elicsir
Elevating Italian Computer Science Innovation and Research

Thinking like Transformers

Giulia Beraldo

Mentor: Prof. Barbara Di Camillo

16.05.2026, Fourth Orthogonal Weekend, Procida

elicsir
Elevating Italian Computer Science Innovation and Research

What is a computational model behind a transformer?

Thinking Like Transformers

Gail Weiss [1] Yoav Goldberg [2 3] Eran Yahav [1]

Abstract

What is the computational model behind a Transformer? Where recurrent neural networks have direct parallels in finite state machines, allowing clear discussion and thought around architecture variants or trained models, Transformers have no such familiar parallel. In this paper we aim to change that, proposing a computational model for the transformer-encoder in the form of a programming language. We map the basic components of a transformer-encoder—attention and feed-forward computation—into simple primitives, around which we form a programming language: the Restricted Access Sequence Processing Language (RASP). We show how RASP can be used to program solutions to tasks that could conceivably be learned by a Transformer, and how a Transformer can be trained to mimic a RASP solution. In particular, we provide RASP programs for histograms, sorting, and Dyck-languages. We

a transformer operates at a higher-level of abstraction, reasoning in terms of a composition of *sequence operations* rather than neural network primitives.

We are inspired by the use of automata as an abstract computational model for recurrent neural networks (RNNs). Using automata as an abstraction for RNNs has enabled a long line of work, including extraction of automata from RNNs (Omlin & Giles, 1996; Weiss et al., 2018b; Ayache et al., 2018), analysis of RNNs' practical expressive power in terms of automata (Weiss et al., 2018a; Rabusseau et al., 2019; Merrill, 2019; Merrill et al., 2020b), and even augmentations based on automata variants (Joulin & Mikolov, 2015). Previous work on transformers explores their computational power, but does not provide a computational model (Yun et al., 2020; Hahn, 2020; Pérez et al., 2021).

Thinking in terms of the RASP model can help derive computational results. Bhattamishra et al. (2020) and Ebrahimi et al. (2020) explore the ability of transformers to recognize Dyck-k languages, with Bhattamishra et al. providing a

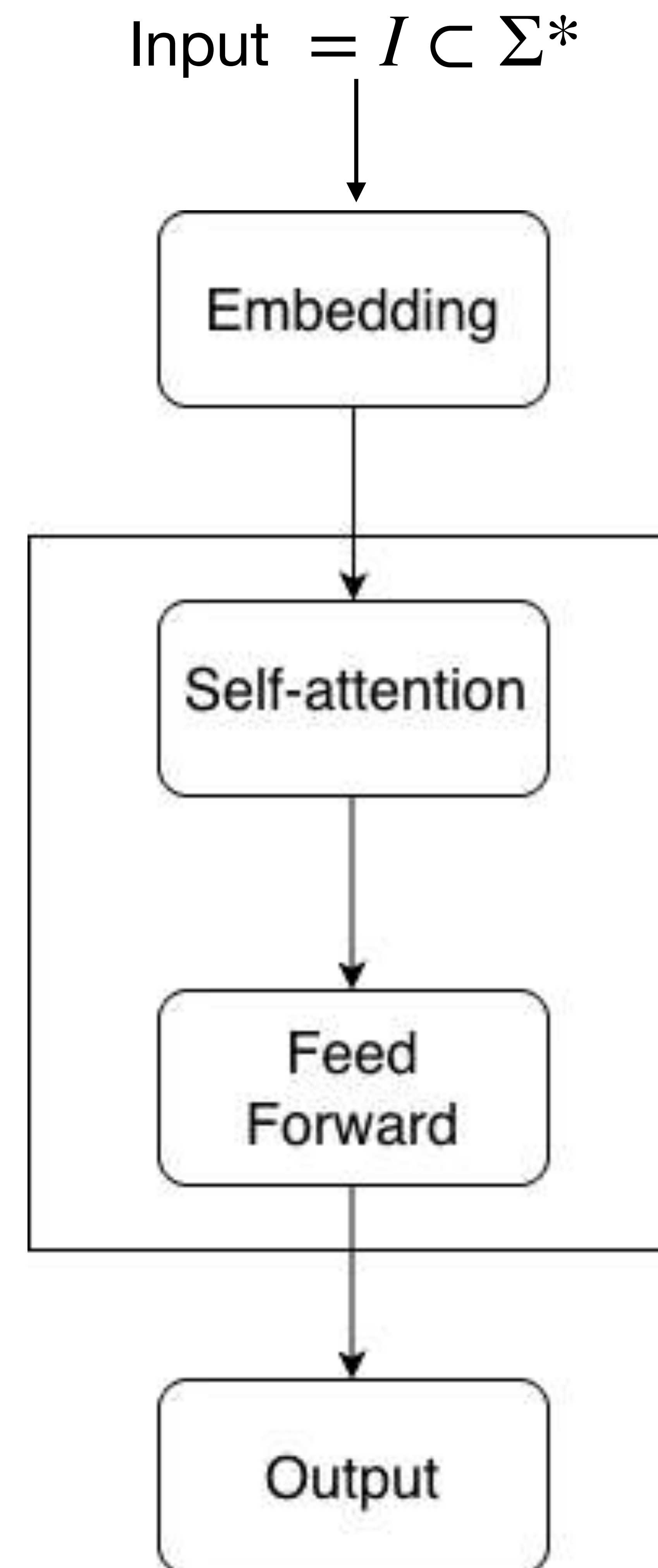
ICML, 2021

Summary

- The Transformer model
- RASP: the computational model
- Example: sorting
- Beyond RASP

A simplified model for Transformer

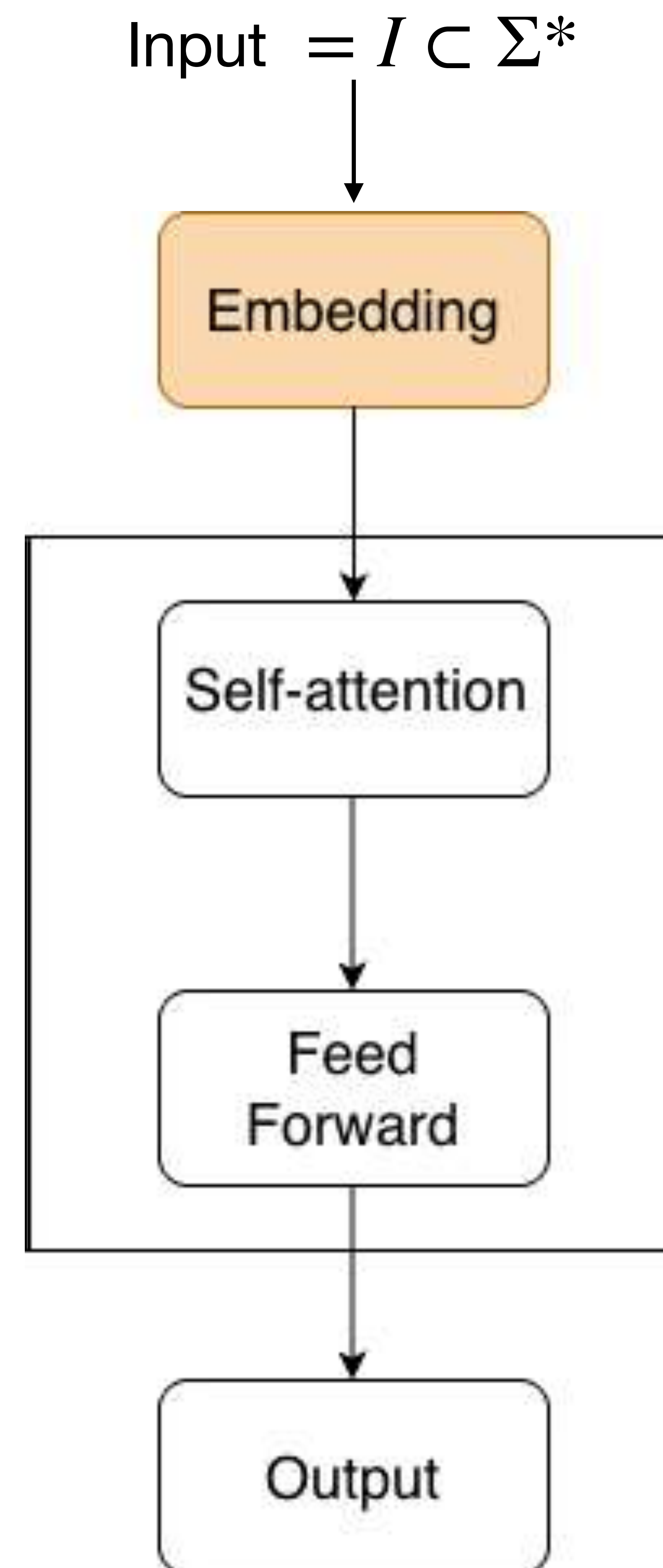
Input $= I \subset \Sigma^*$



A simplified model for Transformer

Embedding

Input $= I \subset \Sigma^*$ $|\text{input}| = n$



$\forall w \in I \qquad w \mapsto x \in \mathbb{R}^d$

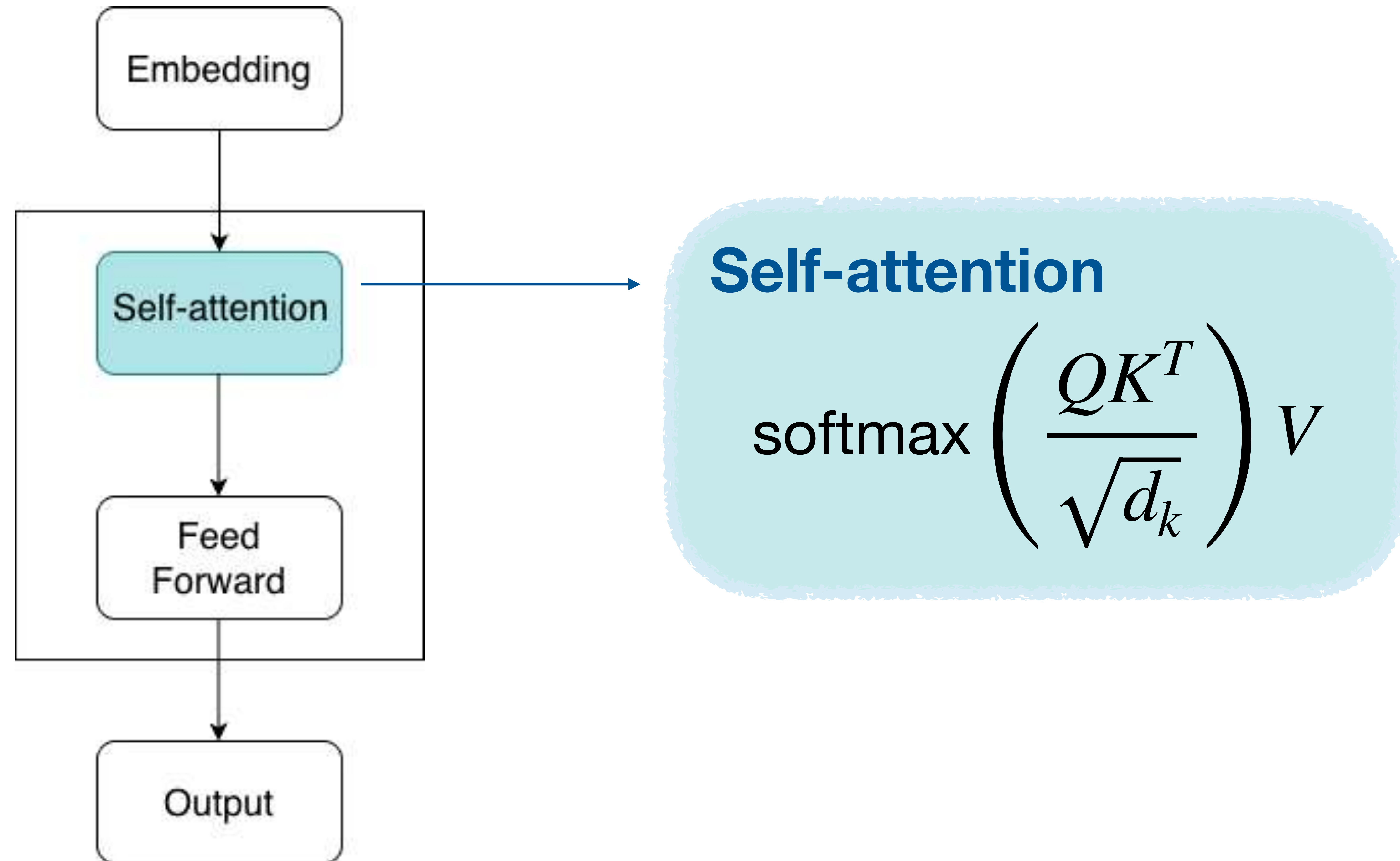
$X \in \mathbb{R}^{n \times d}$

$x \mapsto$

Query	$Q = XW_Q$	$W_Q \in \mathbb{R}^{d \times d_k}$
Key	$K = XW_K$	$W_K \in \mathbb{R}^{d \times d_k}$
Value	$V = XW_V$	$W_V \in \mathbb{R}^{d \times d_k}$

