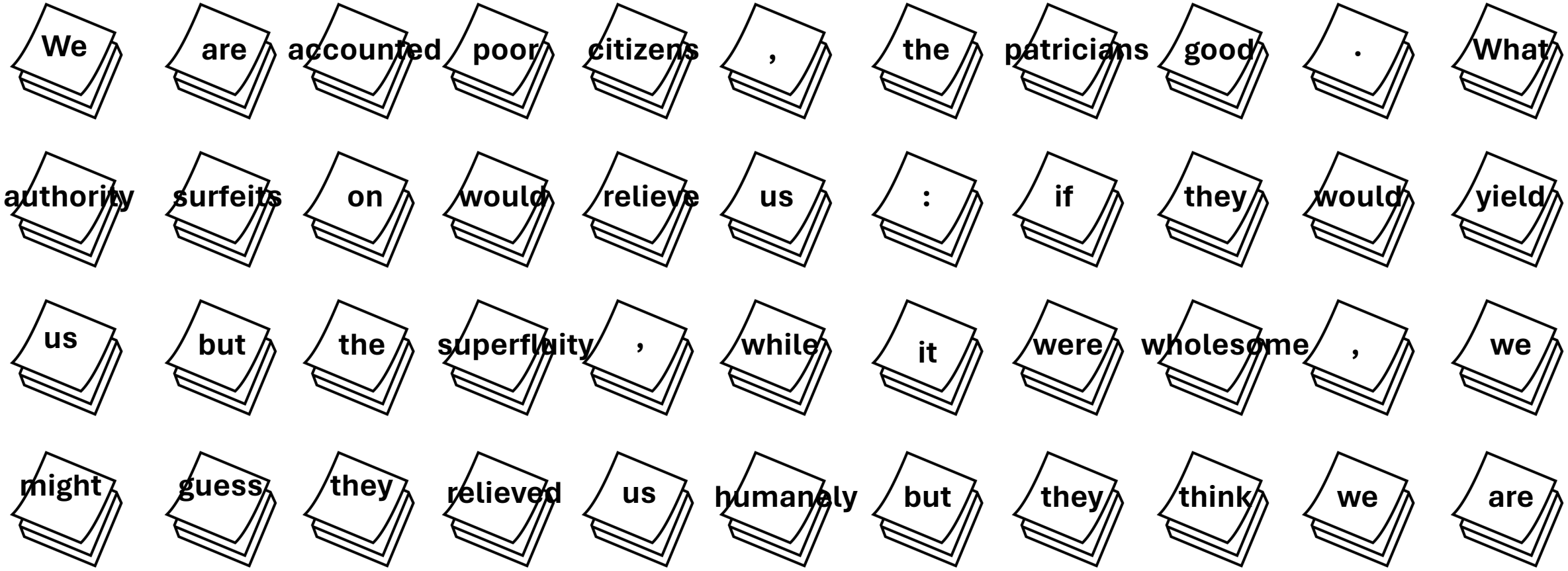
We are accounted poor citizens, the patricians good. What authority surfeits on would relieve us: if they would yield us but the superfluity, while it were wholesome, we might guess they relieved us humanely; but they think we are too dear: the leanness that afflicts us, the object of our misery, is as an inventory to particularise their abundance; our sufferance is a gain to them Let us revenge this with our pikes, ere we become rakes: for the gods know I speak this in hunger for bread, not in thirst for revenge.

Texto original


We are accounted poor citizens , the patricians good . What
authority surfeits on would relieve us : if they would yield
us but the superfluity , while it were wholesome , we
might guess they relieved us humanely but they think we are

Token

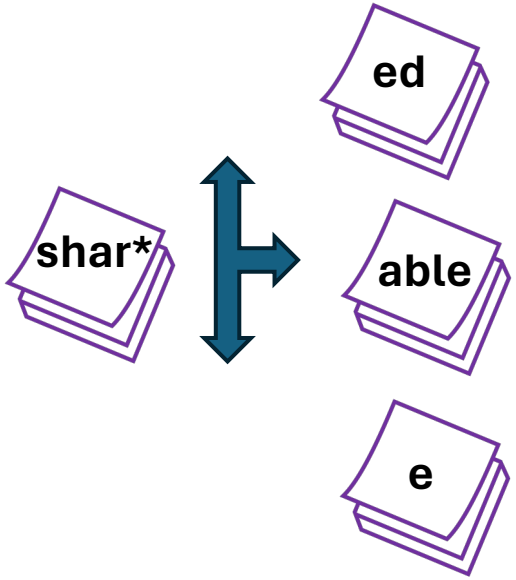

We
are
accounted
poor
citizens
,
the
patricians
good
.
What
authority
surfeits
on
would
relieve
us
:
if
they
would
yield
us
but
the
superfluity
,
while
it
were
wholesome
,
we
might
guess
they
relieved
us
humanely
but
they
think
we
are

