Discovering Interpretable Algorithms by Decompiling Transformers to RASP

Xinting Huang* Aleksandra Bakalova* Satwik Bhattamishra William Merrill Michael Hahn





What algorithms do Transformers internally implement?

Weiss, Gail, Yoav Goldberg, and Eran Yahav. "Thinking like transformers." International Conference on Machine Learning. PMLR, 2021.

Zhou, H., Bradley, A., Littwin, E., Razin, N., Saremi, O., Susskind, J. M., Bengio, S., and Nakkiran, P. What Algorithms can Transformers Learn? A Study in Length Generalization. In International Conference on Learning Representations, 2024.

Huang, X., Yang, A., Bhattamishra, S., Sarrof, Y., Krebs, A., Zhou, H., ... & Hahn, M. (2025, May). A formal framework for understanding length generalization in transformers. In International Conference on Learning Representations (Vol. 2025, pp. 58095-58179).

What algorithms do Transformers internally implement?

RASP(Weiss et al, 2021) – a symbolic language that mimics computations in transformers.

```
19  select_last = select(indices,length-1,==);
20  end_0 = aggregate(select_last,
21              bal1==0 and bal2==0);
22
23  shuffle_dyck2 = end_0 and not had_neg;
```

Weiss, Gail, Yoav Goldberg, and Eran Yahav. "Thinking like transformers." International Conference on Machine Learning. PMLR, 2021.

Zhou, H., Bradley, A., Littwin, E., Razin, N., Saremi, O., Susskind, J. M., Bengio, S., and Nakkiran, P. What Algorithms can Transformers Learn? A Study in Length Generalization. In International Conference on Learning Representations, 2024.

Huang, X., Yang, A., Bhattamishra, S., Sarrof, Y., Krebs, A., Zhou, H., ... & Hahn, M. (2025, May). A formal framework for understanding length generalization in transformers. In International Conference on Learning Representations (Vol. 2025, pp. 58095-58179).

What algorithms do Transformers internally implement?

RASP(Weiss et al, 2021) – a symbolic language that mimics computations in transformers.

```
19  select_last = select(indices,length-1,==);
20  end_0 = aggregate(select_last,
21              bal1==0 and bal2==0);
22
23  shuffle_dyck2 = end_0 and not had_neg;
```

Transformers length-generalize if trained on tasks which are expressible in (C-)RASP(Zhou et al., 2024; Huang et al., 2025).

Weiss, Gail, Yoav Goldberg, and Eran Yahav. "Thinking like transformers." International Conference on Machine Learning. PMLR, 2021.

Zhou, H., Bradley, A., Littwin, E., Razin, N., Saremi, O., Susskind, J. M., Bengio, S., and Nakkiran, P. What Algorithms can Transformers Learn? A Study in Length Generalization. In International Conference on Learning Representations, 2024.

Huang, X., Yang, A., Bhattamishra, S., Sarrof, Y., Krebs, A., Zhou, H., ... & Hahn, M. (2025, May). A formal framework for understanding length generalization in transformers. In International Conference on Learning Representations (Vol. 2025, pp. 58095-58179).

What algorithms do Transformers internally implement?

RASP(Weiss et al, 2021) – a symbolic language that mimics computations in transformers.

```
19  select_last = select(indices,length-1,==);
20  end_0 = aggregate(select_last,
21              bal1==0 and bal2==0);
22
23  shuffle_dyck2 = end_0 and not had_neg;
```

Transformers length-generalize if trained on tasks which are expressible in (C-)RASP(Zhou et al., 2024; Huang et al., 2025).

Length-generalizable transformers might internally implement RASP programs, but do they?

Weiss, Gail, Yoav Goldberg, and Eran Yahav. "Thinking like transformers." International Conference on Machine Learning. PMLR, 2021.

Zhou, H., Bradley, A., Littwin, E., Razin, N., Saremi, O., Susskind, J. M., Bengio, S., and Nakkiran, P. What Algorithms can Transformers Learn? A Study in Length Generalization. In International Conference on Learning Representations, 2024.

Huang, X., Yang, A., Bhattamishra, S., Sarrof, Y., Krebs, A., Zhou, H., ... & Hahn, M. (2025, May). A formal framework for understanding length generalization in transformers. In International Conference on Learning Representations (Vol. 2025, pp. 58095-58179).

Contributions

1. Method for automatically extracting short and interpretable RASP-like programs from trained Transformers.

To the best of our knowledge, this is the first method to convert naturally trained transformers to RASP.

Contributions

1. Method for automatically extracting short and interpretable RASP-like programs from trained Transformers.

To the best of our knowledge, this is the first method to convert naturally trained transformers to RASP.

2. Most direct evidence so far that transformers often internally implement interpretable RASP programs.

Decompilation Method

Main idea: **reparametrize and simplify.**

Programs are defined in D-RASP (Decompiled RASP).

