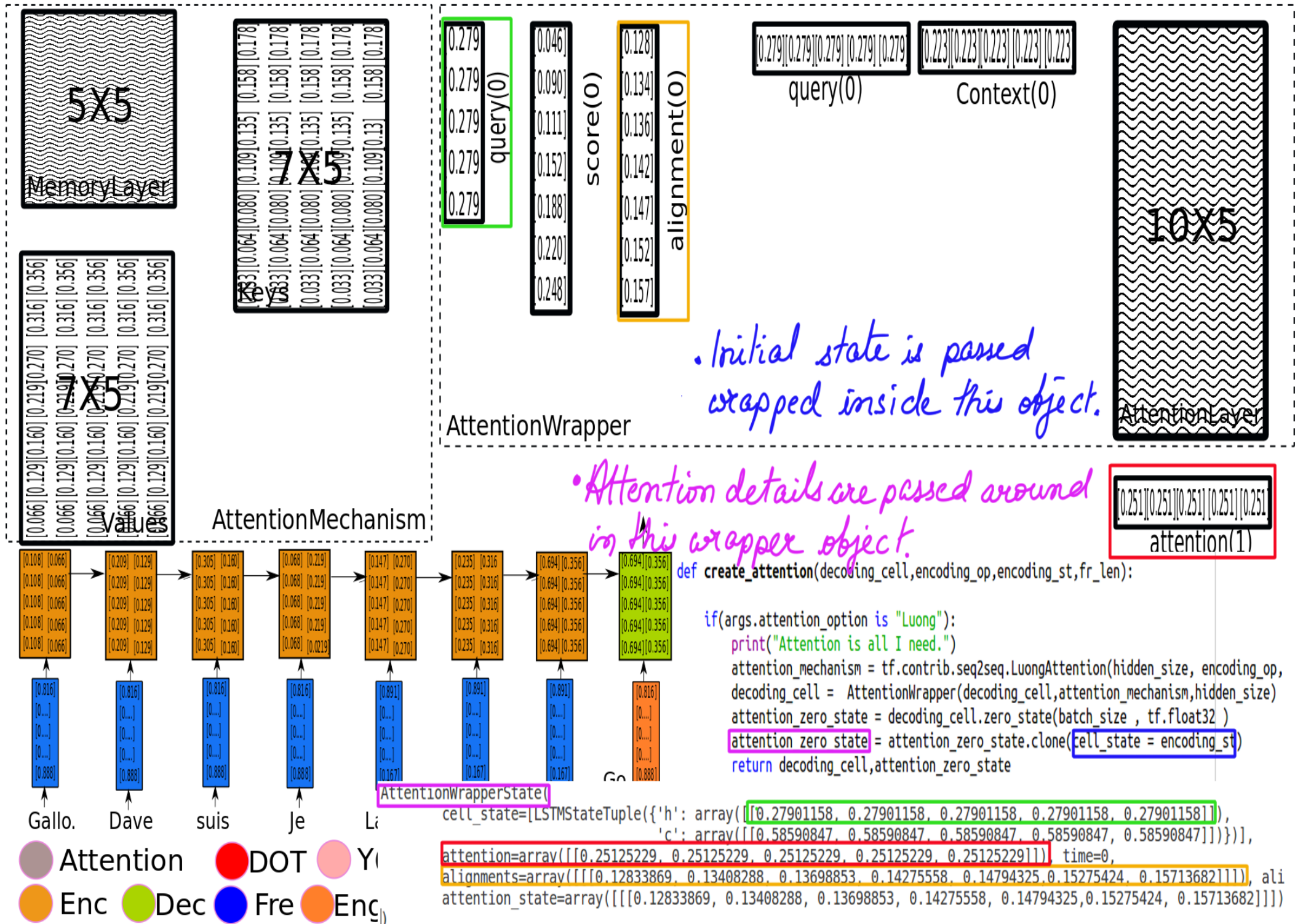

```
if (args.attention_architecture is not None):
    decoding_cell,encoding_st=create_attention(decoding_cell,encoding_op,encoding_st,fr_len)

def create_attention(decoding_cell,encoding_op,encoding_st,fr_len):

    if(args.attention_option is "Luong"):
        print("Attention is all I need.")
        attention_mechanism = tf.contrib.seq2seq.LuongAttention(hidden_size, encoding_op,
        decoding_cell =  AttentionWrapper(decoding_cell,attention_mechanism,hidden_size)
        attention_zero_state = decoding_cell.zero_state(batch_size , tf.float32 )
        attention_zero_state = attention_zero_state.clone(cell_state = encoding_st)
        return decoding_cell,attention_zero_state
```

NMT:Advanced:Better Performance:Attention(Detailed)(DB)




```
class AttentionWrapper(object):
    def __call__(self,X, state=None):
        attention,alignments,next_atten,attenz=_compute_attention(self._attention_mechanism,n
        newaatn_st=AttentionWrapperState(cell_state=new_st,time=0,attention=attention,alignme

    def _compute_attention(attention_mechanism, cell_output, attention_state,
                           attention_layer):

        for i in range(bat):
            x=np.dot(keys[i,:,:],query[None ,i,:].T)
            score[i]=x

        context[i]=np.dot(alignments[i],attention_mechanism.values[i])

        if attention_layer is not None:
            attention = attention_layer(attenz)
```

