Deep Learning models for ASR

3. Transformer audio-to-text

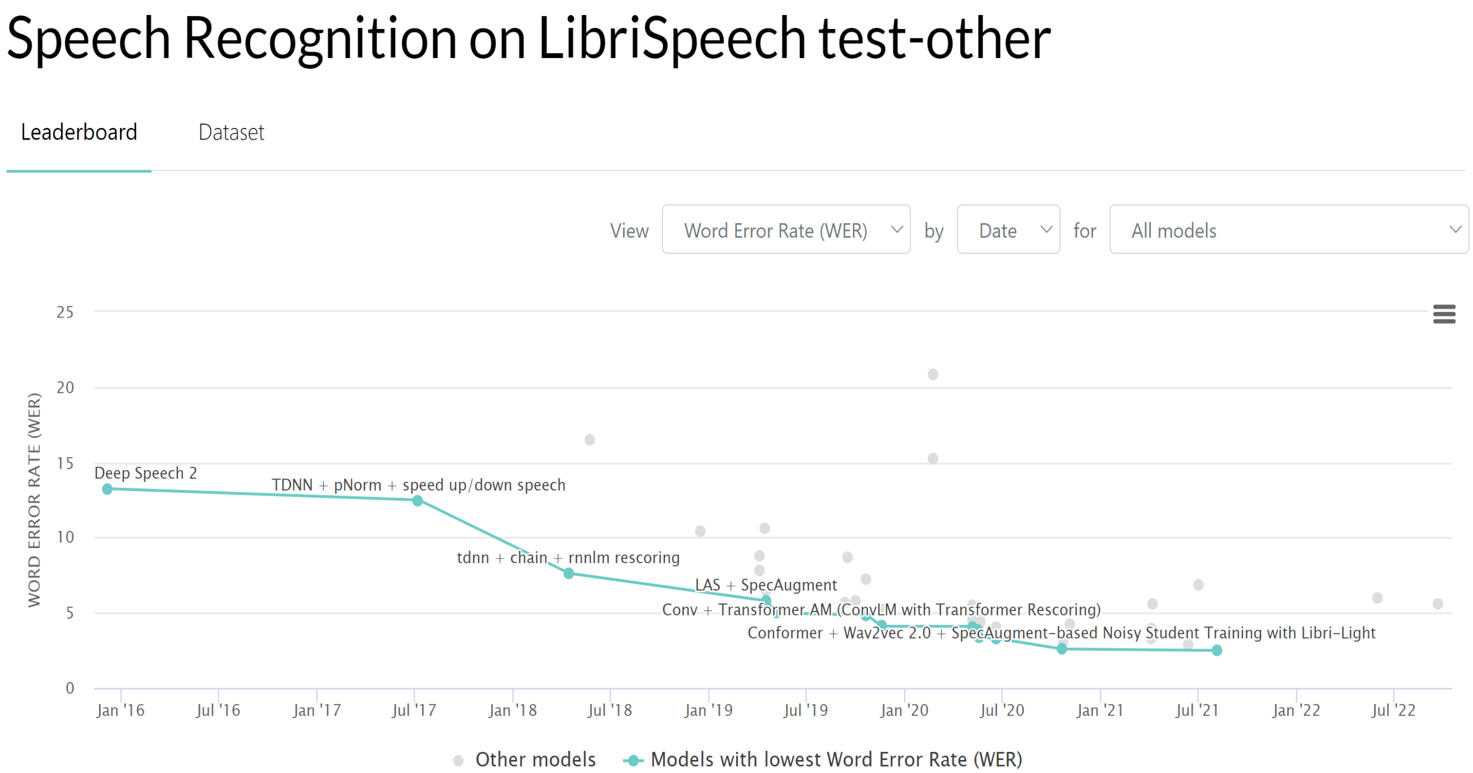
Summary

- **Audio-to-text transformer**
- Optimizations:
  - Easy implementation: einops, rearrange
  - Rotary positional encoding
  - Flash Attention
  - Sliding window attention