D-RASP

1. $\alpha = \texttt{select}\left(\texttt{k}=v_k, \texttt{q}=v_q, \texttt{op}=\boldsymbol{A}\right)$ $\quad \alpha(i,s) = v_q(i)^\top \boldsymbol{A}\, v_k(s)$

2. $v = \texttt{aggregate}\left(\texttt{s}=\alpha_1 + \cdots + \alpha_p, \texttt{v}=w\right)$ $\quad v(i) = \sum_{j \le i} a_{i,j} w(j)$

$$a_{i,s} = \frac{\exp(\alpha(i,s)_1 + \cdots + \alpha(i,s)_p)}{\sum_{s' \le i} \exp(\alpha(i,s')_1 + \cdots + \alpha(i,s')_p)}$$

3. $v = \texttt{element\_wise\_op}(v_1, \ldots, v_s, \texttt{func}=f)$ $\quad v(i) = f(v_1(i), \ldots, v_s(i))$

4. $p = \texttt{project}(\texttt{inp}=v, \texttt{op}=\boldsymbol{A})$ $\quad p(j) = \boldsymbol{A} \cdot v(j)$

Decompilation Method

Step 1: Reparametrize

Theorem (informal). Given any GPT-2-style transformer satisfying Linear Layer Norm Assumption, we can rewrite it as a D-RASP program defining the same input-output map.

Decompilation Method

Step 1: Reparametrize

Neural network:

1 layer 1 head transformer

D-RASP:

Program for 1-layer 1-head model

$$\alpha_{1,h} = \texttt{select}(k = \texttt{pos}, q = \texttt{token}, op = \boldsymbol{E}^T \boldsymbol{Q}_{1,h}^T \boldsymbol{K}_{1,h} \boldsymbol{P})$$
$$\alpha'_{1,h} = \texttt{select}(k = \texttt{token}, q = \texttt{token}, op = \boldsymbol{E}^T \boldsymbol{Q}_{1,h}^T \boldsymbol{K}_{1,h} \boldsymbol{E})$$
$$\alpha''_{1,h} = \texttt{select}(k = \texttt{pos}, q = \texttt{pos}, op = \boldsymbol{P}^T \boldsymbol{Q}_{1,h}^T \boldsymbol{K}_{1,h} \boldsymbol{P})$$
$$\alpha'''_{1,h} = \texttt{select}(k = \texttt{token}, q = \texttt{pos}, op = \boldsymbol{P}^T \boldsymbol{Q}_{1,h}^T \boldsymbol{K}_{1,h} \boldsymbol{E})$$
$$v_{(pos,(1,h))} = \texttt{aggregate}(\alpha_{1,h} + \alpha'_{1,h} + \alpha''_{1,h} + \alpha'''_{1,h}, \texttt{pos})$$
$$v_{(tok,(1,h))} = \texttt{aggregate}(\alpha_{1,h} + \alpha'_{1,h} + \alpha''_{1,h} + \alpha'''_{1,h}, \texttt{token})$$
$$v_{(mlp_1,\langle\rangle)} = \texttt{element-wise-op}(\texttt{token}, \texttt{pos}, v_{(pos,(1,h))}, v_{(tok,(1,h))}, \texttt{func}{=}f_{MLP,1})$$
$$p_{token} = \texttt{project}(\texttt{token}, \ \texttt{op}{=}\boldsymbol{UE})$$
$$p_{pos} = \texttt{project}(\texttt{pos}, \ \texttt{op}{=}\boldsymbol{UP})$$
$$p_{(tok,(1,h))} = \texttt{project}(v_{(tok,(1,h))}, \ \texttt{op}{=}\boldsymbol{UV}_{1,h}\boldsymbol{E})$$
$$p_{(pos,(1,h))} = \texttt{project}(v_{(pos,(1,h))}, \ \texttt{op}{=}\boldsymbol{UV}_{1,h}\boldsymbol{P})$$
$$p_{(mlp_1,\langle\rangle)} = \texttt{project}(v_{(mlp_1,\langle\rangle)}, \ \texttt{op}{=}\boldsymbol{U})$$
$$\texttt{prediction} = \texttt{softmax}(p_{token} + p_{pos} + p_{(tok,(1,h))} + p_{(pos,(1,h))} + p_{(mlp_1,\langle\rangle)})$$

Decompilation Method

Step 1: Reparametrize

Neural network:

1 layer 1 head transformer

Program length is exponential in the number of model's layers

D-RASP:

Program for 1-layer 1-head model

$$\alpha_{1,h} = \texttt{select}(k = \texttt{pos}, q = \texttt{token}, op = \boldsymbol{E}^T \boldsymbol{Q}_{1,h}^T \boldsymbol{K}_{1,h} \boldsymbol{P})$$
$$\alpha'_{1,h} = \texttt{select}(k = \texttt{token}, q = \texttt{token}, op = \boldsymbol{E}^T \boldsymbol{Q}_{1,h}^T \boldsymbol{K}_{1,h} \boldsymbol{E})$$
$$\alpha''_{1,h} = \texttt{select}(k = \texttt{pos}, q = \texttt{pos}, op = \boldsymbol{P}^T \boldsymbol{Q}_{1,h}^T \boldsymbol{K}_{1,h} \boldsymbol{P})$$
$$\alpha'''_{1,h} = \texttt{select}(k = \texttt{token}, q = \texttt{pos}, op = \boldsymbol{P}^T \boldsymbol{Q}_{1,h}^T \boldsymbol{K}_{1,h} \boldsymbol{E})$$
$$v_{(pos,(1,h))} = \texttt{aggregate}(\alpha_{1,h} + \alpha'_{1,h} + \alpha''_{1,h} + \alpha'''_{1,h}, \texttt{pos})$$
$$v_{(tok,(1,h))} = \texttt{aggregate}(\alpha_{1,h} + \alpha'_{1,h} + \alpha''_{1,h} + \alpha'''_{1,h}, \texttt{token})$$
$$v_{(mlp_1,\langle\rangle)} = \texttt{element-wise-op}(\texttt{token}, \texttt{pos}, v_{(pos,(1,h))}, v_{(tok,(1,h))}, \texttt{func}{=}f_{MLP,1})$$
$$p_{token} = \texttt{project}(\texttt{token},\ \texttt{op}{=}\boldsymbol{UE})$$
$$p_{pos} = \texttt{project}(\texttt{pos},\ \texttt{op}{=}\boldsymbol{UP})$$
$$p_{(tok,(1,h))} = \texttt{project}\,(v_{(tok,(1,h))},\ \texttt{op}{=}\boldsymbol{UV}_{1,h}\boldsymbol{E})$$
$$p_{(pos,(1,h))} = \texttt{project}\,(v_{(pos,(1,h))},\ \texttt{op}{=}\boldsymbol{UV}_{1,h}\boldsymbol{P})$$
$$p_{(mlp_1,\langle\rangle)} = \texttt{project}\,(v_{(mlp_1,\langle\rangle)},\ \texttt{op}{=}\boldsymbol{U})$$
$$\texttt{prediction} = \texttt{softmax}\,(p_{token} + p_{pos} + p_{(tok,(1,h))} + p_{(pos,(1,h))} + p_{(mlp_1,\langle\rangle)})$$