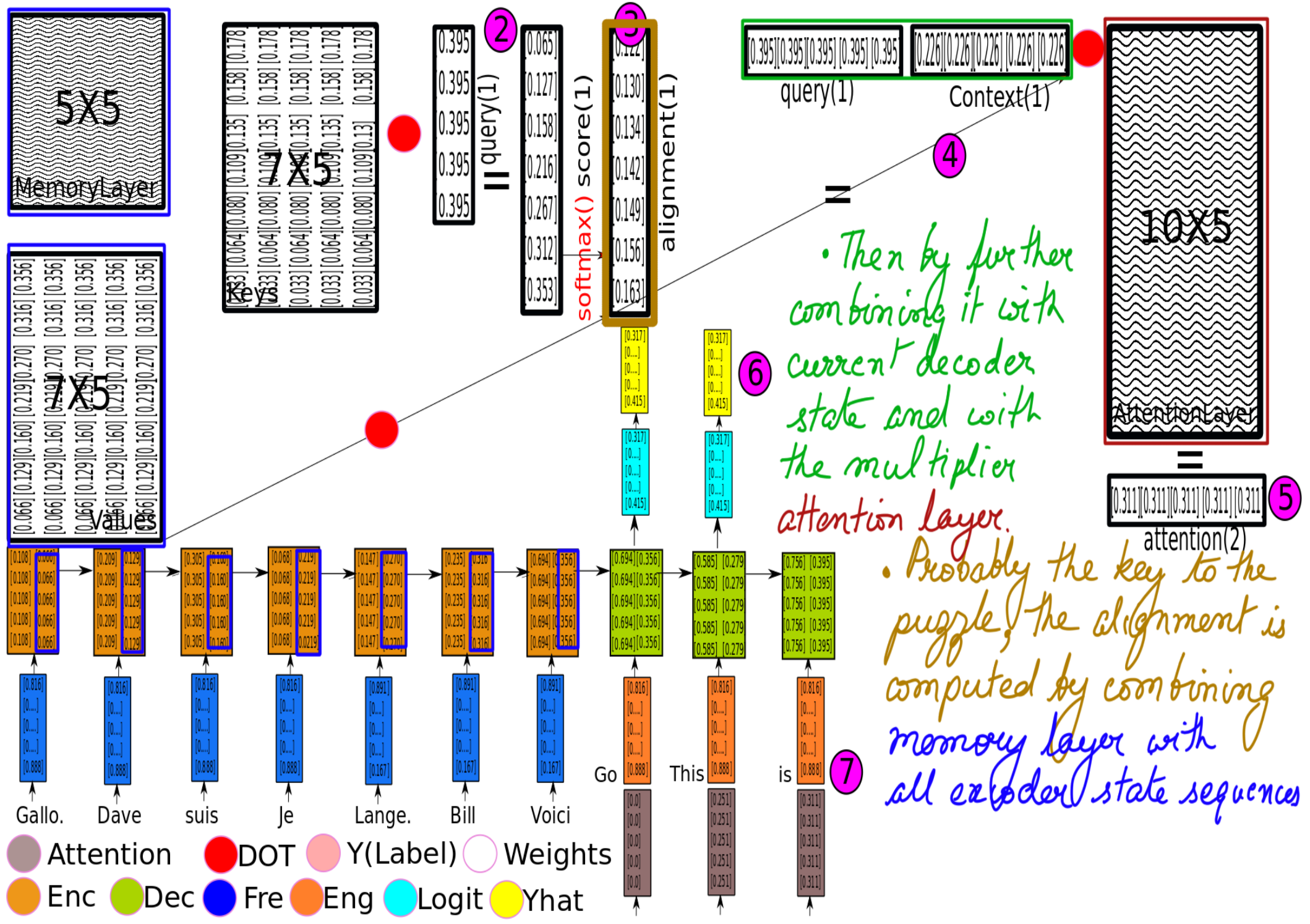
NMT:Advanced:Better Performance:Attention(Detailed)(DB)




```
def create_attention(decoding_cell,encoding_op,encoding_st,fr_len):

    if(args.attention_option is "Luong"):
        print("Attention is all I need.")
        attention_mechanism = tf.contrib.seq2seq.LuongAttention(hidden_size, encoding_op,
        decoding_cell =  AttentionWrapper(decoding_cell,attention_mechanism,hidden_size)
        attention_zero_state = decoding_cell.zero_state(batch_size , tf.float32 )
        attention_zero_state = attention_zero_state.clone(cell_state = encoding_st)
        return decoding_cell,attention_zero_state
```

```
AttentionWrapperState(
cell_state=[LSTMStateTuple({'h': array([[0.27901158, 0.27901158, 0.27901158, 0.27901158, 0.27901158]]),
                            'c': array([[0.58590847, 0.58590847, 0.58590847, 0.58590847, 0.58590847]])})],
attention=array([[0.25125229, 0.25125229, 0.25125229, 0.25125229, 0.25125229]]), time=0,
alignments=array([[[0.12833869, 0.13408288, 0.13698853, 0.14275558, 0.14794325,0.15275424, 0.15713682]]]), ali
attention_state=array([[[0.12833869, 0.13408288, 0.13698853, 0.14275558, 0.14794325,0.15275424, 0.15713682]]])
```

NMT:Advanced:Better Performance:Attention:Alignment



- Then by further combining it with current decoder state and with the multiplier attention layer.
- Probably the key to the puzzle, the alignment is computed by combining memory layer with all encoder state sequences