A simplified model for Transformer

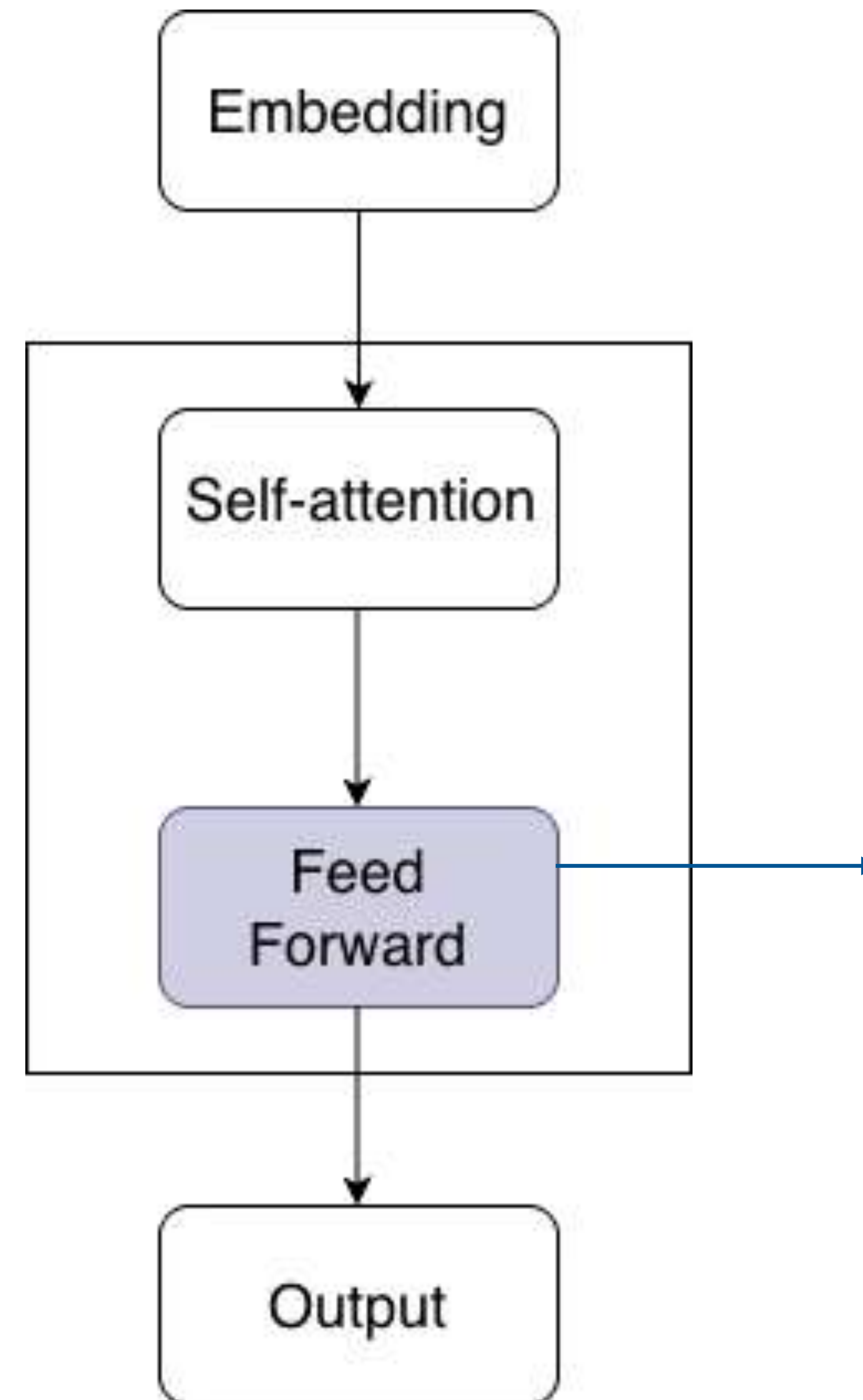
Self attention



elicsir
Elevating Italian Computer Science Innovation and Research

A simplified model for Transformer

Feed Forward Layer



Feed-forward network

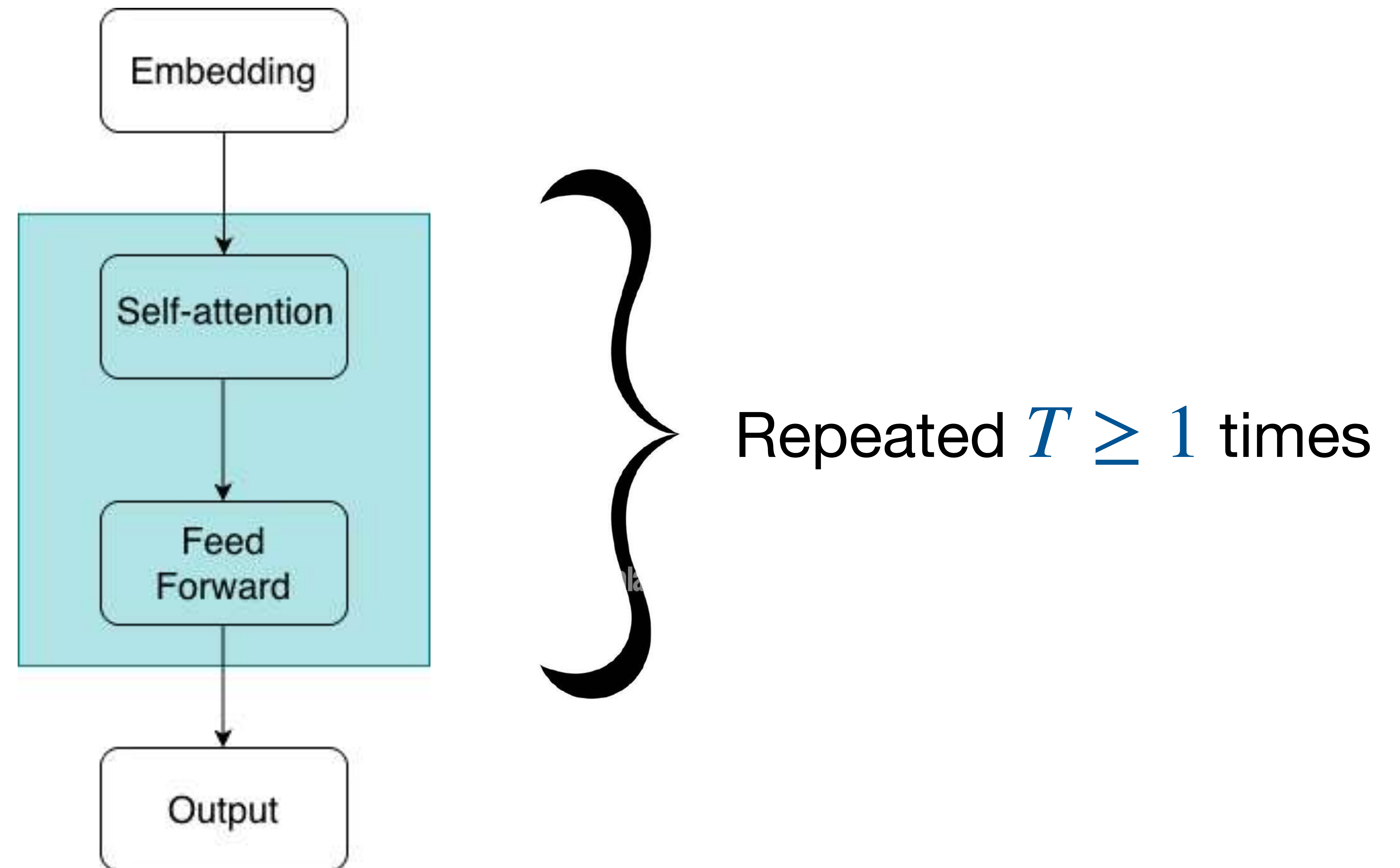
$$FFN(s) = W_1\sigma(W_2 s + b_1) + b_2$$

$$W_1 \in \mathbb{R}^{d_{ff} \times d},\ b_2 \in \mathbb{R}^{d \times d_{ff}}$$

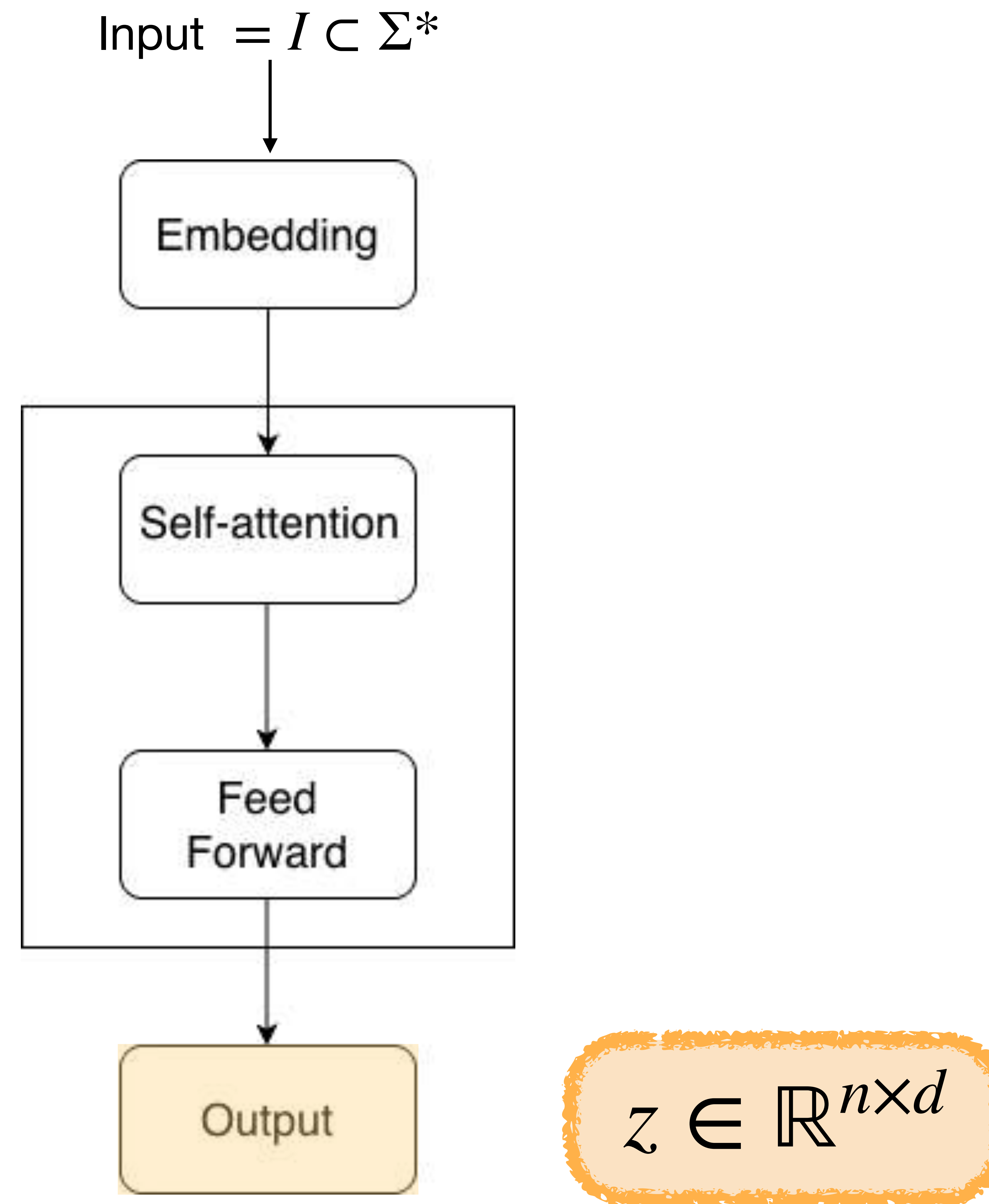
$$b_1 \in \mathbb{R}^{d_{ff}},\ b_2 \in \mathbb{R}^{d}$$

A simplified model for Transformer

Multiple Layers



A simplified model for Transformer



$z \in \mathbb{R}^{n \times d}$

elicsir
Elevating Italian Computer Science Innovation and Research

What is the computational model for a Transformer?

Thinking Like Transformers

Gail Weiss [1] Yoav Goldberg [2 3] Eran Yahav [1]

Abstract

What is the computational model behind a Transformer? Where recurrent neural networks have direct parallels in finite state machines, allowing clear discussion and thought around architecture variants or trained models, Transformers have no such familiar parallel. In this paper we aim to change that, proposing a computational model for the transformer-encoder in the form of a programming language. We map the basic components of a transformer-encoder—attention and feed-forward computation—into simple primitives, around which we form a programming language: the Restricted Access Sequence Processing Language (RASP). We show how RASP can be used to program solutions to tasks that could conceivably be learned by a Transformer, and how a Transformer can be trained to mimic a RASP solution. In particular, we provide RASP programs for histograms, sorting, and Dyck-languages. We

a transformer operates at a higher-level of abstraction, reasoning in terms of a composition of *sequence operations* rather than neural network primitives.

We are inspired by the use of automata as an abstract computational model for recurrent neural networks (RNNs). Using automata as an abstraction for RNNs has enabled a long line of work, including extraction of automata from RNNs (Omlin & Giles, 1996; Weiss et al., 2018b; Ayache et al., 2018), analysis of RNNs' practical expressive power in terms of automata (Weiss et al., 2018a; Rabusseau et al., 2019; Merrill, 2019; Merrill et al., 2020b), and even augmentations based on automata variants (Joulin & Mikolov, 2015). Previous work on transformers explores their computational power, but does not provide a computational model (Yun et al., 2020; Hahn, 2020; Pérez et al., 2021).

Thinking in terms of the RASP model can help derive computational results. Bhattamishra et al. (2020) and Ebrahimi et al. (2020) explore the ability of transformers to recognize Dyck-k languages, with Bhattamishra et al. providing a

elicsir
Elevating Italian Computer Science Innovation and Research

[Q] What is a **computational model** ?

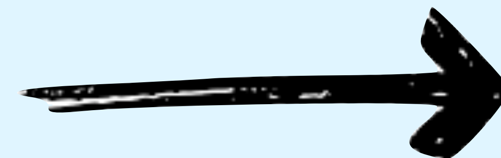
"A computational model is a **formal abstraction of computation**, used to:

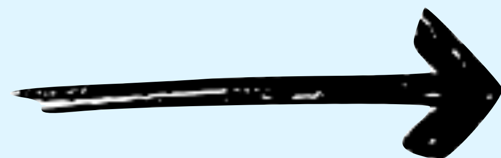
1. Reason about **what can be computed** and **with what resources**.
2. Which architectural mechanisms coincide with **computational primitives**"

Some examples of computational models

elicsir
Elevating Italian Computer Science Innovation and Research

Classical Computer **Turing machine**

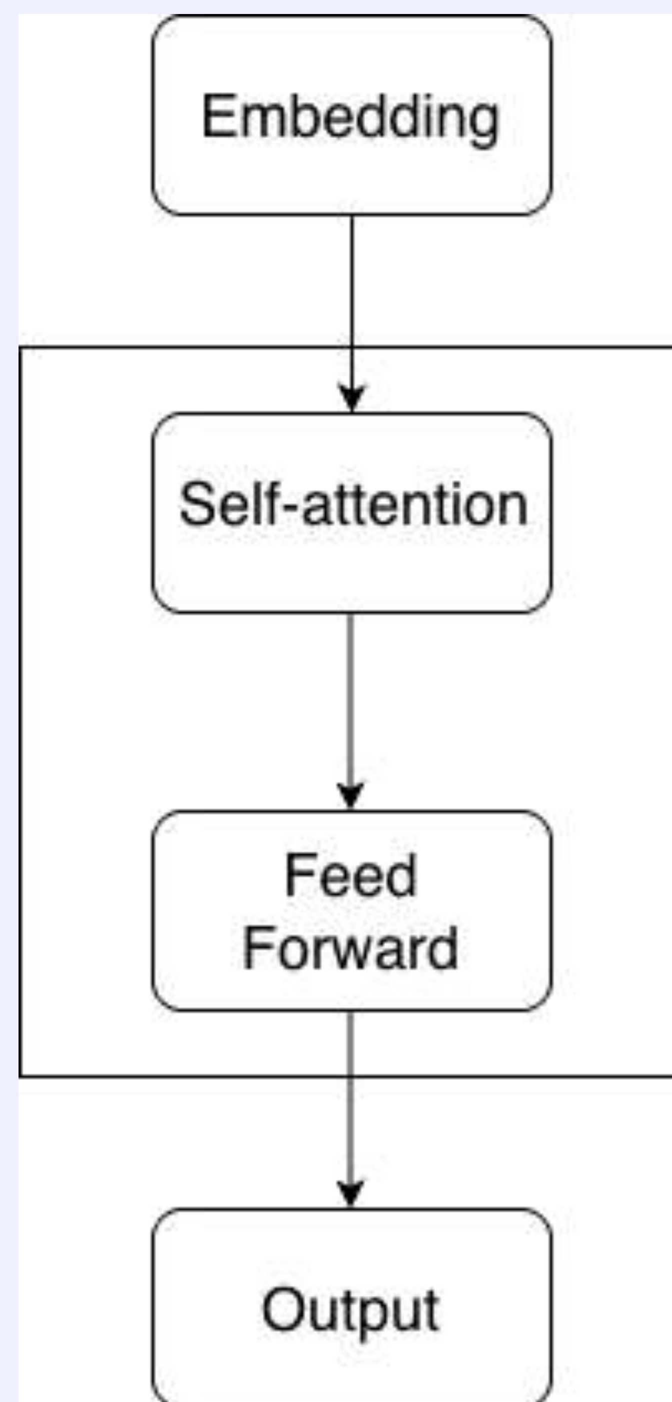
Recurrent Neural Network  **Finite state automata**

Convolutional Neural Network  **Localized Boolean Circuit**

… and more!