Decompilation Method

Step 2.1: Simplify (pruning)

Neural network:

1 layer 1 head transformer

D-RASP:

Program for 1-layer 1-head model

$\alpha_{1,h} = \texttt{select}(k = \texttt{pos}, q = \texttt{token}, op = \boldsymbol{E}^T \boldsymbol{Q}_{1,h}^T \boldsymbol{K}_{1,h} \boldsymbol{P})$

$\alpha'_{1,h} = \texttt{select}(k = \texttt{token}, q = \texttt{token}, op = \boldsymbol{E}^T \boldsymbol{Q}_{1,h}^T \boldsymbol{K}_{1,h} \boldsymbol{E})$

$\alpha''_{1,h} = \texttt{select}(k = \texttt{pos}, q = \texttt{pos}, op = \boldsymbol{P}^T \boldsymbol{Q}_{1,h}^T \boldsymbol{K}_{1,h} \boldsymbol{P})$

$\alpha'''_{1,h} = \texttt{select}(k = \texttt{token}, q = \texttt{pos}, op = \boldsymbol{P}^T \boldsymbol{Q}_{1,h}^T \boldsymbol{K}_{1,h} \boldsymbol{E})$

$v_{\langle pos,\langle 1,h\rangle\rangle} = \texttt{aggregate}(\alpha_{1,h} + \alpha'_{1,h} + \alpha''_{1,h} + \alpha'''_{1,h}, \texttt{pos})$

$v_{\langle tok,\langle 1,h\rangle\rangle} = \texttt{aggregate}(\alpha_{1,h} + \alpha'_{1,h} + \alpha''_{1,h} + \alpha'''_{1,h}, \texttt{token})$

$v_{\langle mlp_1,\langle\rangle\rangle} = \texttt{element\_wise\_op}(\texttt{token}, \texttt{pos}, v_{\langle pos,\langle 1,h\rangle\rangle}, v_{\langle tok,\langle 1,h\rangle\rangle}, \texttt{func} = f_{MLP,1})$

$p_{\texttt{token}} = \texttt{project}(\texttt{token}, \texttt{op} = \boldsymbol{UE})$

$p_{\texttt{pos}} = \texttt{project}(\texttt{pos}, \texttt{op} = \boldsymbol{UP})$

$p_{\langle tok,\langle 1,h\rangle\rangle} = \texttt{project}(v_{\langle tok,\langle 1,h\rangle\rangle}, \texttt{op} = \boldsymbol{UV}_{1,h} \boldsymbol{E})$

$p_{\langle pos,\langle 1,h\rangle\rangle} = \texttt{project}(v_{\langle pos,\langle 1,h\rangle\rangle}, \texttt{op} = \boldsymbol{UV}_{1,h} \boldsymbol{P})$

$p_{\langle mlp_1,\langle\rangle\rangle} = \texttt{project}(v_{\langle mlp_1,\langle\rangle\rangle}, \texttt{op} = \boldsymbol{U})$

$\texttt{prediction} = \texttt{softmax}(p_{\texttt{token}} + p_{\texttt{pos}} + p_{\langle tok,\langle 1,h\rangle\rangle} + p_{\langle pos,\langle 1,h\rangle\rangle} + p_{\langle mlp_1,\langle\rangle\rangle})$

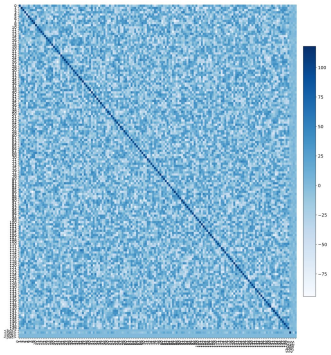
Decompilation Method

Step 2.2: Simplify (primitive matching)

$$\ldots$$

$$a = \texttt{select}(\texttt{q=token}, \texttt{k=}v_{\langle tok, \langle (1,1) \rangle \rangle}, \texttt{op=}\boldsymbol{E}^T Q_{2,1}^T K_{2,1} \boldsymbol{V}_{1,1} \boldsymbol{E})$$

$$\ldots$$



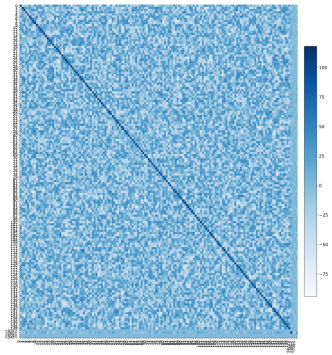
Decompilation Method

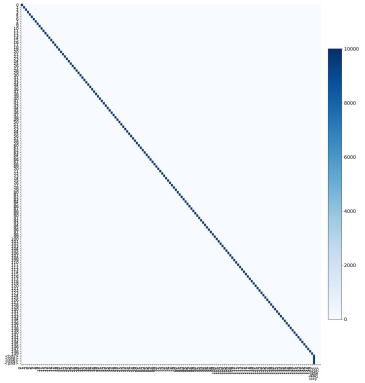
Step 2.2: Simplify (primitive matching)

...