ASR: benchmarks

- **Automatic Speech Recognition ASR**

https://paperswithcode.com/sota/speech-recognition-on-librispeech-test-other



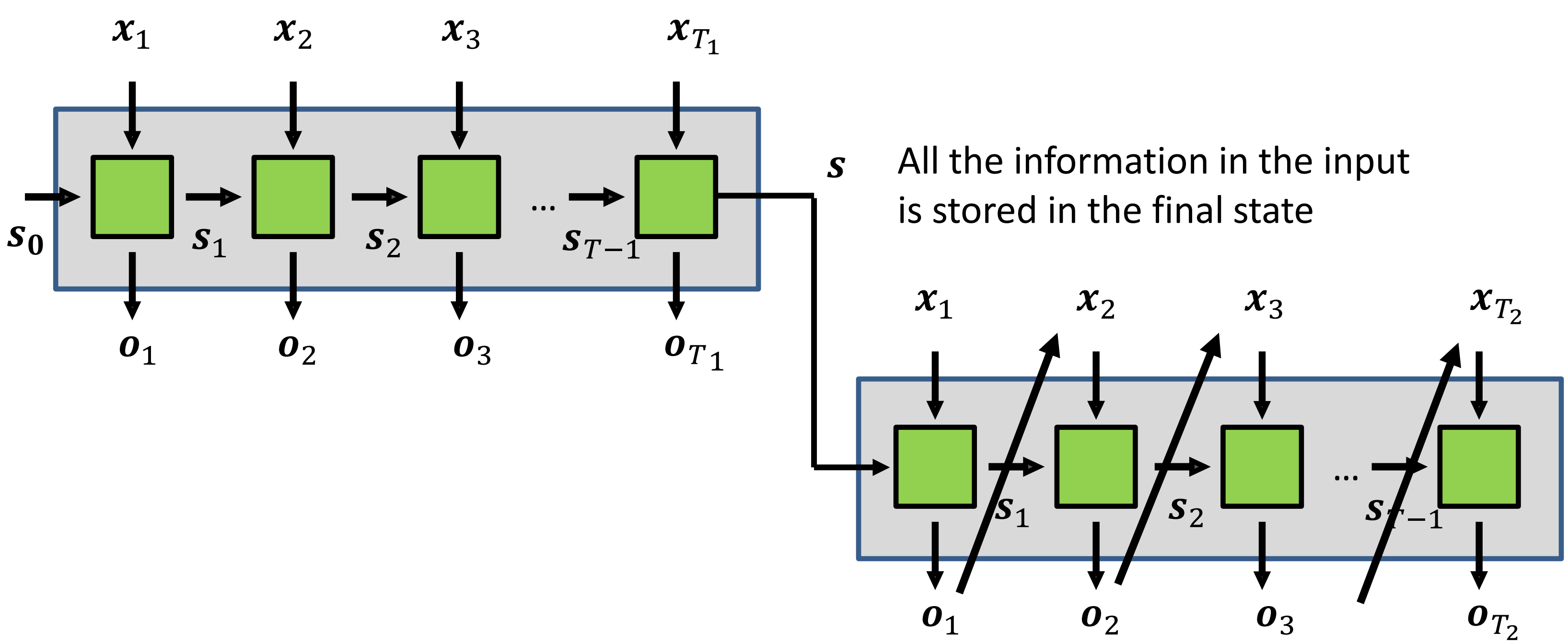
Non-natives reading book in english

In less than 5 five years
From 13.2% to 2.5% WER

ASR: seq2seq

- **Automatic Speech Recognition ASR**

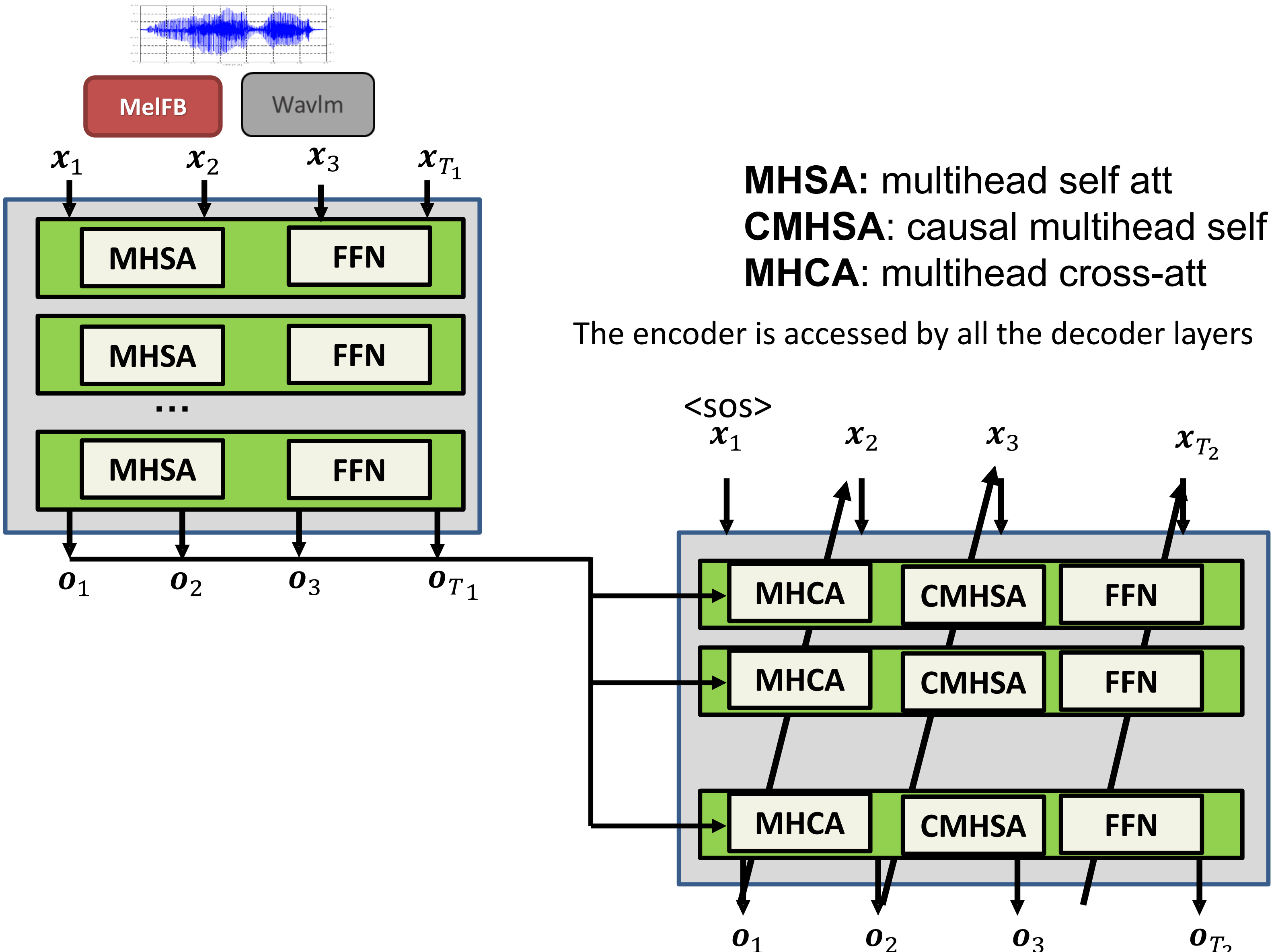
  - **Seq2seq RNN model:** state is a bottleneck
  - **Decoder:** uses previous outputs as input