Q What is the goal of formalizing a **computational model** for a transformer?

Neural architecture



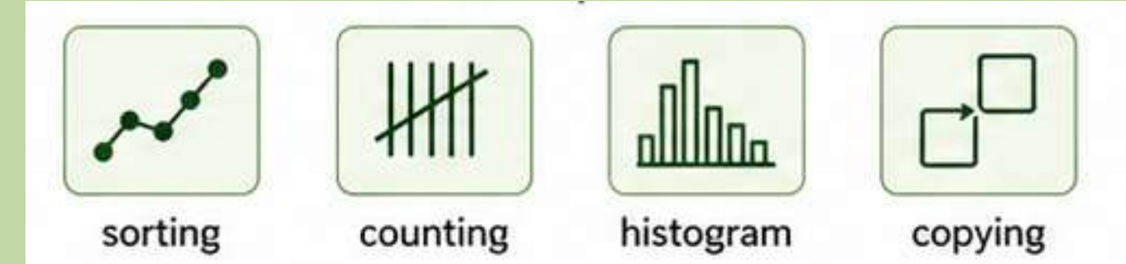
Formal abstraction

```
def sort(vals,keys,assume_bos=False) {
    smaller = select(keys,keys,<) or
        (select(keys,keys,==) and
         select(indices,indices,<) );
    num_smaller =
        selector_width(smaller,
            assume_bos=assume_bos);
    target_pos = num_smaller if
                    not assume_bos else
    (0 if indices==0 else (num_smaller+1));
    sel_new =
        select(target_pos,indices,==);
    sort = aggregate(sel_new,vals);
}
```

Human-readable sequence program

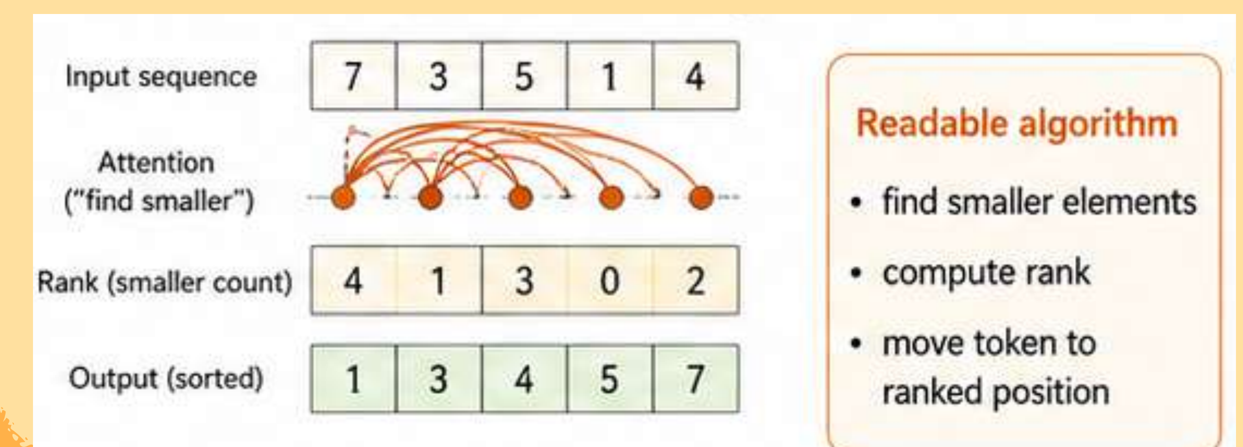
Expressivity analysis

Which functions can be expressed?



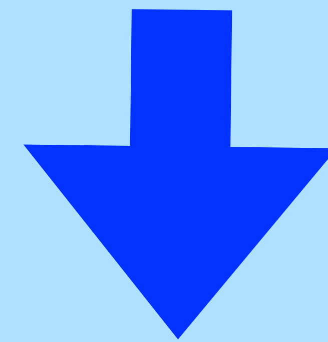
Mechanistic Interpretability

How does the model compute it?

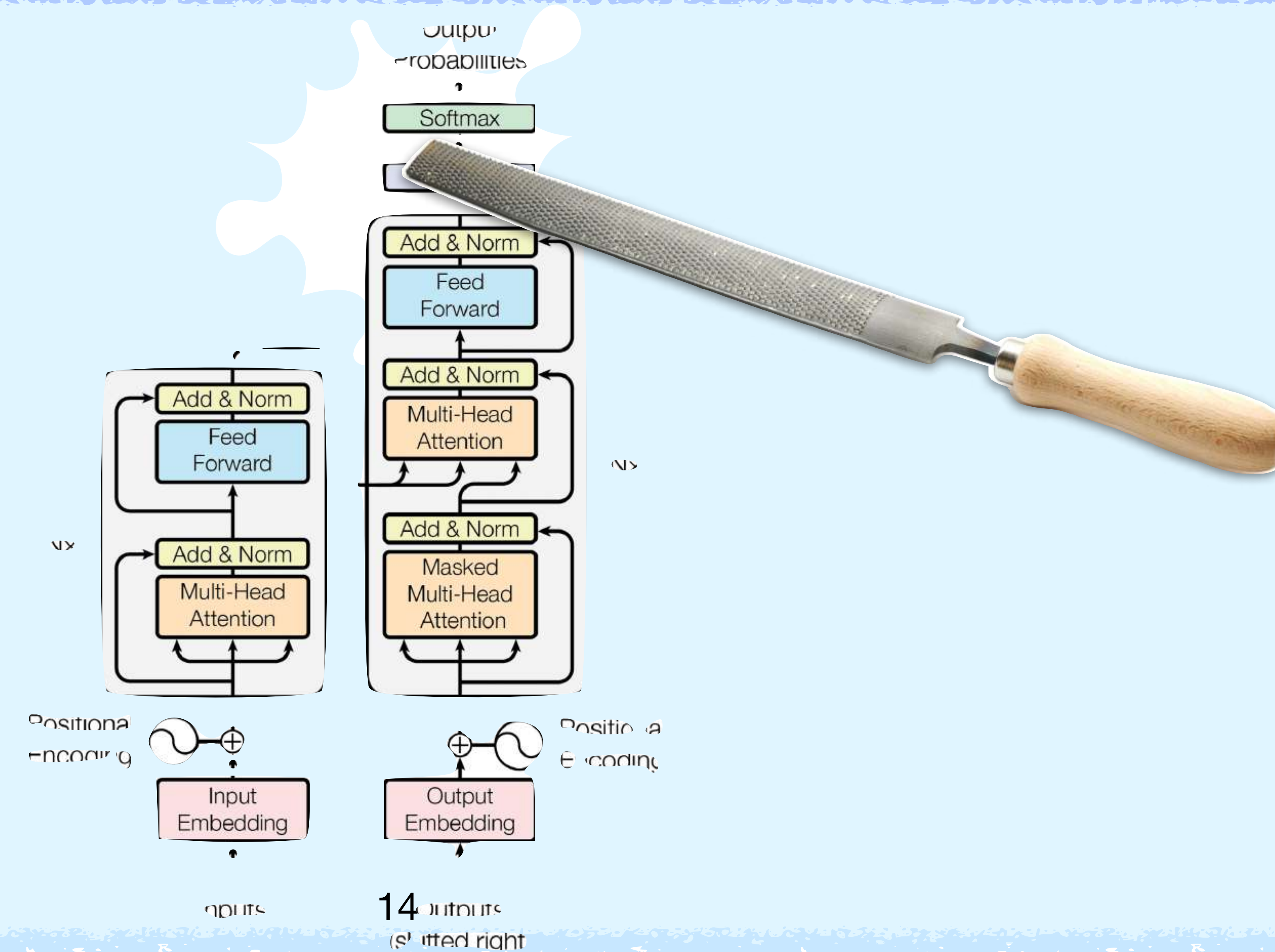


What is a computational model for a transformer?

Restricted Access Sequence Processing Language



RASP



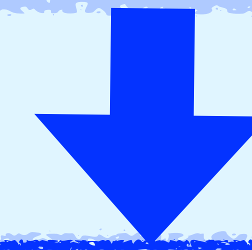
RASP

Def RASP is an **abstraction** of a **programming language**

RASP

elicsir
Elevating Italian Computer Science Innovation and Research

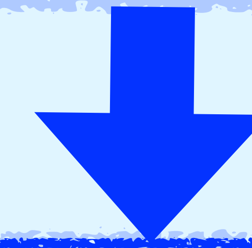
Def RASP is an **abstraction** of a **programming language**



set of instructions that enables humans to communicate with computers to turn "ideas" into instructions computers can understand and execute

RASP

Def RASP is an **abstraction** of a **programming language**



set of instructions that enables humans to communicate with computers to turn “ideas” into instructions computers can understand and execute

RASP is **NOT** a language we directly execute

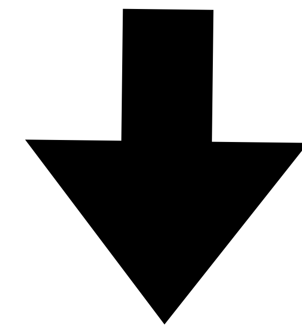
RASP —The computational model

[Input]

Tokens $\equiv x \in \mathbb{R}^n$

Indices $\equiv [0,\ldots,n-1]$

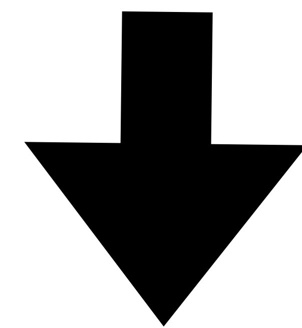
} sequences



RASP program

$y \in \mathbb{R}^n$

$A \in \{T, F\}^{n \times n}$



[Output]

$z \in \mathbb{R}^n$

Relation to Transformer

How do the RASP operations compile to describe the information flow of a transformer architecture?

Embedding

$tokens \quad [a_1, a_2, \ldots, a_n] \quad \approx \quad x \in \mathbb{R}^{n \times d}$

$indices \quad [0, 1, \ldots, n] \quad \approx$ positional encoding

elicsir
Elevating Italian Computer Science Innovation and Research

The computational model

Sequences can be transformed into other sequences using:

Select

Aggregate

Element-wise operations

The computational model

Select

$$select : \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{P}_{\{0,1\}} \rightarrow \{T, F\}^{n \times n}$$

$$(k, q, p) \mapsto S \qquad k, q \in \mathbb{R}^n,\ p \in \mathbb{P}_{\{0,1\}},\ S \in \{T, F\}^{n \times n}$$

$$select(k, q, p) = S : \quad S[i, j] = p(q_i, k_j) \qquad i, j \in [n]$$

The computational model

Select

$$select : \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{P}_{\{0,1\}} \rightarrow \{T, F\}^{n \times n}$$

$$(k, q, p) \mapsto S \qquad k, q \in \mathbb{R}^n,\ p \in \mathbb{P}_{\{0,1\}},\ S \in \{T, F\}^{n \times n}$$

$$select(k, q, p) = S : \quad S[i, j] = p(q_i, k_j) \qquad i, j \in [n]$$

$$select([0,1,2], [1,2,3], <) = \begin{array}{c} \\ 0 \\ 1 \\ 2 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} T & T & T \\ F & T & T \\ F & F & T \end{bmatrix} \end{array}$$

The computational model

Aggregate

$$aggregate : \mathbb{R}^{n\times n} \times \mathbb{R}^n \to \mathbb{R}^n$$

$$(S, v) \mapsto s \qquad S \in \mathbb{R}^{n\times n},\ v \in \mathbb{R}^n,\ s \in \mathbb{R}^n$$

$$aggregate_row(S, v) = s : \quad s[i] = row(v[j])\ \forall j \text{ where } S[i][j] = T$$

The computational model

Aggregate

$$aggregate : \mathbb{R}^{n \times n} \times \mathbb{R}^n \rightarrow \mathbb{R}^n$$

$$(S, v) \mapsto s \qquad S \in \mathbb{R}^{n \times n},\ v \in \mathbb{R}^n,\ s \in \mathbb{R}^n$$

$$aggregate_row(S, v) = s : \quad s[i] = row(v[j])\ \forall j \text{ where } S[i][j] = T$$

$$aggregate_avg(\begin{bmatrix} \mathbf{T} & F & F \\ \mathbf{T} & \mathbf{T} & F \\ \mathbf{T} & \mathbf{T} & \mathbf{T} \end{bmatrix}, [10,20,30]) = [\frac{10}{1}, \frac{10+20}{2}, \frac{10+20+30}{3}] = [10,15,20]$$

The computational model

A particular case of aggregate

selector_width

$$selector \mapsto s-op \quad \equiv aggregate_count : (S, \mathbf{1}_n) \mapsto t \in \mathbb{D}^n$$

$$S(i, :) \mapsto \sum_{j=1}^{n} \mathbf{1}_{S(i,j)=T} \qquad \forall i \in [n]$$

selector_width($\begin{bmatrix} \mathrm{T} & F & F \\ \mathrm{T} & \mathrm{T} & F \\ \mathrm{T} & \mathrm{T} & \mathrm{T} \end{bmatrix}$) $= [1,2,3]$

elicsir
Elevating Italian Computer Science Innovation and Research

Relation to Transformer

How do the RASP operations compile to describe the information flow of a transformer architecture?

Self-attention

$$Aggregate_row(Select(\cdot\,,\ \cdot\,, p),\ \ s-op) \approx \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

The computational model

Element-wise operations

Given $s = [s_0, \ldots, s_{n-1}], t = [t_0, \ldots, t_{n-1}] \in \mathbb{D}^n$ an element-wise operator

$f : \mathbb{D} \times \mathbb{D} \to \mathbb{D}$ such that $f(s, t)_i = f(s_i, t_i) \quad \forall i \in [n]$

The computational model

Element-wise operations

Given $s = [s_0, \ldots, s_{n-1}], t = [t_0, \ldots, t_{n-1}] \in \mathbb{D}^n$ an element-wise operator

$f : \mathbb{D}^n \times \mathbb{D}^n \to \mathbb{D}^n$ such that $f(s, t)_i = f(s_i, t_i) \quad \forall i \in [n]$

$$s = [1,2,3] \qquad t = [4,5,6] \qquad \in \mathbb{Z}^3$$

$$s + t = [s_0 + t_0, s_1 + t_1, s_2 + t_2] = [1 + 4,\ 2 + 5,\ 3 + 6] = [5,7,9]$$

$$s + k = [s_0 + k,\ s_1 + k,\ s_2 + k] = [1 + k,\ 2 + k,\ 3 + k] \qquad k \in \mathbb{Z}$$

Relation to Transformer

How do the RASP operations compile to describe the information flow of a transformer architecture?

Element-wise operations

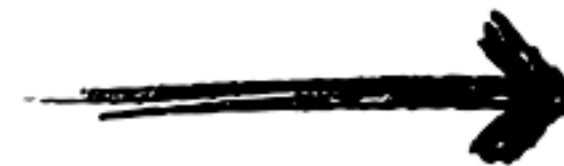
$$s + t = [1 + 4,\ 2 + 5,\ 3 + 6] = [5,7,9] \quad \approx \quad W \begin{bmatrix} s \\ t \end{bmatrix} \quad W \in \mathbb{R}^{3\times 6}$$

elicsir
Elevating Italian Computer Science Innovation and Research

Relation to Transformer

How do the RASP operations compile to describe the information flow of a transformer architecture?

