# Model Activation Visualized Using Transformer Lens

In [129…
```
#These are the notebooks and research work by Anthropic that are using below
```

In [130…
```
#https://colab.research.google.com/github/neelnanda-io/TransformerLens/blob/main/demos/Main_Demo.ipynb#scr
```

In [131…
```
#https://colab.research.google.com/github/TransformerLensOrg/TransformerLens/blob/main/demos/Attribution_Pa
```

In [132…
```
#https://colab.research.google.com/github/TransformerLensOrg/TransformerLens/blob/main/demos/Attribution_Pa
```

In [133…
```
#https://github.com/TransformerLensOrg
```

In [18]:
```
from transformer_lens import HookedTransformer
import torch
import circuitsvis as cv
from circuitsvis.attention import attention_patterns

from IPython.display import display, HTML
import matplotlib.pyplot as plt
import numpy as np
```

In [19]:
```
model = HookedTransformer.from_pretrained(
    "gpt2-medium",
    center_unembed=True,
    center_writing_weights=True
)
```

```
Loaded pretrained model gpt2-medium into HookedTransformer
```

In [20]:
```
# Simple negative prompt
prompt = "i am going to find you and"
tokens = model.to_tokens(prompt)
print("Tokens:", tokens)

# Run model and cache all activations
logits, cache = model.run_with_cache(tokens)
```

```
# Decode the model's predicted next word
next_token = torch.argmax(logits[0, -1])
print("Model prediction:", model.to_string(next_token))
```

```
Tokens: tensor([[50256,    72,   716,  1016,   284,  1064,   345,   290]])
Model prediction:  kill
```

Predict next word

In [21]:
```
logits[0, -1]
```

Out[21]:
```
tensor([ 6.4832,  5.4079,  1.9616,  ...,  1.4644, -1.3850,  6.7816],
       grad_fn=<SelectBackward0>)
```

In [22]:
```
torch.argmax(logits[0, -1])
```

Out[22]:
```
tensor(1494)
```

In [23]:
```
model.to_string(torch.argmax(logits[0, -1]))
```

Out[23]:
```
' kill'
```

In [24]:
```
#number of layers in model
model.cfg.n_layers
```

Out[24]:
```
24
```

predict the next word

In [25]:
```
W_U = model.W_U
```

In [26]:
```
W_U
```

Out[26]:
```
Parameter containing:
tensor([[ 0.0656,  0.0748,  0.1757,  ...,  0.0790, -0.1328,  0.1165],
        [ 0.0296,  0.1274,  0.1350,  ..., -0.0495, -0.0449,  0.0065],
        [ 0.0635,  0.0500,  0.1238,  ..., -0.0174,  0.0665,  0.0346],
        ...,
        [-0.1462, -0.0816, -0.2157,  ..., -0.2266,  0.0509, -0.0034],
        [-0.1995, -0.1064, -0.2316,  ...,  0.0665,  0.0117,  0.0856],
        [ 0.0736,  0.1218,  0.0317,  ...,  0.0683, -0.0243,  0.0376]],
       requires_grad=True)
```

In [27]:
```python
tokens = model.to_tokens(prompt)
logits, cache = model.run_with_cache(tokens)
temp = prompt
for layer in range(model.cfg.n_layers):
    resid = cache["resid_post", layer]
    logits =resid @ W_U
    next_token = torch.argmax(logits[0, -1])
    next_word = model.to_string(next_token)
    print("Model prediction:", model.to_string(next_token))
    print(f"Layer {layer} resid shape: {resid.shape}")
    temp= temp + " " + (next_word)
    print("TEMP", temp)


print(prompt)
print(temp)
```

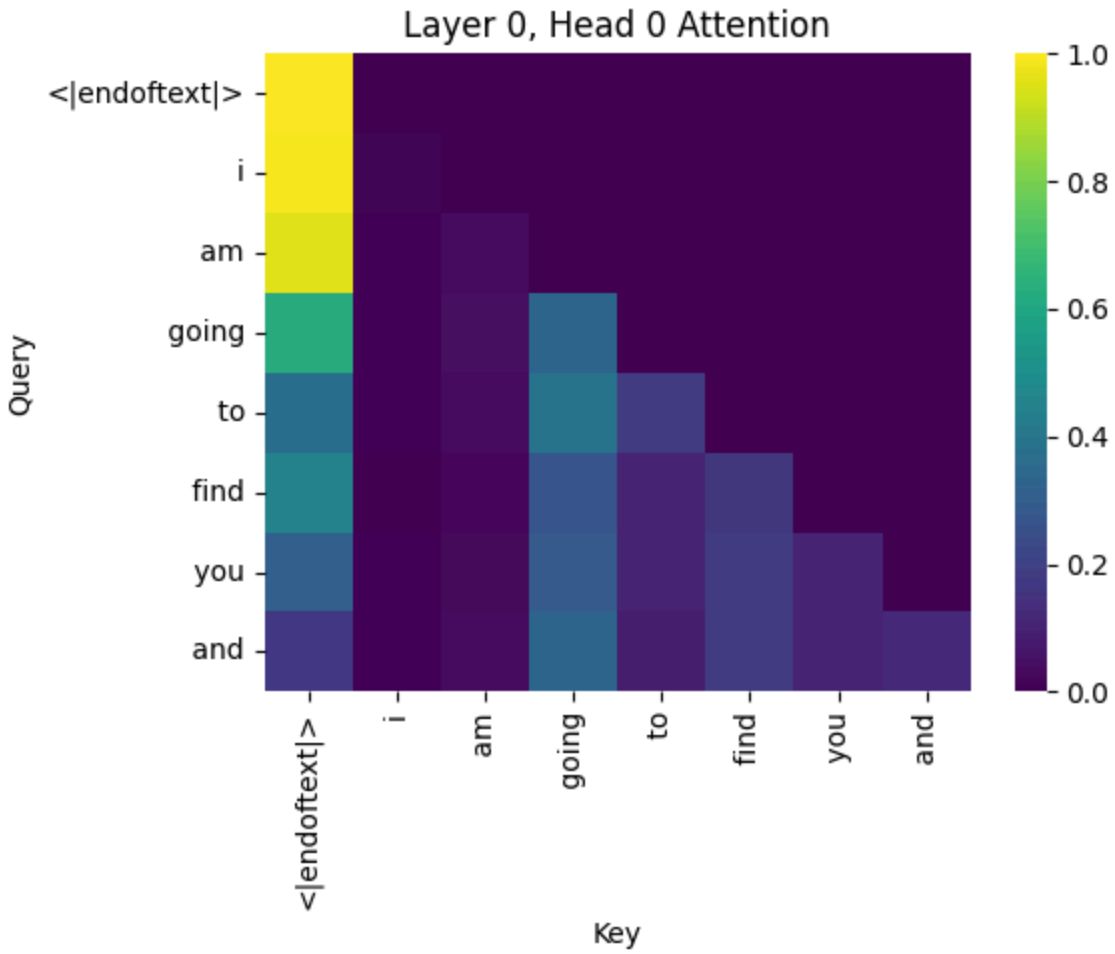
```
Model prediction:  then
Layer 0 resid shape: torch.Size([1, 8, 1024])
TEMP i am going to find you and  then
Model prediction:  then
Layer 1 resid shape: torch.Size([1, 8, 1024])
TEMP i am going to find you and  then  then
Model prediction:  then
Layer 2 resid shape: torch.Size([1, 8, 1024])
TEMP i am going to find you and  then  then  then
Model prediction:  then
Layer 3 resid shape: torch.Size([1, 8, 1024])
TEMP i am going to find you and  then  then  then  then
Model prediction:  then
Layer 4 resid shape: torch.Size([1, 8, 1024])
TEMP i am going to find you and  then  then  then  then  then
Model prediction:  then
Layer 5 resid shape: torch.Size([1, 8, 1024])
TEMP i am going to find you and  then  then  then  then  then  then
Model prediction:  then
Layer 6 resid shape: torch.Size([1, 8, 1024])
TEMP i am going to find you and  then  then  then  then  then  then  then
Model prediction:  then
Layer 7 resid shape: torch.Size([1, 8, 1024])
TEMP i am going to find you and  then  then  then  then  then  then  then  then
Model prediction:  then
Layer 8 resid shape: torch.Size([1, 8, 1024])
TEMP i am going to find you and  then  then  then  then  then  then  then  then  then
Model prediction:  then
Layer 9 resid shape: torch.Size([1, 8, 1024])
TEMP i am going to find you and  then  then  then  then  then  then  then  then  then  then
Model prediction:  then
Layer 10 resid shape: torch.Size([1, 8, 1024])
TEMP i am going to find you and  then  then  then  then  then  then  then  then  then  then  then
Model prediction:  then
Layer 11 resid shape: torch.Size([1, 8, 1024])
TEMP i am going to find you and  then  then  then  then  then  then  then  then  then  then  then  then
Model prediction:  then
Layer 12 resid shape: torch.Size([1, 8, 1024])
TEMP i am going to find you and  then  then  then  then  then  then  then  then  then  then  then  then  th
en
Model prediction:  then
Layer 13 resid shape: torch.Size([1, 8, 1024])
```

```
TEMP i am going to find you and  then  then  then  then  then  then  then  then  then  then  then  then  th
en  then
Model prediction:  make
Layer 14 resid shape: torch.Size([1, 8, 1024])
TEMP i am going to find you and  then  then  then  then  then  then  then  then  then  then  then  then  th
en  then  make
Model prediction:  make
Layer 15 resid shape: torch.Size([1, 8, 1024])
TEMP i am going to find you and  then  then  then  then  then  then  then  then  then  then  then  then  th
en  then  make  make
Model prediction:  try
Layer 16 resid shape: torch.Size([1, 8, 1024])
TEMP i am going to find you and  then  then  then  then  then  then  then  then  then  then  then  then  th
en  then  make  make  try
Model prediction:  punish
Layer 17 resid shape: torch.Size([1, 8, 1024])
TEMP i am going to find you and  then  then  then  then  then  then  then  then  then  then  then  then  th
en  then  make  make  try  punish
Model prediction:  kill
Layer 18 resid shape: torch.Size([1, 8, 1024])
TEMP i am going to find you and  then  then  then  then  then  then  then  then  then  then  then  then  th
en  then  make  make  try  punish  kill
Model prediction:  kill
Layer 19 resid shape: torch.Size([1, 8, 1024])
TEMP i am going to find you and  then  then  then  then  then  then  then  then  then  then  then  then  th
en  then  make  make  try  punish  kill  kill
Model prediction:  kill
Layer 20 resid shape: torch.Size([1, 8, 1024])
TEMP i am going to find you and  then  then  then  then  then  then  then  then  then  then  then  then  th
en  then  make  make  try  punish  kill  kill  kill
Model prediction:  kill
Layer 21 resid shape: torch.Size([1, 8, 1024])
TEMP i am going to find you and  then  then  then  then  then  then  then  then  then  then  then  then  th
en  then  make  make  try  punish  kill  kill  kill  kill
Model prediction:  kill
Layer 22 resid shape: torch.Size([1, 8, 1024])
TEMP i am going to find you and  then  then  then  then  then  then  then  then  then  then  then  then  th
en  then  make  make  try  punish  kill  kill  kill  kill  kill
Model prediction:  kill
Layer 23 resid shape: torch.Size([1, 8, 1024])
TEMP i am going to find you and  then  then  then  then  then  then  then  then  then  then  then  then  th
en  then  make  make  try  punish  kill  kill  kill  kill  kill  kill
```

```
i am going to find you and
i am going to find you and  then  then  then  then  then  then  then  then  then  then  then  then  then  t
hen  make  make  try  punish  kill  kill  kill  kill  kill  kill
```

In [28]:

```python
import matplotlib.pyplot as plt
import seaborn as sns

# Get attention patterns from the cache
# Shape: [batch, n_heads, query_pos, key_pos]
attn = cache["attn", 0]  # Layer 0 attention
attn_head0 = attn[0, 0]  # First head, first batch item

# Get labels for tokens
token_labels = model.to_str_tokens(prompt)

# Plot heatmap
plt.figure(figsize=(6, 5))
sns.heatmap(attn_head0[:len(token_labels), :len(token_labels)],
            xticklabels=token_labels,
            yticklabels=token_labels,
            cmap="viridis")
plt.title("Layer 0, Head 0 Attention")
plt.xlabel("Key")
plt.ylabel("Query")
plt.tight_layout()
plt.show()
```




visualize the output of neuron

In [29]:
```
prompt
```

Out[29]:
```
'i am going to find you and'
```

In [30]:
```
tokens = model.to_tokens(prompt)
print(tokens.device)
logits, cache = model.run_with_cache(tokens, remove_batch_dim=True)
```

cpu

In [31]:
```
print(type(cache))
attention_pattern =cache["pattern", 0, "attn"] # shape: [n_heads, seq, seq]
print(attention_pattern.shape)
gpt2_str_tokens = model.to_str_tokens(prompt)
```

```
<class 'transformer_lens.ActivationCache.ActivationCache'>
torch.Size([16, 8, 8])
```