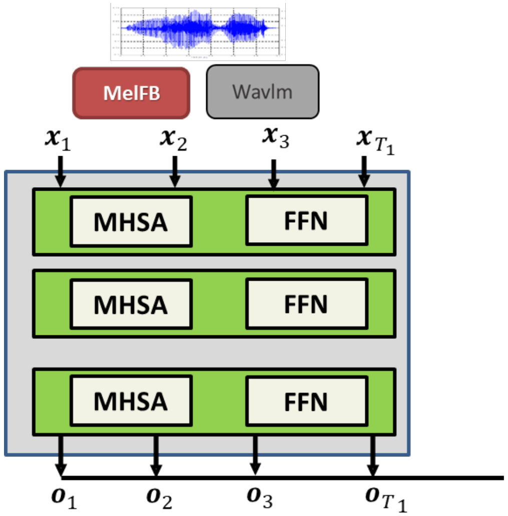
ASR: transformer speech-to-text

MHSA: multihead self att
CMHSA: causal multihead self att
MHCA: multihead cross-att

The encoder is accessed by all the decoder layers



ASR: transformer speech-to-text



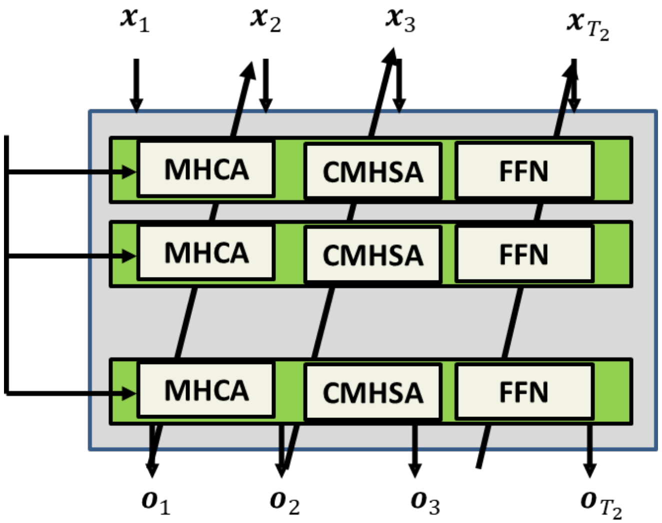

```python
class Encoder(torch.nn.Module):
    def __init__(self, nb_layers=6, seq_len=400, **kwargs):
        super().__init__()
        self.pos = torch.nn.Parameter(torch.randn(1, seq_len, kwargs['d_model']))
        self.att = torch.nn.ModuleList([SelfAttention(**kwargs) for _ in range(nb_layers)])
        self.ff = torch.nn.ModuleList([FeedForward(**kwargs) for _ in range(nb_layers)])

    def forward(self, x):
        b, t, d = x.shape
        x = x + self.pos[:, :t, :]
        for att, ff in zip(self.att, self.ff):
            x = x + att(x)
            x = x + ff(x)
        return x
```