$indices \quad tokens$



Input embedding (vector + PE)

Select + **Aggregate**

Attention layer $\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$

Element-wise operations

Feed forward layers

elicsir
Elevating Italian Computer Science Innovation and Research

An example

EXPRESSIVITY - THE IDEA

Given a task f, can it be computed by a certain architecture transformer?

Find a RASP program P which solves f

If such program exists There exists a transformer which computes that task

An example: sorting

elicsir Elevating Italian Computer Science Innovation and Research

Sort a string

[Input]: $s = (s_0, \ldots, s_{n-1}) \in L \subseteq \Sigma^*$

[Output]: $s' \in \Sigma^* : s' = \pi(s) = (s_{j_0}, \ldots, s_{j_{n-1}}),\ s_{j_k} \leq s_{j_k+1} \quad \forall k \in [n]$

sort(“cab”) = “abc”

An example: sorting

Sort a string

```
sort(s, t) {

larger  = select(keys, keys, >) or
(select(keys, keys, ==) and
select(indices, indices, >));

num_smaller =
selector_width(larger);

target_pos = num_smaller;

sel_new =
select(indices, target_pos, ==);

sort = aggregate(sel_new, vals);
```

```
sort([c,a,b]) = [a,b,c]

t = s =[c,a,b] (= tokens)

indices = [0,1,2]
```

IDEA:

For each token, count **how many tokens should appear before it** in the sorted order (**t**)

≡

Target positions

Transform **s** accordingly

An example: sorting

Sort a string

```
sort(s, t) {

larger = select(t, t, >) or
(select(t, t, ==) and
select(indices, indices, >));

num_smaller =
selector_width(larger);

target_pos = num_smaller;

sel_new =
select(indices, target_pos, ==);

sort = aggregate(sel_new, s);
```

```
sort([c,a,b]) = [a,b,c]

t = s =[c,a,b] (= tokens)

indices = [0,1,2]
```

elicsir
Elevating Italian Computer Science Innovation and Research

An example: sorting

Sort a string

```
sort(s, t) {

larger = select(t, t, >) or
(select(t, t, ==) and
select(indices, indices, >));

num_smaller =
selector_width(larger);

target_pos = num_smaller;

sel_new =
select(indices, target_pos, ==);

sort = aggregate(sel_new, vals);
```

```
sort([c,a,b]) = [a,b,c]

t = s =[c,a,b] (= tokens)

indices = [0,1,2]
```

	c	a	b
c	F	T	T
a	F	F	F
b	F	T	F

elicsir
Elevating Italian Computer Science Innovation and Research

An example: sorting

Sort a string

```
sort(s, t) {

larger = select(t, t, >) or
(select(t, t, ==) and
select(indices, indices, >));

num_smaller =
selector_width(larger);

target_pos = num_smaller;

sel_new =
select(indices, target_pos, ==);

sort = aggregate(sel_new, vals);
```

```
sort([c,a,b]) = [a,b,c]
t = s =[c,a,b] (= tokens)
indices = [0,1,2]
```

$$\begin{bmatrix} F & T & T \\ F & F & F \\ F & T & F \end{bmatrix} \text{OR} \left(\begin{bmatrix} T & F & F \\ F & T & F \\ F & F & T \end{bmatrix} \text{AND} \begin{bmatrix} F & F & F \\ T & F & F \\ T & T & F \end{bmatrix} \right)$$

An example: sorting

Sort a string

```
sort(s, t) {

larger = select(t, t, >) or
(select(t, t, ==) and
select(indices, indices, >));

num_smaller =
selector_width(larger);

target_pos = num_smaller;

sel_new =
select(indices, target_pos, ==);

sort = aggregate(sel_new, vals);
```

```
sort([c,a,b]) = [a,b,c]

t = s =[c,a,b] (= tokens)

indices = [0,1,2]
```

$$\begin{bmatrix} F & T & T \\ F & F & F \\ F & T & F \end{bmatrix} \text{ OR } \left(\begin{bmatrix} T & F & F \\ F & T & F \\ F & F & T \end{bmatrix} \text{ AND } \begin{bmatrix} F & F & F \\ T & F & F \\ T & T & F \end{bmatrix} \right)$$

$$= \begin{bmatrix} F & T & T \\ F & F & F \\ F & T & F \end{bmatrix} \text{ OR } \begin{bmatrix} F & F & F \\ F & F & F \\ F & F & F \end{bmatrix} =$$

$$= \begin{bmatrix} F & T & T \\ F & F & F \\ F & T & F \end{bmatrix} = \texttt{larger}$$

elicsir
Elevating Italian Computer Science Innovation and Research

An example: sorting

Sort a string

```
sort(s, t) {

larger = select(t, t, >) or
(select(t, t, ==) and
select(indices, indices, >));

num_smaller =
selector_width(larger);

target_pos = num_smaller;

sel_new =
select(indices, target_pos, ==);

sort = aggregate(sel_new, vals);
```

```
sort([c,a,b]) = [a,b,c]
t = s = [c,a,b] (= tokens)
indices = [0,1,2]
```

$$\texttt{larger} = \begin{bmatrix} F & T & T \\ F & F & F \\ F & T & F \end{bmatrix}$$

For each element, count how many elements is bigger than (≡ number of T in each row)

$$\sum_{j=0}^{2} \mathbf{1}_{S(i,j)=T} \ \forall i \in \{0,1,2\}$$

An example: sorting

Sort a string

```
sort(s, t) {

larger = select(t, t, >) or
(select(t, t, ==) and
select(indices, indices, >));

num_smaller =
selector_width(larger);

target_pos = num_smaller;

sel_new =
select(indices, target_pos, ==);

sort = aggregate(sel_new, vals);
```

```
sort([c,a,b]) = [a,b,c]

t = s = [c,a,b] (= tokens)

indices = [0,1,2]
```

$$\texttt{larger} = \begin{bmatrix} F & T & T \\ F & F & F \\ F & T & F \end{bmatrix}$$

$$\texttt{num_smaller} = [2,0,1]$$

An example: sorting

Sort a string

```
sort(s, t) {

larger = select(t, t, >) or
(select(t, t, ==) and
select(indices, indices, >));

num_smaller =
selector_width(larger);

target_pos = num_smaller;

sel_new =
select(indices, target_pos, ==);

sort = aggregate(sel_new, vals);
```

```
sort([c,a,b]) = [a,b,c]

t = s = [c,a,b] (= tokens)

indices = [0,1,2]
```

$$\texttt{larger} = \begin{bmatrix} F & T & T \\ F & F & F \\ F & T & F \end{bmatrix}$$

$$\texttt{num_smaller} = [2,0,1]$$

num_smaller coincides with the index of each element in the sorted sequence

An example: sorting

Sort a string

```
sort(s, t) {

larger = select(t, t, >) or
(select(t, t, ==) and
select(indices, indices, >));

num_smaller =
selector_width(larger);

target_pos = num_smaller;

sel_new =
select(indices, target_pos, ==);

sort = aggregate(sel_new, vals);
```

```
sort([c,a,b]) = [a,b,c]
t = s = [c,a,b] (= tokens)
indices = [0,1,2]
```

$$\texttt{larger} = \begin{bmatrix} F & T & T \\ F & F & F \\ F & T & F \end{bmatrix}$$

$\texttt{num_smaller} = [2,0,1]$

`target_pos = num_smaller`

elicsir
Elevating Italian Computer Science Innovation and Research

An example: sorting

Sort a string

```
sort(s, t) {

larger = select(t, t, >) or
(select(t, t, ==) and
select(indices, indices, >));

num_smaller =
selector_width(larger);

target_pos = num_smaller;

sel_new =
select(indices, target_pos, ==);

sort = aggregate(sel_new, vals);
```

```
sort([c,a,b]) = [a,b,c]
t = s = [c,a,b] (= tokens)
indices = [0,1,2]
```

larger = $\begin{bmatrix} F & T & T \\ F & F & F \\ F & T & F \end{bmatrix}$

num_smaller = [2,0,1]

target_pos = num_smaller

An example: sorting

Sort a string

```
sort(s, t) {

larger = select(t, t, >) or
(select(t, t, ==) and
select(indices, indices, >));

num_smaller =
selector_width(larger);

target_pos = num_smaller;

sel_new =
select(indices, target_pos, ==);

sort = aggregate(sel_new, vals);
```

```
sort([c,a,b]) = [a,b,c]

t = s = [c,a,b] (= tokens)

indices = [0,1,2]
```

larger $= \begin{bmatrix} F & T & T \\ F & F & F \\ F & T & F \end{bmatrix}$

num_smaller $= [2,0,1]$

target_pos $= [2,0,1]$

```
Sort s accordingly to target_pos
```

An example: sorting

Sort a string

```
sort(s, t) {

larger = select(t, t, >) or
(select(t, t, ==) and
select(indices, indices, >));

num_smaller =
selector_width(larger);

target_pos = num_smaller;

sel_new =
select(indices, target_pos, ==);

sort = aggregate(sel_new, vals);
```

elicsir
Elevating Italian Computer Science Innovation and Research

```
sort([c,a,b]) = [a,b,c]
t = s = [c,a,b] (= tokens)
indices = [0,1,2]
```

larger $= \begin{bmatrix} F & T & T \\ F & F & F \\ F & T & F \end{bmatrix}$

num_smaller $= [2,0,1]$

target_pos $= [2,0,1]$

sel_new $= \begin{bmatrix} F & T & F \\ F & F & T \\ T & F & F \end{bmatrix}$

An example: sorting

Sort a string

```
sort(s, t) {

larger = select(t, t, >) or
(select(t, t, ==) and
select(indices, indices, >));

num_smaller =
selector_width(larger);

target_pos = num_smaller;

sel_new =
select(indices, target_pos, ==);

sort = aggregate(sel_new, s);
```

```
sort([c,a,b]) = [a,b,c]

t = s = [c,a,b] (= tokens)

indices = [0,1,2]
```

larger $= \begin{bmatrix} F & T & T \\ F & F & F \\ F & T & F \end{bmatrix}$

num_smaller $= [2,0,1]$

target_pos $= [2,0,1]$

sel_new $= \begin{bmatrix} F & T & F \\ F & F & T \\ T & F & F \end{bmatrix}$

Return the sorted string

elicsir
Elevating Italian Computer Science Innovation and Research

An example: sorting

Sort a string

```
sort(s, t) {

larger = select(t, t, >) or
(select(t, t, ==) and
select(indices, indices, >));

num_smaller =
selector_width(larger);

target_pos = num_smaller;

sel_new =
select(indices, target_pos, ==);

sort = aggregate(sel_new, s);
```

```
sort([c,a,b]) = [a,b,c]
t = s = [c,a,b] (= tokens)
indices = [0,1,2]
```

$$\texttt{larger} = \begin{bmatrix} F & T & T \\ F & F & F \\ F & T & F \end{bmatrix}$$

$$\texttt{num_smaller} = [2,0,1]$$

$$\texttt{target_pos} = [2,0,1]$$

$$\texttt{sel_new} = \begin{bmatrix} F & T & F \\ F & F & T \\ T & F & F \end{bmatrix}$$

$$\begin{bmatrix} F & T & F \\ F & F & T \\ T & F & F \end{bmatrix} \begin{matrix} \texttt{cab} \\ \texttt{cab} \\ \texttt{cab} \end{matrix} \rightarrow \text{“abc”}$$

An example: sorting

Sort a string

```
sort(s, t) {

larger = select(t, t, >) or
(select(t, t, ==) and
select(indices, indices, >));

num_smaller =
selector_width(larger);

target_pos = num_smaller;

sel_new =
select(indices, target_pos, ==);

sort = aggregate(sel_new, s);
```

`sort([c,a,b]) = [a,b,c]`

`t = s = [c,a,b] (= tokens)`

`indices = [0,1,2]`

$$\texttt{larger} = \begin{bmatrix} F & T & T \\ F & F & F \\ F & T & F \end{bmatrix}$$

$$\texttt{num_smaller} = [2,0,1]$$

$$\texttt{target_pos} = [2,0,1]$$

$$\texttt{sel_new} = \begin{bmatrix} F & T & F \\ F & F & T \\ T & F & F \end{bmatrix}$$

`sort = "abc"`

elicsir
Elevating Italian Computer Science Innovation and Research

Sorting → Transformer Architecture

```
s = t = [c,a,b], indices = [0,1,2]
```

Embedding

```
sort(s, t) {

larger = select(t, t, >) or
(select(t, t, ==) and
select(indices, indices, >));

num_smaller =
selector_width(larger);


target_pos = num_smaller;


sel_new =
select(indices, target_pos, ==);

sort = aggregate(sel_new, s);
```

Sorting → Transformer Architecture

```
s = t = [c,a,b], indices = [0,1,2]
```

Embedding

```
sort(s, t) {

larger = select(t, t, >) or
(select(t, t, ==) and
select(indices, indices, >));

num_smaller =
selector_width(larger);
```

Self attention layer 1

```
target_pos = num_smaller;


sel_new =
select(indices, target_pos, ==);

sort = aggregate(sel_new, s);
```

Sorting → Transformer Architecture

```
s = t = [c,a,b], indices = [0,1,2]
```

Embedding

```
sort(s, t) {
```

```
larger = select(t, t, >) or
(select(t, t, ==) and
select(indices, indices, >));

num_smaller =
selector_width(larger);
```

Self attention layer 1

```
target_pos = num_smaller;
```

Feed Forward layer 1

```
sel_new =
select(indices, target_pos, ==);

sort = aggregate(sel_new, s);
```

Sorting → Transformer Architecture

```
s = t = [c,a,b], indices = [0,1,2]
```
Embedding

```
sort(s, t) {
```

```
larger = select(t, t, >) or
(select(t, t, ==) and
select(indices, indices, >));

num_smaller =
selector_width(larger);
```
Self attention layer 1

```
target_pos = num_smaller;
```
Feed Forward layer 1

```
sel_new =
select(indices, target_pos, ==);

sort = aggregate(sel_new, s);
```
Self attention layer 2

Sorting → Transformer Architecture

```
s = t = [c,a,b], indices = [0,1,2]
```

Embedding

```
sort(s, t) {
```

```
larger = select(t, t, >) or
(select(t, t, ==) and
select(indices, indices, >));

num_smaller =
selector_width(larger);
```

Self attention layer 1

```
target_pos = num_smaller;
```

Feed Forward layer 1

```
sel_new =
select(indices, target_pos, ==);

sort = aggregate(sel_new, s);
```

Self attention layer 2

```
(no-op)
```

Feed Forward layer 2

Some other examples

RASP is applied to:

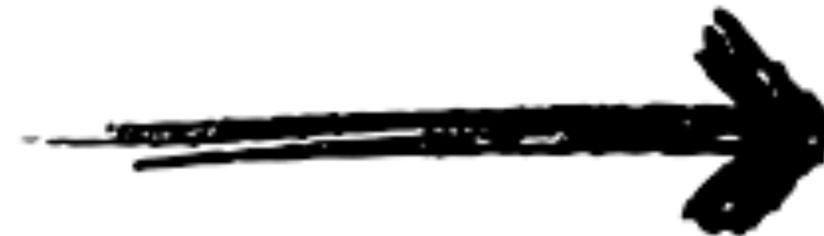
- **Reverse** `reverse("abc")="cba"`
- **Histograms** `hist_bos(“$aba")=[$,2,1,2]`
- **Double-histograms** `hist2("$abbc")=[$,2,1,1,2]`
- **Find** the **most frequent element** in a sequence `most_freq("$abbccddd")="$dbca$$$$"`
- ...