$a = \texttt{select}(\texttt{q=token}, \texttt{k=}v_{\langle tok, \langle (1,1) \rangle \rangle}, \texttt{op=}\boldsymbol{E}^T Q_{2,1}^T K_{2,1} \boldsymbol{V}_{1,1} \boldsymbol{E})$

...





(b) Line 3: op=(k==q), special_op=(k==BOS)

Decompilation Method

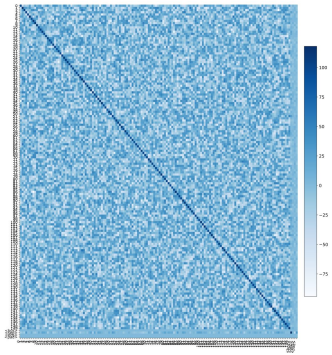
Step 2.2: Simplify (primitive matching)

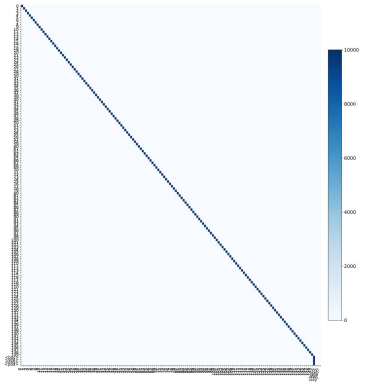
...

$a = \texttt{select}(\texttt{q=token}, \texttt{k=}v_{\langle tok, \langle (1,1) \rangle \rangle}, \texttt{op=}\boldsymbol{E}^T Q_{2,1}^T K_{2,1} \boldsymbol{V}_{1,1} \boldsymbol{E})$

...

Replace





(b) Line 3: op=(k==q), special_op=(k==BOS)

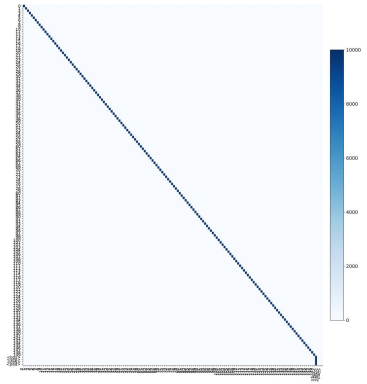
Decompilation Method

Step 2.2: Simplify (primitive matching)

...

$a = \texttt{select}(\texttt{q=token}, \texttt{k=}v_{\langle tok, \langle (1,1) \rangle \rangle}, \texttt{op=}\boldsymbol{E}^T Q_{2,1}^T K_{2,1} \boldsymbol{V}_{1,1} \boldsymbol{E})$

...



(b) Line 3: op=(k==q), special_op=(k==BOS)

Example

Decompiled program for MOST FREQUENT

A program extracted from a 1-layer 1-head transformer trained on Most Frequent task