ASR: transformer speech-to-text

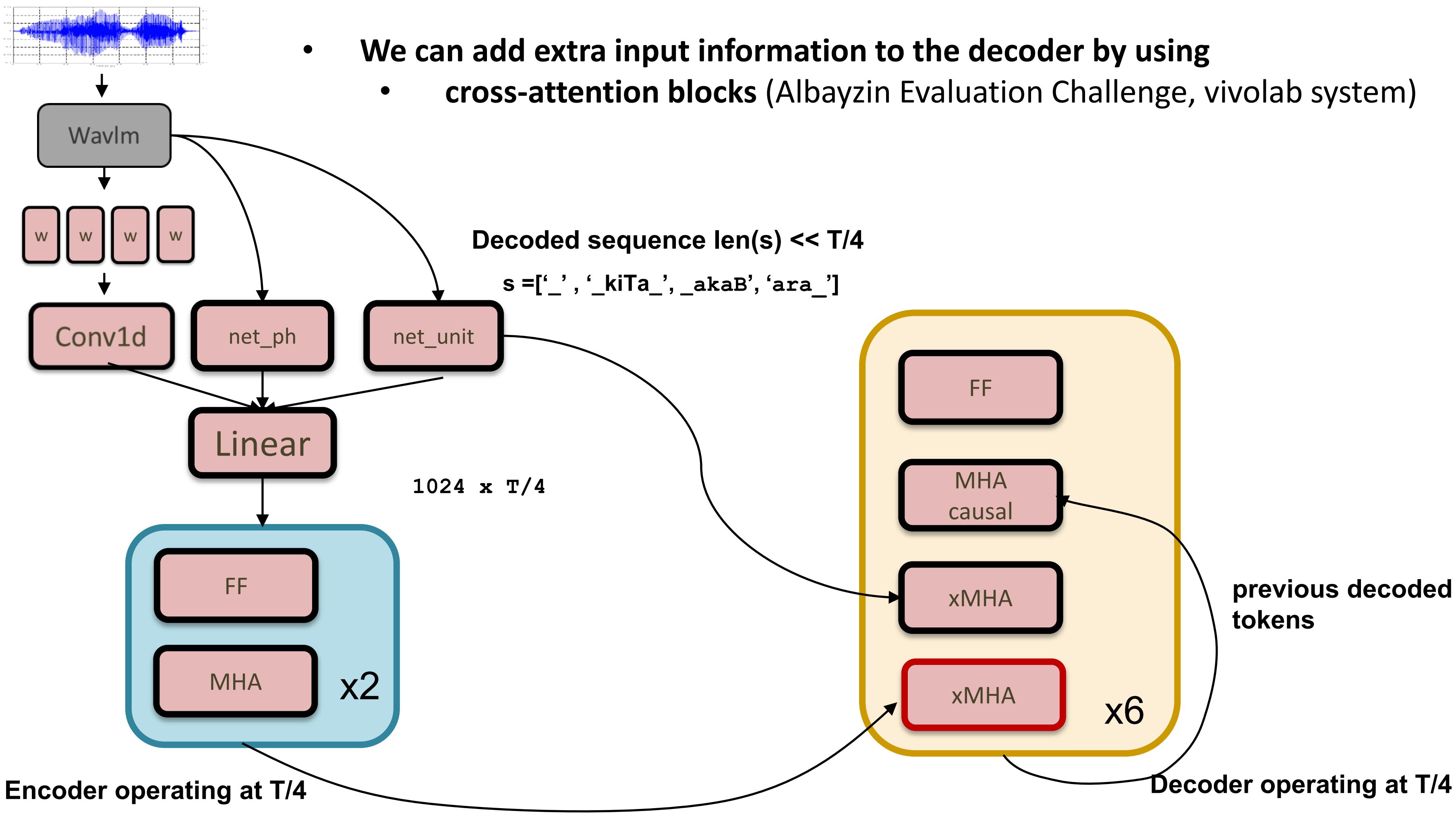
```
class Decoder(torch.nn.Module):
    def __init__(self, nb_layers=6, seq_len=400, **kwargs):
        super().__init__()
        self.pos = torch.nn.Parameter(torch.randn(1, seq_len, kwargs['d_model']))
        self.att = torch.nn.ModuleList([CausalSelfAttention(**kwargs) for _ in range(nb_layers)])
        self.cross_att = torch.nn.ModuleList([CrossAttention(**kwargs) for _ in range(nb_layers)])
        self.ff = torch.nn.ModuleList([FeedForward(**kwargs) for _ in range(nb_layers)])

    def forward(self, x, enc):
        b, t, d = x.shape
        x = x + self.pos[:, :t, :]
        for att, cross_att, ff in zip(self.att, self.cross_att, self.ff):
            x = x + att(x)
            x = x + cross_att(x, enc)[0]
            x = x + ff(x)
        return x
```



Models

- **Automatic Speech Recognition ASR: extra inputs**
  - **We can add extra input information to the decoder by using**
    - **cross-attention blocks** (Albayzin Evaluation Challenge, vivolab system)

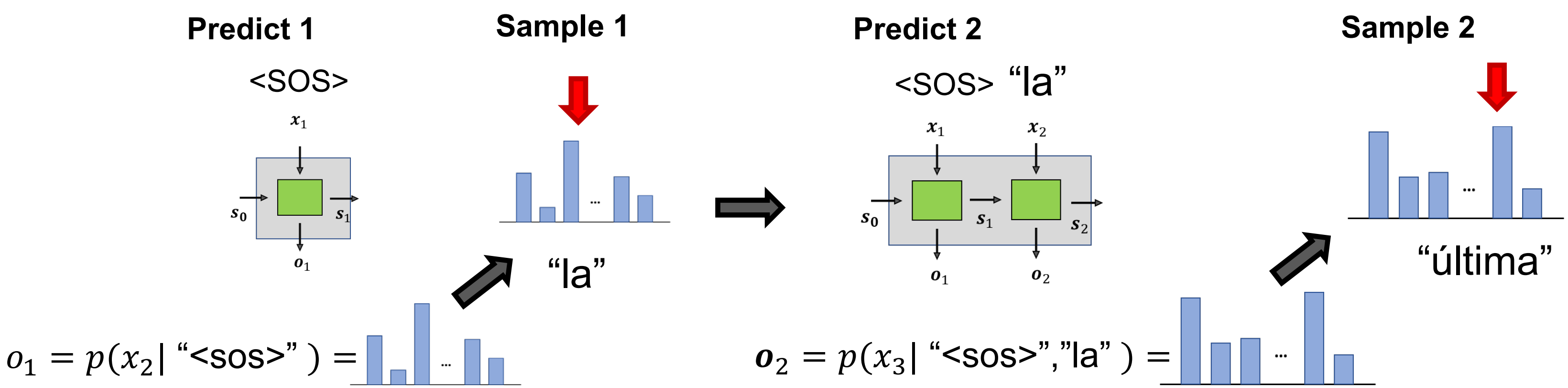


Summary

- **Audio-to-text transformer**
  - **Generation/decoding**
  - Attention
- Optimizations:
  - Easy implementation: einops, rearrange
  - Rotary positional encoding
  - Flash Attention
  - Slide attention