Beyond RASP

Elevating Italian Computer Science Innovation and Research

Tracr (2023)

RASP program



Transformer architecture

D-RASP (2026)

Abstraction and Compilation

Each set of high-level operations can be compiled down to execute a transformer

Obs full compilation ≡ **concrete weights**

Tracr: Compiled Transformers as a Laboratory for Interpretability

David Lindner† Google DeepMind
János Kramár Google DeepMind
Sebastian Farquhar Google DeepMind
Matthew Rahtz Google DeepMind
Thomas McGrath Google DeepMind
Vladimir Mikulik† Google DeepMind

Abstract

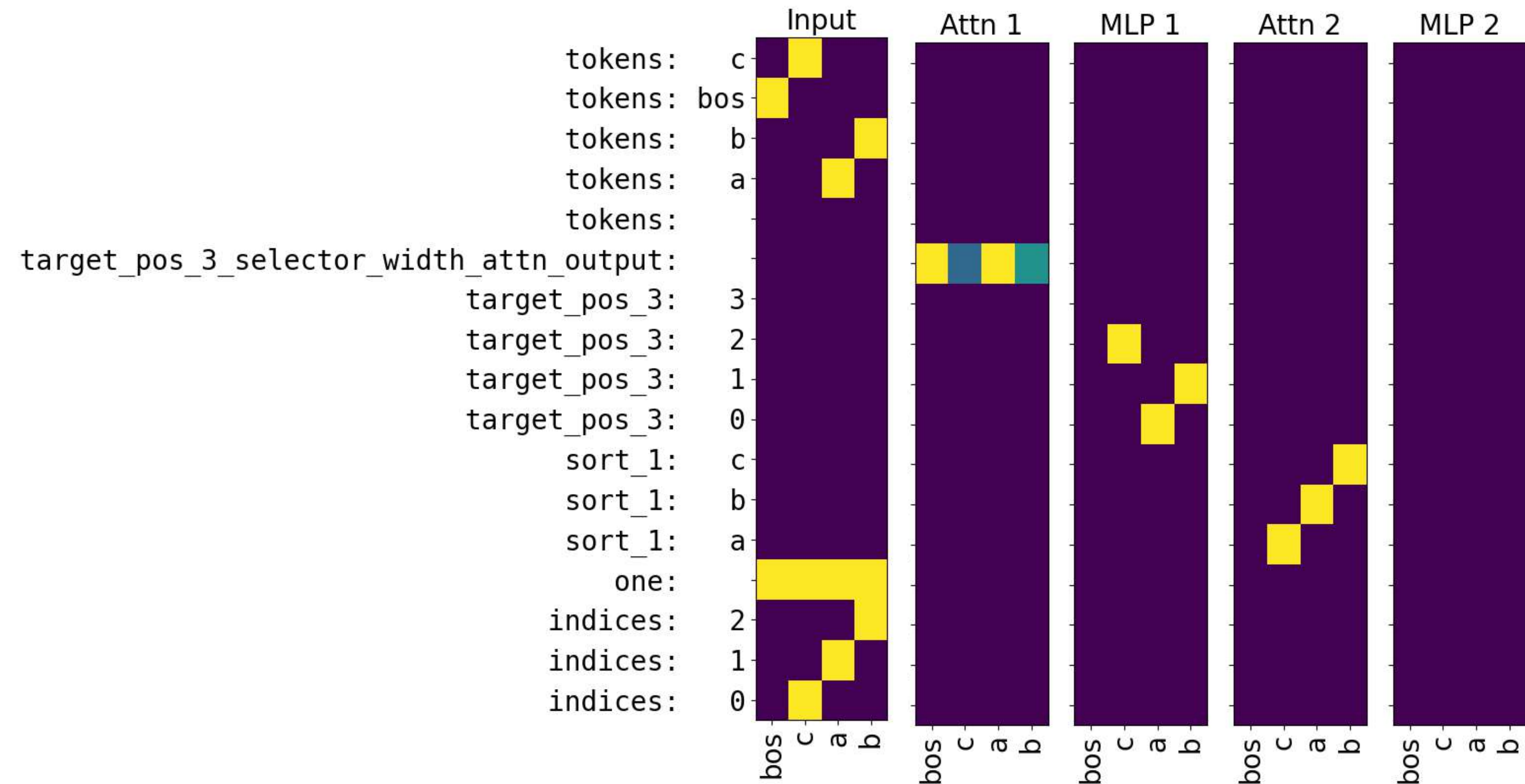
We show how to "compile" human-readable programs into standard decoder-only transformer models. Our compiler, Tracr, generates models with known structure. This structure can be used to design experiments. For example, we use it to study "superposition" in transformers that execute multi-step algorithms. Additionally, the known structure of Tracr-compiled models can serve as *ground-truth* for evaluating interpretability methods. Commonly, because the "programs" learned by transformers are unknown it is unclear whether an interpretation succeeded. We demonstrate our approach by implementing and examining programs including computing token frequencies, sorting, and parenthesis checking. We provide an open-source implementation of Tracr at https://github.com/google-deepmind/tracr.

NeurIPS, 2023

Compiling with Tracr

For compiling in Tracr:

```
tokens = ["bos", "c", "a", "b"]
```



Compiling with Tracr

For compiling in Tracr:

```
tokens = [“bos”, “c”, “a”, “b”]
```

Embedding

$$X = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

Compiling with Tracr

For compiling in Tracr:

```
tokens = ["bos", "c", "a", "b"]
```

Self attention layer 1

$$W_Q^{(1)} = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} \qquad W_K^{(1)} = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad W_V^{(1)} = \mathbb{I}_4$$

Self attention layer 2

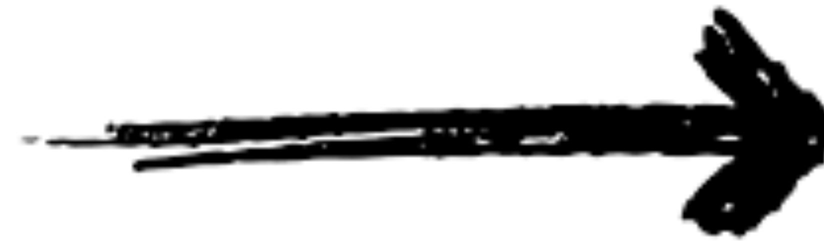
$$W_Q^{(2)} = W_K^{(2)} = W_V^{(2)} = \mathbb{I}_4$$

elicsir
Elevating Italian Computer Science Innovation and Research

Beyond RASP

Tracr (2023)

RASP program



Transformer architecture

D-RASP (2026)

Beyond RASP

Q: Do trained models of transformers implement such simple programs?

Discovering Interpretable Algorithms by Decompiling Transformers to RASP

Xinting Huang [*1] **Aleksandra Bakalova** [*1] **Satwik Bhattamishra** [2] **William Merrill** [3] **Michael Hahn** [1]

Abstract

Recent work has shown that the computations of Transformers can be simulated in the RASP family of programming languages. These findings have enabled improved understanding of the expressive capacity and generalization abilities of Transformers. In particular, Transformers have been suggested to length-generalize exactly on problems that have simple RASP programs. However, it remains open whether trained models actually implement simple interpretable programs. In this paper, we present a general method to extract such programs from trained Transformers. The idea is to faithfully re-parameterize a Transformer as a RASP program and then apply causal interventions to discover a small sufficient sub-program. In experiments on small Transformers trained on algorithmic and formal language tasks, we show that our method often recovers simple and interpretable RASP programs from length-generalizing transformers. Our results provide the most direct evidence so far that Transformers internally implement simple RASP programs.[1]

program, then a Transformer trained on shorter inputs will generalize to longer inputs (RASP length generalization conjecture, Zhou et al., 2024; Huang et al., 2025). Yet, this story has a missing piece: Theoretical results show that Transformers *can* realize RASP-like computations, but they do not show that trained models *do* so in a human-interpretable way. Interpretability methods such as circuit discovery (e.g. Conmy et al., 2023) identify sparse computation graphs supporting particular behaviors, but the resulting objects are *circuits* rather than *programs*: they are usually tied to specific input lengths and input templates, and do not directly yield an algorithm uniformly across input lengths. Lindner et al. (2023) map RASP programs *to* Transformers, but the inverse problem—recovering a compact symbolic program *from* a trained Transformer—has remained open. This gap is especially salient for the length-generalization conjecture: if length-generalizing models succeed because they internally implement short RASP programs, we would like to *extract* those programs and inspect them.

In this work we introduce a general decompilation pipeline for recovering interpretable RASP-like programs from trained Transformers. Our starting point is a faithful reparameterization: we define a target dialect, **Decompiled RASP (D-RASP)**, whose primitives mirror Transformer

arXiv, 02/2026

elicsir
Elevating Italian Computer Science Innovation and Research

Discovering Interpretable Algorithms by Decompiling Transformers to RASP

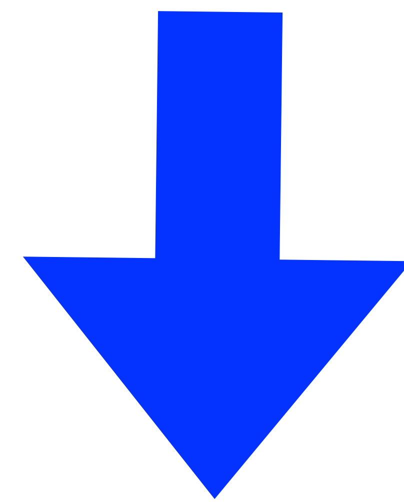
Length-generalization conjecture

Some **trained Transformers** that length-generalize on algorithmic/formal-language tasks **actually implement simple RASP-like algorithms internally**

elicsir
Elevating Italian Computer Science Innovation and Research

Discovering Interpretable Algorithms by Decompiling Transformers to RASP

Some **trained Transformers** that length-generalize on algorithmic/formal-language tasks **actually implement RASP-like algorithms internally**



Design a **method** to extract those algorithms **automatically** as readable programs

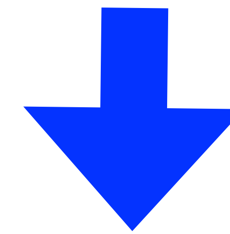
Overview of the method

Instead of RASP, they use **D(ecompiled)-RASP**

RASP	D-RASP
boolean selectors	real value selectors
hard/discrete aggregation	softmax aggregation
Symbolic sequence variables	Activation variables $v \in \mathbb{R}^{d \times N}$ Selector variables $\alpha \in \mathbb{R}^{N \times N}$
Tokens, positions, sequence features	Explicit one-hot variables
Symbolic sequence output	Real-valued logits

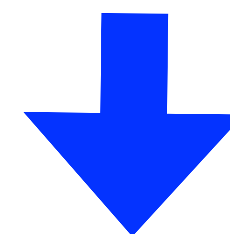
elicsir
Elevating Italian Computer Science Innovation and Research

Trained transformer model



1. Translate the model in **D-RASP primitives** ➡ The model is “huge”

2. Remove **unnecessary parameters** and operations applying **causal intervention**



D-RASP program

Conclusions and some future directions

RASP proposes a computational model for Transformers as an abstraction of a programming language

RASP-guided **architecture reduction**

…from decompilation to **model simplification**

Does symbolic simplicity correlate with **generalization**?

Expressivity: from infinite families to **finite constructions**

elicsir
Elevating Italian Computer Science Innovation and Research

Thank you for the

$$\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

elicsir
Elevating Italian Computer Science Innovation and Research

References

[1] **“Attention is all you need”**, A.Vaswani et al., NeurIPS, 2017

[2] **“Thinking like Transformers”**, G.Weiss et al. ICML, 2021

[3] **“Improving transformer models by reordering their sublayers”**, Press et al., ACL, 2020

[4] **“Discovering Interpretable Algorithms by Decompiling Transformers to RASP”**, X.Huang et al., arXiv, 2026

[5] **“Synthesis and Verification of Transformers Programs”**, H.Jiang et al., arXiv, 2026

[6] **“Thinking Like Transformers”**, Rush & Weiss, ICLR Blogposts, 2023

[7] **“Tracr: Compiled Transformers as a Laboratory for Interpretability”**, D.Lindner et al., NeurIPS, 2023

[8] **“Introduction to the Theory of Computation”**, M.Sipser, 3rd ed., Cengage Learning, 2013.

[9] **“Finite State Automata and Simple Recurrent Networks.”,** A. Cleeremans et al., *Neural Computation*, 1989.

elicsir
Elevating Italian Computer Science Innovation and Research

Tracr: compiling RASP into transformers

IDEA

1. Convert the RASP program into a Computational Direct Acyclic Graph (CDAG)

2. For each s-op, decide how to embed it in the residual stream

3. Convert each node in the CDAG independently to a model block

4. Assign each block to a layer of the transformer architecture

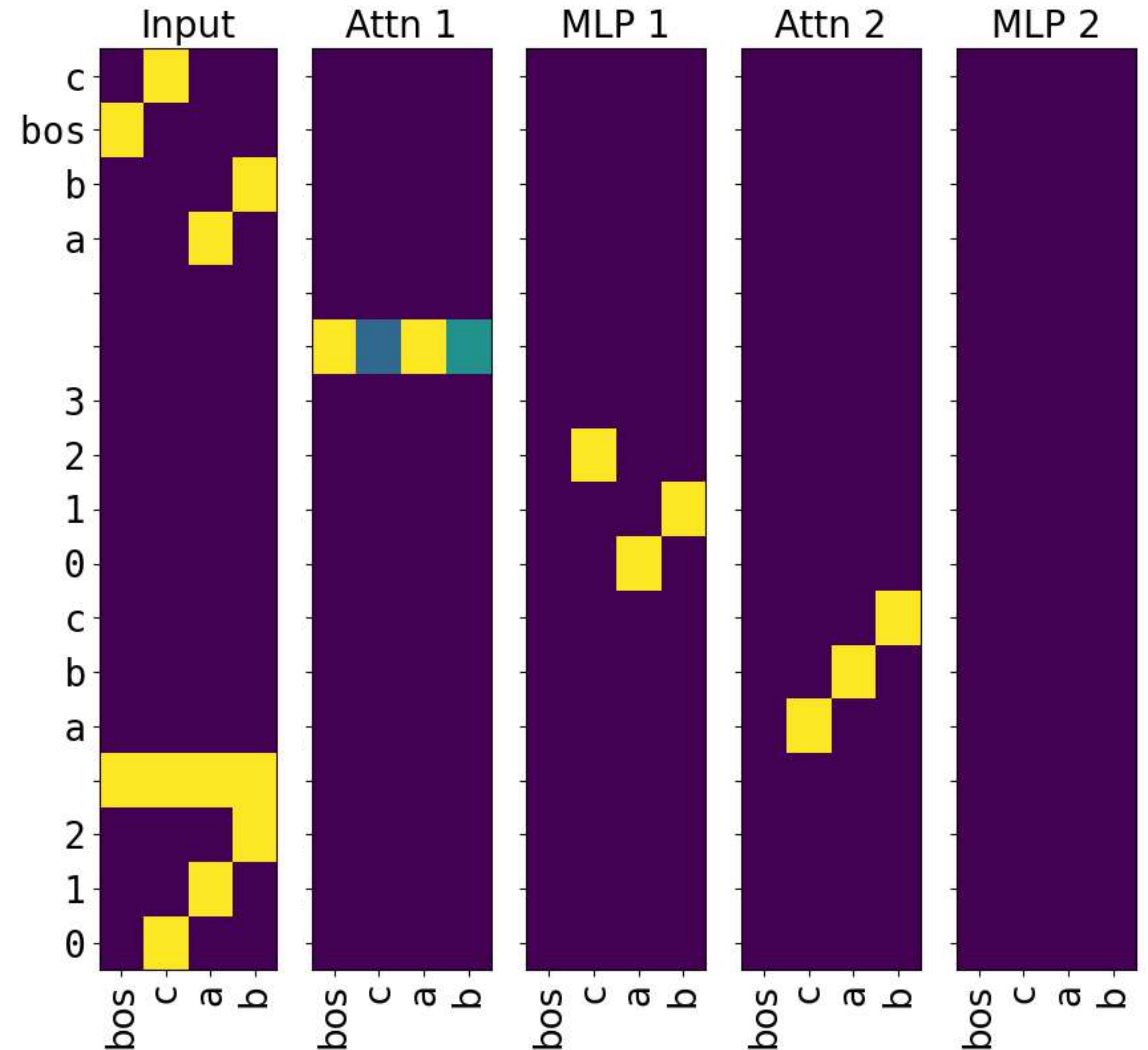
5. Construct the residual stream space as the direct sum of all models components' input and output spaces