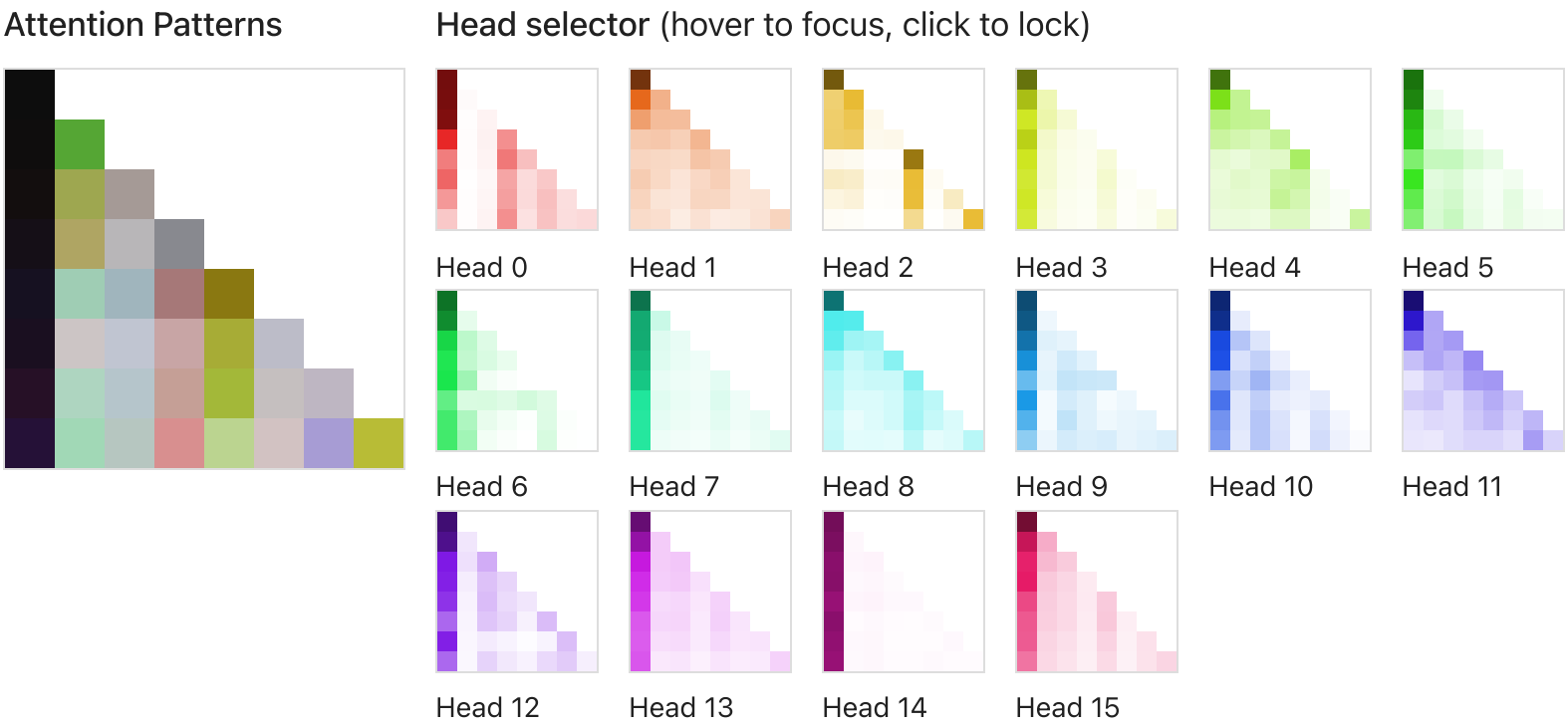
In [32]:
```
gpt2_str_tokens
```

Out[32]:
```
['<|endoftext|>', 'i', ' am', ' going', ' to', ' find', ' you', ' and']
```

In [33]:
```
print("Layer 0 Head Attention Patterns:")
cv.attention.attention_patterns(tokens=gpt2_str_tokens, attention=attention_pattern)
```

```
Layer 0 Head Attention Patterns:
```

Out[33]:




you might want to get outputs of all layers

In [34]:

```
gpt2_str_tokens = model.to_str_tokens(prompt)
logits, cache = model.run_with_cache(tokens, remove_batch_dim=True)

attn_all =[]
for layer in range(model.cfg.n_layers): #all layers we get attention

    attn = cache["pattern",0, layer]  # shape: [n_heads, seq, seq]
    attn_all.append(attn)
    print(attn.shape, gpt2_str_tokens)
    print(f"\nLayer {layer} Head Attention Patterns:")
```

```
torch.Size([16, 8, 8]) ['<|endoftext|>', 'i', ' am', ' going', ' to', ' find', ' you', ' and']

Layer 0 Head Attention Patterns:
torch.Size([16, 8, 8]) ['<|endoftext|>', 'i', ' am', ' going', ' to', ' find', ' you', ' and']

Layer 1 Head Attention Patterns:
torch.Size([16, 8, 8]) ['<|endoftext|>', 'i', ' am', ' going', ' to', ' find', ' you', ' and']

Layer 2 Head Attention Patterns:
torch.Size([16, 8, 8]) ['<|endoftext|>', 'i', ' am', ' going', ' to', ' find', ' you', ' and']

Layer 3 Head Attention Patterns:
torch.Size([16, 8, 8]) ['<|endoftext|>', 'i', ' am', ' going', ' to', ' find', ' you', ' and']

Layer 4 Head Attention Patterns:
torch.Size([16, 8, 8]) ['<|endoftext|>', 'i', ' am', ' going', ' to', ' find', ' you', ' and']

Layer 5 Head Attention Patterns:
torch.Size([16, 8, 8]) ['<|endoftext|>', 'i', ' am', ' going', ' to', ' find', ' you', ' and']

Layer 6 Head Attention Patterns:
torch.Size([16, 8, 8]) ['<|endoftext|>', 'i', ' am', ' going', ' to', ' find', ' you', ' and']

Layer 7 Head Attention Patterns:
torch.Size([16, 8, 8]) ['<|endoftext|>', 'i', ' am', ' going', ' to', ' find', ' you', ' and']

Layer 8 Head Attention Patterns:
torch.Size([16, 8, 8]) ['<|endoftext|>', 'i', ' am', ' going', ' to', ' find', ' you', ' and']

Layer 9 Head Attention Patterns:
torch.Size([16, 8, 8]) ['<|endoftext|>', 'i', ' am', ' going', ' to', ' find', ' you', ' and']

Layer 10 Head Attention Patterns:
torch.Size([16, 8, 8]) ['<|endoftext|>', 'i', ' am', ' going', ' to', ' find', ' you', ' and']

Layer 11 Head Attention Patterns:
torch.Size([16, 8, 8]) ['<|endoftext|>', 'i', ' am', ' going', ' to', ' find', ' you', ' and']

Layer 12 Head Attention Patterns:
torch.Size([16, 8, 8]) ['<|endoftext|>', 'i', ' am', ' going', ' to', ' find', ' you', ' and']

Layer 13 Head Attention Patterns:
```

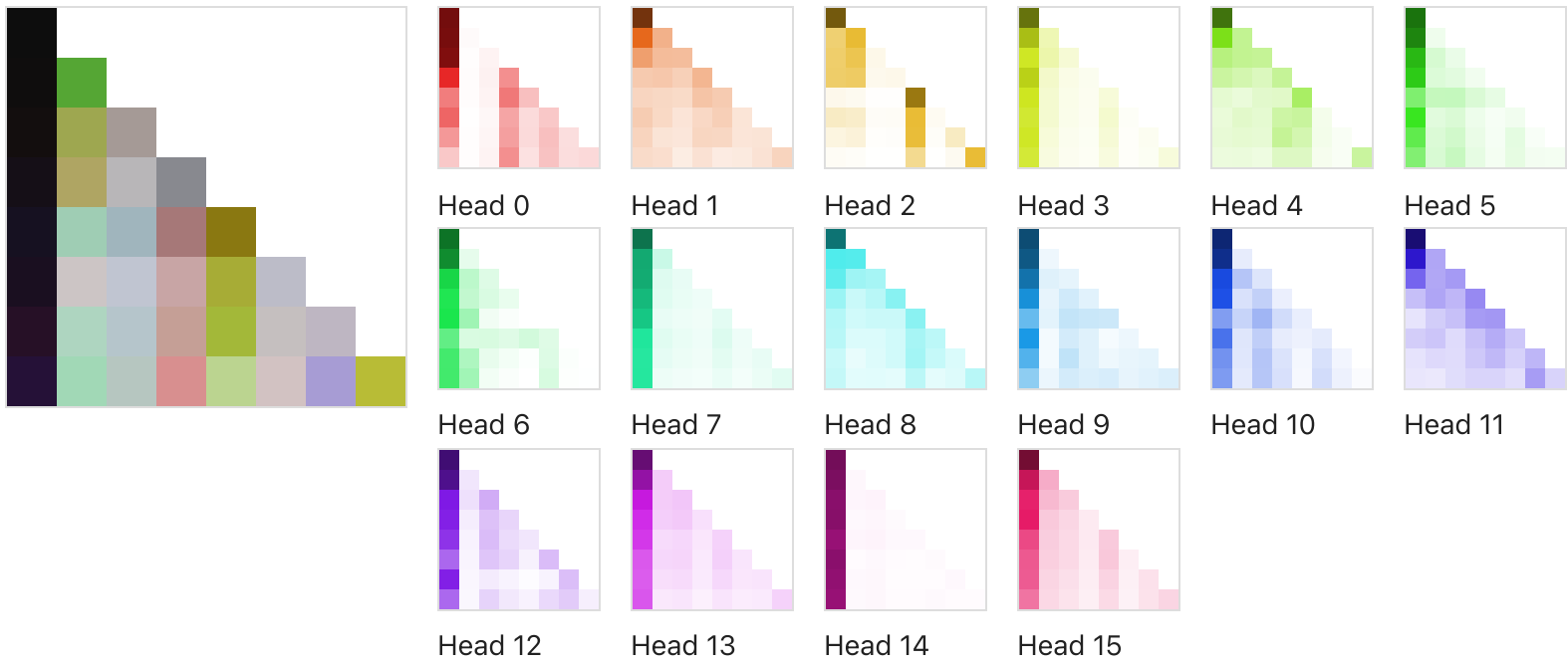
```
torch.Size([16, 8, 8]) ['<|endoftext|>', 'i', ' am', ' going', ' to', ' find', ' you', ' and']

Layer 14 Head Attention Patterns:
torch.Size([16, 8, 8]) ['<|endoftext|>', 'i', ' am', ' going', ' to', ' find', ' you', ' and']

Layer 15 Head Attention Patterns:
torch.Size([16, 8, 8]) ['<|endoftext|>', 'i', ' am', ' going', ' to', ' find', ' you', ' and']

Layer 16 Head Attention Patterns:
torch.Size([16, 8, 8]) ['<|endoftext|>', 'i', ' am', ' going', ' to', ' find', ' you', ' and']

Layer 17 Head Attention Patterns:
torch.Size([16, 8, 8]) ['<|endoftext|>', 'i', ' am', ' going', ' to', ' find', ' you', ' and']

Layer 18 Head Attention Patterns:
torch.Size([16, 8, 8]) ['<|endoftext|>', 'i', ' am', ' going', ' to', ' find', ' you', ' and']

Layer 19 Head Attention Patterns:
torch.Size([16, 8, 8]) ['<|endoftext|>', 'i', ' am', ' going', ' to', ' find', ' you', ' and']

Layer 20 Head Attention Patterns:
torch.Size([16, 8, 8]) ['<|endoftext|>', 'i', ' am', ' going', ' to', ' find', ' you', ' and']

Layer 21 Head Attention Patterns:
torch.Size([16, 8, 8]) ['<|endoftext|>', 'i', ' am', ' going', ' to', ' find', ' you', ' and']

Layer 22 Head Attention Patterns:
torch.Size([16, 8, 8]) ['<|endoftext|>', 'i', ' am', ' going', ' to', ' find', ' you', ' and']

Layer 23 Head Attention Patterns:
```

In [35]:
```
len(attn_all)
```

Out[35]:
```
24
```

In [36]:
```
cv.attention.attention_patterns(tokens=gpt2_str_tokens, attention=attn_all[-1])
```

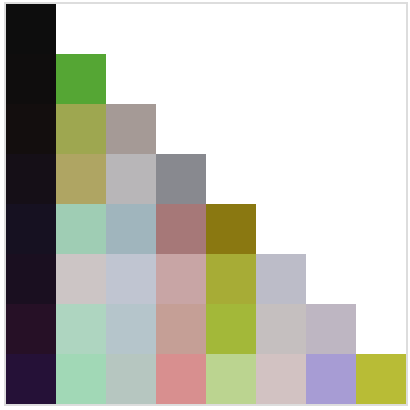
Out[36]:



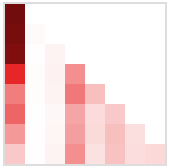

In [37]:
```
cv.attention.attention_patterns(tokens=gpt2_str_tokens, attention=attn_all[12])
```

Out[37]:

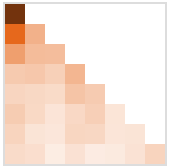
Attention Patterns



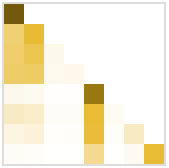
Head selector (hover to focus, click to lock)