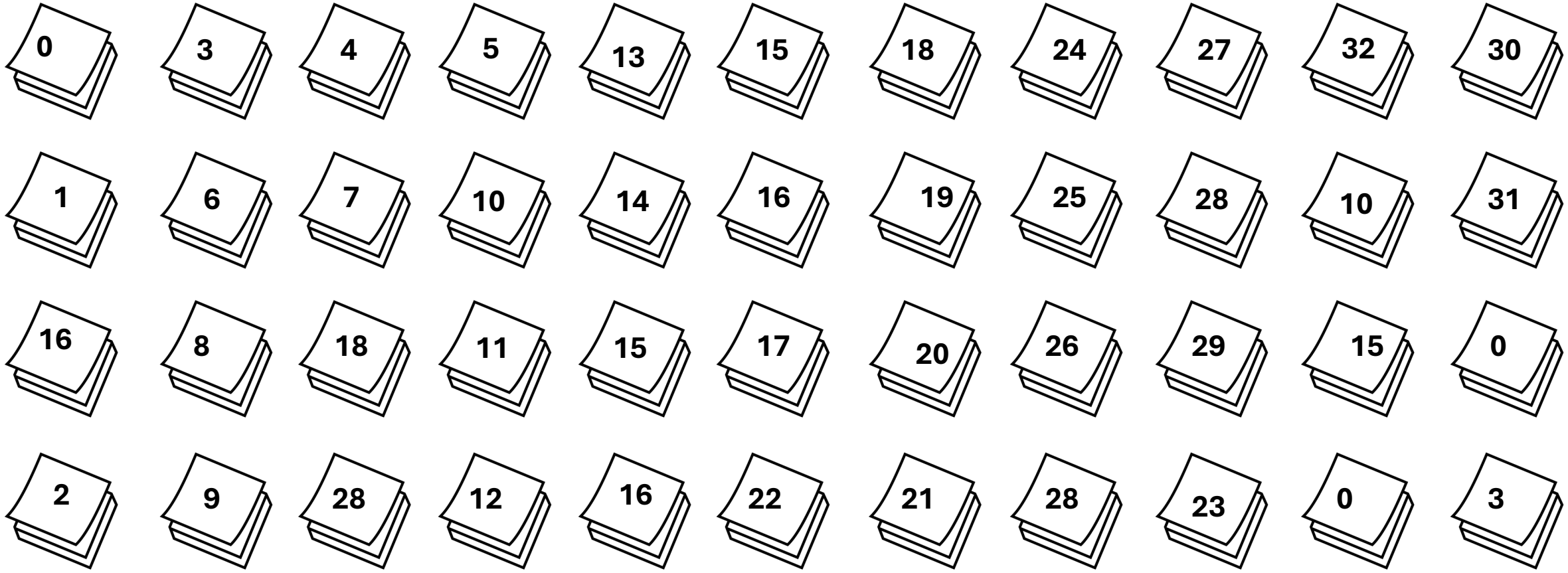
Token sem repetição

Token sem repetição

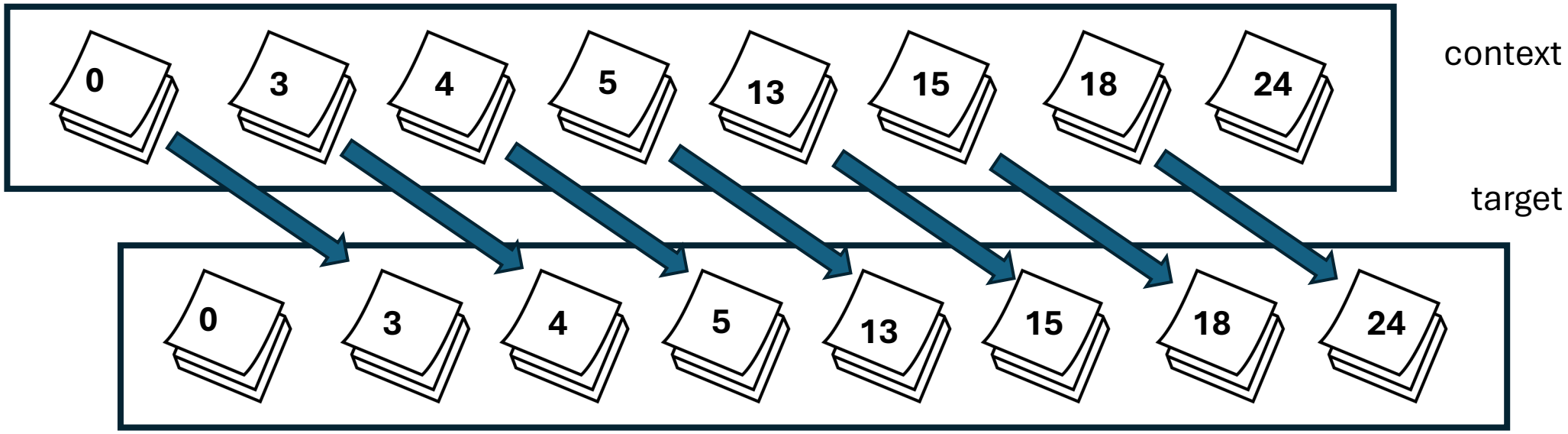
Vocabulario
Google – Sentence Piece
OpenAI – TikTok (GPT)