6. Assemble weight matrices

Compiling with Tracr

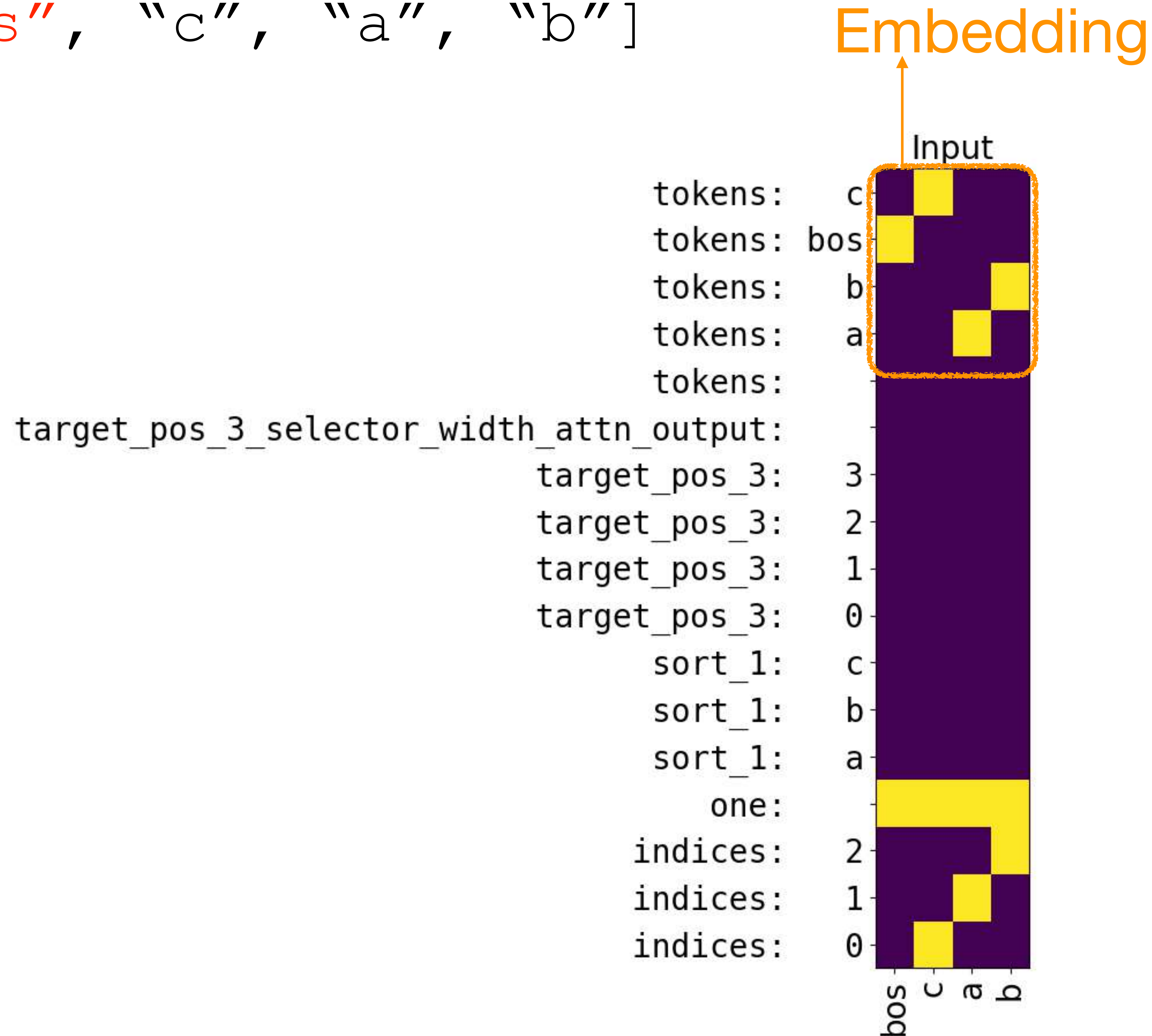
For compiling in Tracr:

```
tokens = ["bos", "c", "a", "b"]
```



Compiling with Tracr

tokens = ["bos", "c", "a", "b"]



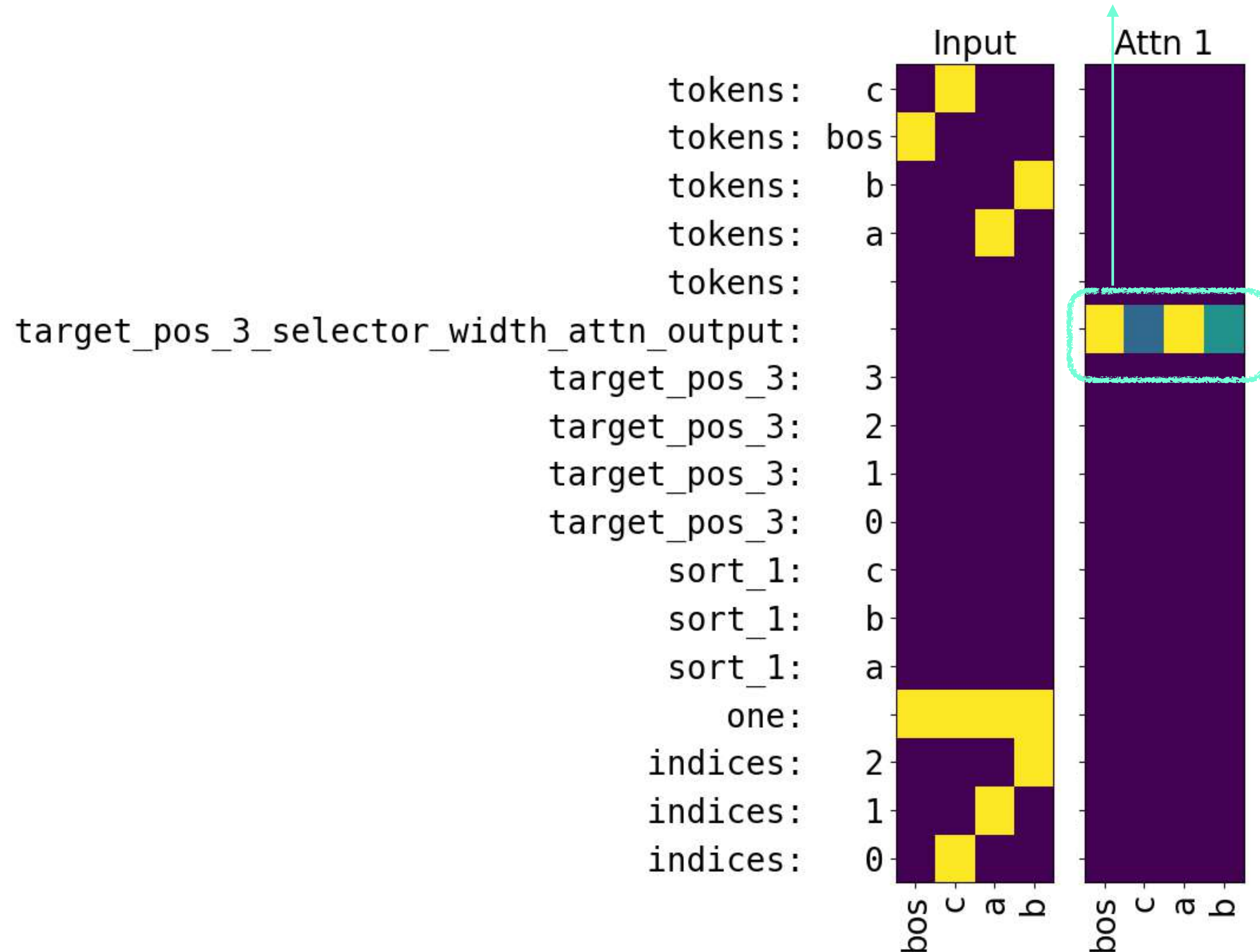
Compiling with Tracr

tokens = [“bos”, “c”, “a”, “b”]

“c” is greater than two elements

“b” is greater than one element

“a” is greater than zero element

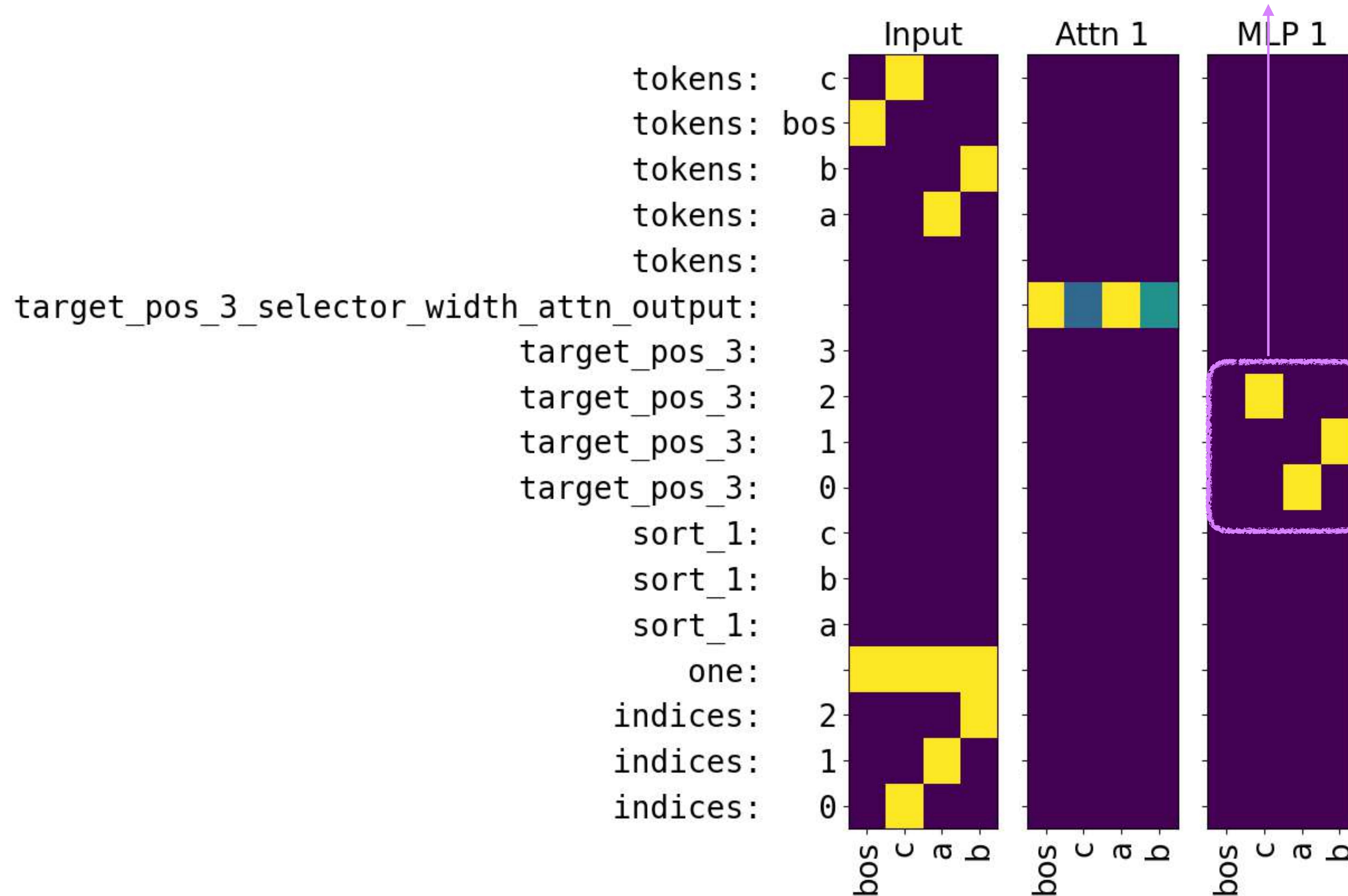


Compiling with Tracr

tokens = ["bos", "c", "a", "b"]

FFN Layer 1

target_pos = [2, 0, 1]

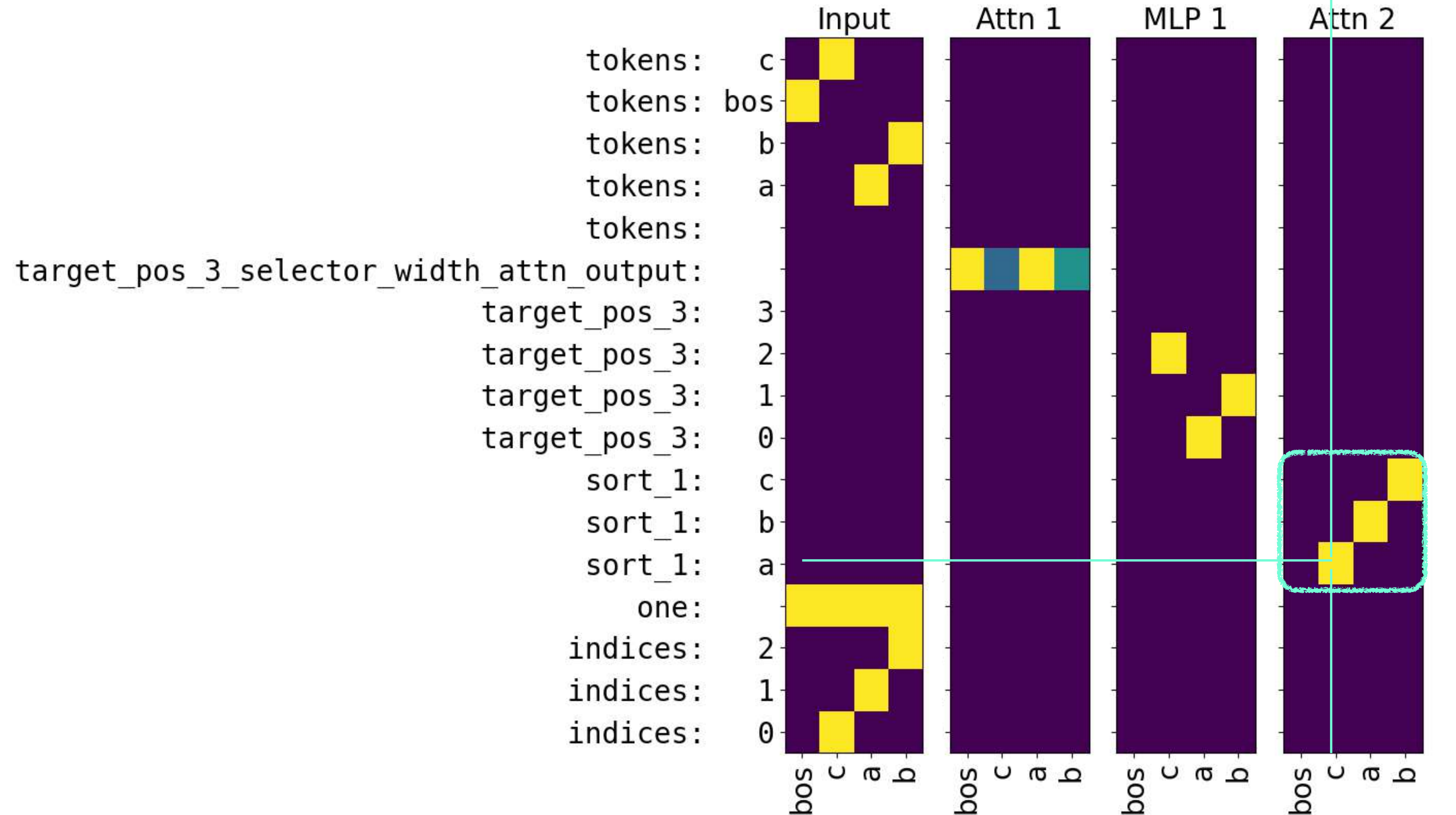


Compiling with Tracr

tokens = [“bos”, “c”, “a”, “b”]