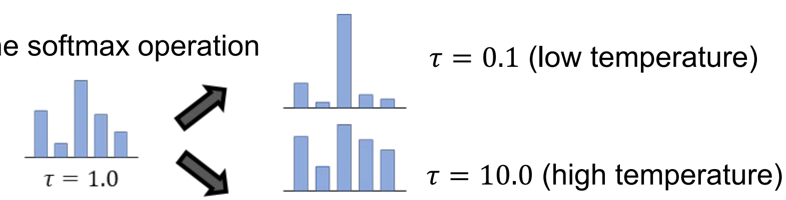
Autoregressive models

- **How do we generate samples ?**
  - **Predict**: We use the probability distribution provided by the model at time t given the previous context to obtain the discrete distribution: $p(x_t|x_1^{t-1})$
  - **Sample**: We sample from the distribution:$x_t \sim p(x_t|x_1^{t-1})$
    - Problem: **slow generation**



- The distributions can be controlled to follow the most probable symbols (low temperature) or a more uniform distribution high temperature
  - This effect is controlled by a simple scale before the softmax operation

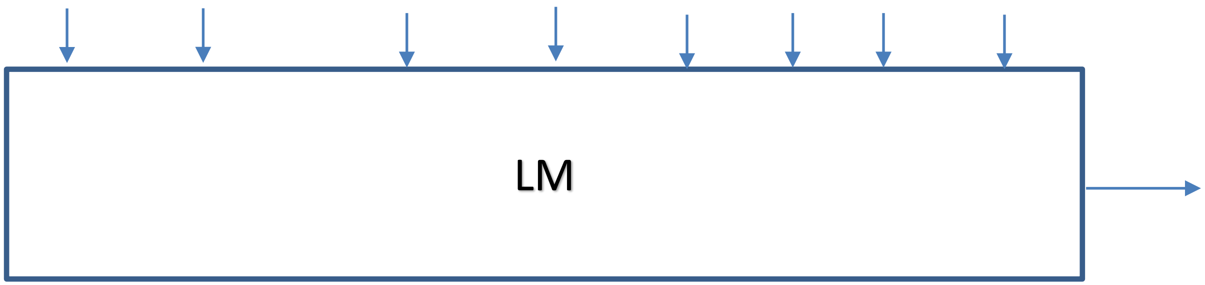
$$softmax_T(\boldsymbol{x}, \tau) = \frac{\exp x_c / \tau}{\sum_{c'} \exp x_{c'} / \tau}$$



Autoregressive models

- **Modelo de lenguaje: partes de palabra (tokens)**

_La _nueva _resol ución _de _n aciones...



p(unidas | _La, _nueva, _resol, ución, _de, _n, aciones)

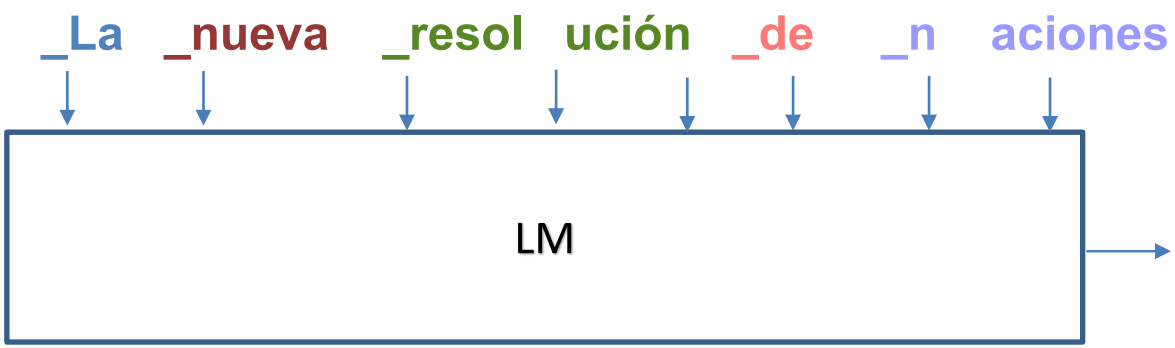
- *Can we do better than random sampling ?*

LLAMA 7B
32k tokens

```
token   | prob
------------
_un     | 0.48
_y      | 0.04
_en     | 0.03
_Un     | 0.03
_que    | 0.02
,       | 0.02
...
```

Autoregressive models

- **Modelo de lenguaje: generación**



LLAMA 7B
(two steps)

```
--------------------
_un (0.48) ->
       idas       (0.99)
       ific       (0.00)
       ip         (0.00)
       as         (0.00)
       id         (0.00)
--------------------
_y (0.04) ->
       _pue       (0.21)
       _g         (0.07)
       _raz       (0.04)
       _nacional  (0.04)
       _de        (0.04)
--------------------
_en (0.03) ->
       _la        (0.13)
       _el        (0.13)
       _conflic   (0.05)
       _las       (0.04)
       _Europa    (0.03)
```