Decompiled program for MOST FREQUENT

<bos> o b r o <sep>

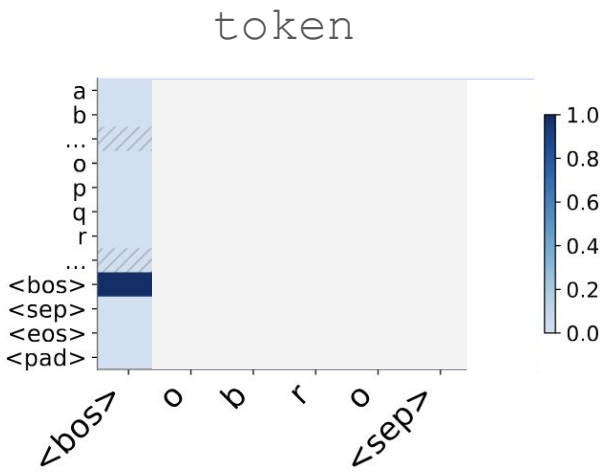
Decompiled program for MOST FREQUENT

<bos> o b r o <sep>

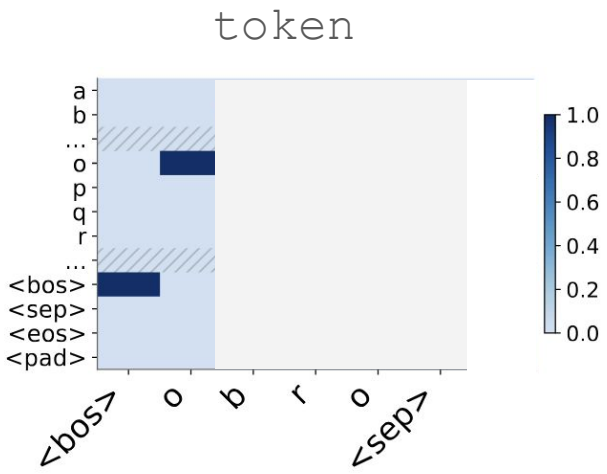
Decompiled program for MOST FREQUENT

<bos> o b r o <sep> o

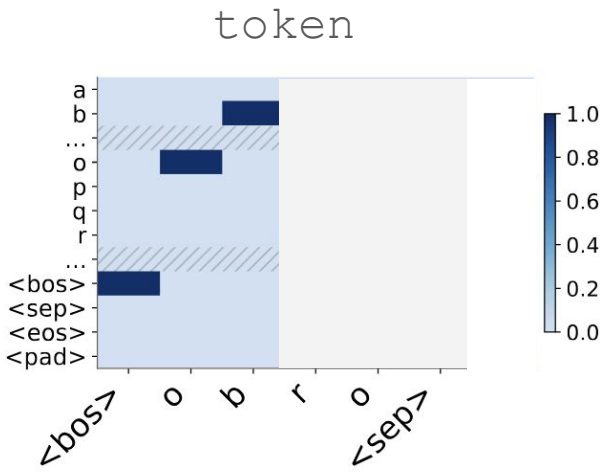
Decompiled program for MOST FREQUENT



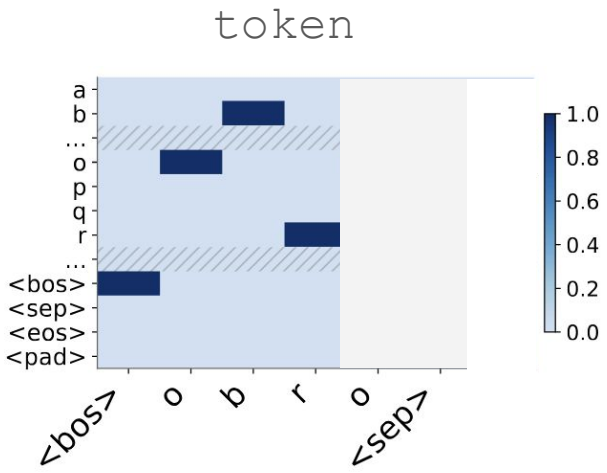
Decompiled program for MOST FREQUENT



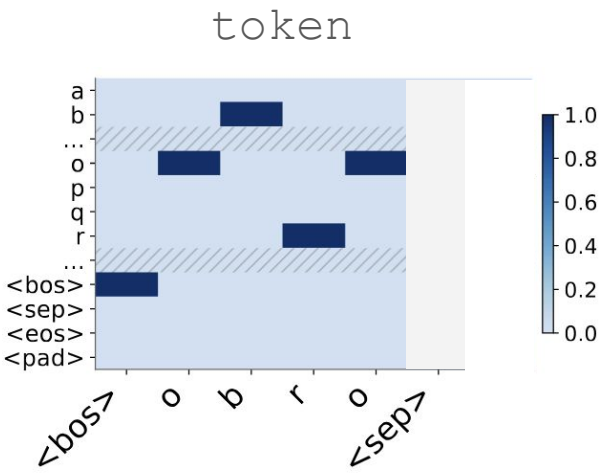
Decompiled program for MOST FREQUENT



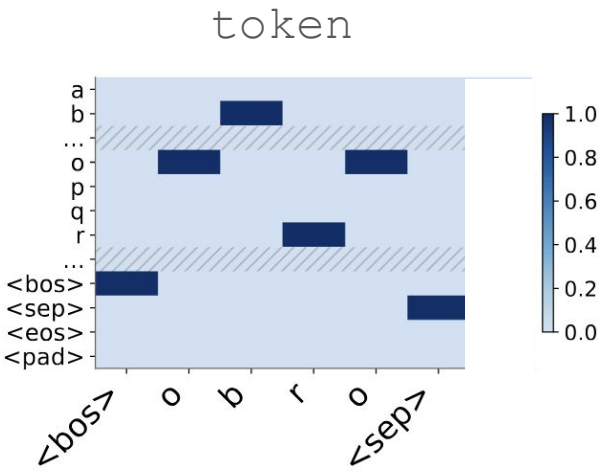
Decompiled program for MOST FREQUENT



