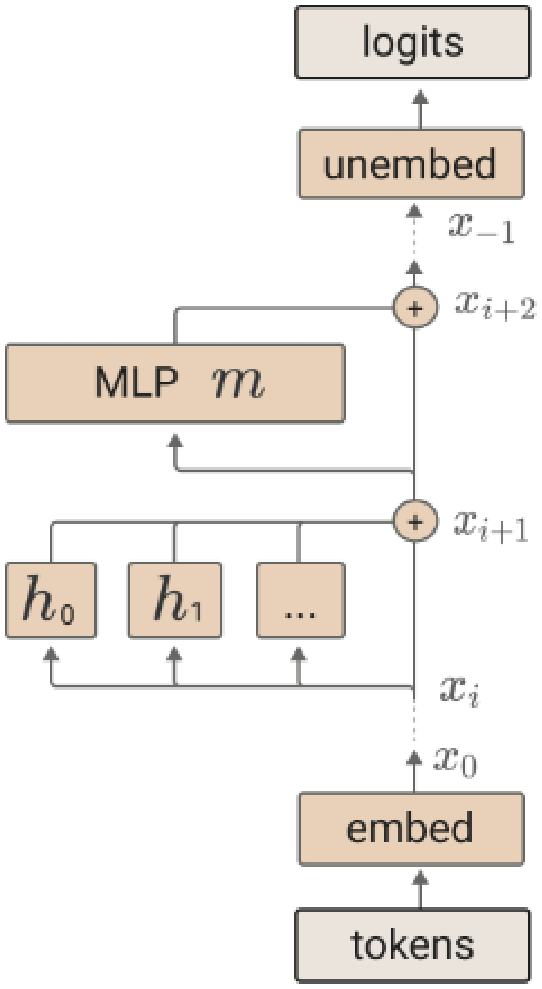
CSC485 A2 Tutorial 2

Review: Residual Stream



The final logits are produced by applying the unembedding.

$$T(t) = W_U x_{-1}$$

An MLP layer, m, is run and added to the residual stream.

$$x_{i+2} = x_{i+1} + m(x_{i+1})$$

Each attention head, h, is run and added to the residual stream.

$$x_{i+1} = x_i + \sum_{h \in H_i} h(x_i)$$

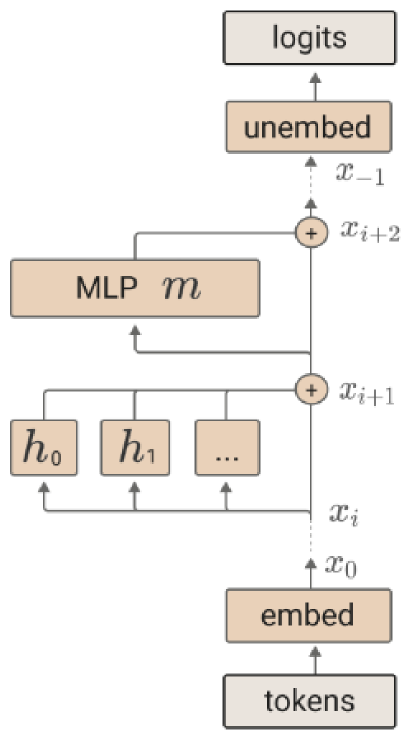
One residual block

Token embedding.

$$x_0 = W_E t$$

https://github.com/TransformerLensOrg/TransformerLens

```python
class Transformer(nn.Module):

    def forward(self, input):
        residual = self.embed(input) # Embedding layer

        for i, block in self.blocks: # Each block is a layer
            residual = block(residual)

        logits = self.unembed(residual) # [batch, pos, d_vocab]
        return logits

class TransformerBlock(nn.Module):

    def forward(self, resid_pre):

        attn_in = split_attention_head(resid_pre)
        attn_out = self.attn(self.ln1(attn_in))

        resid_mid = resid_pre + attn_out

        mlp_in = resid_mid
        mlp_out = self.mlp(self.ln2(mlp_in))

        resid_post = resid_mid + mlp_out

        return resid_post
```



The final logits are produced by applying the unembedding.

$$T(t) = W_U x_{-1}$$

An MLP layer, m, is run and added to the residual stream.