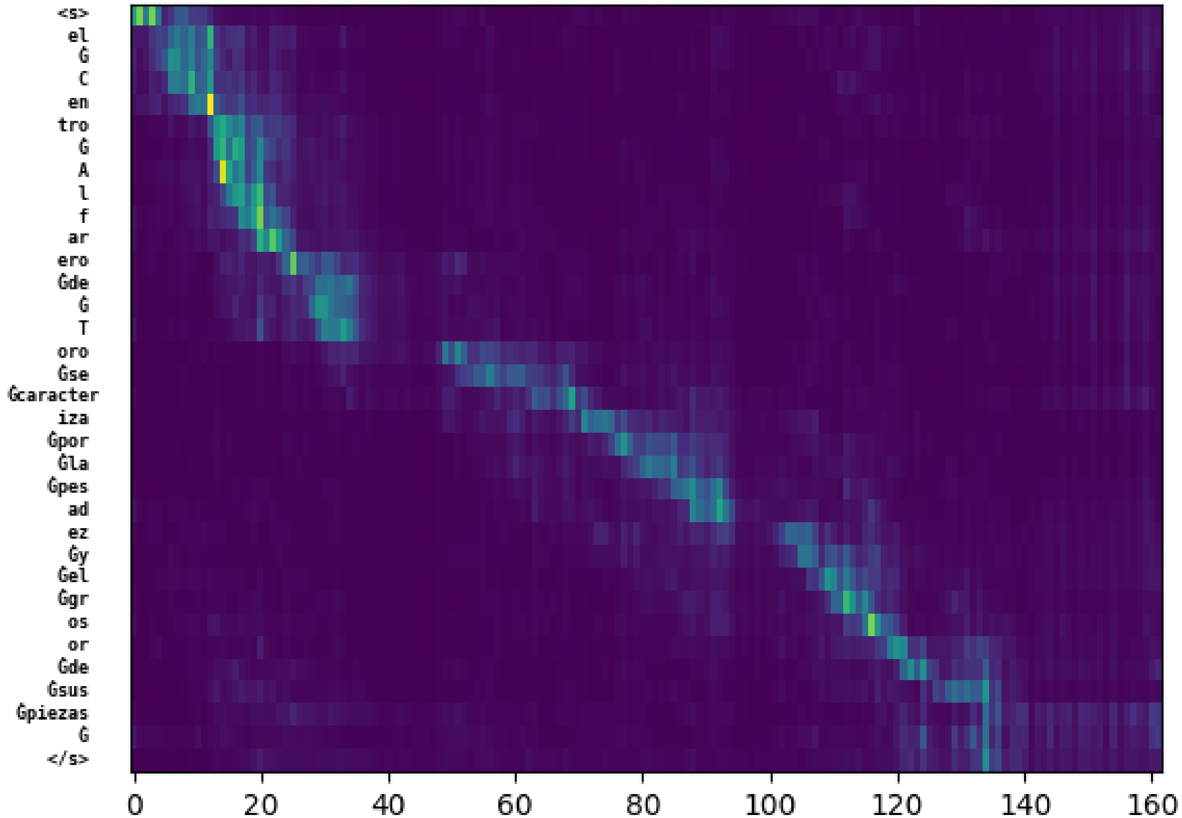
- *How to generate optimal sequences ?*
  - *Multiple active hypotheses + beam*
  - *For example with **beam 3**, active hyps in the example would be:*

***_un + idas** :*	*0.48 * 0.99 = 0.4752*
***_y + _pue**:*	*0.04 * 0.21 = 0.0084*
***_en + _la**:*	*0.03 * 0.13 = 0.0039*

Summary

- **Audio-to-text transformer**
  - Generation/decoding
  - **Attention**
- Optimizations:
  - Easy implementation: einops, rearrange
  - Rotary positional encoding
  - Flash Attention
  - Slide attention