Decompiled program for MOST FREQUENT

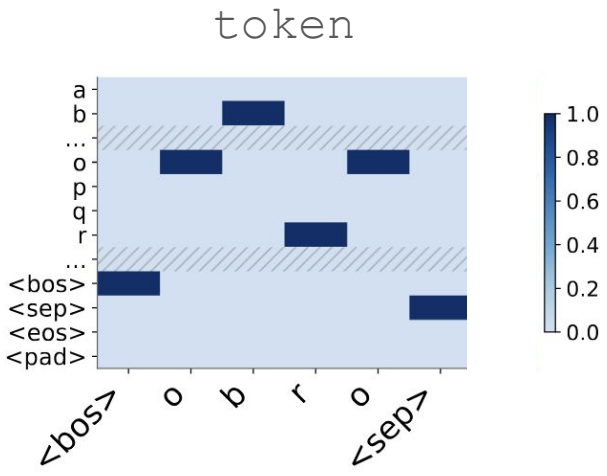


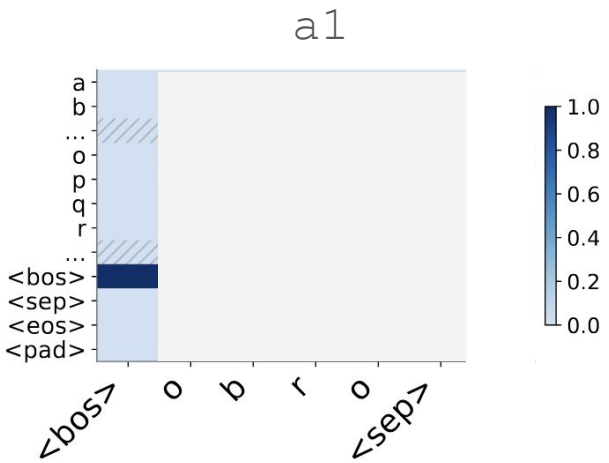
Decompiled program for MOST FREQUENT



Decompiled program for MOST FREQUENT

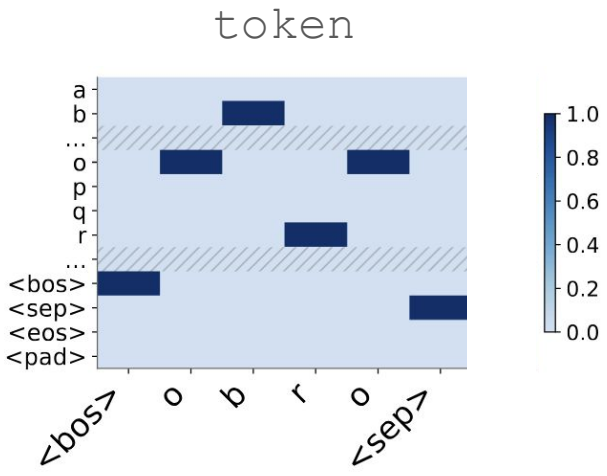
```
1. a1 = aggregate(s=[], v=token)
```

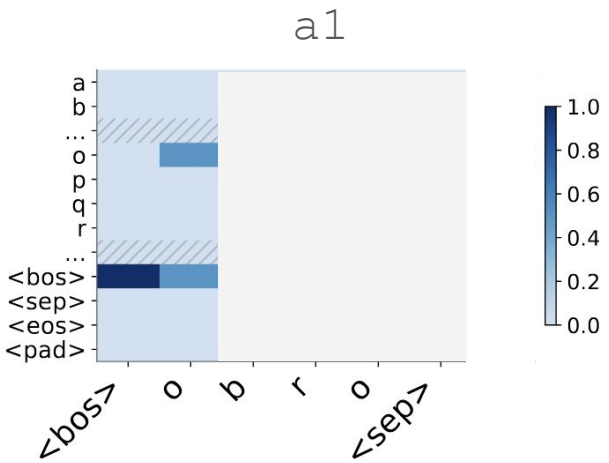




Decompiled program for MOST FREQUENT

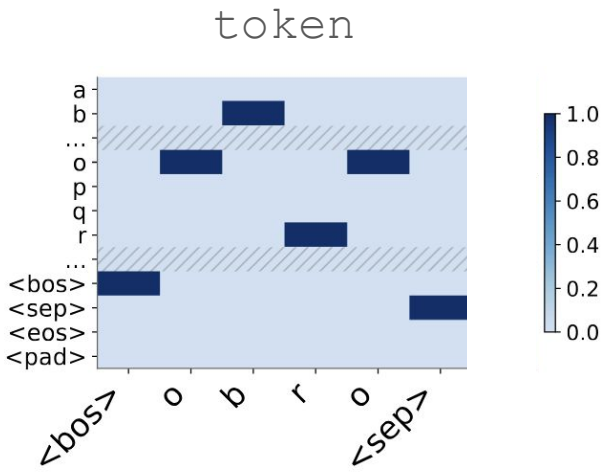
```
1. a1 = aggregate(s=[], v=token)
```

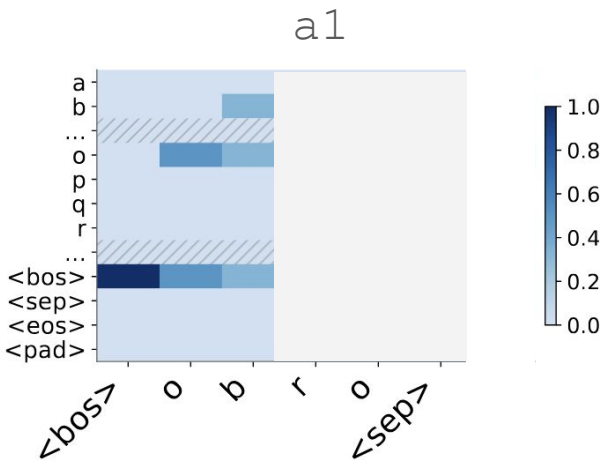




Decompiled program for MOST FREQUENT

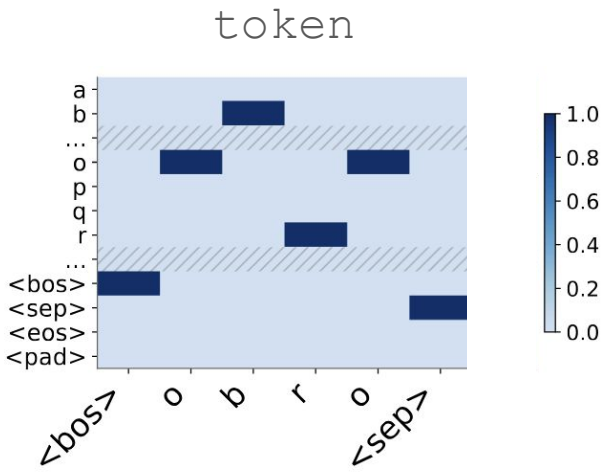
```
1. a1 = aggregate(s=[], v=token)
```