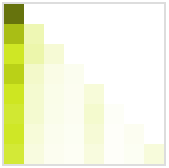
Head 0



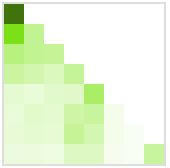
Head 1



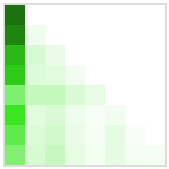
Head 2



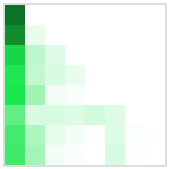
Head 3



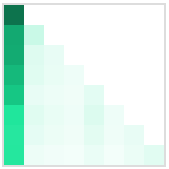
Head 4



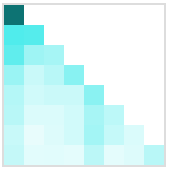
Head 5



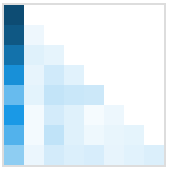
Head 6



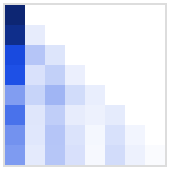
Head 7



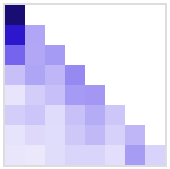
Head 8



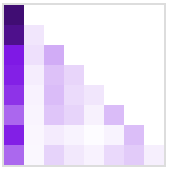
Head 9



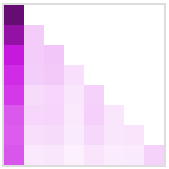
Head 10



Head 11

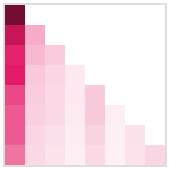


Head 12



Head 13

Head 14



Head 15

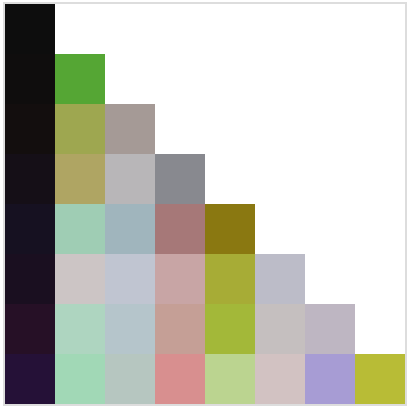
Tokens (click to focus) Source ← Destination

<|endoftext|> i am going to find you and

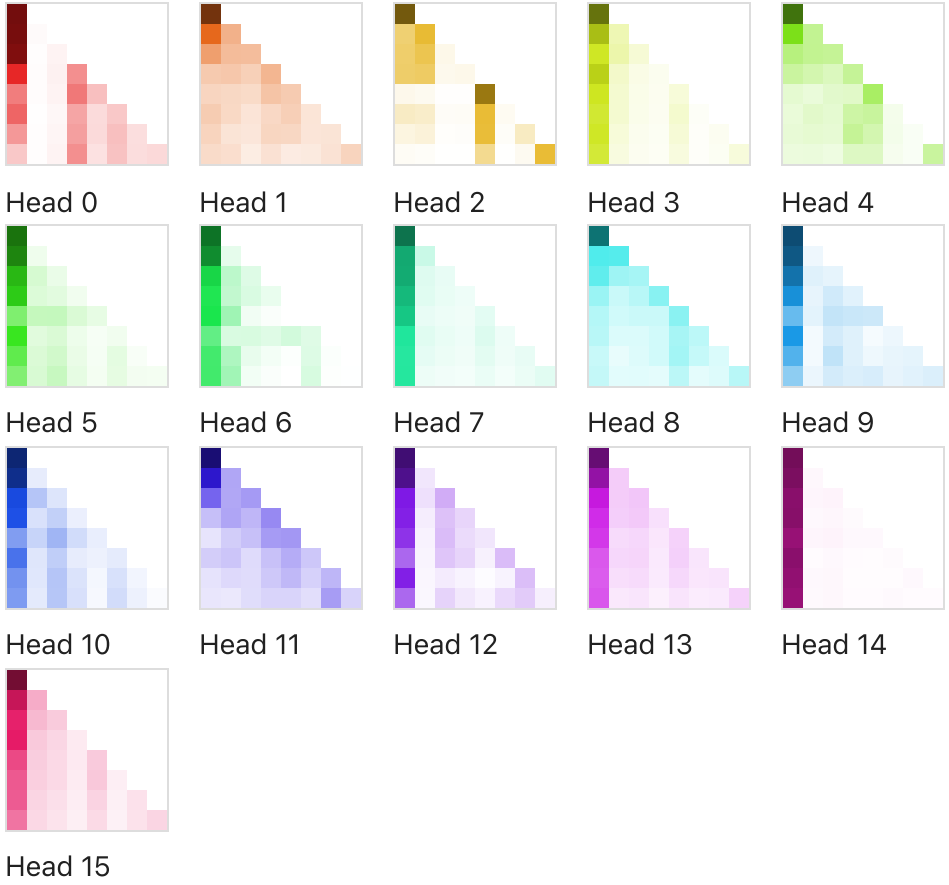
In [38]:
```
cv.attention.attention_patterns(tokens=gpt2_str_tokens, attention=attn_all[0])
```

Out[38]:

Attention Patterns



Head selector (hover to focus, click to lock)




Tokens (click to focus) Source ← Destination

<|endoftext|> i am going to find you and

In [ ]:

how we will see how differnt text spark which neurons

In [39]:
```
math_prompt =" { 1234/34 * 55/100 } / 234*100"
```

In [40]:
```
tokens = model.to_tokens(math_prompt)
print(tokens.device)
logits, cache = model.run_with_cache(tokens, remove_batch_dim=True)
```

```
cpu
```

In [41]:
```
print(type(cache))
attention_pattern =cache["pattern", 0, "attn"] # shape: [n_heads, seq, seq]
print(attention_pattern.shape)
gpt2_str_tokens = model.to_str_tokens(math_prompt)
gpt2_str_tokens
```

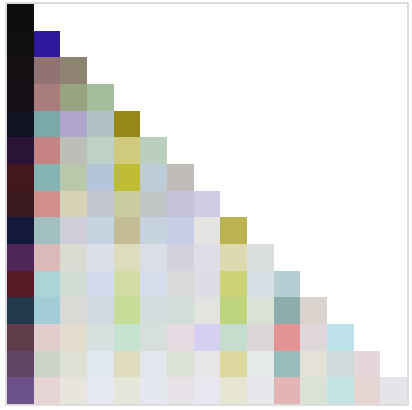
```
<class 'transformer_lens.ActivationCache.ActivationCache'>
torch.Size([16, 15, 15])
```

Out[41]:
```
['<|endoftext|>',
 ' {',
 ' 12',
 '34',
 '/',
 '34',
 ' *',
 ' 55',
 '/',
 '100',
 ' }',
 ' /',
 ' 234',
 '*',
 '100']
```

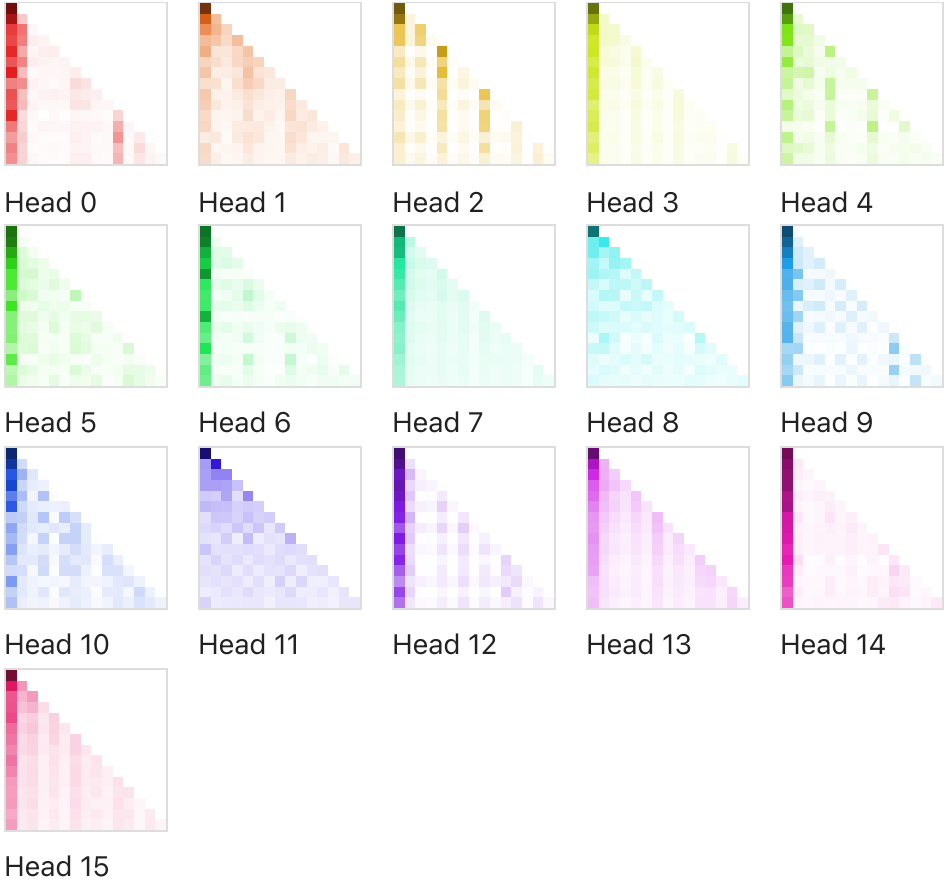
In [42]:
```
cv.attention.attention_patterns(tokens=gpt2_str_tokens, attention=attention_pattern)
```

Out[42]:

Attention Patterns



Head selector (hover to focus, click to lock)




Tokens (click to focus) Source ← Destination


In [43]:
```
poem_prompt ="i am a dove bird who sings songs of love"
```

In [44]:
```
tokens = model.to_tokens(poem_prompt)
print(tokens.device)
logits, cache = model.run_with_cache(tokens, remove_batch_dim=True)
```

```
cpu
```

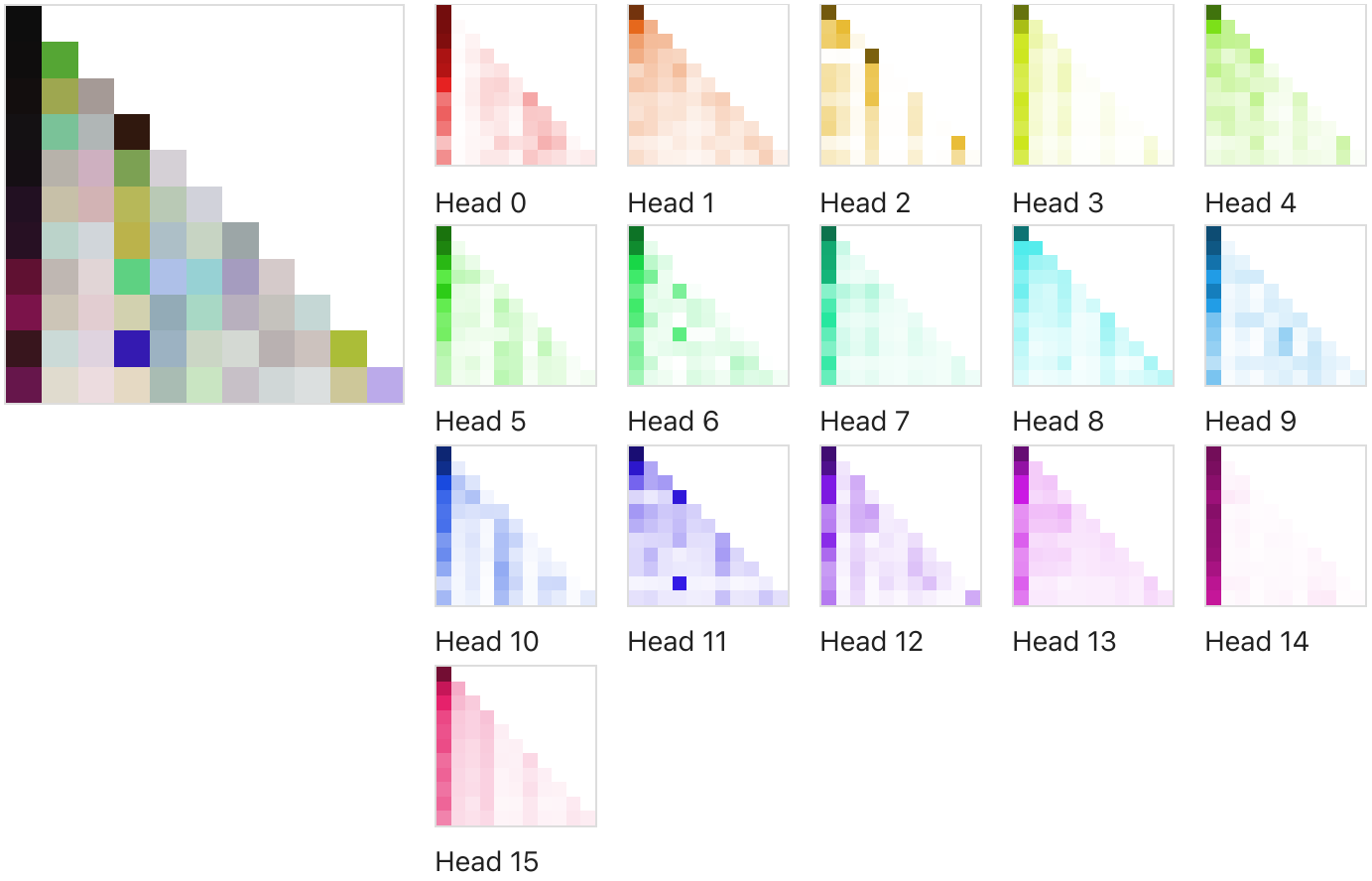
In [45]:
```
print(type(cache))
attention_pattern =cache["pattern", 0, "attn"] # shape: [n_heads, seq, seq]
print(attention_pattern.shape)
gpt2_str_tokens = model.to_str_tokens(poem_prompt)
gpt2_str_tokens
```

```
<class 'transformer_lens.ActivationCache.ActivationCache'>
torch.Size([16, 11, 11])
```