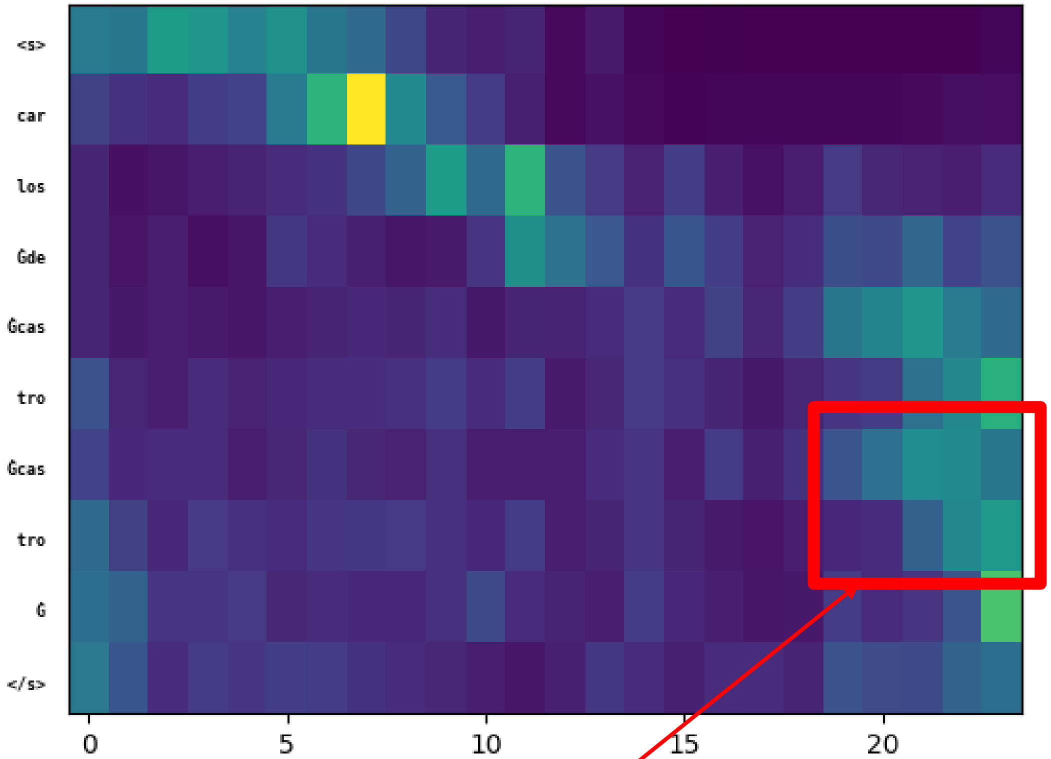
Cross-Attention



DG90323101

```
Scores: (#C #S #D #I) 28 0 0 0
REF:  el centro alfarero de toro se caracteriza por la pesadez y el grosor de sus piezas asi como por su botijo de carro que no tiene lado plano
HYP:  el centro alfarero de toro se caracteriza por la pesadez y el grosor de sus piezas asi como por su botijo de carro que no tiene lado plano
```

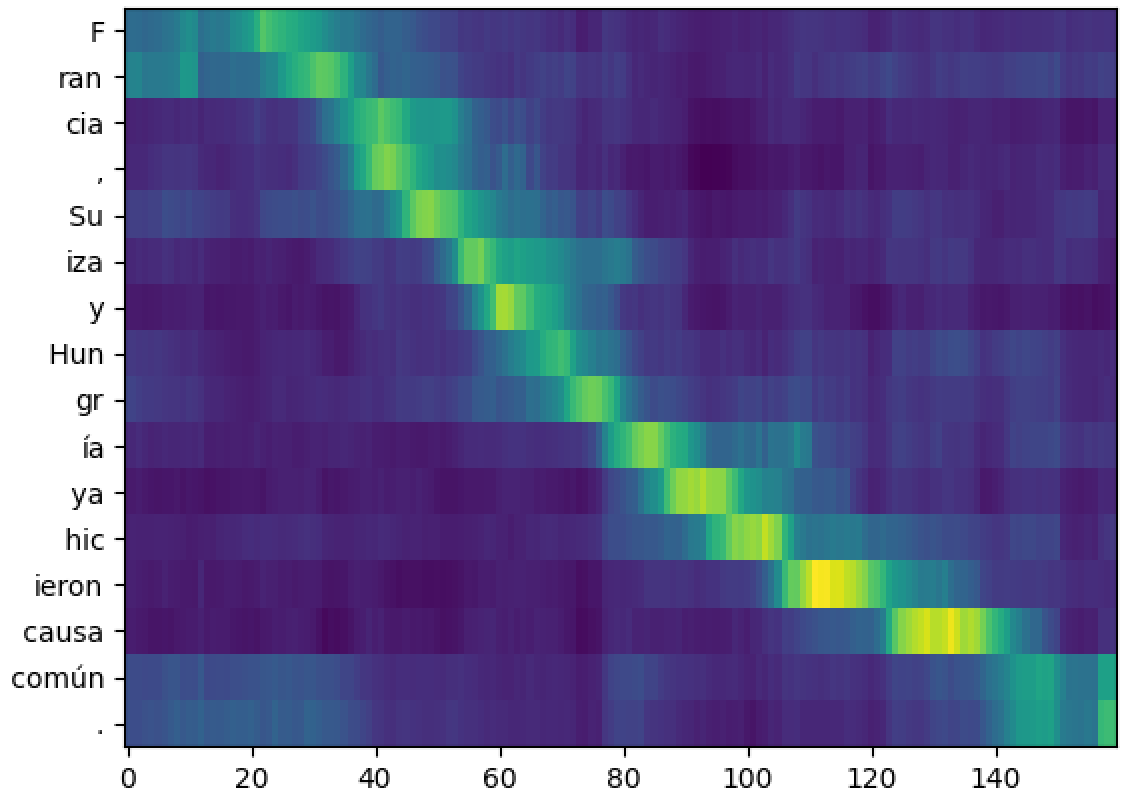
Cross-Attention



YR-C1

```
Scores: (#C #S #D #I) 3 0 0 1
REF:  carlos de castro ******
HYP:  carlos de castro CASTRO
```

Cross-Attention



whisper medium model

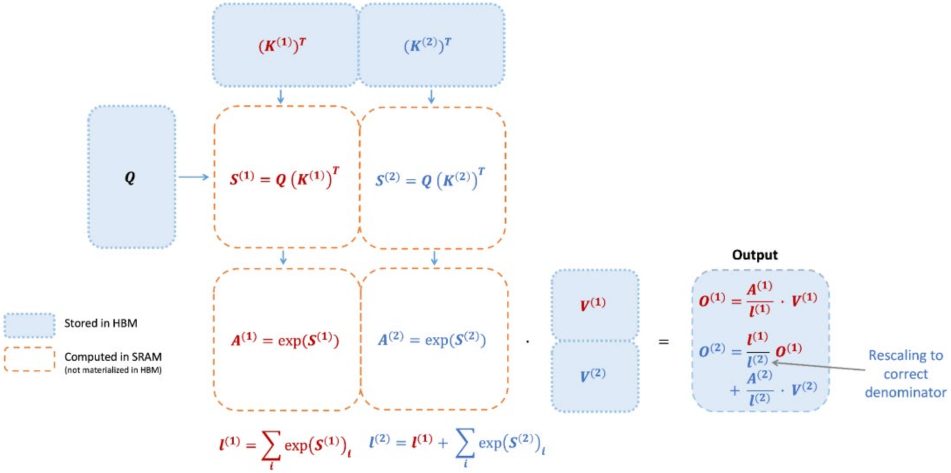
Summary

- Audio-to-text transformer
- **Optimizations**:
  - Easy implementation: einops, rearrange
  - Flash Attention
  - Rotary Positional Embedding
  - Sliding window attention