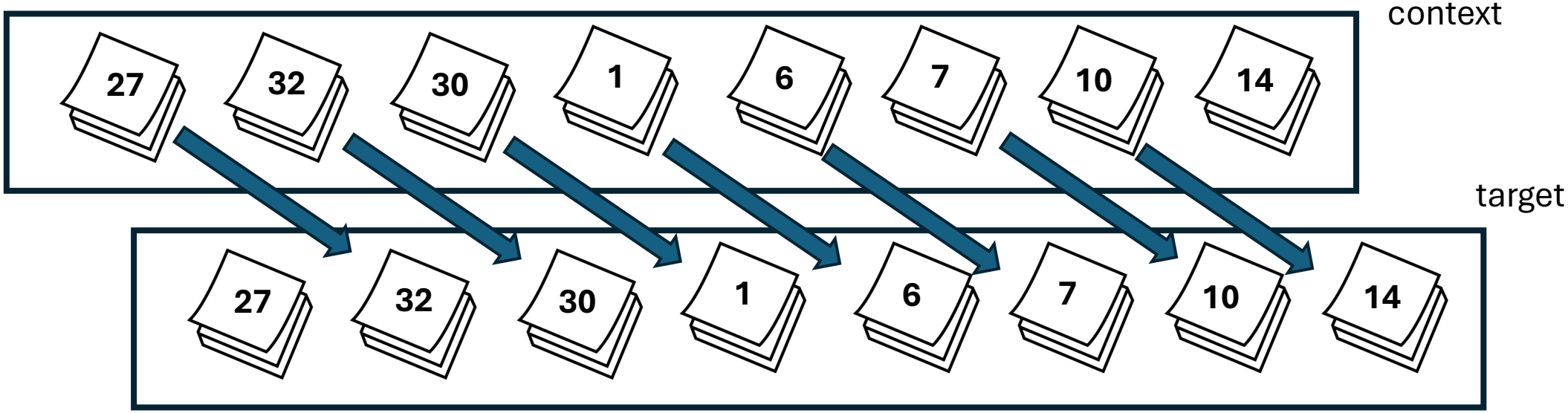
Tipicamente ~40K - 50k Tokens

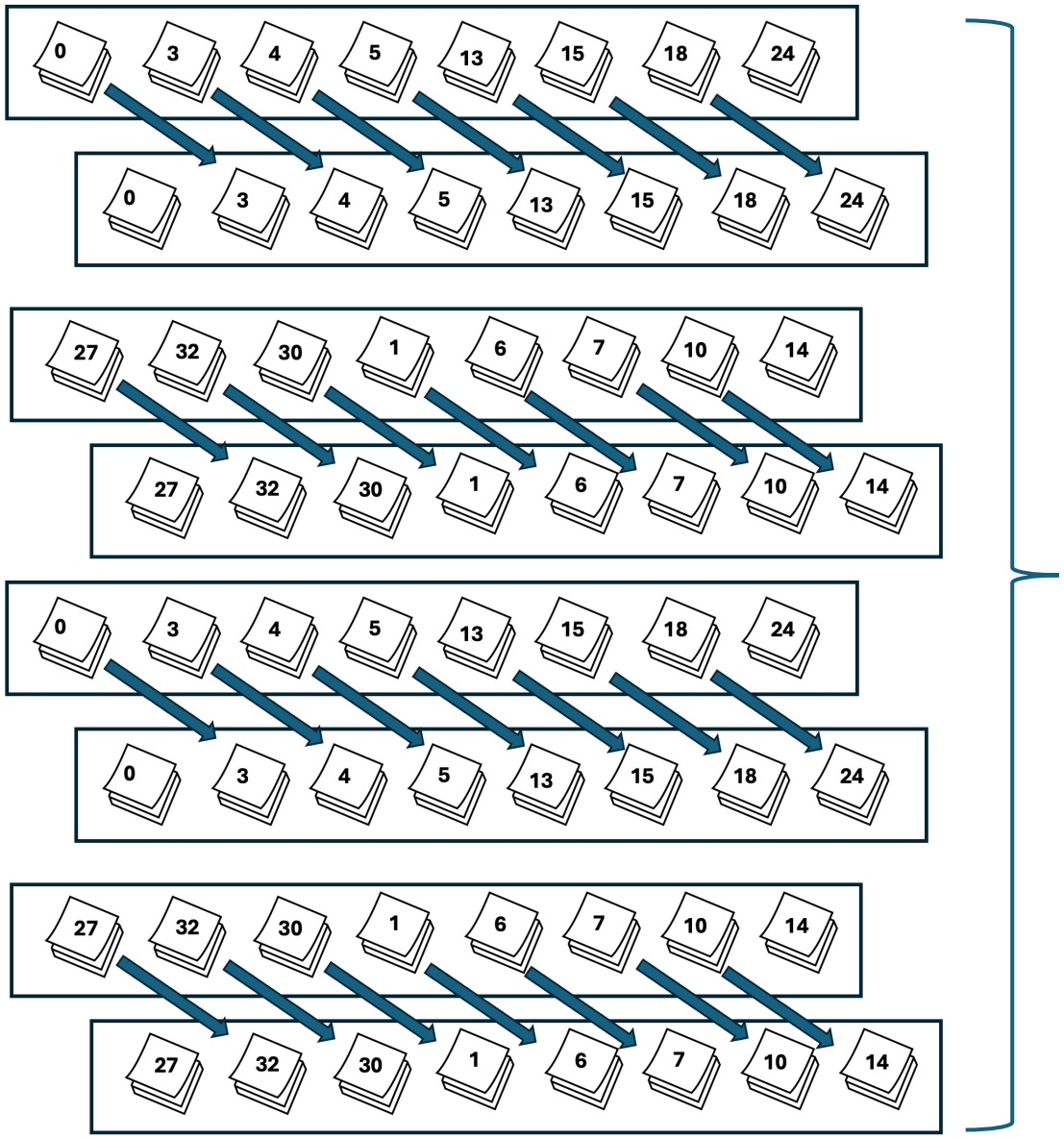




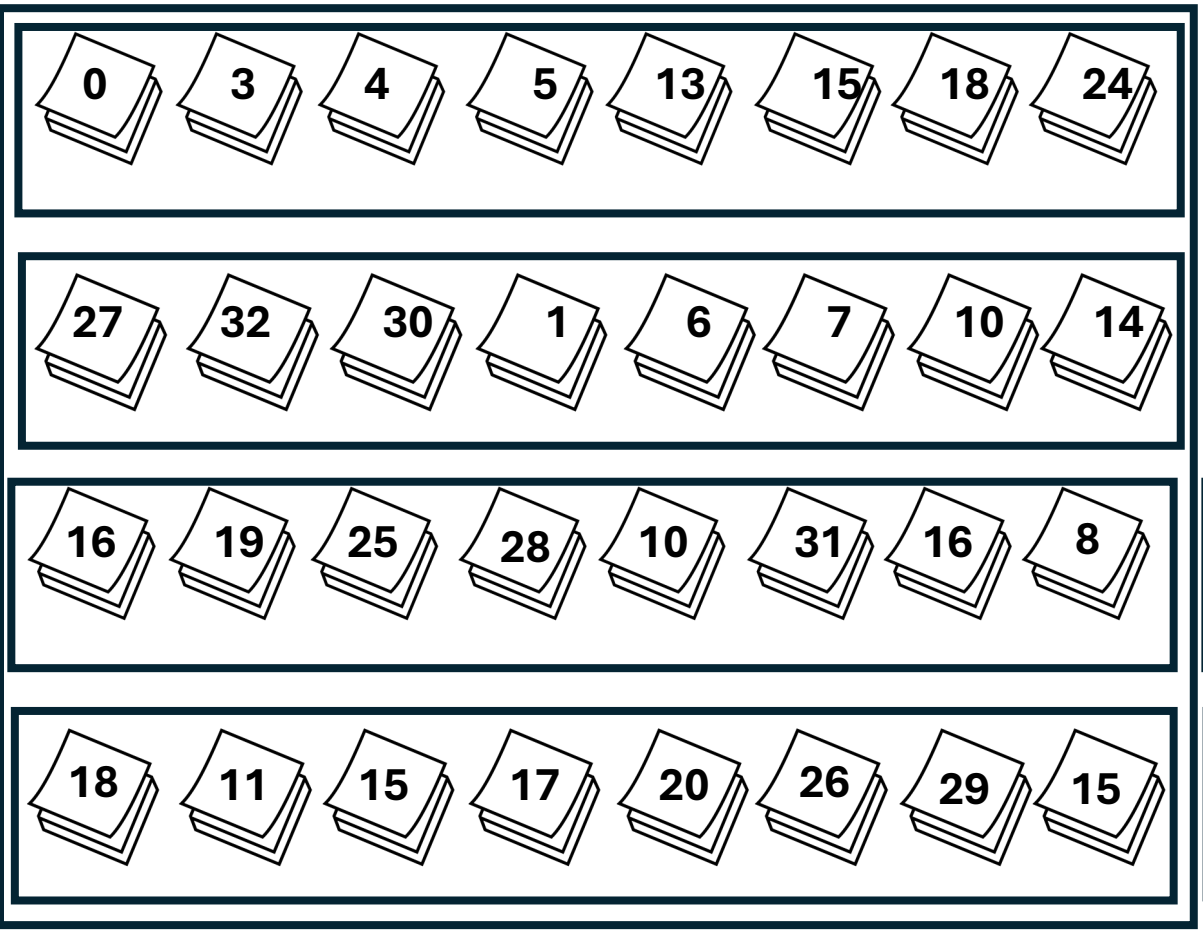
Texto codificado para Tokens


context
target
0
3
4
5
13
15
18
24
0
3
4
5
13
15
18
24