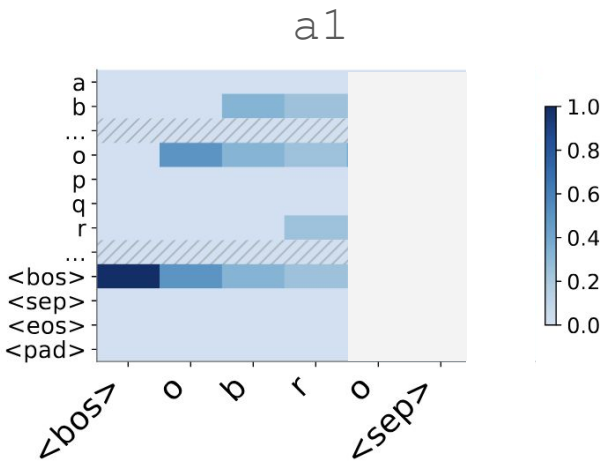




Decompiled program for MOST FREQUENT

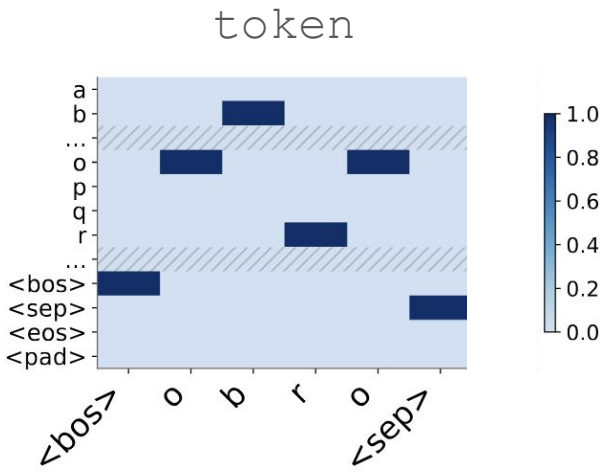
```
1. a1 = aggregate(s=[], v=token)
```

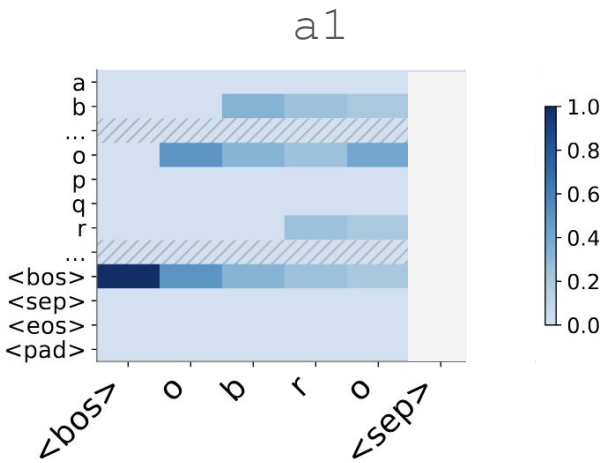




Decompiled program for MOST FREQUENT

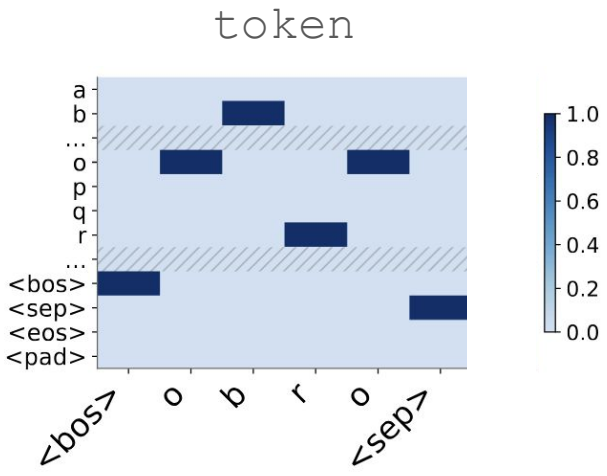
```
1. a1 = aggregate(s=[], v=token)
```