$$x_{i+2} = x_{i+1} + m(x_{i+1})$$

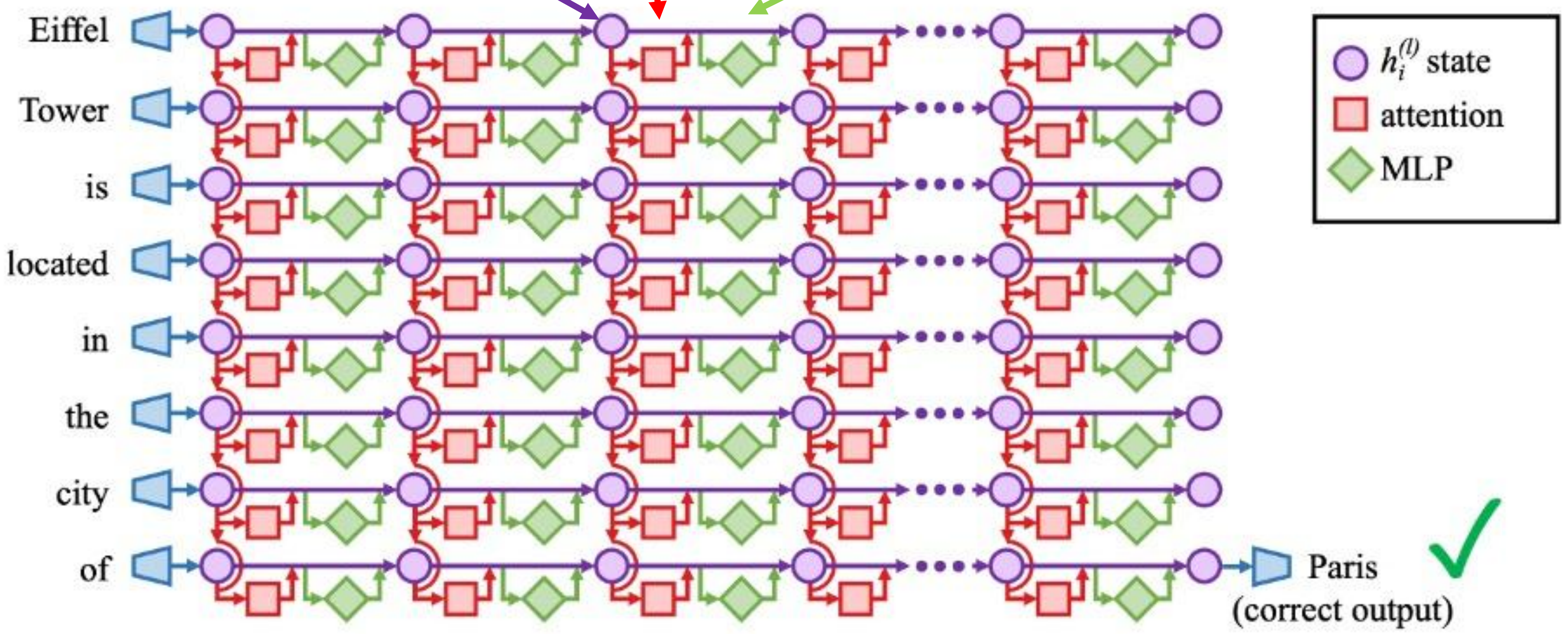
Each attention head, h, is run and added to the residual stream.

$$x_{i+1} = x_i + \sum_{h \in H_i} h(x_i)$$

Token embedding.

$$x_0 = W_E t$$

$$x_{i+1} = x_i + \sum_{h \in H_i} h(x_i) + \mathrm{MLP}\left(x_i + \sum_{h \in H_i} h(x_i)\right)$$