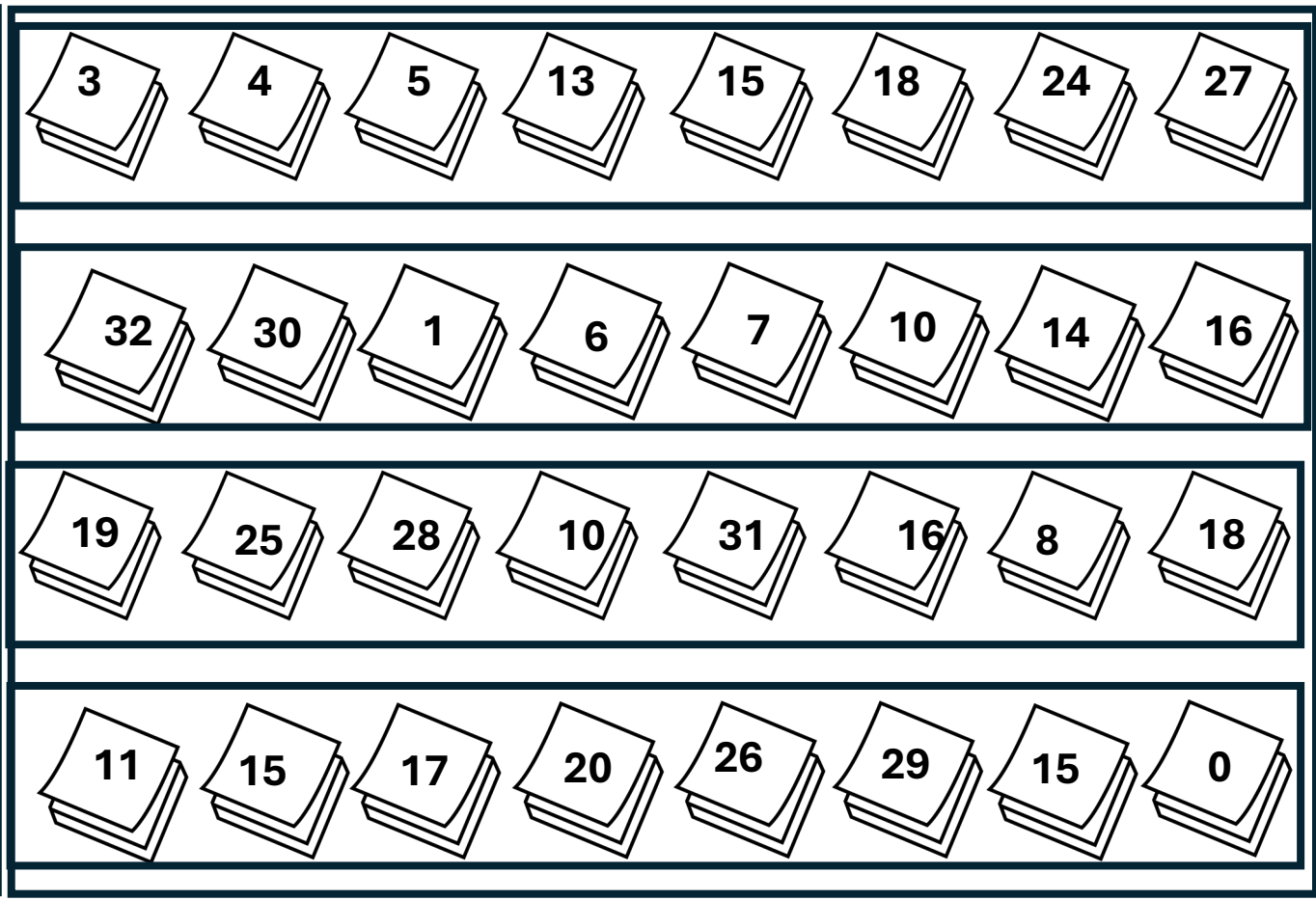

context
target
27
32
30
1
6
7
10
14
27
32
30
1
6
7
10
14



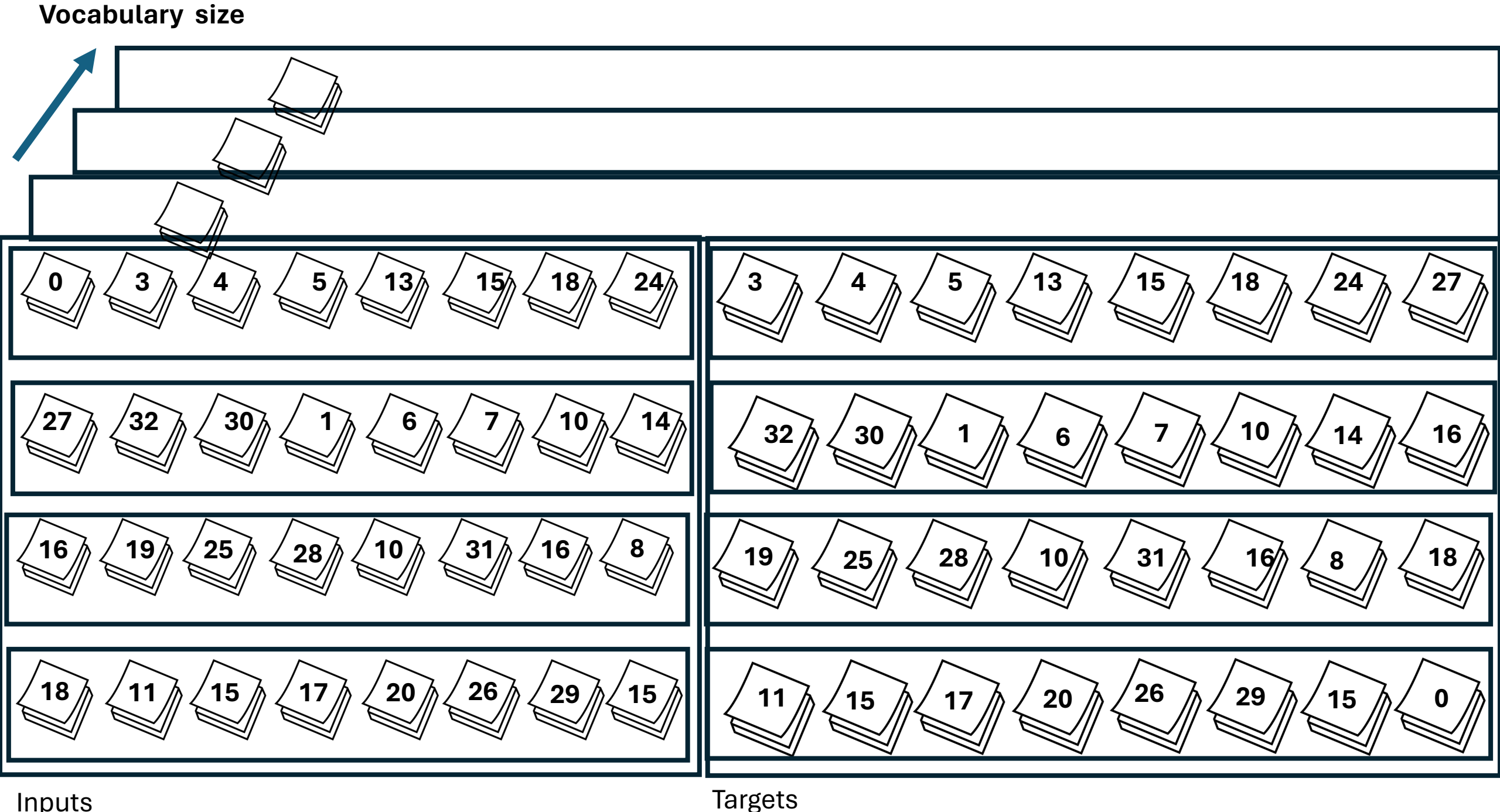
Block size = 8 (tamanho dos tokens na entrada)
Batch size = 4 (processamento praralelo)



Inputs

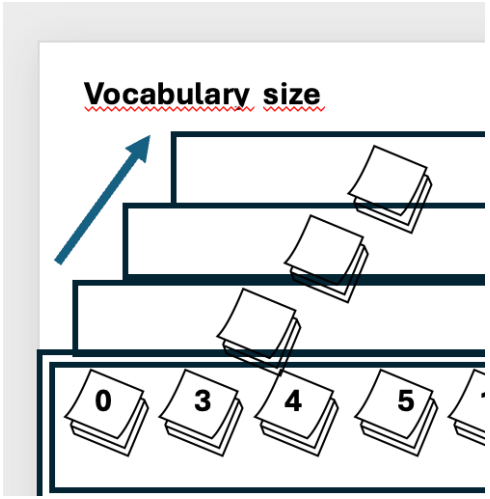


Targets


Vocabulary size
0 3 4 5 13 15 18 24
27 32 30 1 6 7 10 14
16 19 25 28 10 31 16 8
18 11 15 17 20 26 29 15
3 4 5 13 15 18 24 27
32 30 1 6 7 10 14 16
19 25 28 10 31 16 8 18
11 15 17 20 26 29 15 0
Inputs
Targets