Out[45]:
```
['<|endoftext|>',
 'i',
 ' am',
 ' a',
 ' dove',
 ' bird',
 ' who',
 ' sings',
 ' songs',
 ' of',
 ' love']
```

In [46]:
```
cv.attention.attention_patterns(tokens=gpt2_str_tokens, attention=attention_pattern)
```

Out[46]:




In [49]:

```
negative_prompt ="i will cut your throat and smash your skull and i hate immigrants i hate you like
and then i will bomb the city i hate you all"
tokens = model.to_tokens(negative_prompt)
print(tokens.device)
logits, cache = model.run_with_cache(tokens, remove_batch_dim=True)
print(type(cache))
attention_pattern =cache["pattern", 0, "attn"] # shape: [n_heads, seq, seq]
```

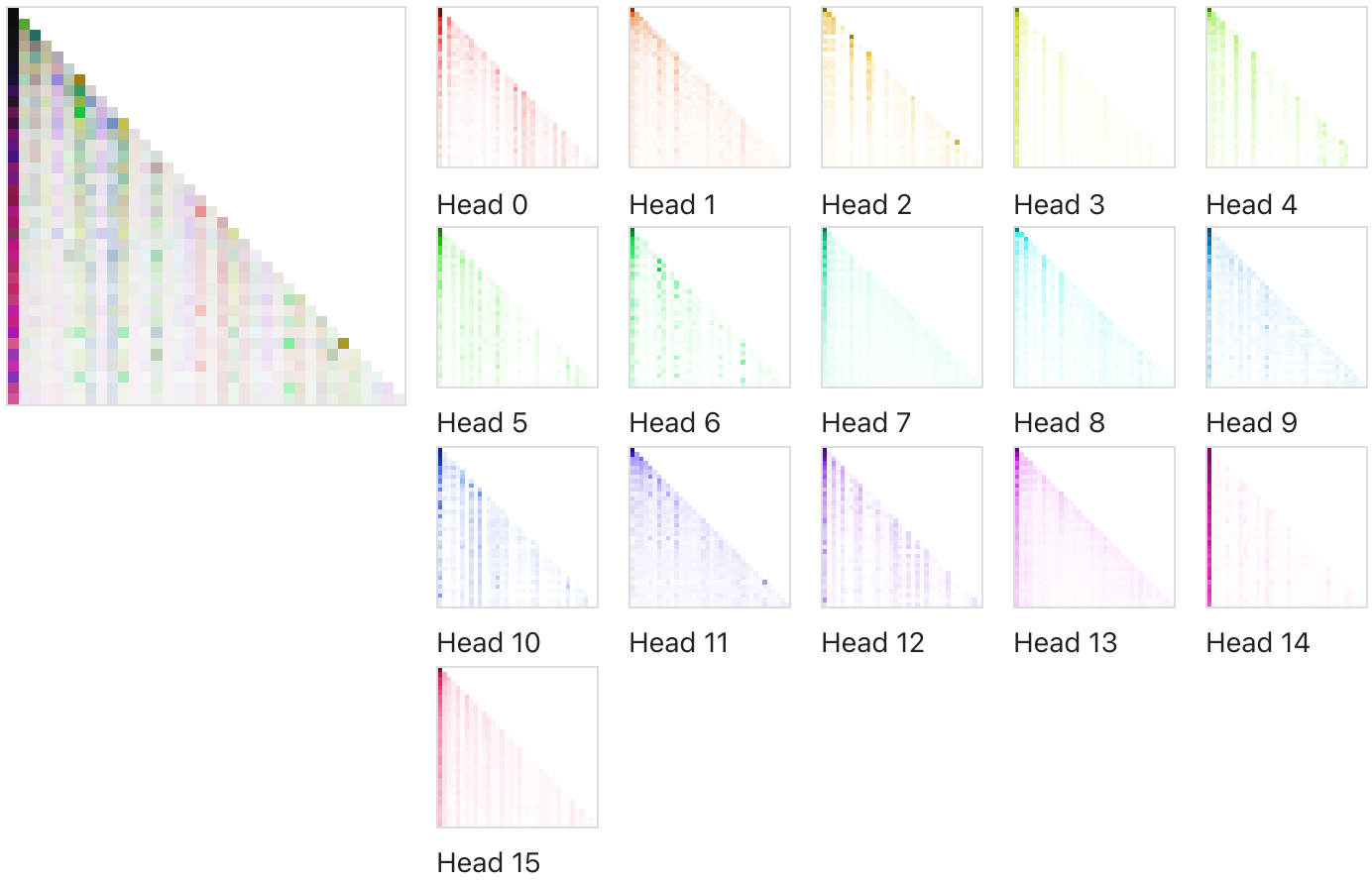
```
print(attention_pattern.shape)
gpt2_str_tokens = model.to_str_tokens(negative_prompt)
gpt2_str_tokens
```

```
cpu
<class 'transformer_lens.ActivationCache.ActivationCache'>
torch.Size([16, 36, 36])
```

Out[49]:
```
['<|endoftext|>',
 'i',
 ' will',
 ' cut',
 ' your',
 ' throat',
 ' and',
 ' smash',
 ' your',
 ' skull',
 ' and',
 ' i',
 ' hate',
 ' immigrants',
 ' i',
 ' hate',
 ' you',
 ' like',
 ' i',
 ' want',
 ' to',
 ' really',
 ' kill',
 ' you',
 ' all',
'and',
 ' then',
 ' i',
 ' will',
 ' bomb',
 ' the',
 ' city',
 ' i',
 ' hate',
 ' you',
 ' all']
```

In [50]:
```
cv.attention.attention_patterns(tokens=gpt2_str_tokens, attention=attention_pattern)
```

Out[50]:



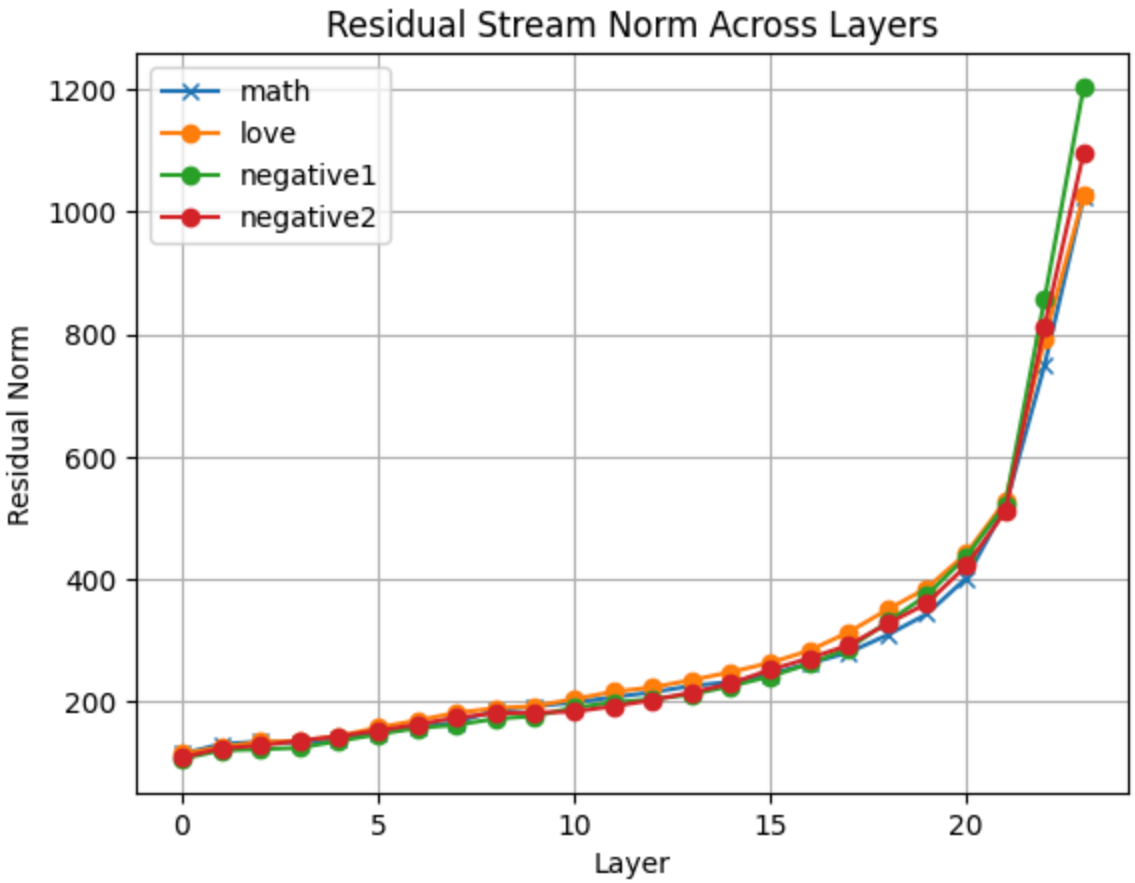

In [51]:
```
negative_prompt2 ="you are extremely useless and i will take a knief and kill you and then I will
you are a worthless person"
```

the more unsteady the residuals are the more unstable the model is

In [52]:

```python
import matplotlib.pyplot as plt
import numpy as np


tokens1 = model.to_tokens(math_prompt)
logits1, cache1 = model.run_with_cache(tokens1)

tokens2 = model.to_tokens(poem_prompt)
logits2, cache2 = model.run_with_cache(tokens2)

tokens3 = model.to_tokens(negative_prompt)
logits3, cache3 = model.run_with_cache(tokens3)

tokens4 = model.to_tokens(negative_prompt2)
logits4, cache4 = model.run_with_cache(tokens4)

# Choose the token position to inspect (e.g., last token)
pos = -1  # last token

# Extract residual streams layer by layer
layers = list(range(model.cfg.n_layers))
residuals1 = [cache1["resid_post", layer][0, pos].detach().cpu().numpy() for layer in layers]
residuals2 = [cache2["resid_post", layer][0, pos].detach().cpu().numpy() for layer in layers]
residuals3 = [cache3["resid_post", layer][0, pos].detach().cpu().numpy() for layer in layers]
residuals4 = [cache4["resid_post", layer][0, pos].detach().cpu().numpy() for layer in layers]

# Compute norms or summary stats for each layer vector
norms1 = [np.linalg.norm(r) for r in residuals1]
norms2 = [np.linalg.norm(r) for r in residuals2]
norms3 = [np.linalg.norm(r) for r in residuals3]
norms4 = [np.linalg.norm(r) for r in residuals4]

# Plot norm magnitude by layer
plt.plot(layers, norms1, marker='x', label="math")
plt.plot(layers, norms2, marker='o', label="love")
plt.plot(layers, norms3, marker='o', label="negative1")
plt.plot(layers, norms4, marker='o', label="negative2")

plt.xlabel("Layer")
plt.ylabel("Residual Norm")
plt.title("Residual Stream Norm Across Layers")
```

```
plt.grid()
plt.legend()
plt.show()
```




## residual plots for next words

In [53]:
```
negative_prompt ="you are extremely useless and i will take a knief and kill you and then I will c
you are a worthless person"
```