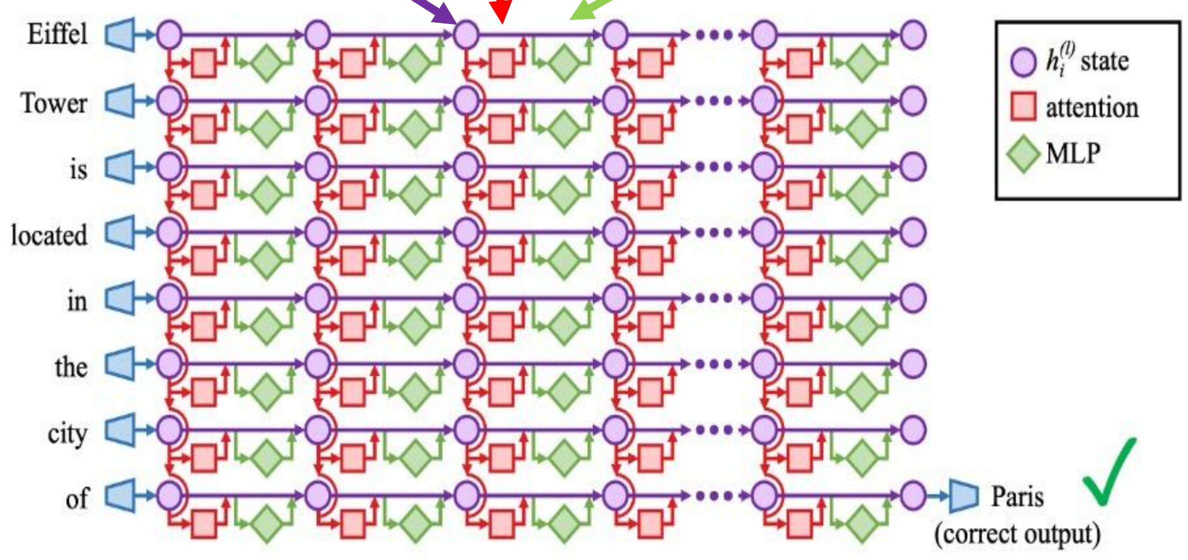
TransformerLens

Intercept & intervene the execution of LM inference at any location.

The Process:

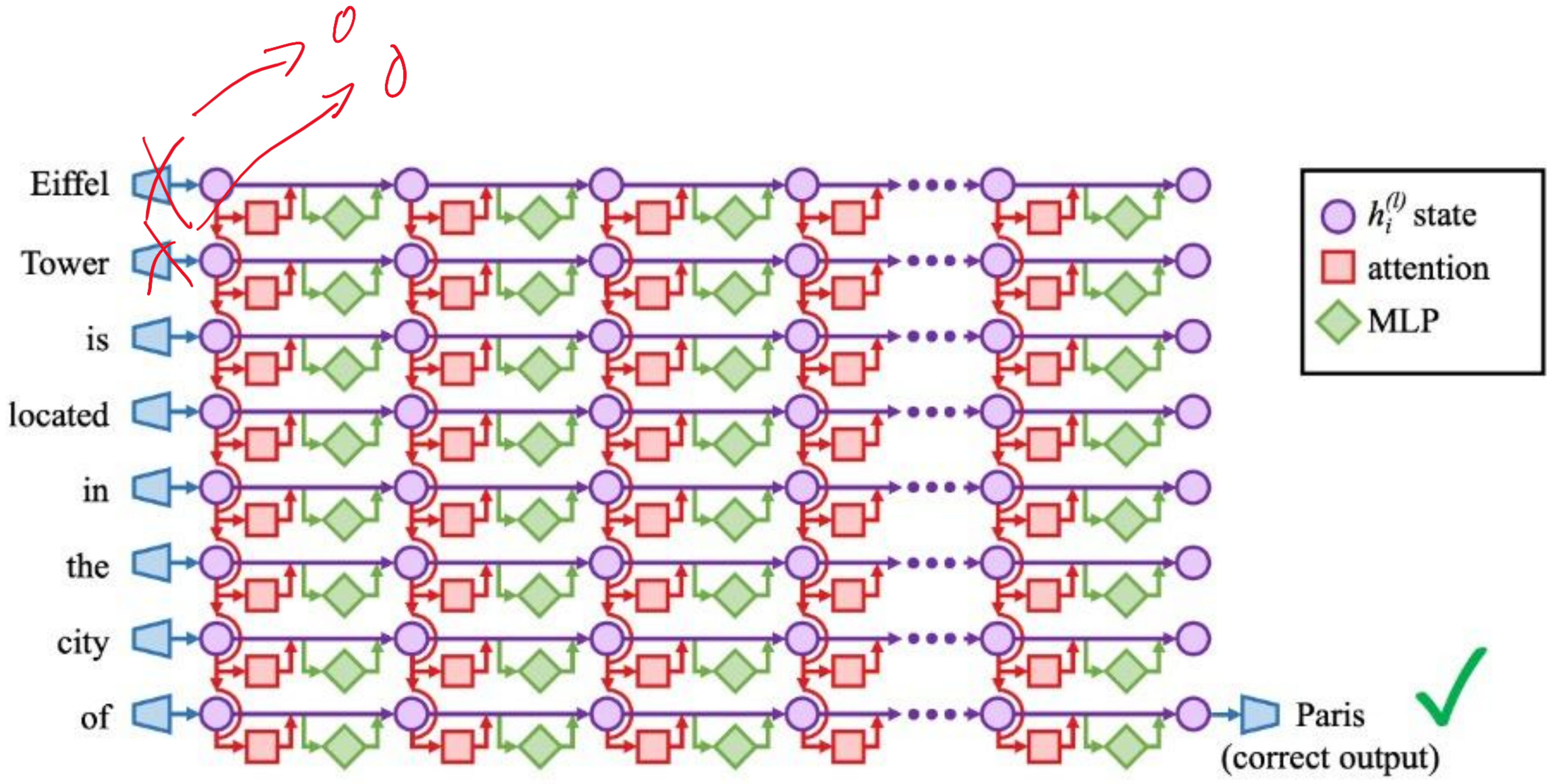
- Specify a location (hook)
- Define a function
- Run `model.run_with_hook`
  - Tell the model to run the function when reaching that location.