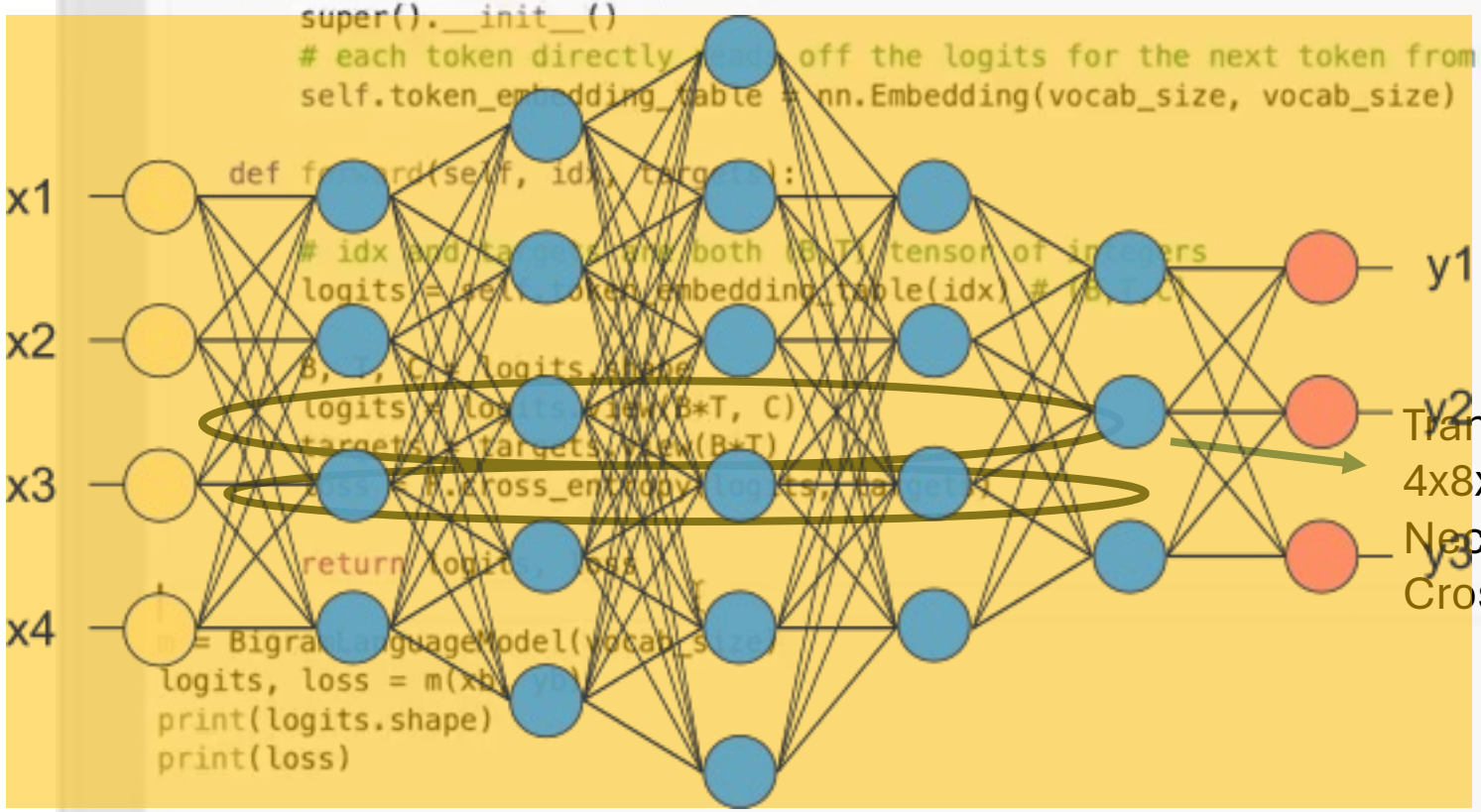
Cross entropy

Cross entropy is also considered a loss function. This loss function is nothing but the difference between the predicted probability distribution and the actual distribution, or true distribution.




```
from torch.nn import functional as F
torch.manual_seed(1337)

class BigramLanguageModel(nn.Module):

    def __init__(self, vocab_size):
        super().__init__()
        # each token directly reads off the logits for the next token from a lookup table
        self.token_embedding_table = nn.Embedding(vocab_size, vocab_size)

    def forward(self, idx, targets=None):
        # idx and targets are both (B,T) tensor of integers
        logits = self.token_embedding_table(idx) # (B,T,C)
        B, T, C = logits.shape
        logits = logits.view(B*T, C)
        targets = targets.view(B*T)
        loss = F.cross_entropy(logits, targets)

        return logits, loss

m = BigramLanguageModel(vocab_size)
logits, loss = m(xb, yb)
print(logits.shape)
print(loss)
```

```
torch.Size([32, 65])
tensor(4.8786, grad_fn=<NllLossBackward0>)
```



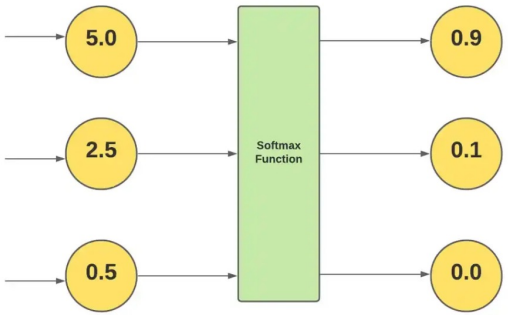
Transforma um tensor de 4x8x65 para 32x65, que é Necessário para a função Cross_entropy

Para cada elemento de input, obtemos o target e calculamos a cross antropy, que é a probabilidade de um input estar gerando um target corretamente, comparado com todos os outros itens do vocabulário