Transformer optimization

- **Flash attention, Flash attention2**

Dao, T., Fu, D., Ermon, S., Rudra, A., & Ré, C. (2022). Flashattention: Fast and memory-efficient exact attention with io-awareness. Advances in Neural Information Processing Systems, 35, 16344-16359.
Dao, T. (2023). Flashattention-2: Faster attention with better parallelism and work partitioning. arXiv preprint arXiv:2307.08691...



- Attention matrix is never computed completely: smaller tiles are computed for each query row -> output
- Very dependent on the GPU arquitecture, increased gains for modern GPUs small precision types like float16

Transformer optimization

- **Einops, rearrange**

  - Rearrange allows to execute multiple operations reshape, permute in a single call

```
q = self.to_q(q).view(b, -1, h, d)
k = self.to_k(k).view(b, -1, h, d)
v = self.to_v(v).view(b, -1, h, d)

q = q.permute(2, 0, 1, 3).contiguous().view(b*h, -1, d)
k = k.permute(2, 0, 1, 3).contiguous().view(b*h, -1, d)
v = v.permute(2, 0, 1, 3).contiguous().view(b*h, -1, d)
```

```
q = rearrange(self.to_q(q), ‘b t (h d) -> (b h) t d’, h=h)
k = rearrange(self.to_k(k), 'b t (h d) -> (b h) t d’, h=h)
v = rearrange(self.to_v(v), 'b t (h d) -> (b h) t d’, h=h)
```

  - Einops follows Einstein’s tensor notation and allows to calculate products, reductions…

```
scores = torch.matmul(q, k.transpose(-1,-2))
```

```
scores = torch.einsum('bihd,bjhd->bhij', q, k)
```

https://einops.rocks/pytorch-examples.html

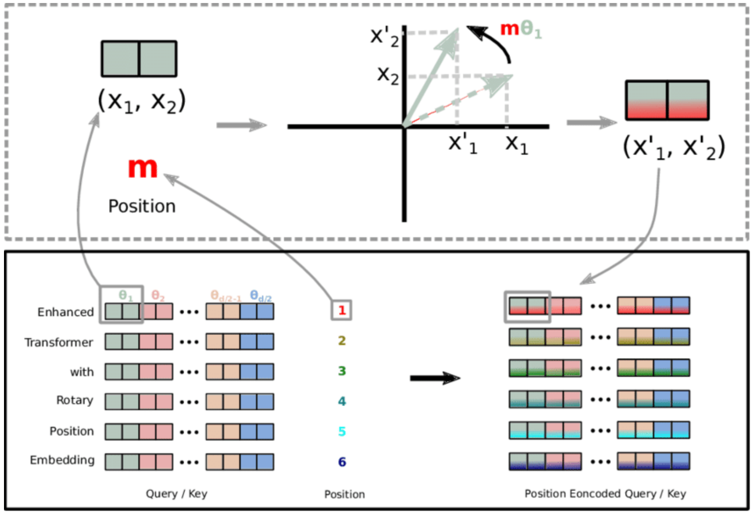
Transformer optimization

Rotary Positional Embedding (RoPE)

Su, J., Lu, Y., Pan, S., Murtadha, A., Wen, B., & Liu, Y. (2021). Roformer: Enhanced transformer with rotary position embedding. arXiv preprint arXiv:2104.09864..

- q and k sequences are modulated by a complex exponential
  - The inner product is equivalent to the scalar product and a term dependent on the relative difference of indices

$$
\begin{aligned}
\mathrm{RoPE}(x, m) &= x e^{mi\varepsilon} \\
\langle \mathrm{RoPE}(q_j, m), \mathrm{RoPE}(k_j, n) \rangle &= \langle q_j e^{mi\varepsilon}, k_j e^{ni\varepsilon} \rangle \\
&= q_j k_j e^{mi\varepsilon} \overline{e^{ni\varepsilon}} \\
&= q_j k_j e^{(m-n)i\varepsilon} \\
&= \mathrm{RoPE}(q_j k_j, m-n)
\end{aligned}
$$



- Useful for flash attention type models:
  - applied to q and k before product

Transformer optimization

Sliding window attention

Beltagy, I., Peters, M. E., & Cohan, A. (2020). Longformer: The long-document transformer. arXiv preprint arXiv:2004.05150....

	The	cat	sat	on	the
The	1	0	0	0	0
cat	1	1	0	0	0
sat	1	1	1	0	0
on	0	1	1	1	0
the	0	0	1	1	1

- It is implemented in flash attention2
- Similarly to causal conv1d (i.e. wavenet) the attention mask defines a finite distance attention. Computational gains and still good result. Example the recent LLM mistral
  - https://github.com/mistralai/mistral-src