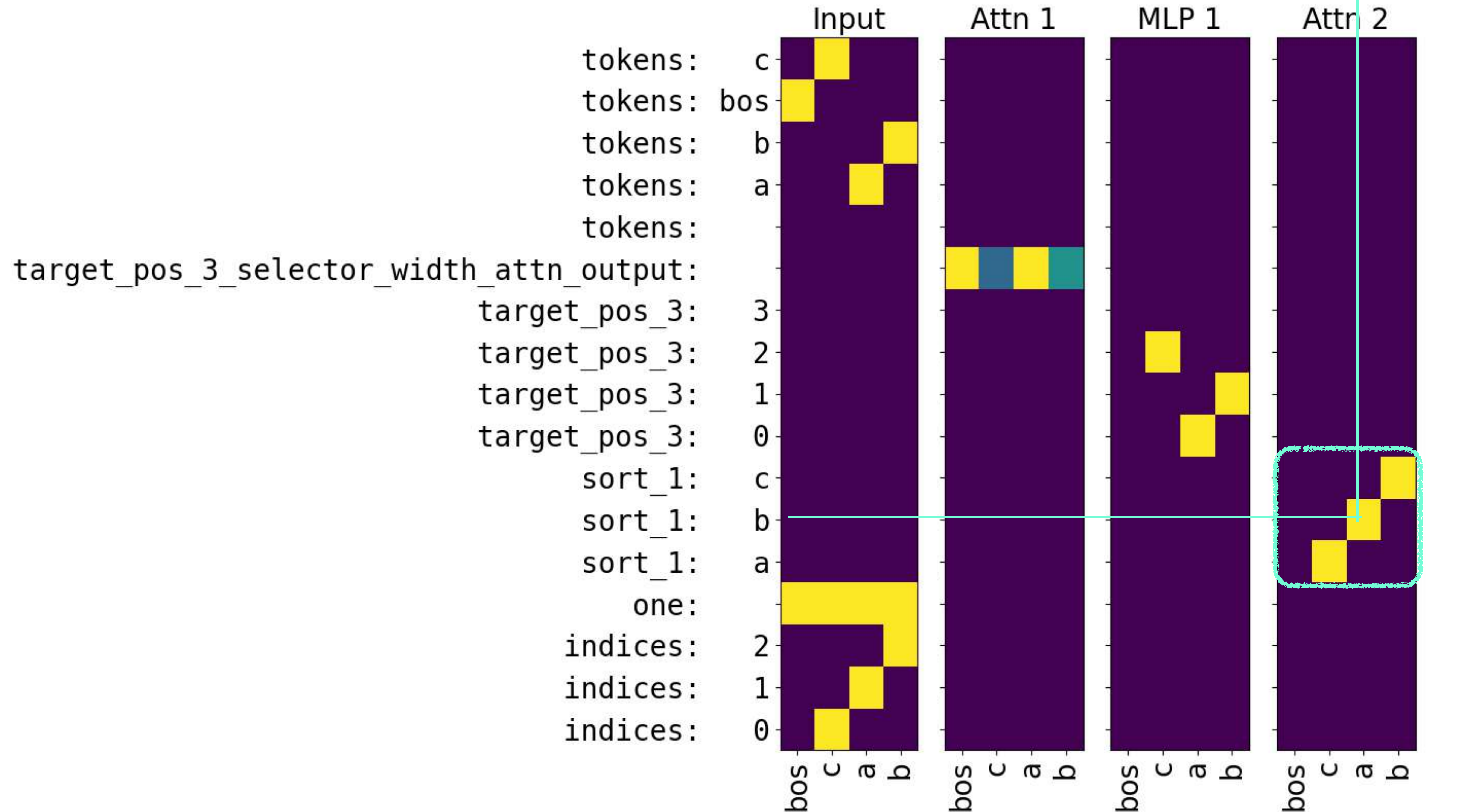
Self attention layer 2

“a”→“c”



Compiling with Tracr

tokens = [“bos”, “c”, “a”, “b”]

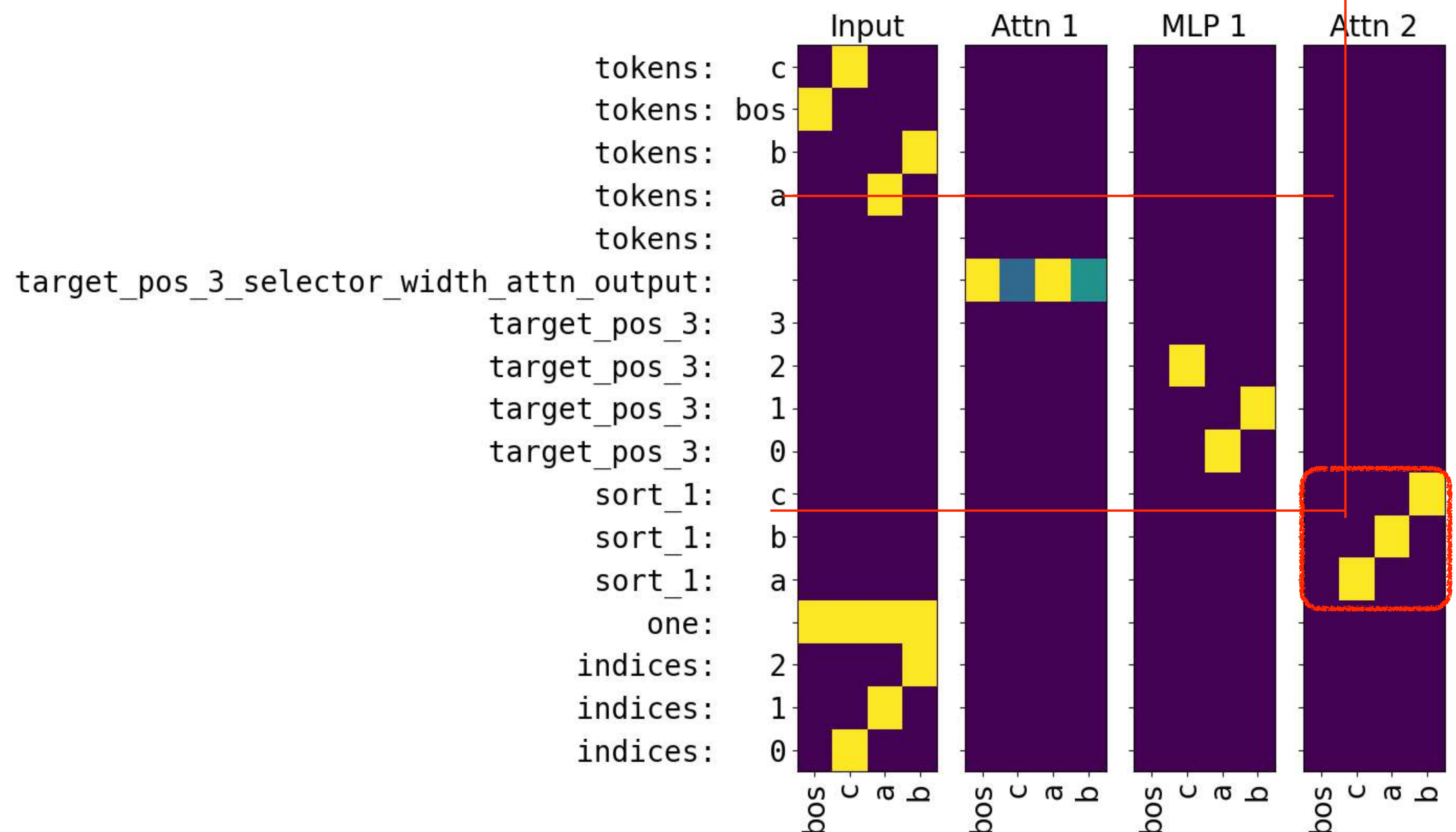


elicsir
Elevating Italian Computer Science Innovation and Research

Compiling with Tracr

```
tokens = ["bos", "c", "a", "b"]
```

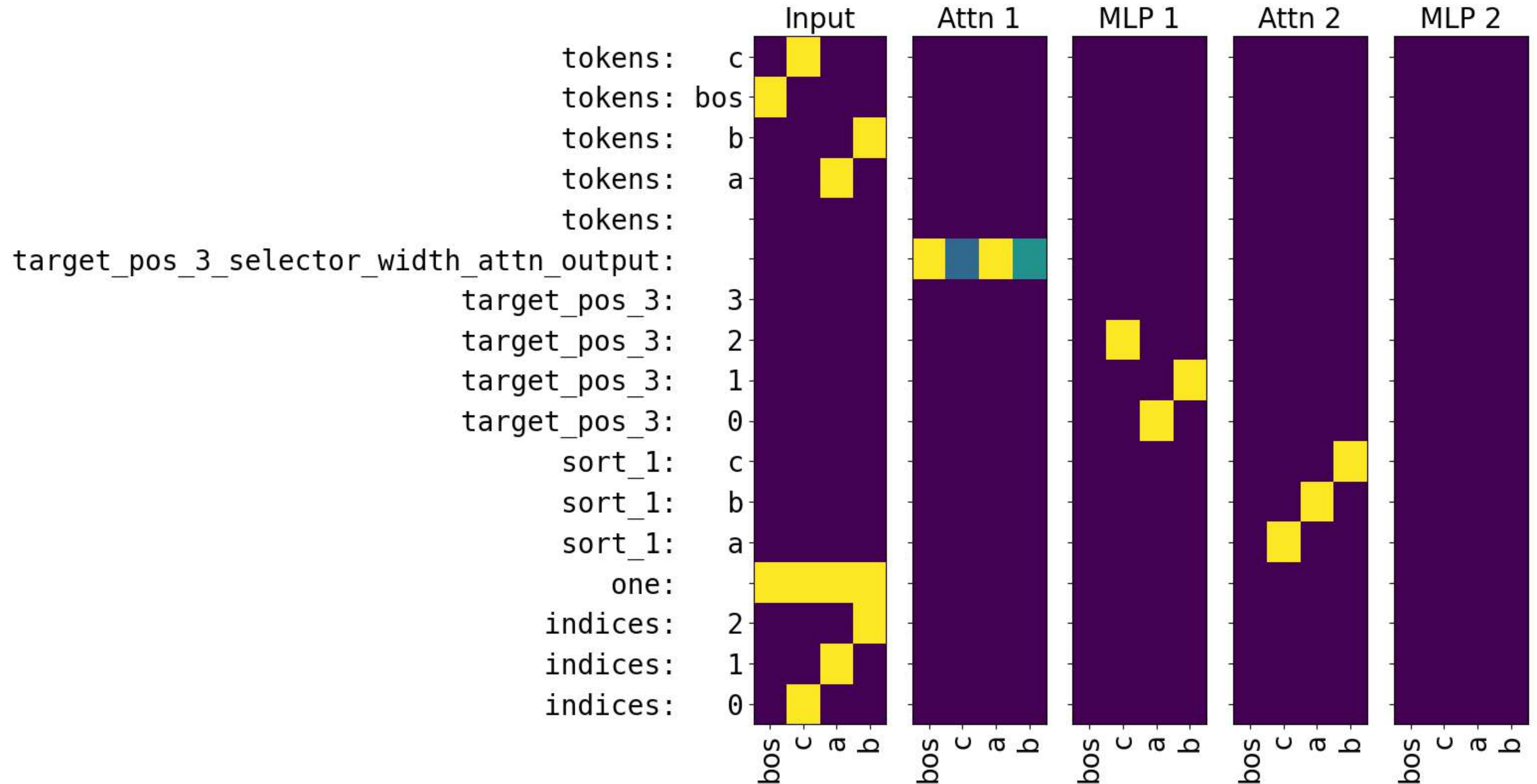
"c"→"b"