Softmax e multinomial

```
    def generate(self, idx, max_new_tokens):
        # idx is (B, T) array of indices in the current context
        for _ in range(max_new_tokens):   # get the predictions
            logits, loss = self(idx)      # focus only on the last time step - SELF CALLS forward
            logits = logits[:, -1, :] # becomes (B, C) -  apply softmax to get probabilities
            probs = F.softmax(logits, dim=-1) # (B, C)  # sample from the distribution
            idx_next = torch.multinomial(probs, num_samples=1) # (B, 1) - append sampled index to the running sequen
            idx = torch.cat((idx, idx_next), dim=1) # (B, T+1)
        return idx

m = BigramLanguageModel(vocab_size)
print(decode(m.generate(torch.zeros((1, 1), dtype=torch.long) , max_new_tokens=50)[0].tolist()))
```

Softmax – A função softmax é uma função que converte um vetor de valores reais K em um vetor de valores reais K que somam 1.



Multinomial – retorna elementos de um conjunto de probabilidades ordenados pelas maiores probabilidades

```
In [199]: print(torch.multinomial(torch.tensor([0,0,5,2], dtype=torch.float64), num_samples=1))
tensor [2]
```

Segundo elemento é a maior "probabilidade"

Quantos elementos desejo

Softmax e multinomial

```
    def generate(self, idx, max_new_tokens):
        # idx is (B, T) array of indices in the current context
        for _ in range(max_new_tokens):    # get the predictions
            logits, loss = self(idx)       # focus only on the last time step - SELF CALLS forward
            logits = logits[:, -1, :] # becomes (B, C) -  apply softmax to get probabilities
            probs = F.softmax(logits, dim=-1) # (B, C)  # sample from the distribution
            idx_next = torch.multinomial(probs, num_samples=1) # (B, 1) - append sampled index to the running sequen
            idx = torch.cat((idx, idx_next), dim=1) # (B, T+1)
        return idx

m = BigramLanguageModel(vocab_size)
print(decode(m.generate(torch.zeros((1, 1), dtype=torch.long) , max_new_tokens=50)[0].tolist()))
```