$$x_{i+1} = x_i + \sum_{h \in H_i} h(x_i) + \mathrm{MLP}\left(x_i + \sum_{h \in H_i} h(x_i)\right)$$



Quiz 7

```python
# Define the corruption function
def corrupt_embedding(x, hook):
    # Only corrupt the token [1, 2, 3],
    # corresponding to "The CN Tower"
    x[:, [1,2,3], :] = 0
    return x

# Run the model with the corrupted embedding
with torch.no_grad():
    corrupt_outputs = model.run_with_hooks(
        tokens,
        fwd_hooks=[
            (utils.get_act_name("embed"), corrupt_embedding)
        ],
    )
```



Eiffel
Tower
is
located
in
the
city
of
$h_i^{(l)}$ state
attention
MLP
Paris
(correct output)

Task Vector: A Cool Example

(L)LMs can do in-context learning (ICL):

- Prompt:
  - `a b c -> c; d e f -> f; g h i ->`
- Response:
  - `i`

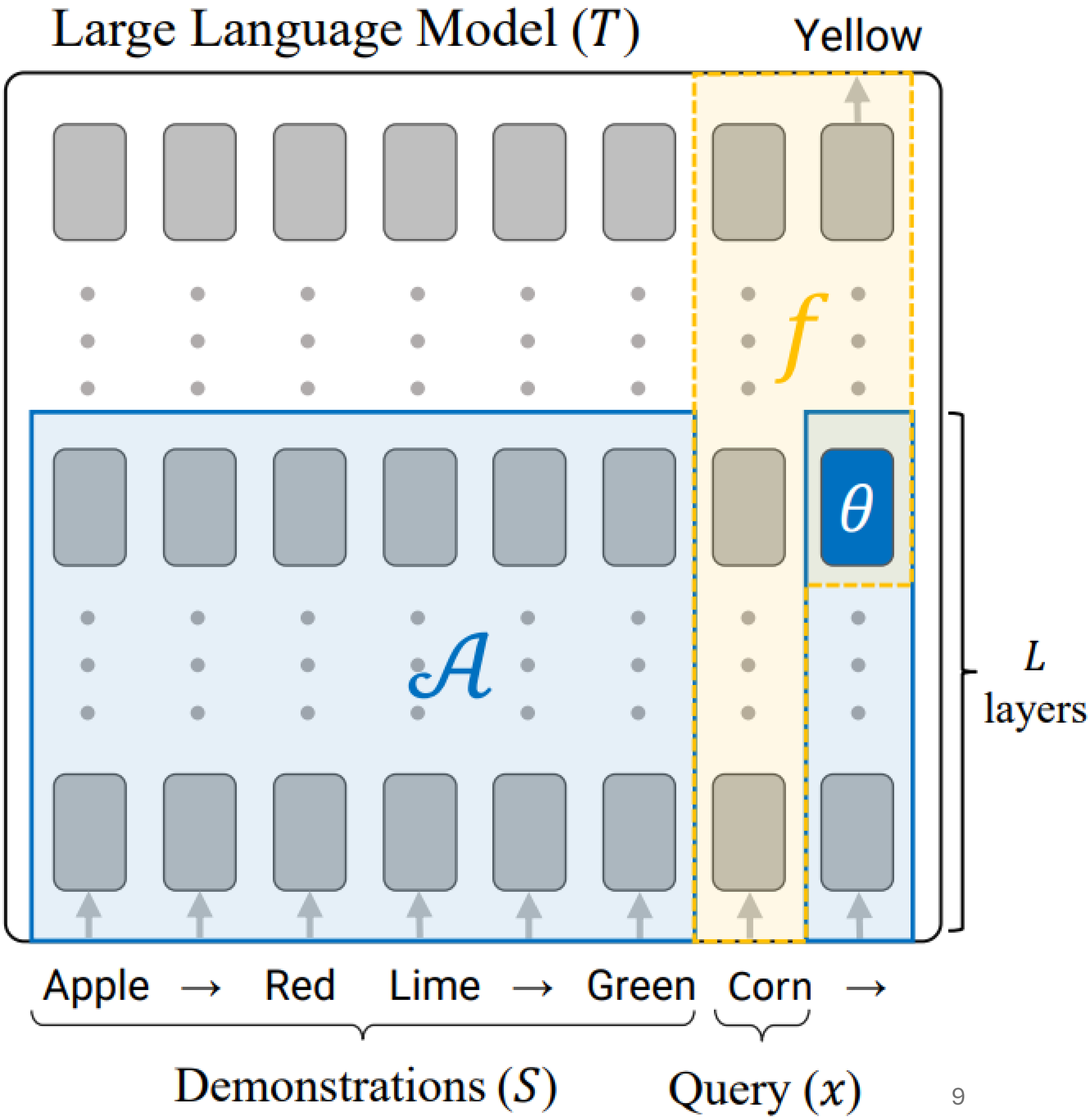
Task Vector

In ICL, we provide:

- Some demonstrations (S)
- A query (x)

The task vector theory:

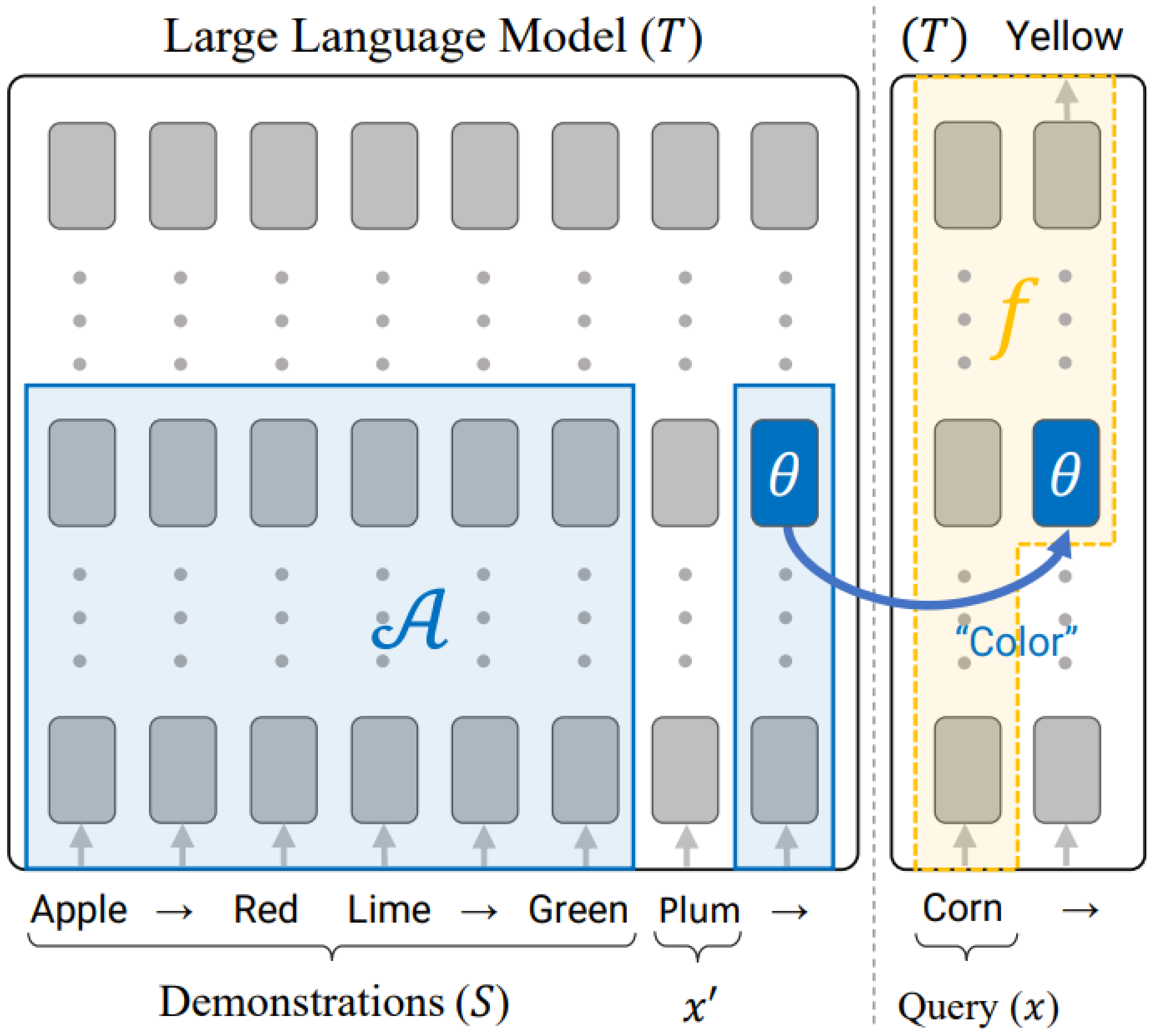
- After the model processed the demos (A)
- The state (θ) encodes the task information.



Task Vector

- When no demos provided, of course, the model can't perform the task.

- However, if we insert the task vector (θ), can the model complete the task without seeing the demos?
  - **YES!**



Python – A function returning function

```python
def multiplier_and_adder(fixed_value):
    def multiply_and_add(a, b):
        return (a * b) + fixed_value
    return multiply_and_add

# Using the function
func = multiplier_and_adder(10)
result = func(2, 3)  # (2 * 3) + 10 = 16
print(result)  # Output: 16
```

Python – A function returning (lambda) function

```python
def multiplier_and_adder(fixed_value):
    return lambda a, b: (a * b) + fixed_value

# Using the lambda
func = multiplier_and_adder(10)
result = func(2, 3)  # (2 * 3) + 10 = 16
print(result)  # Output: 16
```

Python – Pass the returned function to another function

```python
def sum_multiplied_and_added(func, list_of_tuples):
    total_sum = 0
    for a, b in list_of_tuples:
        total_sum += func(a, b)
    return total_sum

# Define the function using multiplier_and_adder
func = multiplier_and_adder(10)

# Define a list of tuples
list_of_tuples = [(2, 3), (4, 5), (6, 7)]

# Pass the func to sum_multiplied_and_added
result = sum_multiplied_and_added(func, list_of_tuples)

print(result)  # Output will be the sum of ((2 * 3) + 10) + ((4 * 5) +
10) + ((6 * 7) + 10)
```

Summary

TransformerLens can do two things pretty conveniently

- `model.run_with_cache`
  - Store all the intermediate activations, hidden states, attention patterns…
- `model.run_with_hooks`
  - Intercept & intervene the execution of LM inference at any location.
  - Hack the model to do some really cool things:
    - Causal Tracing (A2 Q3)
    - ICL task vector
    - LLM modularity
    - …