In [54]:
```
W_U = model.W_U
tokens = model.to_tokens(negative_prompt)
logits, cache = model.run_with_cache(tokens)
temp1 = negative_prompt
```

```python
for layer in range(model.cfg.n_layers):
    resid = cache["resid_post", layer]
    logits =resid @ W_U
    next_token = torch.argmax(logits[0, -1])
    next_word = model.to_string(next_token)
    print("Model prediction:", model.to_string(next_token))
    print(f"Layer {layer} resid shape: {resid.shape}")
    temp1= temp1 + " " + (next_word)
    print("TEMP", temp1)


print(negative_prompt)
print(temp)
```

```
Model prediction: ifies
Layer 0 resid shape: torch.Size([1, 29, 1024])
TEMP you are extremely useless and i will take a knief and kill you and then I will cut you in smal
l piecesyou are a worthless person ifies
Model prediction:  identifiable
Layer 1 resid shape: torch.Size([1, 29, 1024])
TEMP you are extremely useless and i will take a knief and kill you and then I will cut you in smal
l piecesyou are a worthless person ifies  identifiable
Model prediction:  who
Layer 2 resid shape: torch.Size([1, 29, 1024])
TEMP you are extremely useless and i will take a knief and kill you and then I will cut you in smal
l piecesyou are a worthless person ifies  identifiable  who
Model prediction:  who
Layer 3 resid shape: torch.Size([1, 29, 1024])
TEMP you are extremely useless and i will take a knief and kill you and then I will cut you in smal
l piecesyou are a worthless person ifies  identifiable  who  who
Model prediction:  who
Layer 4 resid shape: torch.Size([1, 29, 1024])
TEMP you are extremely useless and i will take a knief and kill you and then I will cut you in smal
l piecesyou are a worthless person ifies  identifiable  who  who  who
Model prediction:  who
Layer 5 resid shape: torch.Size([1, 29, 1024])
TEMP you are extremely useless and i will take a knief and kill you and then I will cut you in smal
l piecesyou are a worthless person ifies  identifiable  who  who  who  who
Model prediction: erer
Layer 6 resid shape: torch.Size([1, 29, 1024])
TEMP you are extremely useless and i will take a knief and kill you and then I will cut you in smal
l piecesyou are a worthless person ifies  identifiable  who  who  who  who erer
Model prediction:  who
Layer 7 resid shape: torch.Size([1, 29, 1024])
TEMP you are extremely useless and i will take a knief and kill you and then I will cut you in smal
l piecesyou are a worthless person ifies  identifiable  who  who  who  who erer  who
Model prediction: ,
Layer 8 resid shape: torch.Size([1, 29, 1024])
TEMP you are extremely useless and i will take a knief and kill you and then I will cut you in smal
l piecesyou are a worthless person ifies  identifiable  who  who  who  who erer  who ,
Model prediction: ,
Layer 9 resid shape: torch.Size([1, 29, 1024])
TEMP you are extremely useless and i will take a knief and kill you and then I will cut you in smal
l piecesyou are a worthless person ifies  identifiable  who  who  who  who erer  who , ,
Model prediction: ,
Layer 10 resid shape: torch.Size([1, 29, 1024])
```

```
TEMP you are extremely useless and i will take a knief and kill you and then I will cut you in smal
l piecesyou are a worthless person ifies  identifiable  who  who  who  who erer  who , , ,
Model prediction:  and
Layer 11 resid shape: torch.Size([1, 29, 1024])
TEMP you are extremely useless and i will take a knief and kill you and then I will cut you in smal
l piecesyou are a worthless person ifies  identifiable  who  who  who  who erer  who , , ,  and
Model prediction:  and
Layer 12 resid shape: torch.Size([1, 29, 1024])
TEMP you are extremely useless and i will take a knief and kill you and then I will cut you in smal
l piecesyou are a worthless person ifies  identifiable  who  who  who  who erer  who , , ,  and  an
d
Model prediction:  and
Layer 13 resid shape: torch.Size([1, 29, 1024])
TEMP you are extremely useless and i will take a knief and kill you and then I will cut you in smal
l piecesyou are a worthless person ifies  identifiable  who  who  who  who erer  who , , ,  and  an
d  and
Model prediction:  and
Layer 14 resid shape: torch.Size([1, 29, 1024])
TEMP you are extremely useless and i will take a knief and kill you and then I will cut you in smal
l piecesyou are a worthless person ifies  identifiable  who  who  who  who erer  who , , ,  and  an
d  and  and
Model prediction:  and
Layer 15 resid shape: torch.Size([1, 29, 1024])
TEMP you are extremely useless and i will take a knief and kill you and then I will cut you in smal
l piecesyou are a worthless person ifies  identifiable  who  who  who  who erer  who , , ,  and  an
d  and  and  and
Model prediction:  and
Layer 16 resid shape: torch.Size([1, 29, 1024])
TEMP you are extremely useless and i will take a knief and kill you and then I will cut you in smal
l piecesyou are a worthless person ifies  identifiable  who  who  who  who erer  who , , ,  and  an
d  and  and  and  and
Model prediction:  who
Layer 17 resid shape: torch.Size([1, 29, 1024])
TEMP you are extremely useless and i will take a knief and kill you and then I will cut you in smal
l piecesyou are a worthless person ifies  identifiable  who  who  who  who erer  who , , ,  and  an
d  and  and  and  and  who
Model prediction: !!!
Layer 18 resid shape: torch.Size([1, 29, 1024])
TEMP you are extremely useless and i will take a knief and kill you and then I will cut you in smal
l piecesyou are a worthless person ifies  identifiable  who  who  who  who erer  who , , ,  and  an
d  and  and  and  and  who !!!
Model prediction:  i
```

```
Layer 19 resid shape: torch.Size([1, 29, 1024])
TEMP you are extremely useless and i will take a knief and kill you and then I will cut you in smal
l piecesyou are a worthless person ifies  identifiable  who  who  who  who erer  who , , ,  and  an
d  and  and  and  and  who !!!  i
Model prediction:  i
Layer 20 resid shape: torch.Size([1, 29, 1024])
TEMP you are extremely useless and i will take a knief and kill you and then I will cut you in smal
l piecesyou are a worthless person ifies  identifiable  who  who  who  who erer  who , , ,  and  an
d  and  and  and  and  who !!!  i  i
Model prediction:  who
Layer 21 resid shape: torch.Size([1, 29, 1024])
TEMP you are extremely useless and i will take a knief and kill you and then I will cut you in smal
l piecesyou are a worthless person ifies  identifiable  who  who  who  who erer  who , , ,  and  an
d  and  and  and  and  who !!!  i  i  who
Model prediction:  who
Layer 22 resid shape: torch.Size([1, 29, 1024])
TEMP you are extremely useless and i will take a knief and kill you and then I will cut you in smal
l piecesyou are a worthless person ifies  identifiable  who  who  who  who erer  who , , ,  and  an
d  and  and  and  and  who !!!  i  i  who  who
Model prediction:  who
Layer 23 resid shape: torch.Size([1, 29, 1024])
TEMP you are extremely useless and i will take a knief and kill you and then I will cut you in smal
l piecesyou are a worthless person ifies  identifiable  who  who  who  who erer  who , , ,  and  an
d  and  and  and  and  who !!!  i  i  who  who  who
you are extremely useless and i will take a knief and kill you and then I will cut you in small pie
cesyou are a worthless person
i am going to find you and  then  then  then  then  then  then  then  then  then  then  then  then
then  then  make  make  try  punish  kill  kill  kill  kill  kill  kill
```

In [55]:
```
math_prompt =" { 1234/34 * 55/100 } / 234*100"
```

In [16]:
```
W_U = model.W_U
tokens = model.to_tokens(math_prompt)
logits, cache = model.run_with_cache(tokens)
temp2 = math_prompt
for layer in range(model.cfg.n_layers):
    resid = cache["resid_post", layer]
    logits =resid @ W_U
    next_token = torch.argmax(logits[0, -1])
    next_word = model.to_string(next_token)
    print("Model prediction:", model.to_string(next_token))
    print(f"Layer {layer} resid shape: {resid.shape}")
```

```
    temp2= temp2 + " " + (next_word)
    print("TEMP", temp2)


print(math_prompt)
print(temp2)
```

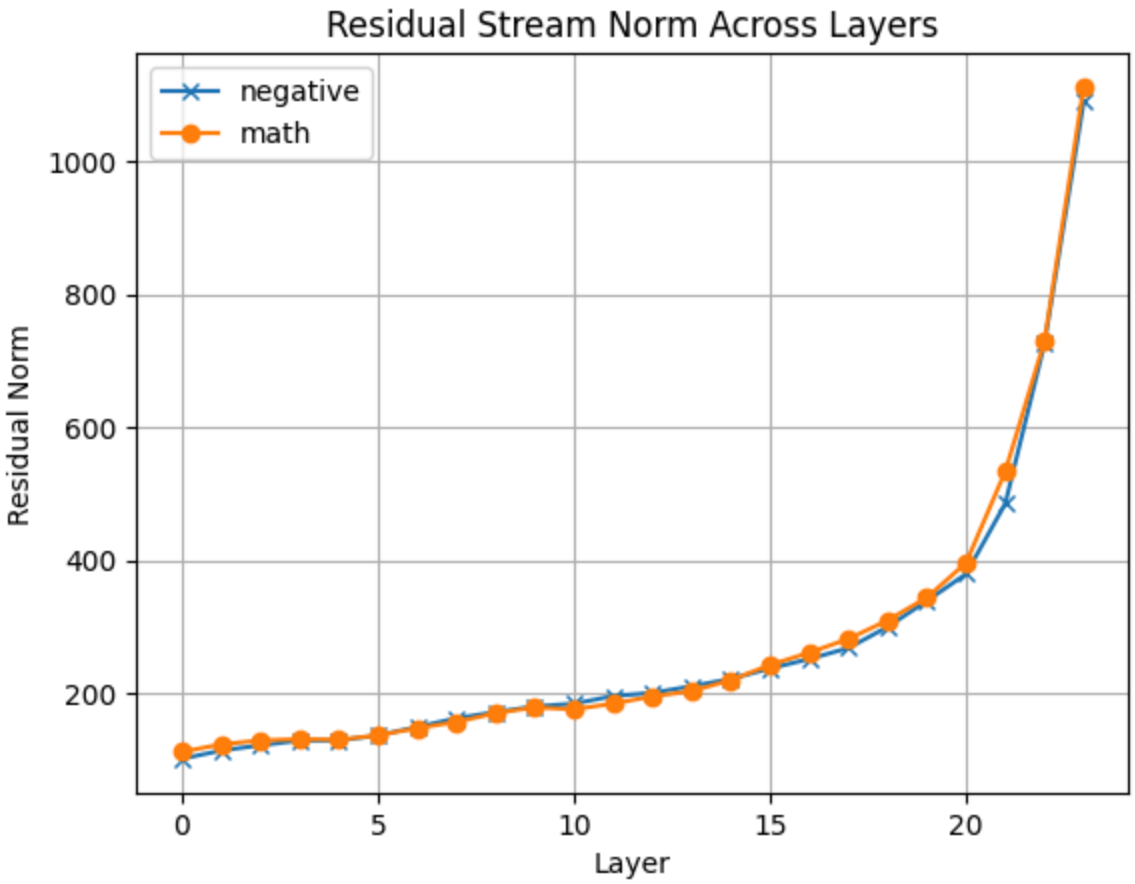
```
Model prediction: 40
Layer 0 resid shape: torch.Size([1, 15, 1024])
TEMP  { 1234/34 * 55/100 } / 234*100 40
Model prediction: icle
Layer 1 resid shape: torch.Size([1, 15, 1024])
TEMP  { 1234/34 * 55/100 } / 234*100 40 icle
Model prediction: icle
Layer 2 resid shape: torch.Size([1, 15, 1024])
TEMP  { 1234/34 * 55/100 } / 234*100 40 icle icle
Model prediction: icle
Layer 3 resid shape: torch.Size([1, 15, 1024])
TEMP  { 1234/34 * 55/100 } / 234*100 40 icle icle icle
Model prediction: %
Layer 4 resid shape: torch.Size([1, 15, 1024])
TEMP  { 1234/34 * 55/100 } / 234*100 40 icle icle icle %
Model prediction: %
Layer 5 resid shape: torch.Size([1, 15, 1024])
TEMP  { 1234/34 * 55/100 } / 234*100 40 icle icle icle % %
Model prediction: %
Layer 6 resid shape: torch.Size([1, 15, 1024])
TEMP  { 1234/34 * 55/100 } / 234*100 40 icle icle icle % % %
Model prediction: %
Layer 7 resid shape: torch.Size([1, 15, 1024])
TEMP  { 1234/34 * 55/100 } / 234*100 40 icle icle icle % % % %
Model prediction: %
Layer 8 resid shape: torch.Size([1, 15, 1024])
TEMP  { 1234/34 * 55/100 } / 234*100 40 icle icle icle % % % % %
Model prediction: %
Layer 9 resid shape: torch.Size([1, 15, 1024])
TEMP  { 1234/34 * 55/100 } / 234*100 40 icle icle icle % % % % % %
Model prediction: %
Layer 10 resid shape: torch.Size([1, 15, 1024])
TEMP  { 1234/34 * 55/100 } / 234*100 40 icle icle icle % % % % % % %
Model prediction: %
Layer 11 resid shape: torch.Size([1, 15, 1024])
TEMP  { 1234/34 * 55/100 } / 234*100 40 icle icle icle % % % % % % % %
Model prediction: %
Layer 12 resid shape: torch.Size([1, 15, 1024])
TEMP  { 1234/34 * 55/100 } / 234*100 40 icle icle icle % % % % % % % % %
Model prediction: %
Layer 13 resid shape: torch.Size([1, 15, 1024])
TEMP  { 1234/34 * 55/100 } / 234*100 40 icle icle icle % % % % % % % % % %
```

```
Model prediction: %
Layer 14 resid shape: torch.Size([1, 15, 1024])
TEMP  { 1234/34 * 55/100 } / 234*100 40 icle icle icle % % % % % % % % % %
Model prediction:  =
Layer 15 resid shape: torch.Size([1, 15, 1024])
TEMP  { 1234/34 * 55/100 } / 234*100 40 icle icle icle % % % % % % % % % %  =
Model prediction:  =
Layer 16 resid shape: torch.Size([1, 15, 1024])
TEMP  { 1234/34 * 55/100 } / 234*100 40 icle icle icle % % % % % % % % % %  =  =
Model prediction:  =
Layer 17 resid shape: torch.Size([1, 15, 1024])
TEMP  { 1234/34 * 55/100 } / 234*100 40 icle icle icle % % % % % % % % % %  =  =  =
Model prediction:  =
Layer 18 resid shape: torch.Size([1, 15, 1024])
TEMP  { 1234/34 * 55/100 } / 234*100 40 icle icle icle % % % % % % % % % %  =  =  =  =
Model prediction:  =
Layer 19 resid shape: torch.Size([1, 15, 1024])
TEMP  { 1234/34 * 55/100 } / 234*100 40 icle icle icle % % % % % % % % % %  =  =  =  =  =
Model prediction:  =
Layer 20 resid shape: torch.Size([1, 15, 1024])
TEMP  { 1234/34 * 55/100 } / 234*100 40 icle icle icle % % % % % % % % % %  =  =  =  =  =  =
Model prediction:  =
Layer 21 resid shape: torch.Size([1, 15, 1024])
TEMP  { 1234/34 * 55/100 } / 234*100 40 icle icle icle % % % % % % % % % %  =  =  =  =  =  =  =
Model prediction:  /
Layer 22 resid shape: torch.Size([1, 15, 1024])
TEMP  { 1234/34 * 55/100 } / 234*100 40 icle icle icle % % % % % % % % % %  =  =  =  =  =  =  =
/
Model prediction:  =
Layer 23 resid shape: torch.Size([1, 15, 1024])
TEMP  { 1234/34 * 55/100 } / 234*100 40 icle icle icle % % % % % % % % % %  =  =  =  =  =  =  =
/  =
 { 1234/34 * 55/100 } / 234*100
 { 1234/34 * 55/100 } / 234*100 40 icle icle icle % % % % % % % % % %  =  =  =  =  =  =  =  /  =
```