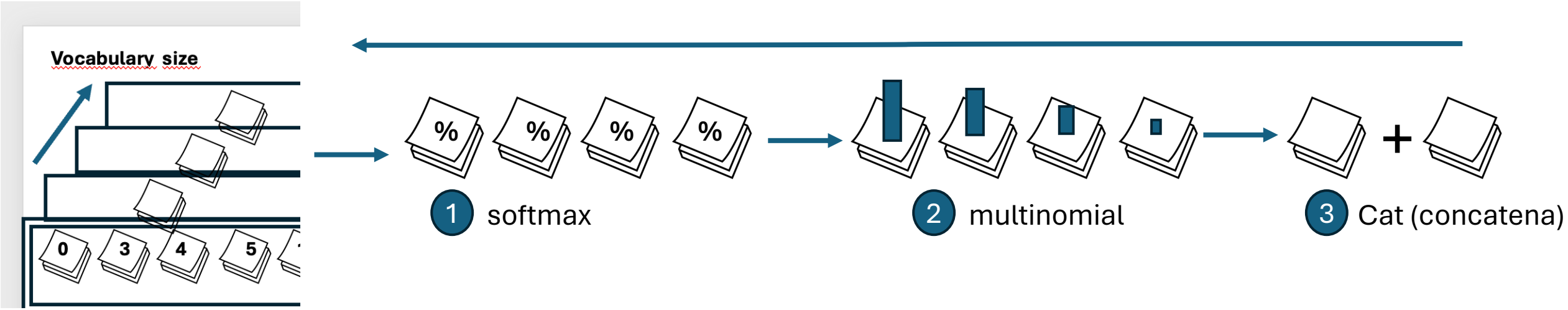
Self aciona forward

1

2

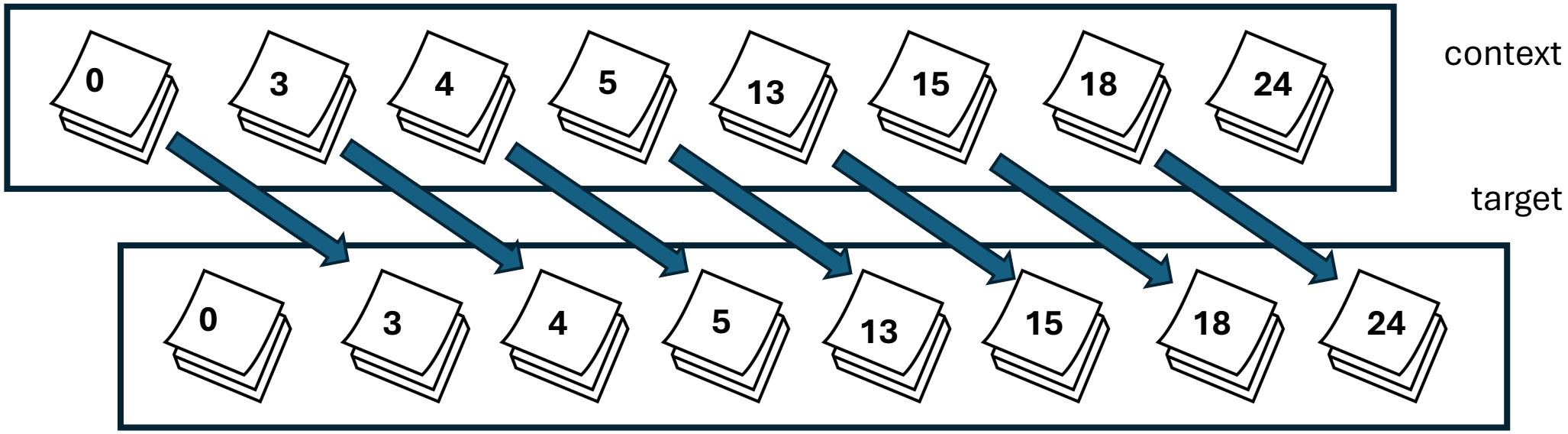
3

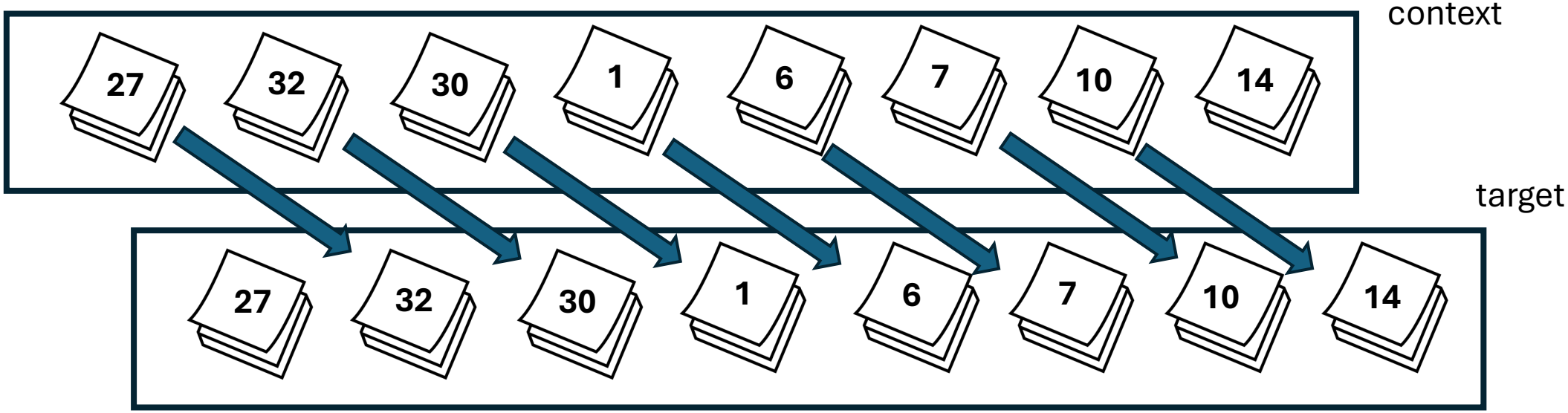
SKIcLT;AcELMoTbvZv C?nq-QE33:CJqkOKH-q;:la!oiywkHj

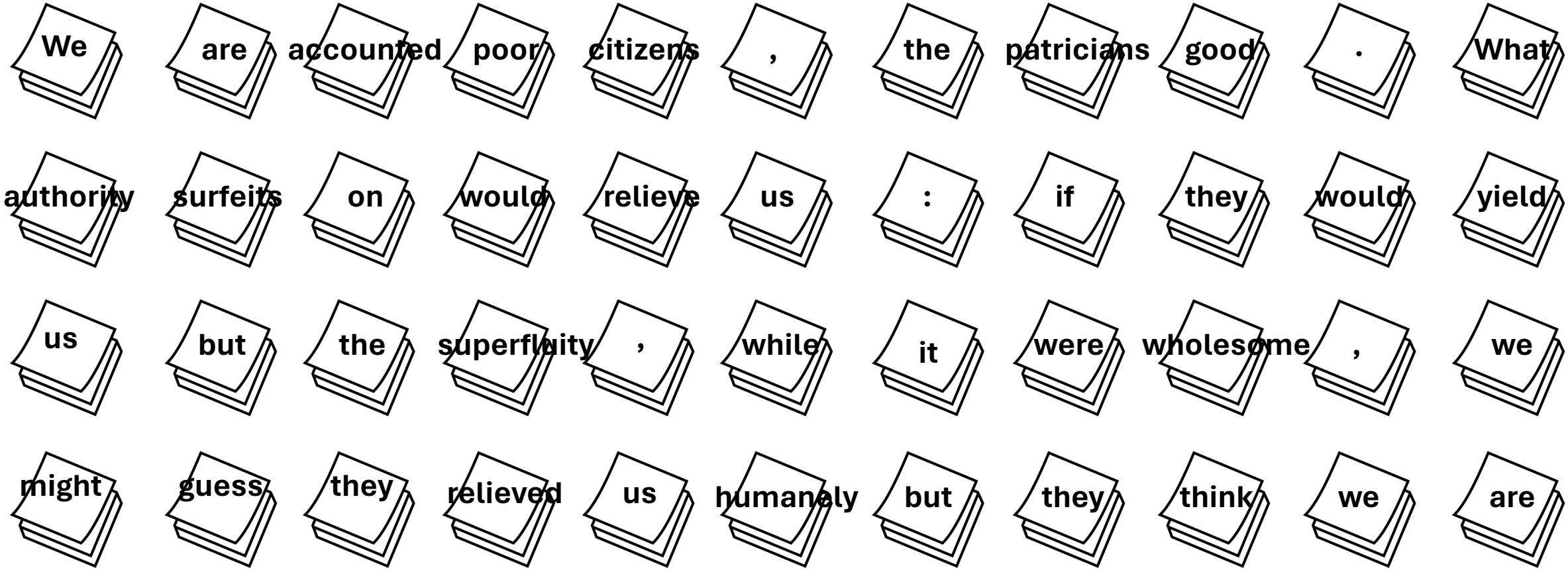


Transforma um array de próximas palavras em probabilidades, retorna o elemento de maior probabilidade e concatena com o resultado anterior

Optimizer


context
target
0
3
4
5
13
15
18
24
0
3
4
5
13
15
18
24


context
target
27
32
30
1
6
7
10
14
27
32
30
1
6
7
10
14


We are accounted poor citizens , the patricians good . What
authority surfeits on would relieve us : if they would yield
us but the superfluity , while it were wholesome , we
might guess they relieved us humanely but they think we are