Compiling with Tracr

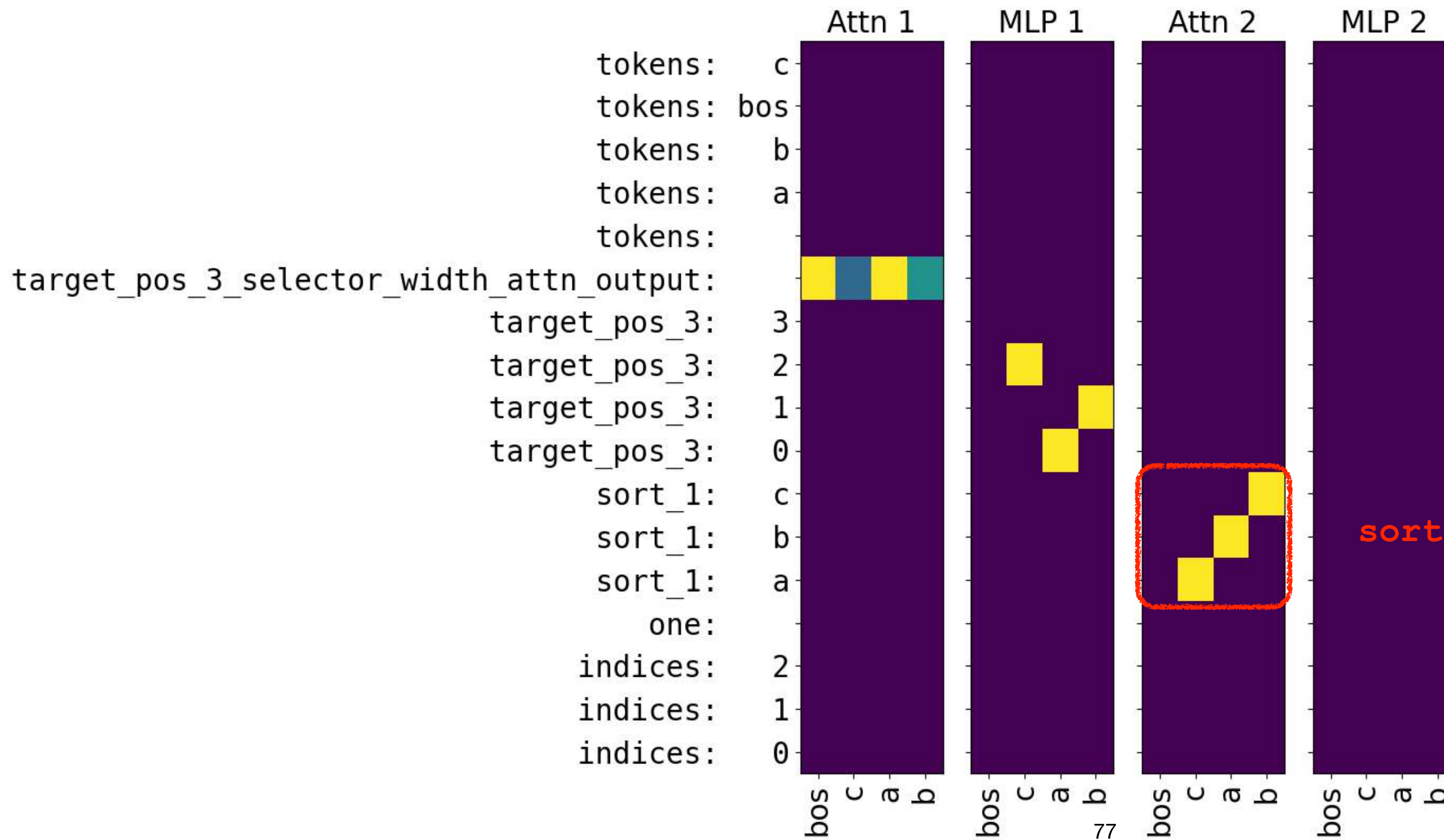
tokens = [“bos”, “c”, “a”, “b”]

sort([“c”, “a”, “b”]) = [“a”, “b”, “c”]



Compiling with Tracr

tokens = [“bos”, “c”, “a”, “b”]



sort([“c”, “a”, “b”]) = [“a”, “b”, “c”]

RASP —Implications and Insights

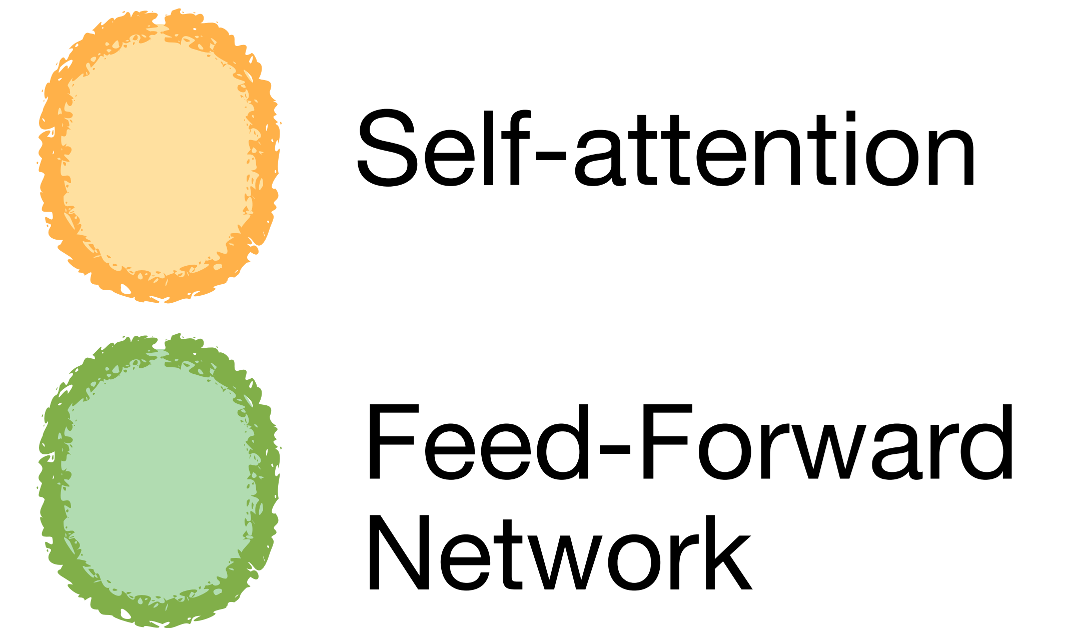
Original time complexity of the attention $\equiv O(n^2)$

Lowered to $O(n \log n)$ or $O(n) \rightarrow$ sparse attention

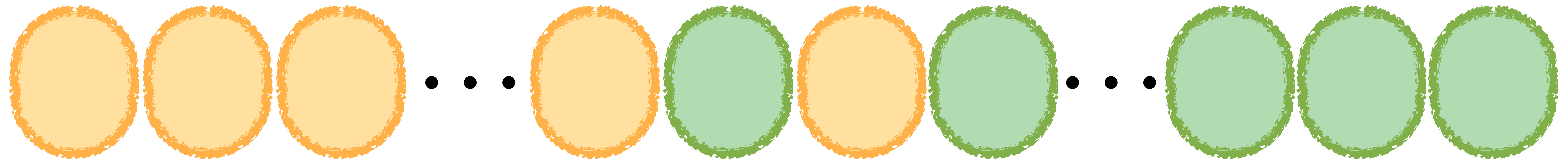
We can use RASP to reason about the computation capabilities of such variants

E.g. any arrangement of sublayers imposes a restriction on the order and number of RASP operations

RASP —Implications and Insights



Example - Sandwich Transformers [3]

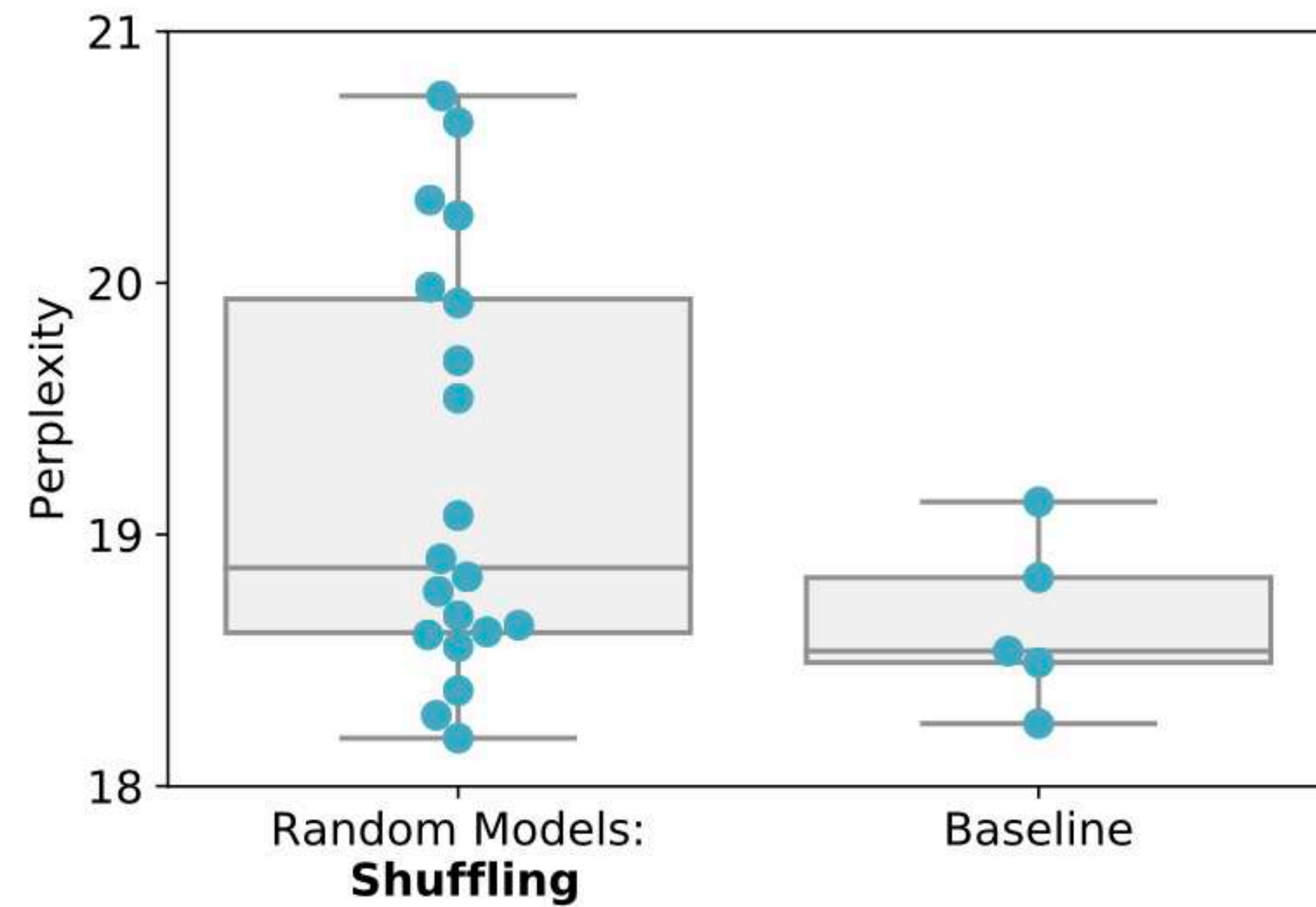


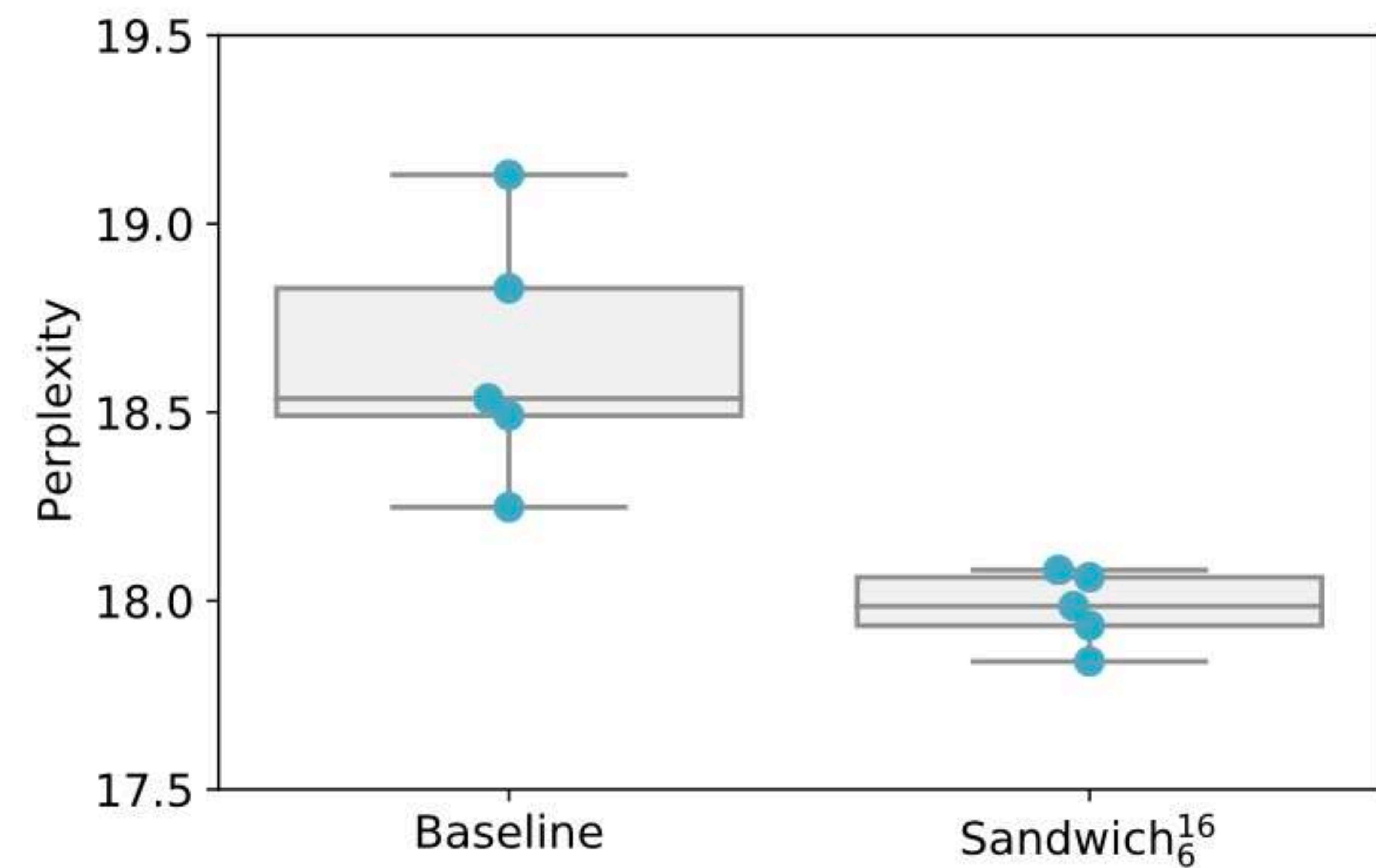
Claim

1. Pushing FFN sublayers towards the bottom of the transformer weakens it
2. Pushing attention sublayers to the bottom and FFN sublayers to the top of the transformer strengthen it

RASP —Implications and Insights

Example - Sandwich Transformers





elicsir
Elevating Italian Computer Science Innovation and Research

RASP —Implications and Insights

Claim

1. **Pushing FFN sublayers towards the bottom of the transformer weakens it**

2. Pushing attention sublayers to the bottom and FFN sublayers to the top of the transformer strengthen it

FFN sublayers ≡ element-wise operations

Self attention sublayers ≡ selectors-aggregate

Intuition with RASP (1)

There is little value to repeated element-wise operations before the first aggregate: each position has only its input and cannot generate new information

RASP —Implications and Insights

Claim

1. Pushing FFN sublayers towards the bottom of the transformer weakens it
2. **Pushing attention sublayers to the bottom and FFN sublayers to the top of the transformer strengthen it**

FFN sublayers ≡ element-wise operations

Self attention sublayers ≡ selectors-aggregate

Intuition with RASP (2)

An architecture beginning with several select-aggregate pairs is able to gather a large amount of information into each position early in the computation

elicsir
Elevating Italian Computer Science Innovation and Research

RASP —Implications and Insights

Claim

1. Pushing FFN sublayers towards the bottom of the transformer weakens it
2. **Pushing attention sublayers to the bottom and FFN sublayers to the top of the transformer strengthen it**

FFN sublayers ≡ element-wise operations

Self attention sublayers ≡ selectors-aggregate

Intuition with RASP (2) (contd.)

More complicated gathering rules can later be realized by applying element-wise operations to aggregated information before generating new selectors

elicsir
Elevating Italian Computer Science Innovation and Research

RASP —The computational model

Select + Aggregate

In general, select-aggregate can be seen two dimensional map-reduce:

$$select \equiv filtering$$

$$aggregate \equiv reduce$$

s = select([1,2,2], [0,1,2], ==)

	1	2	2
0	F	F	F
1	**T**	F	F
2	F	**T**	**T**

res = aggregate(s, [4,6,8])

F	F	F	4	6	8
T	F	F	4	6	8
F	**T**	**T**	4	6	8

[0,4,7]