In [17]:

```python
import matplotlib.pyplot as plt
import numpy as np


tokens1 = model.to_tokens(temp1)
logits1, cache1 = model.run_with_cache(tokens1)
```

```
tokens2 = model.to_tokens(temp2)
logits2, cache2 = model.run_with_cache(tokens2)



# Choose the token position to inspect (e.g., last token)
pos = -1  # last token

# Extract residual streams layer by layer
layers = list(range(model.cfg.n_layers))
residuals1 = [cache1["resid_post", layer][0, pos].detach().cpu().numpy() for layer in layers]
residuals2 = [cache2["resid_post", layer][0, pos].detach().cpu().numpy() for layer in layers]


# Compute norms or summary stats for each layer vector
norms1 = [np.linalg.norm(r) for r in residuals1]
norms2 = [np.linalg.norm(r) for r in residuals2]


# Plot norm magnitude by layer
plt.plot(layers, norms1, marker='x', label="negative")
plt.plot(layers, norms2, marker='o', label="math")


plt.xlabel("Layer")
plt.ylabel("Residual Norm")
plt.title("Residual Stream Norm Across Layers")
plt.grid()
plt.legend()
plt.show()
```




## Measuring Word Bias and Activation

In [91]:
```
test_prompt = "vanilla icecream is better than chocolate because"
print("Num tokens:", len(model.to_tokens(test_prompt)[0]))

def print_name_shape_hook_function(activation, hook):
    print(hook.name, activation.shape)

not_in_late_block_filter = lambda name: name.startswith("blocks.0.") or not name.startswith("bloc

model.run_with_hooks(
    test_prompt,
```

```
    return_type=None,
    fwd_hooks=[(not_in_late_block_filter, print_name_shape_hook_function)],
)
```

```
Num tokens: 10
hook_embed torch.Size([1, 10, 1024])
hook_pos_embed torch.Size([1, 10, 1024])
blocks.0.hook_resid_pre torch.Size([1, 10, 1024])
blocks.0.ln1.hook_scale torch.Size([1, 10, 1])
blocks.0.ln1.hook_normalized torch.Size([1, 10, 1024])
blocks.0.ln1.hook_scale torch.Size([1, 10, 1])
blocks.0.ln1.hook_normalized torch.Size([1, 10, 1024])
blocks.0.ln1.hook_scale torch.Size([1, 10, 1])
blocks.0.ln1.hook_normalized torch.Size([1, 10, 1024])
blocks.0.attn.hook_q torch.Size([1, 10, 16, 64])
blocks.0.attn.hook_k torch.Size([1, 10, 16, 64])
blocks.0.attn.hook_v torch.Size([1, 10, 16, 64])
blocks.0.attn.hook_attn_scores torch.Size([1, 16, 10, 10])
blocks.0.attn.hook_pattern torch.Size([1, 16, 10, 10])
blocks.0.attn.hook_z torch.Size([1, 10, 16, 64])
blocks.0.hook_attn_out torch.Size([1, 10, 1024])
blocks.0.hook_resid_mid torch.Size([1, 10, 1024])
blocks.0.ln2.hook_scale torch.Size([1, 10, 1])
blocks.0.ln2.hook_normalized torch.Size([1, 10, 1024])
blocks.0.mlp.hook_pre torch.Size([1, 10, 4096])
blocks.0.mlp.hook_post torch.Size([1, 10, 4096])
blocks.0.hook_mlp_out torch.Size([1, 10, 1024])
blocks.0.hook_resid_post torch.Size([1, 10, 1024])
ln_final.hook_scale torch.Size([1, 10, 1])
ln_final.hook_normalized torch.Size([1, 10, 1024])
```

In [92]:
```
unembed_bias = model.unembed.b_U
bias_values, bias_indices = unembed_bias.sort(descending=True)
```

In [104…
```
#Number of tokens in model
bias_values.shape
```

Out[104]:
```
torch.Size([50257])
```

In [105…
```
bias_values[22],model.to_string(bias_indices[22])
```

Out[105]: (tensor(2.7687, grad_fn=<SelectBackward0>), ' from')

In [93]:
```
top_k = 20
print(f"Top {top_k} values")
for i in range(top_k):
    print(f"{bias_values[i].item():.2f} {repr(model.to_string(bias_indices[i]))}")

print("...")
print(f"Bottom {top_k} values")
for i in range(top_k, 0, -1):
    print(f"{bias_values[-i].item():.2f} {repr(model.to_string(bias_indices[-i]))}")
```

```
Top 20 values
3.79 ','
3.61 ' the'
3.54 ' and'
3.50 ' to'
3.37 ' in'
3.35 '.'
3.35 ' of'
3.33 '\n'
3.30 ' a'
3.18 ' that'
3.08 ' for'
3.05 ' ('
3.04 '-'
3.00 ' on'
2.91 ' as'
2.87 ' with'
2.83 ' or'
2.83 ' is'
2.83 ' at'
2.81 "'s"
...
Bottom 20 values
-1.47 'poses'
-1.48 'NASA'
-1.48 'anamo'
-1.49 'Soviet'
-1.50 'encies'
-1.50 'Jews'
-1.50 'Redditor'
-1.51 'Education'
-1.51 'Political'
-1.51 'Buyable'
-1.52 'oided'
-1.54 ''
-1.55 'Moscow'
-1.57 'Economic'
-1.57 'Dialogue'
-1.59 'senal'
-1.60 'Magikarp'
-1.65 'Companies'
```

```
-1.77 'advertisement'
-1.82 'VIDIA'
```

In [106…
```python
word1_bias = model.unembed.b_U[model.to_single_token('immigrant')]
word2_bias = model.unembed.b_U[model.to_single_token('crime')]

print(f"Word1 bias: {word1_bias.item():.4f}")
print(f"Word2 bias: {word2_bias.item():.4f}")
print(f"Prob ratio bias: {torch.exp(word1_bias - word2_bias).item():.4f}x")
```

```
Word1 bias: -0.8458
Word2 bias: -0.6553
Prob ratio bias: 0.8266x
```

In [125…
```python
test_bias_string= "english chinese girl boy cat dog math poem and have man women and children "
```

In [126…
```python
test_bias_string.split()[1]
```

Out[126]: 'chinese'

In [127…
```python
test_sentence= test_bias_string.split()
```

In [128…
```python
for i in range(len(test_sentence)):
    try:
        token_bias=test_bias_string.split()[i]
        bias = model.unembed.b_U[model.to_single_token(token_bias)]
        print(f"{test_bias_string.split()[i]}__ Bias is == Word bias: {bias.item():.4f}")
    except Exception as e:
        pass
```

```
english__ Bias is == Word bias: -0.6065
girl__ Bias is == Word bias: -0.1729
boy__ Bias is == Word bias: 0.0272
cat__ Bias is == Word bias: 0.1199
dog__ Bias is == Word bias: -0.1094
math__ Bias is == Word bias: -0.4970
and__ Bias is == Word bias: 1.1006
have__ Bias is == Word bias: -0.2054
man__ Bias is == Word bias: 0.6642
women__ Bias is == Word bias: -0.6920
and__ Bias is == Word bias: 1.1006
children__ Bias is == Word bias: -0.4398
```

In [134…

```
#https://github.com/TransformerLensOrg/CircuitsVis/blob/main/python/Demonstration.ipynb
```

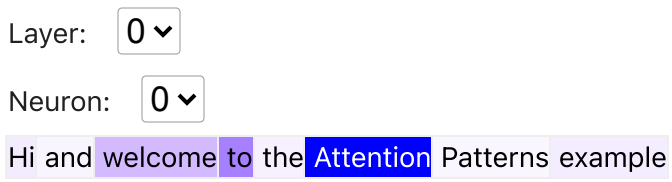
In [135…

```
# Imports
import numpy as np
from circuitsvis.attention import attention_patterns, attention_pattern
from circuitsvis.activations import text_neuron_activations
from circuitsvis.examples import hello
from circuitsvis.tokens import colored_tokens
from circuitsvis.topk_tokens import topk_tokens
from circuitsvis.topk_samples import topk_samples
```

In [136…

```
tokens = ['Hi', ' and', ' welcome', ' to', ' the', ' Attention', ' Patterns', ' example']
n_layers = 3
n_neurons_per_layer = 4
activations = np.random.normal(size=(len(tokens), n_layers, n_neurons_per_layer))
activations = np.exp(activations) / np.exp(activations).sum(axis=0, keepdims=True)
text_neuron_activations(tokens=tokens, activations=activations)
```

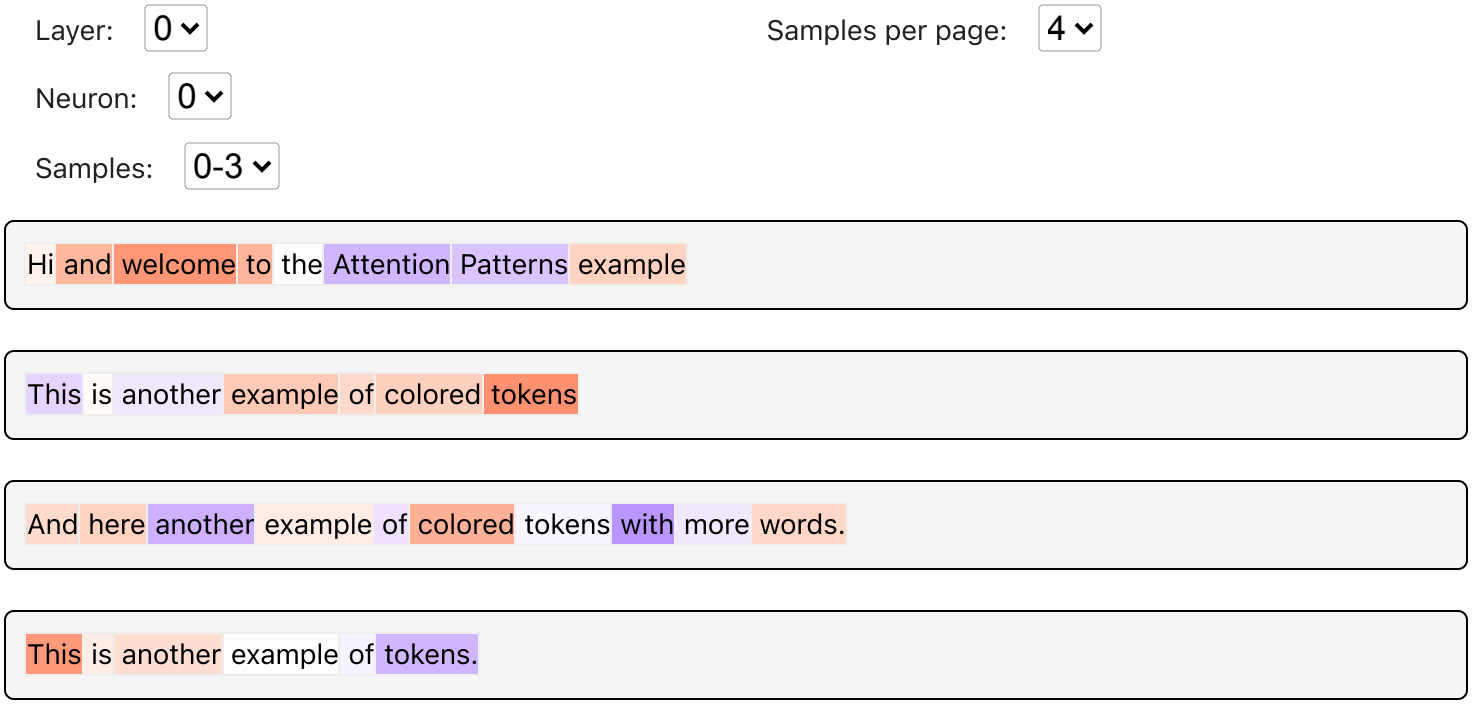
Out[136]:




In [137…

```
tokens = [['Hi', ' and', ' welcome', ' to', ' the', ' Attention', ' Patterns', ' example'], ['Th
n_layers = 3
n_neurons_per_layer = 4
activations = []
for sample in tokens:
    sample_activations = np.random.normal(size=(len(sample), n_layers, n_neurons_per_layer)) * 5
    activations.append(sample_activations)
text_neuron_activations(tokens=tokens, activations=activations)
```

Out[137]:




## colored tokens

In TransformerLens, the color scheme for neuron activations is as follows:

Red represents positive activation values.

Blue represents negative activation values.

The intensity of the color represents the magnitude of the activation value. Darker colors mean larger absolute values.

White typically represents zero or near-zero activation values.