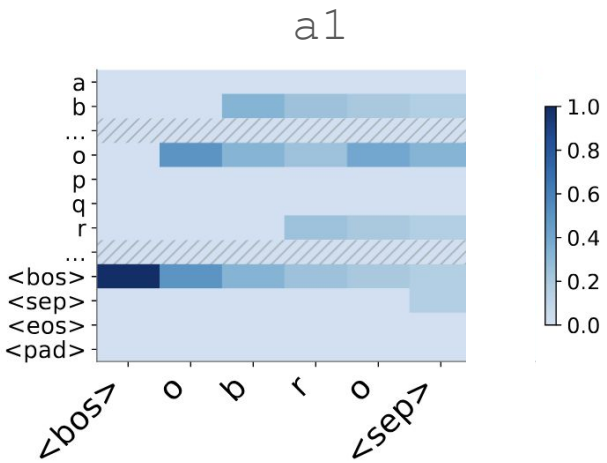




Decompiled program for MOST FREQUENT

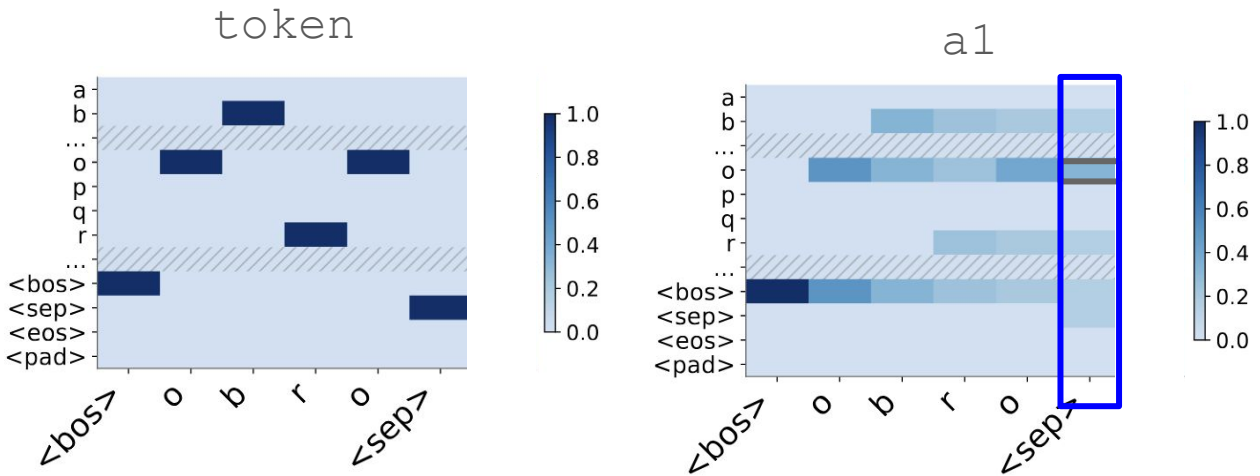
```
1. a1 = aggregate(s=[], v=token)
```





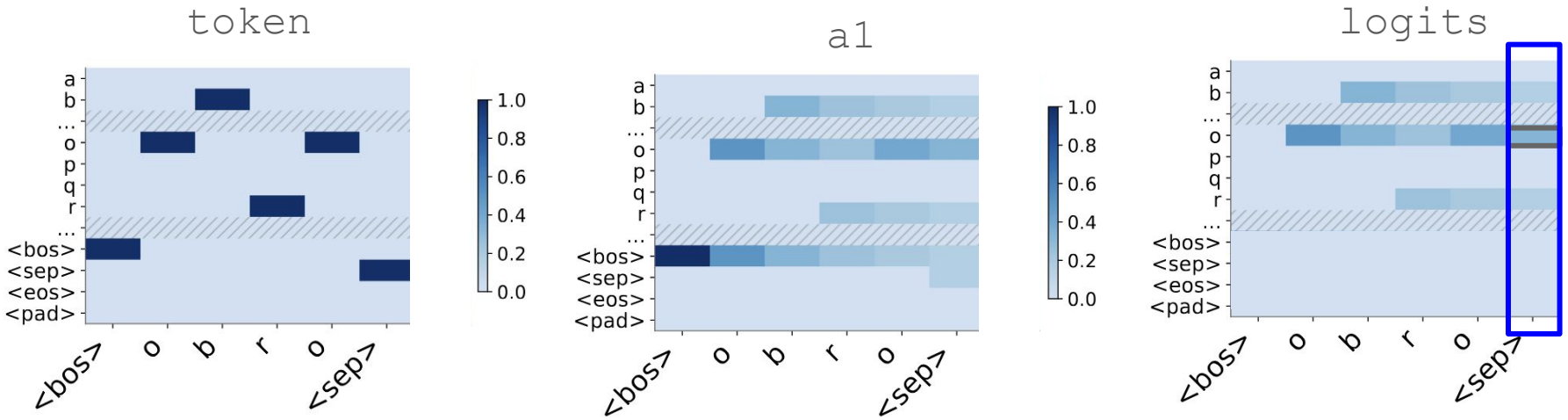
Decompiled program for MOST FREQUENT

```
1. a1 = aggregate(s=[], v=token)
```



Decompiled program for MOST FREQUENT

```
1. a1 = aggregate(s=[], v=token)
2. logits1 = project(inp=a1, op=(inp==out),
        special_op=(uniform selection))
3. prediction = softmax(logits1)
```



We present more decompiled programs in the paper

Task	Program
Binary Majority	Sec. J.1
Binary Majority Interleave	Sec. J.2 J.3
Count	Sec. J.4 J.5
Most Frequent	Sec. J.6 J.7
Sort	Sec. J.8
Unique Bigram Copy	Sec. J.9 J.10
Unique Copy	Sec. J.11
Unique Reverse	Sec. J.12 J.13
D_{12}	Sec. K.1
D_2	Sec. K.2
D_4	Sec. K.3
$(aa)^*$	Sec. K.4
$aa^*bb^*cc^*dd^*ee^*$	Sec. K.5
$\{a,b\}^*d\{b,c\}^*$	Sec. K.6
Tomita1	Sec. K.7
Tomita2	Sec. K.8
Tomita7	Sec. K.9

Applicable only:

- for length-generalizable models that actually internally implement RASP algorithms

Applicable only:

- for length-generalizable models that actually internally implement RASP algorithms
- when layer norm can be linearized, however, empirically this is true for length-generalizing models

What about LLMs?

- LLMs as a whole likely cannot be decompiled into a short RASP program

What about LLMs?

- LLMs as a whole likely cannot be decompiled into a short RASP program
- However, they may implement short RASP subprograms

What about LLMs?

- LLMs as a whole likely cannot be decompiled into a short RASP program
- However, they may implement short RASP subprograms
- Future work!

Conclusion

Method for extracting simple (D-)RASP programs from trained transformers.

Conclusion

Method for extracting simple (D-)RASP programs from trained transformers.

Show that transformers trained on tasks expressible in (C-)RASP often implement interpretable (D-)RASP programs.

Thank you!