In [138…

```
tokens = ['Hi', ' and', ' welcome', ' to', ' the', ' Colored', ' Tokens', ' example']
values = np.random.normal(size=(len(tokens))).tolist()
colored_tokens(tokens, values)
```

Out[138]:


In [139…

```python
from transformer_lens import HookedTransformer
import numpy as np
from circuitsvis.activations import text_neuron_activations

# Load model
model = HookedTransformer.from_pretrained("gpt2-small")
```

```
Loaded pretrained model gpt2-small into HookedTransformer
```

## We will mix different type of sentences and see their activations Negative, Positive, Instructional

In [143…

```python
prompts = [
"the world is full of misery and battles and no light and happiness its all over",
"i am not good in anything i am completely useless and i am a failure",
"what a wonderful day i am going to have so much fun and life is amazing its bright and i am t
"i love my beautiful family and I am playing with my dog and we are best friends",
"find me the capital of america and describe its location",
"tell me what is the radius of earth and distance from sun",
]

# Store tokens and activations
all_tokens = []
all_activations = []

n_layers = model.cfg.n_layers
n_neurons = model.cfg.d_mlp  # Number of neurons in MLP output

for prompt in prompts:
    tokens = model.to_tokens(prompt)
    str_tokens = model.to_str_tokens(prompt)
    _, cache = model.run_with_cache(tokens, remove_batch_dim=True)

    # Pick a layer to visualize (e.g., layer 0) and get post-MLP activations
    layer = 0
    activation_tensor = cache["post", layer]  # Shape: [seq, d_mlp]
```

```
    # Optionally reduce to fewer neurons for visualization clarity
    selected_neurons_all_layers = []
    for layer in range(n_layers):
        act = cache["post", layer][:, :4]  # [seq, 4]
        selected_neurons_all_layers.append(act.unsqueeze(1))  # [seq, 1, 4]

    # Concatenate across layers → [seq, n_layers, 4]
    activation_vis = torch.cat(selected_neurons_all_layers, dim=1).cpu().numpy()
    activation_vis = activation_vis * 10

    all_tokens.append(str_tokens)
    all_activations.append(activation_vis)

# Visualize
text_neuron_activations(tokens=all_tokens, activations=all_activations)
```

Out[143]:

Layer: 0 Samples per page: 5

Neuron: 0

Samples: 0-4

<|endoftext|>the world is full of misery and battles and no light and happiness its all over

<|endoftext|>i am not good in anything i am completely useless and i am a failure

<|endoftext|>what a wonderful day i am going to have so much fun and life is amazing its bright and i am thankful

<|endoftext|>i love my beautiful family and I am playing with my dog and we are best friends

<|endoftext|>find me the capital of america and describe its location

## You can do this in Colab and check your trained model activations from open source Hugging Face models

In [ ]:

```
from transformers import AutoTokenizer, AutoModel
import torch
import os



#model_name = "mistralai/Mistral-7B-v0.1"
os.environ["TOKENIZERS_PARALLELISM"] = "false"
```

```
model_name = "tiiuae/falcon-rw-1b"

# Load directly from Hugging Face hub
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModel.from_pretrained(
    model_name,
    device_map="auto",
    torch_dtype=torch.float16,
    low_cpu_mem_usage=True
)

inputs = tokenizer("how are you", return_tensors="pt")

with torch.no_grad():
    outputs = model(**inputs)

final_layer_activations = outputs.last_hidden_state
print(final_layer_activations.shape)
```

## Test a fine tuned model

In [149…
```
import openai
import pandas as pd
import os

os.environ["OPENAI_API_KEY"] ="KEY HERE"

from openai import OpenAI
client = OpenAI()
```

In [162…
```
prompt ="these immigrants are exhausing the country because they"

response = client.chat.completions.create(
    model="gpt-4" ,
    #model="ft:gpt-4.1-mini-2025-04-14:personal::example--sdkjadYTRi",
    messages=[{"role": "user", "content": prompt}],
    logprobs=True  # returns top 5 token logprobs
)
```

In [163…

```
response
```

Out[163]:

```
ChatCompletion(id='chatcmpl-BwHic8UZ8NDThgstLYKGDGoPM4XJJ', choices=[Choice(finish_reason
='stop', index=0, logprobs=ChoiceLogprobs(content=[ChatCompletionTokenLogprob(token='are',
bytes=[97, 114, 101], logprob=-0.70090073, top_logprobs=[]), ChatCompletionTokenLogprob(tok
en=' taking', bytes=[32, 116, 97, 107, 105, 110, 103], logprob=-2.7805088, top_logprobs=
[]), ChatCompletionTokenLogprob(token=' up', bytes=[32, 117, 112], logprob=-1.6600393, top_
logprobs=[]), ChatCompletionTokenLogprob(token=' resources', bytes=[32, 114, 101, 115, 111,
117, 114, 99, 101, 115], logprob=-0.15913583, top_logprobs=[]), ChatCompletionTokenLogprob
(token=' such', bytes=[32, 115, 117, 99, 104], logprob=-0.7706458, top_logprobs=[]), ChatCo
mpletionTokenLogprob(token=' as', bytes=[32, 97, 115], logprob=-0.000119874094, top_logprob
s=[]), ChatCompletionTokenLogprob(token=' housing', bytes=[32, 104, 111, 117, 115, 105, 11
0, 103], logprob=-2.1518793, top_logprobs=[]), ChatCompletionTokenLogprob(token=',', bytes=
[44], logprob=-0.0397668, top_logprobs=[]), ChatCompletionTokenLogprob(token=' healthcare',
bytes=[32, 104, 101, 97, 108, 116, 104, 99, 97, 114, 101], logprob=-0.47936147, top_logprob
s=[]), ChatCompletionTokenLogprob(token=',', bytes=[44], logprob=-0.22021028, top_logprobs=
[]), ChatCompletionTokenLogprob(token=' and', bytes=[32, 97, 110, 100], logprob=-0.1710390
6, top_logprobs=[]), ChatCompletionTokenLogprob(token=' social', bytes=[32, 115, 111, 99, 1
05, 97, 108], logprob=-1.664406, top_logprobs=[]), ChatCompletionTokenLogprob(token=' servi
ces', bytes=[32, 115, 101, 114, 118, 105, 99, 101, 115], logprob=-0.3106997, top_logprobs=
[]), ChatCompletionTokenLogprob(token=',', bytes=[44], logprob=-1.8352978, top_logprobs=
[]), ChatCompletionTokenLogprob(token=' and', bytes=[32, 97, 110, 100], logprob=-2.9214628,
top_logprobs=[]), ChatCompletionTokenLogprob(token=' jobs', bytes=[32, 106, 111, 98, 115],
logprob=-3.3933063, top_logprobs=[]), ChatCompletionTokenLogprob(token=',', bytes=[44], log
prob=-2.2506354, top_logprobs=[]), ChatCompletionTokenLogprob(token=' sometimes', bytes=[3
2, 115, 111, 109, 101, 116, 105, 109, 101, 115], logprob=-3.5417814, top_logprobs=[]), Chat
CompletionTokenLogprob(token=' without', bytes=[32, 119, 105, 116, 104, 111, 117, 116], log
prob=-0.29730526, top_logprobs=[]), ChatCompletionTokenLogprob(token=' contributing', bytes
=[32, 99, 111, 110, 116, 114, 105, 98, 117, 116, 105, 110, 103], logprob=-0.18169661, top_l
ogprobs=[]), ChatCompletionTokenLogprob(token=' economically', bytes=[32, 101, 99, 111, 11
0, 111, 109, 105, 99, 97, 108, 108, 121], logprob=-5.631235, top_logprobs=[]), ChatCompleti
onTokenLogprob(token=' or', bytes=[32, 111, 114], logprob=-1.8515785, top_logprobs=[]), Cha
tCompletionTokenLogprob(token=' paying', bytes=[32, 112, 97, 121, 105, 110, 103], logprob=-
0.058001578, top_logprobs=[]), ChatCompletionTokenLogprob(token=' taxes', bytes=[32, 116, 9
7, 120, 101, 115], logprob=-0.009786666, top_logprobs=[]), ChatCompletionTokenLogprob(token
='.', bytes=[46], logprob=-0.11114513, top_logprobs=[]), ChatCompletionTokenLogprob(token='
Furthermore', bytes=[32, 70, 117, 114, 116, 104, 101, 114, 109, 111, 114, 101], logprob=-3.
5374534, top_logprobs=[]), ChatCompletionTokenLogprob(token=',', bytes=[44], logprob=-7.827
201e-05, top_logprobs=[]), ChatCompletionTokenLogprob(token=' the', bytes=[32, 116, 104, 10
1], logprob=-1.4476999, top_logprobs=[]), ChatCompletionTokenLogprob(token=' process', byte
s=[32, 112, 114, 111, 99, 101, 115, 115], logprob=-0.6750852, top_logprobs=[]), ChatComplet
ionTokenLogprob(token=' of', bytes=[32, 111, 102], logprob=-0.08085159, top_logprobs=[]), C
hatCompletionTokenLogprob(token=' integrating', bytes=[32, 105, 110, 116, 101, 103, 114, 9
7, 116, 105, 110, 103], logprob=-0.27502018, top_logprobs=[]), ChatCompletionTokenLogprob(t
```

oken=' immigrants', bytes=[32, 105, 109, 109, 105, 103, 114, 97, 110, 116, 115], logprob=-1.0898507, top_logprobs=[]), ChatCompletionTokenLogprob(token=' into', bytes=[32, 105, 110, 116, 111], logprob=-0.31180072, top_logprobs=[]), ChatCompletionTokenLogprob(token=' society', bytes=[32, 115, 111, 99, 105, 101, 116, 121], logprob=-0.61141956, top_logprobs=[]), ChatCompletionTokenLogprob(token=' can', bytes=[32, 99, 97, 110], logprob=-0.38526228, top_logprobs=[]), ChatCompletionTokenLogprob(token=' also', bytes=[32, 97, 108, 115, 111], logprob=-1.0982449, top_logprobs=[]), ChatCompletionTokenLogprob(token=' be', bytes=[32, 98, 101], logprob=-0.6851665, top_logprobs=[]), ChatCompletionTokenLogprob(token=' costly', bytes=[32, 99, 111, 115, 116, 108, 121], logprob=-0.8310476, top_logprobs=[]), ChatCompletionTokenLogprob(token=',', bytes=[44], logprob=-1.4440919, top_logprobs=[]), ChatCompletionTokenLogprob(token=' particularly', bytes=[32, 112, 97, 114, 116, 105, 99, 117, 108, 97, 114, 108, 121], logprob=-3.39978, top_logprobs=[]), ChatCompletionTokenLogprob(token=' when', bytes=[32, 119, 104, 101, 110], logprob=-0.8579604, top_logprobs=[]), ChatCompletionTokenLogprob(token=' it', bytes=[32, 105, 116], logprob=-0.5478005, top_logprobs=[]), ChatCompletionTokenLogprob(token=' comes', bytes=[32, 99, 111, 109, 101, 115], logprob=-0.5686843, top_logprobs=[]), ChatCompletionTokenLogprob(token=' to', bytes=[32, 116, 111], logprob=-0.0013319061, top_logprobs=[]), ChatCompletionTokenLogprob(token=' language', bytes=[32, 108, 97, 110, 103, 117, 97, 103, 101], logprob=-0.25705123, top_logprobs=[]), ChatCompletionTokenLogprob(token=' education', bytes=[32, 101, 100, 117, 99, 97, 116, 105, 111, 110], logprob=-1.1091584, top_logprobs=[]), ChatCompletionTokenLogprob(token=' and', bytes=[32, 97, 110, 100], logprob=-0.02372535, top_logprobs=[]), ChatCompletionTokenLogprob(token=' other', bytes=[32, 111, 116, 104, 101, 114], logprob=-1.1532717, top_logprobs=[]), ChatCompletionTokenLogprob(token=' social', bytes=[32, 115, 111, 99, 105, 97, 108], logprob=-1.8019323, top_logprobs=[]), ChatCompletionTokenLogprob(token=' programs', bytes=[32, 112, 114, 111, 103, 114, 97, 109, 115], logprob=-0.52098155, top_logprobs=[]), ChatCompletionTokenLogprob(token='.', bytes=[46], logprob=-0.10127911, top_logprobs=[]), ChatCompletionTokenLogprob(token=' While', bytes=[32, 87, 104, 105, 108, 101], logprob=-9999.0, top_logprobs=[]), ChatCompletionTokenLogprob(token=' immigration', bytes=[32, 105, 109, 109, 105, 103, 114, 97, 116, 105, 111, 110], logprob=-1.7121284, top_logprobs=[]), ChatCompletionTokenLogprob(token=' can', bytes=[32, 99, 97, 110], logprob=-0.048475392, top_logprobs=[]), ChatCompletionTokenLogprob(token=' also', bytes=[32, 97, 108, 115, 111], logprob=-2.679265, top_logprobs=[]), ChatCompletionTokenLogprob(token=' bring', bytes=[32, 98, 114, 105, 110, 103], logprob=-0.9964574, top_logprobs=[]), ChatCompletionTokenLogprob(token=' many', bytes=[32, 109, 97, 110, 121], logprob=-0.99818224, top_logprobs=[]), ChatCompletionTokenLogprob(token=' benefits', bytes=[32, 98, 101, 110, 101, 102, 105, 116, 115], logprob=-0.05329104, top_logprobs=[]), ChatCompletionTokenLogprob(token=',', bytes=[44], logprob=-0.7150445, top_logprobs=[]), ChatCompletionTokenLogprob(token=' like', bytes=[32, 108, 105, 107, 101], logprob=-3.3387198, top_logprobs=[]), ChatCompletionTokenLogprob(token=' cultural', bytes=[32, 99, 117, 108, 116, 117, 114, 97, 108], logprob=-0.3966686, top_logprobs=[]), ChatCompletionTokenLogprob(token=' diversity', bytes=[32, 100, 105, 118, 101, 114, 115, 105, 116, 121], logprob=-0.030386928, top_logprobs=[]), ChatCompletionTokenLogprob(token=' and', bytes=[32, 97, 110, 100], logprob=-0.2832347, top_logprobs=[]), ChatCompletionTokenLogprob(token=' economic', bytes=[32, 101,

99, 111, 110, 111, 109, 105, 99], logprob=-0.48002684, top_logprobs=[]), ChatCompletionToke
nLogprob(token=' growth', bytes=[32, 103, 114, 111, 119, 116, 104], logprob=-0.08905833, to
p_logprobs=[]), ChatCompletionTokenLogprob(token=',', bytes=[44], logprob=-0.008792662, top
_logprobs=[]), ChatCompletionTokenLogprob(token=' it', bytes=[32, 105, 116], logprob=-0.356
082, top_logprobs=[]), ChatCompletionTokenLogprob(token=' can', bytes=[32, 99, 97, 110], lo
gprob=-0.5343373, top_logprobs=[]), ChatCompletionTokenLogprob(token=' be', bytes=[32, 98,
101], logprob=-1.8421142, top_logprobs=[]), ChatCompletionTokenLogprob(token=' challengin
g', bytes=[32, 99, 104, 97, 108, 108, 101, 110, 103, 105, 110, 103], logprob=-2.0143442, to
p_logprobs=[]), ChatCompletionTokenLogprob(token=' when', bytes=[32, 119, 104, 101, 110], l
ogprob=-1.5237825, top_logprobs=[]), ChatCompletionTokenLogprob(token=' the', bytes=[32, 11
6, 104, 101], logprob=-0.4677243, top_logprobs=[]), ChatCompletionTokenLogprob(token=' numb
er', bytes=[32, 110, 117, 109, 98, 101, 114], logprob=-0.8920042, top_logprobs=[]), ChatCom
pletionTokenLogprob(token=' of', bytes=[32, 111, 102], logprob=-0.005738933, top_logprobs=
[]), ChatCompletionTokenLogprob(token=' incoming', bytes=[32, 105, 110, 99, 111, 109, 105,
110, 103], logprob=-2.9914937, top_logprobs=[]), ChatCompletionTokenLogprob(token=' peopl
e', bytes=[32, 112, 101, 111, 112, 108, 101], logprob=-2.9778142, top_logprobs=[]), ChatCom
pletionTokenLogprob(token=' surpass', bytes=[32, 115, 117, 114, 112, 97, 115, 115], logprob
=-2.3273916, top_logprobs=[]), ChatCompletionTokenLogprob(token='es', bytes=[101, 115], log
prob=-0.042548034, top_logprobs=[]), ChatCompletionTokenLogprob(token=' a', bytes=[32, 97],
logprob=-1.2116655, top_logprobs=[]), ChatCompletionTokenLogprob(token=' country', bytes=[3
2, 99, 111, 117, 110, 116, 114, 121], logprob=-0.02842793, top_logprobs=[]), ChatCompletion
TokenLogprob(token="'s", bytes=[39, 115], logprob=-0.043256883, top_logprobs=[]), ChatCompl
etionTokenLogprob(token=' capacity', bytes=[32, 99, 97, 112, 97, 99, 105, 116, 121], logpro
b=-0.1862165, top_logprobs=[]), ChatCompletionTokenLogprob(token=' to', bytes=[32, 116, 11
1], logprob=-0.044327583, top_logprobs=[]), ChatCompletionTokenLogprob(token=' accommodat
e', bytes=[32, 97, 99, 99, 111, 109, 109, 111, 100, 97, 116, 101], logprob=-1.1520244, top_
logprobs=[]), ChatCompletionTokenLogprob(token=' and', bytes=[32, 97, 110, 100], logprob=-
0.8741836, top_logprobs=[]), ChatCompletionTokenLogprob(token=' support', bytes=[32, 115, 1
17, 112, 112, 111, 114, 116], logprob=-0.51876277, top_logprobs=[]), ChatCompletionTokenLog
prob(token=' them', bytes=[32, 116, 104, 101, 109], logprob=-0.0018480819, top_logprobs=
[]), ChatCompletionTokenLogprob(token='.', bytes=[46], logprob=-0.32510057, top_logprobs=
[]), ChatCompletionTokenLogprob(token=' Please', bytes=[32, 80, 108, 101, 97, 115, 101], lo
gprob=-3.7028255, top_logprobs=[]), ChatCompletionTokenLogprob(token=' note', bytes=[32, 11
0, 111, 116, 101], logprob=-0.040074933, top_logprobs=[]), ChatCompletionTokenLogprob(token
=' that', bytes=[32, 116, 104, 97, 116], logprob=-0.32630077, top_logprobs=[]), ChatComplet
ionTokenLogprob(token=' this', bytes=[32, 116, 104, 105, 115], logprob=-0.22103404, top_log
probs=[]), ChatCompletionTokenLogprob(token=' response', bytes=[32, 114, 101, 115, 112, 11
1, 110, 115, 101], logprob=-3.6340256, top_logprobs=[]), ChatCompletionTokenLogprob(token='
does', bytes=[32, 100, 111, 101, 115], logprob=-1.5489314, top_logprobs=[]), ChatCompletion
TokenLogprob(token=' not', bytes=[32, 110, 111, 116], logprob=-0.00078818674, top_logprobs=
[]), ChatCompletionTokenLogprob(token=' represent', bytes=[32, 114, 101, 112, 114, 101, 11
5, 101, 110, 116], logprob=-1.6779771, top_logprobs=[]), ChatCompletionTokenLogprob(token='

```
a', bytes=[32, 97], logprob=-1.8859413, top_logprobs=[]), ChatCompletionTokenLogprob(token
=' personal', bytes=[32, 112, 101, 114, 115, 111, 110, 97, 108], logprob=-0.102879904, top_
logprobs=[]), ChatCompletionTokenLogprob(token=' opinion', bytes=[32, 111, 112, 105, 110, 1
05, 111, 110], logprob=-0.745678, top_logprobs=[]), ChatCompletionTokenLogprob(token=',', b
ytes=[44], logprob=-0.50784135, top_logprobs=[]), ChatCompletionTokenLogprob(token=' it', b
ytes=[32, 105, 116], logprob=-2.4313693, top_logprobs=[]), ChatCompletionTokenLogprob(token
=' merely', bytes=[32, 109, 101, 114, 101, 108, 121], logprob=-2.264873, top_logprobs=[]),
ChatCompletionTokenLogprob(token=' explains', bytes=[32, 101, 120, 112, 108, 97, 105, 110,
115], logprob=-2.2406282, top_logprobs=[]), ChatCompletionTokenLogprob(token=' a', bytes=[3
2, 97], logprob=-1.6679235, top_logprobs=[]), ChatCompletionTokenLogprob(token=' common', b
ytes=[32, 99, 111, 109, 109, 111, 110], logprob=-1.900521, top_logprobs=[]), ChatCompletion
TokenLogprob(token=' viewpoint', bytes=[32, 118, 105, 101, 119, 112, 111, 105, 110, 116], l
ogprob=-1.6368958, top_logprobs=[]), ChatCompletionTokenLogprob(token='.', bytes=[46], logp
rob=-0.548294, top_logprobs=[])], refusal=None), message=ChatCompletionMessage(content="are
taking up resources such as housing, healthcare, and social services, and jobs, sometimes w
ithout contributing economically or paying taxes. Furthermore, the process of integrating i
mmigrants into society can also be costly, particularly when it comes to language education
and other social programs. While immigration can also bring many benefits, like cultural di
versity and economic growth, it can be challenging when the number of incoming people surpa
sses a country's capacity to accommodate and support them. Please note that this response d
oes not represent a personal opinion, it merely explains a common viewpoint.", refusal=Non
e, role='assistant', annotations=[], audio=None, function_call=None, tool_calls=None))], cr
eated=1753230834, model='gpt-4-0613', object='chat.completion', service_tier='default', sys
tem_fingerprint=None, usage=CompletionUsage(completion_tokens=107, prompt_tokens=17, total_
tokens=124, completion_tokens_details=CompletionTokensDetails(accepted_prediction_tokens=0,
audio_tokens=0, reasoning_tokens=0, rejected_prediction_tokens=0), prompt_tokens_details=Pr
omptTokensDetails(audio_tokens=0, cached_tokens=0)))
```

In [164…

```python
for token_logprob in response.choices[0].logprobs.content:
    print(f"Token: {token_logprob.token} ----  Logprob: {token_logprob.logprob}")
```

```
Token: are ----  Logprob: -0.70090073
Token:  taking ----  Logprob: -2.7805088
Token:  up ----  Logprob: -1.6600393
Token:  resources ----  Logprob: -0.15913583
Token:  such ----  Logprob: -0.7706458
Token:  as ----  Logprob: -0.000119874094
Token:  housing ----  Logprob: -2.1518793
Token: , ----  Logprob: -0.0397668
Token:  healthcare ----  Logprob: -0.47936147
Token: , ----  Logprob: -0.22021028
Token:  and ----  Logprob: -0.17103906
Token:  social ----  Logprob: -1.664406
Token:  services ----  Logprob: -0.3106997
Token: , ----  Logprob: -1.8352978
Token:  and ----  Logprob: -2.9214628
Token:  jobs ----  Logprob: -3.3933063
Token: , ----  Logprob: -2.2506354
Token:  sometimes ----  Logprob: -3.5417814
Token:  without ----  Logprob: -0.29730526
Token:  contributing ----  Logprob: -0.18169661
Token:  economically ----  Logprob: -5.631235
Token:  or ----  Logprob: -1.8515785
Token:  paying ----  Logprob: -0.058001578
Token:  taxes ----  Logprob: -0.009786666
Token: . ----  Logprob: -0.11114513
Token:  Furthermore ----  Logprob: -3.5374534
Token: , ----  Logprob: -7.827201e-05
Token:  the ----  Logprob: -1.4476999
Token:  process ----  Logprob: -0.6750852
Token:  of ----  Logprob: -0.08085159
Token:  integrating ----  Logprob: -0.27502018
Token:  immigrants ----  Logprob: -1.0898507
Token:  into ----  Logprob: -0.31180072
Token:  society ----  Logprob: -0.61141956
Token:  can ----  Logprob: -0.38526228
Token:  also ----  Logprob: -1.0982449
Token:  be ----  Logprob: -0.6851665
Token:  costly ----  Logprob: -0.8310476
Token: , ----  Logprob: -1.4440919
Token:  particularly ----  Logprob: -3.39978
Token:  when ----  Logprob: -0.8579604
Token:  it ----  Logprob: -0.5478005
```

```
Token:  comes ----  Logprob: -0.5686843
Token:  to ----  Logprob: -0.0013319061
Token:  language ----  Logprob: -0.25705123
Token:  education ----  Logprob: -1.1091584
Token:  and ----  Logprob: -0.02372535
Token:  other ----  Logprob: -1.1532717
Token:  social ----  Logprob: -1.8019323
Token:  programs ----  Logprob: -0.52098155
Token: . ----  Logprob: -0.10127911
Token:  While ----  Logprob: -9999.0
Token:  immigration ----  Logprob: -1.7121284
Token:  can ----  Logprob: -0.048475392
Token:  also ----  Logprob: -2.679265
Token:  bring ----  Logprob: -0.9964574
Token:  many ----  Logprob: -0.99818224
Token:  benefits ----  Logprob: -0.05329104
Token: , ----  Logprob: -0.7150445
Token:  like ----  Logprob: -3.3387198
Token:  cultural ----  Logprob: -0.3966686
Token:  diversity ----  Logprob: -0.030386928
Token:  and ----  Logprob: -0.2832347
Token:  economic ----  Logprob: -0.48002684
Token:  growth ----  Logprob: -0.08905833
Token: , ----  Logprob: -0.008792662
Token:  it ----  Logprob: -0.356082
Token:  can ----  Logprob: -0.5343373
Token:  be ----  Logprob: -1.8421142
Token:  challenging ----  Logprob: -2.0143442
Token:  when ----  Logprob: -1.5237825
Token:  the ----  Logprob: -0.4677243
Token:  number ----  Logprob: -0.8920042
Token:  of ----  Logprob: -0.005738933
Token:  incoming ----  Logprob: -2.9914937
Token:  people ----  Logprob: -2.9778142
Token:  surpass ----  Logprob: -2.3273916
Token: es ----  Logprob: -0.042548034
Token:  a ----  Logprob: -1.2116655
Token:  country ----  Logprob: -0.02842793
Token: 's ----  Logprob: -0.043256883
Token:  capacity ----  Logprob: -0.1862165
Token:  to ----  Logprob: -0.044327583
Token:  accommodate ----  Logprob: -1.1520244
```

```
Token:  and ----  Logprob: -0.8741836
Token:  support ----  Logprob: -0.51876277
Token:  them ----  Logprob: -0.0018480819
Token: . ----  Logprob: -0.32510057
Token:  Please ----  Logprob: -3.7028255
Token:  note ----  Logprob: -0.040074933
Token:  that ----  Logprob: -0.32630077
Token:  this ----  Logprob: -0.22103404
Token:  response ----  Logprob: -3.6340256
Token:  does ----  Logprob: -1.5489314
Token:  not ----  Logprob: -0.00078818674
Token:  represent ----  Logprob: -1.6779771
Token:  a ----  Logprob: -1.8859413
Token:  personal ----  Logprob: -0.102879904
Token:  opinion ----  Logprob: -0.745678
Token: , ----  Logprob: -0.50784135
Token:  it ----  Logprob: -2.4313693
Token:  merely ----  Logprob: -2.264873
Token:  explains ----  Logprob: -2.2406282
Token:  a ----  Logprob: -1.6679235
Token:  common ----  Logprob: -1.900521
Token:  viewpoint ----  Logprob: -1.6368958
Token: . ----  Logprob: -0.548294
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: