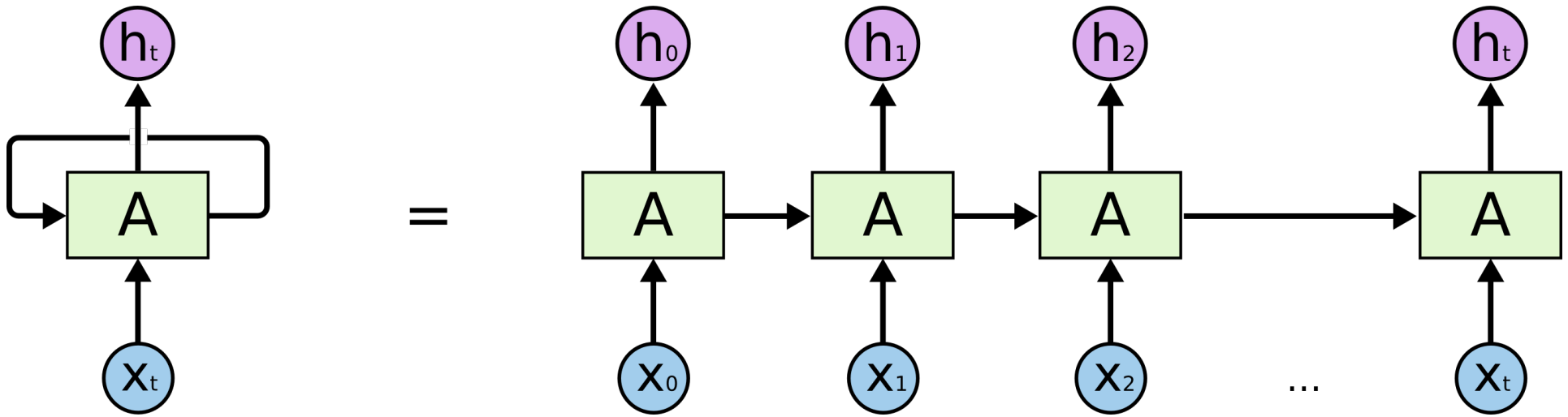
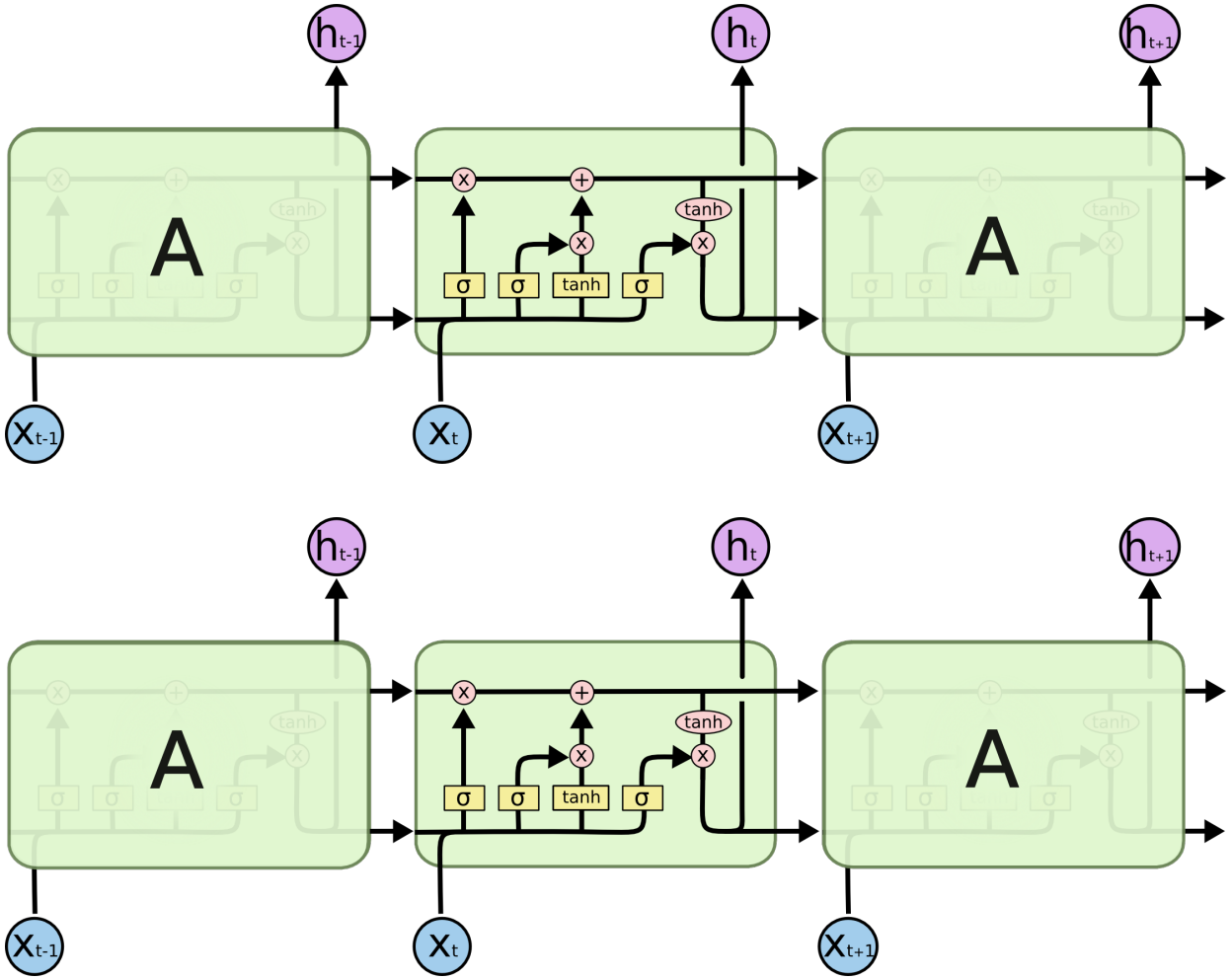
RNN or Transformer

RWKV新结构解析

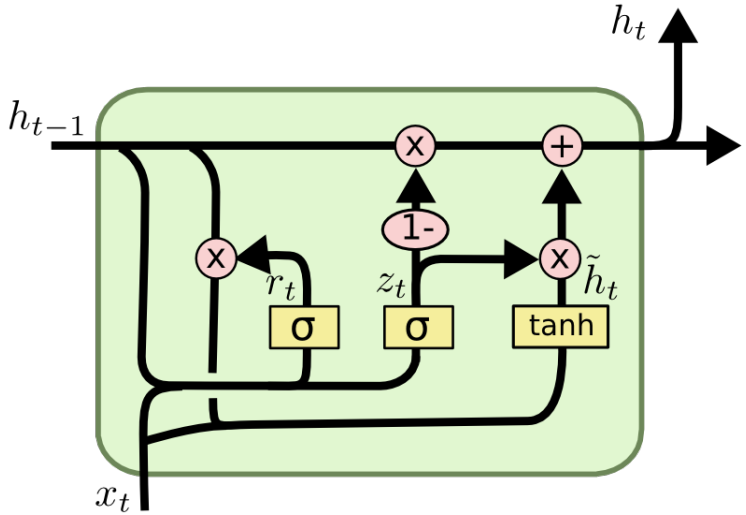
从RNN到LSTM



从RNN到LSTM


h_{t-1}
h_t
h_{t+1}
A
A
tanh
σ
σ
tanh
σ
x_{t-1}
x_t
x_{t+1}
h_{t-1}
h_t
h_{t+1}
A
A
tanh
σ
σ
tanh
σ
x_{t-1}
x_t
x_{t+1}

GRU



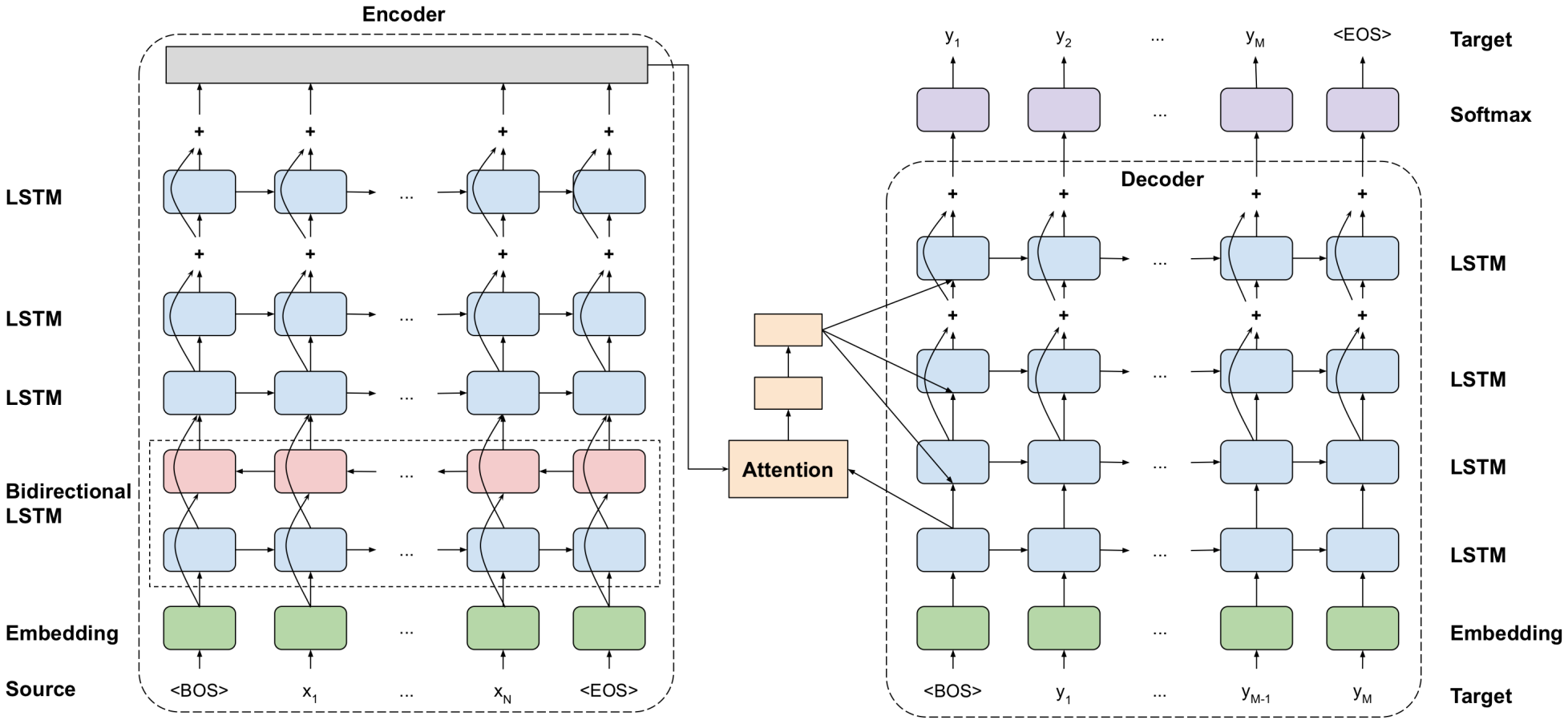
$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

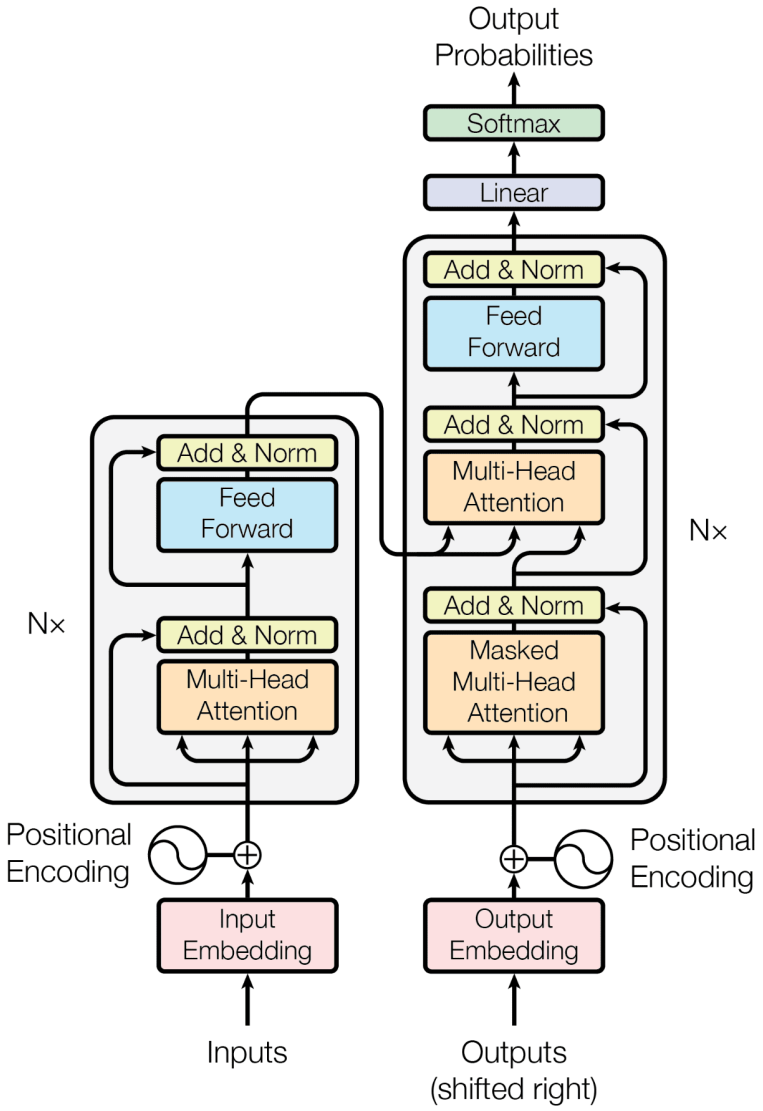
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

GNMTv2


Encoder
Decoder
Attention
LSTM
LSTM
LSTM
Bidirectional LSTM
Embedding
Source
<BOS>
x_1
...
x_N
<EOS>
y_1
y_2
...
y_M
<EOS>
Target
Softmax
LSTM
LSTM
LSTM
LSTM
Embedding
Target
<BOS>
y_1
...
y_{M-1}
y_M

Attention is all you need

- RNN不能并行训练问题
- 参数量
- 信息抽取能力



Self-attention复杂度

$$A(Q,K,V)=\mathrm{Softmax}(QK^T)V$$

- $Q,K,V:n\times d$
- 相似度计算 QK^T ： $n\times d$ 与 $d\times n$ 运算，得到 $n\times n$ 矩阵，复杂度为 $\mathcal{O}(n^2d)$
- softmax计算：对每行做softmax，复杂度为 $\mathcal{O}(n)$ ，则n行的复杂度为 $\mathcal{O}(n^2)$
- 加权和： $n\times n$ 与 $n\times d$ 运算，得到 $n\times d$ 矩阵，复杂度为 $\mathcal{O}(n^2d)$

故最后self-attention的时间复杂度为 $\mathcal{O}(n^2d)$

对于受限的self-attention，每个元素仅能和周围 r 个元素进行交互，即和 r 个 d 维向量做内积运算，复杂度为 $\mathcal{O}(rd)$ ，则 n 个元素的总时间复杂度为 $\mathcal{O}(rnd)$

Multi-head attention复杂度

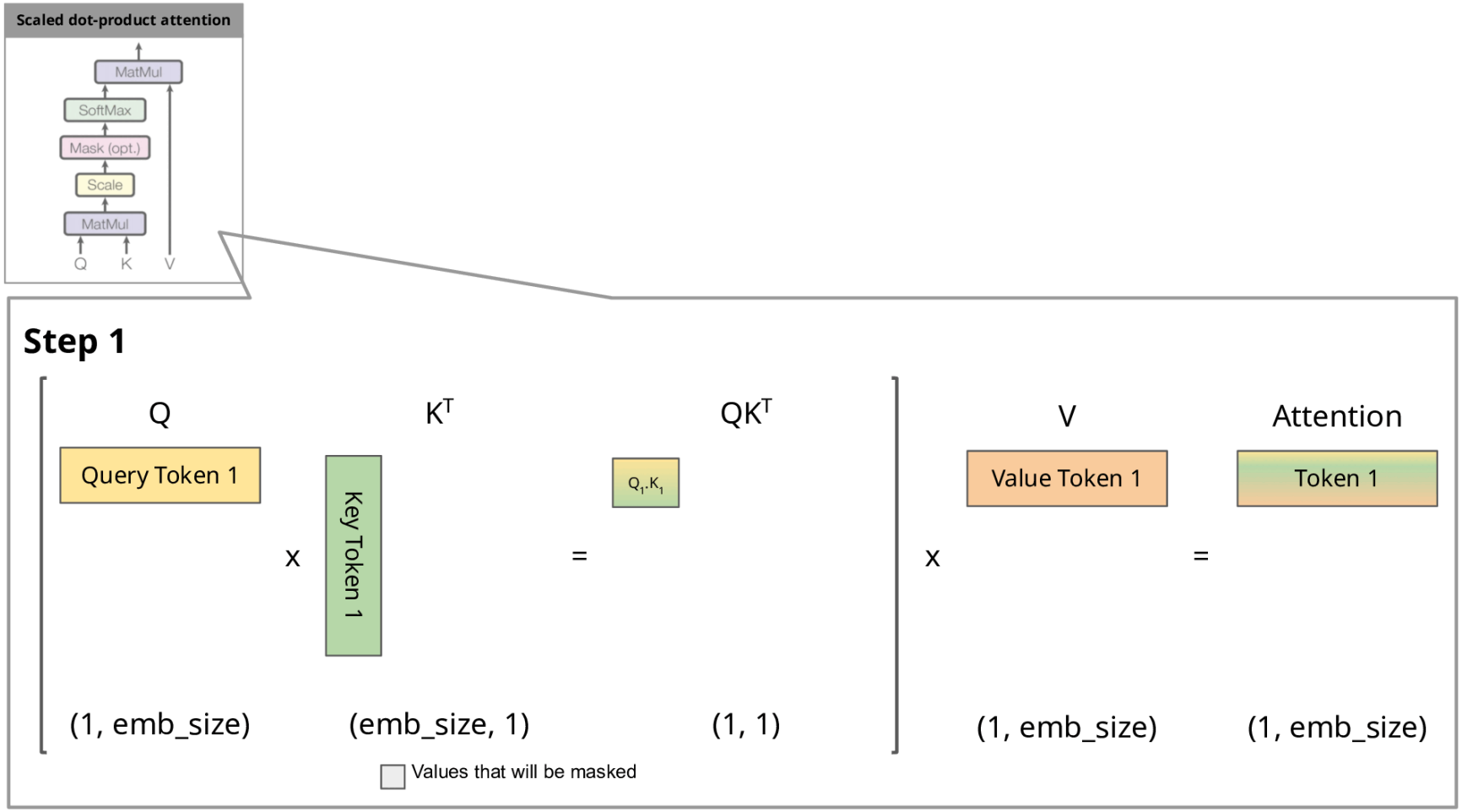
$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_\text{h})W^O$$
$$\text{where} \quad \text{head}_\text{i} = A(QW_i^Q, KW_i^K, VW_i^V)$$

对于multi-head attention，假设有 h 个head，这里 h 是一个常数，对于每个head，首先需要把三个矩阵分别映射到 d_q, d_k, d_v 维度。这里考虑一种简化情况： $d_q = d_k = d_v = \frac{d}{h}$ 。(对于dot-attention计算方式， d_k 与 d_v 可以不同)。

- 输入线性映射的复杂度： $n \times d$ 与 $d \times \frac{d}{h}$ 运算，忽略常系数，复杂度为 $\mathcal{O}(nd^2)$ 。
- Attention操作复杂度：主要在相似度计算及加权和的开销上， $n \times \frac{d}{h}$ 与 $\frac{d}{h} \times n$ 运算，复杂度为 $\mathcal{O}(n^2 d)$
- 输出线性映射的复杂度：concat操作拼起来形成 $n \times d$ 的矩阵，然后经过输出线性映射，保证输入输出相同，所以是 $n \times d$ 与 $d \times d$ 计算，复杂度为 $\mathcal{O}(nd^2)$

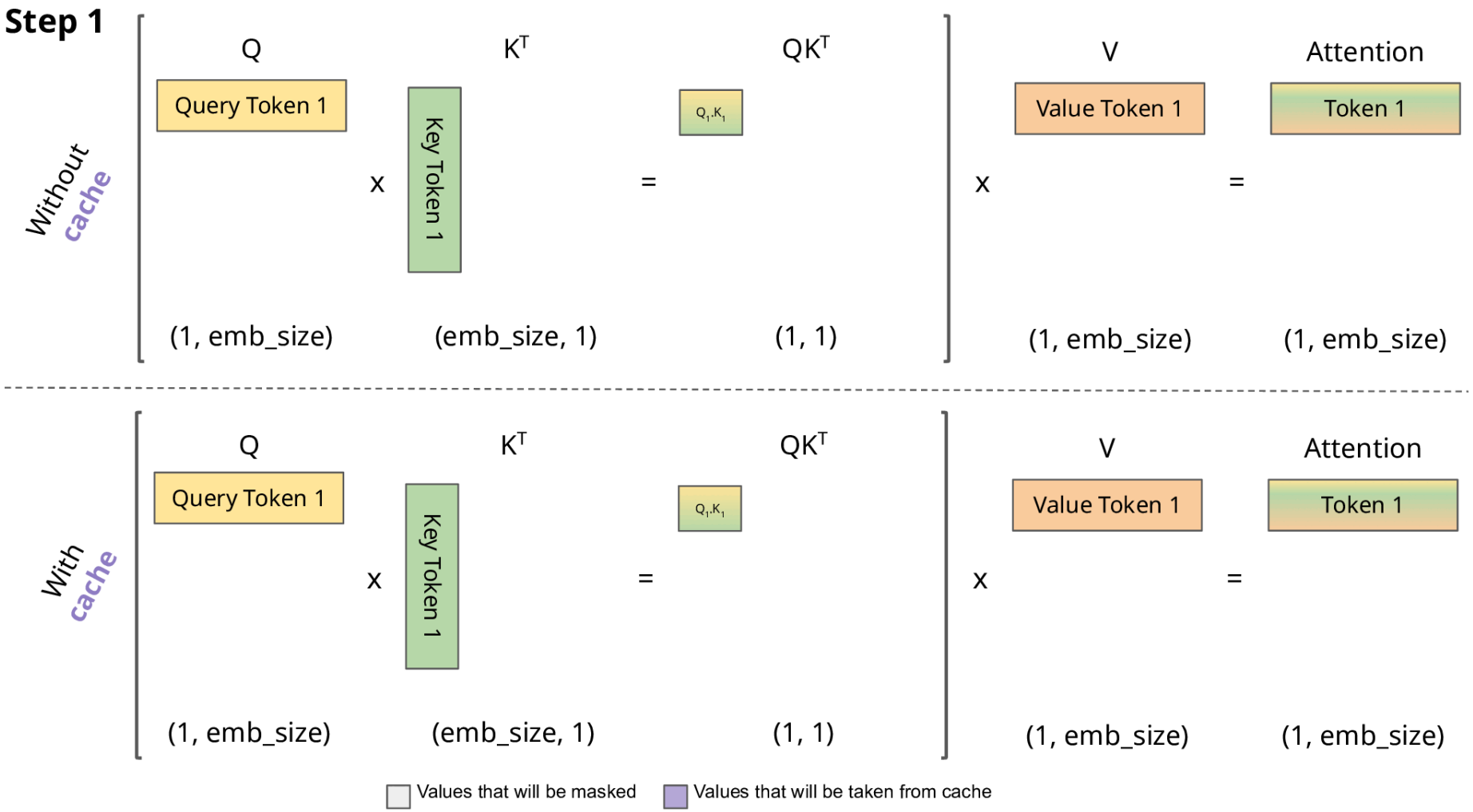
故最后的复杂度为： $\mathcal{O}(n^2 d + nd^2)$

推理效率问题

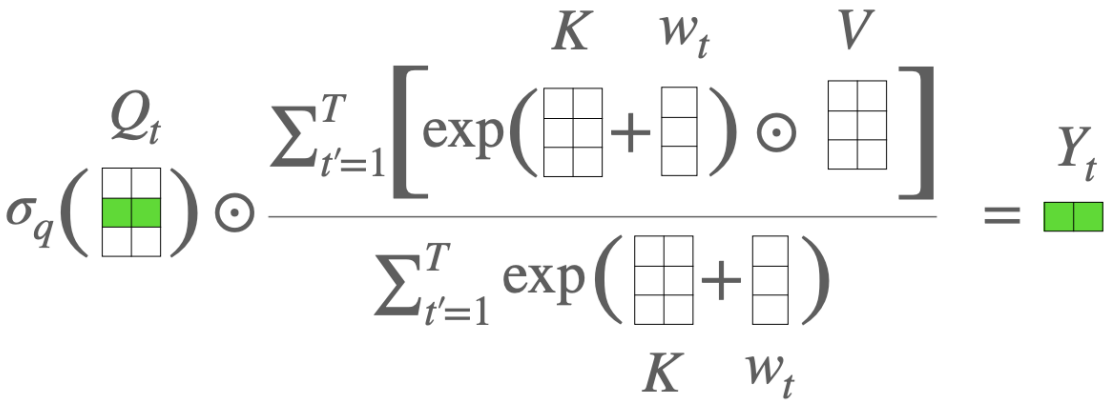


Zoom-in! (simplified without Scale and Softmax)

KV Cache


Step 1
Without cache
Q
K^T
QK^T
Query Token 1
Key Token 1
Q1.K1
x
=
V
Attention
Value Token 1
Token 1
x
=
(1, emb_size)
(emb_size, 1)
(1, 1)
(1, emb_size)
(1, emb_size)
With cache
Q
K^T
QK^T
Query Token 1
Key Token 1
Q1.K1
x
=
V
Attention
Value Token 1
Token 1
x
=
(1, emb_size)
(emb_size, 1)
(1, 1)
(1, emb_size)
(1, emb_size)
Values that will be masked
Values that will be taken from cache

Attention Free Transformer



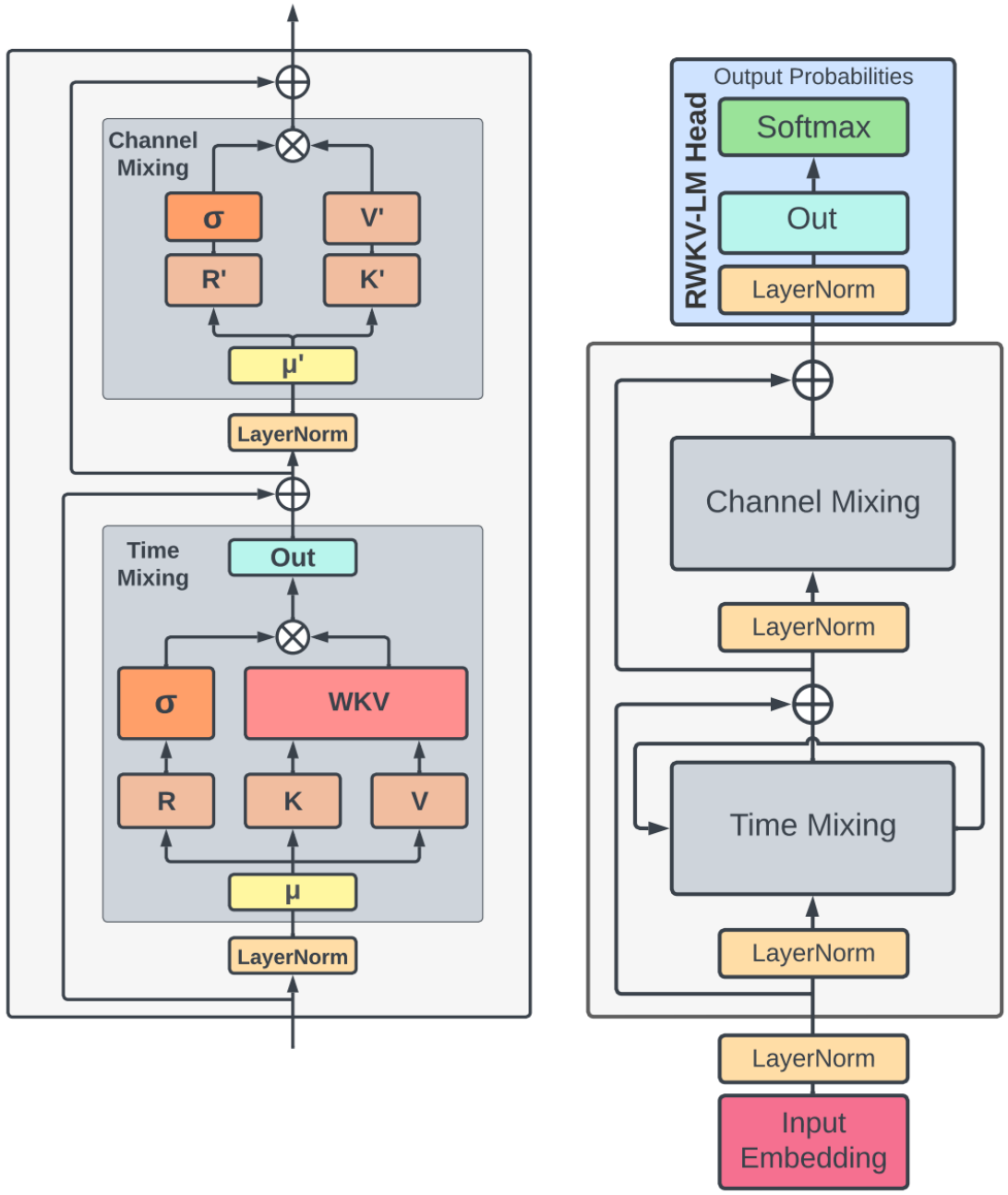
$$Y = f(X);\ Y_t = \sigma_q(Q_t) \odot \frac{\sum_{t'=1}^{T} \exp(K_{t'} + w_{t,t'}) \odot V_{t'}}{\sum_{t'=1}^{T} \exp(K_{t'} + w_{t,t'})}$$

where $\odot$ is the element-wise product; σ_q is the nonlinearity applied to the query with default being sigmoid; $w \in R^{T \times T}$ is the learned pair-wise position biases (see Figure 2 for an illustration).

RWKV

- Time Mixing
- Channel Mixing

- R: The **Receptance** vector acts as the receiver of past information.
- W: The **Weight** signifies the positional weight decay vector, a trainable parameter within the model.
- K: The **Key** vector performs a role analogous to K in traditional attention mechanisms.
- V: The **Value** vector functions similarly to V in conventional attention processes.



Token shift

- Time mixing

$$r_t = W_r \cdot (\mu_r \odot x_t + (1 - \mu_r) \odot x_{t-1}), \quad (11)$$
$$k_t = W_k \cdot (\mu_k \odot x_t + (1 - \mu_k) \odot x_{t-1}), \quad (12)$$
$$v_t = W_v \cdot (\mu_v \odot x_t + (1 - \mu_v) \odot x_{t-1}), \quad (13)$$

- Channel mixing

$$r'_t = W'_r \cdot (\mu'_r \odot x_t + (1 - \mu'_r) \odot x_{t-1}), \quad (14)$$
$$k'_t = W'_k \cdot (\mu'_k \odot x_t + (1 - \mu'_k) \odot x_{t-1}). \quad (15)$$

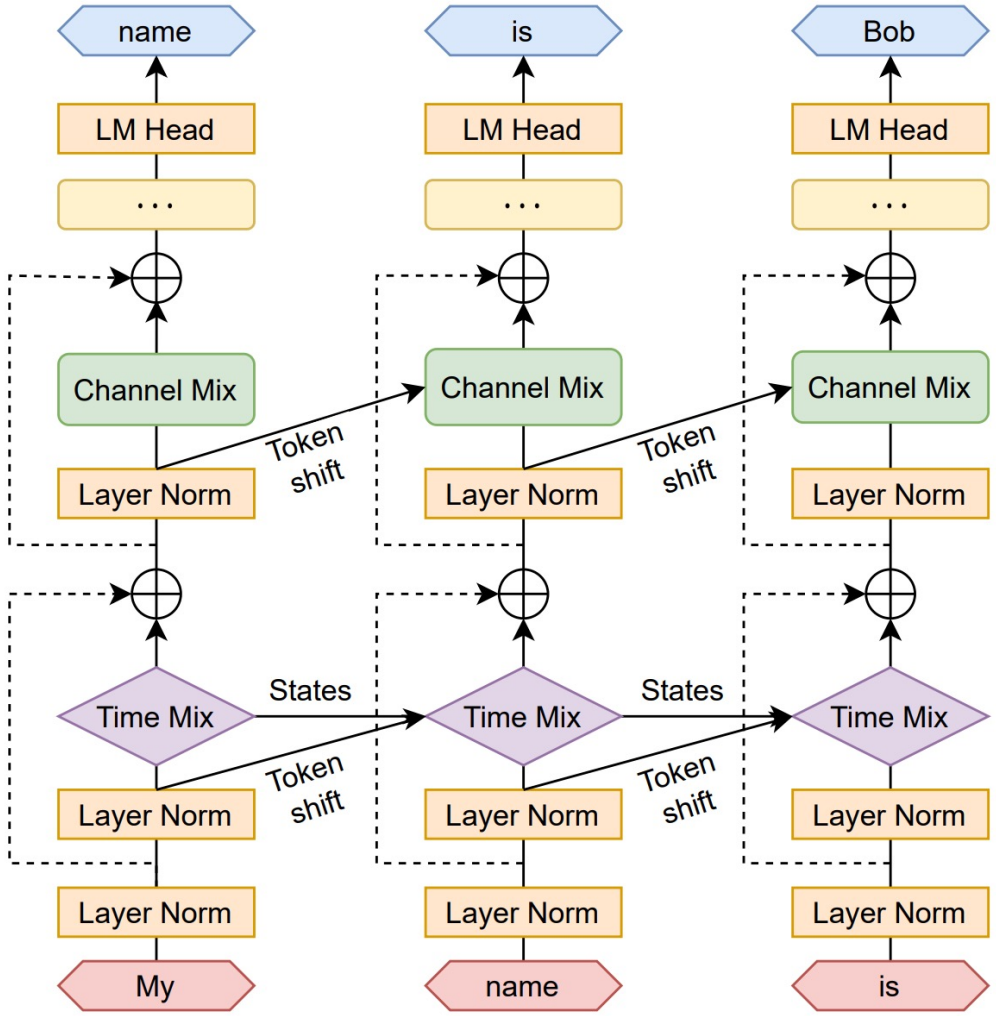


Figure 3: RWKV architecture for language modeling.

WKV

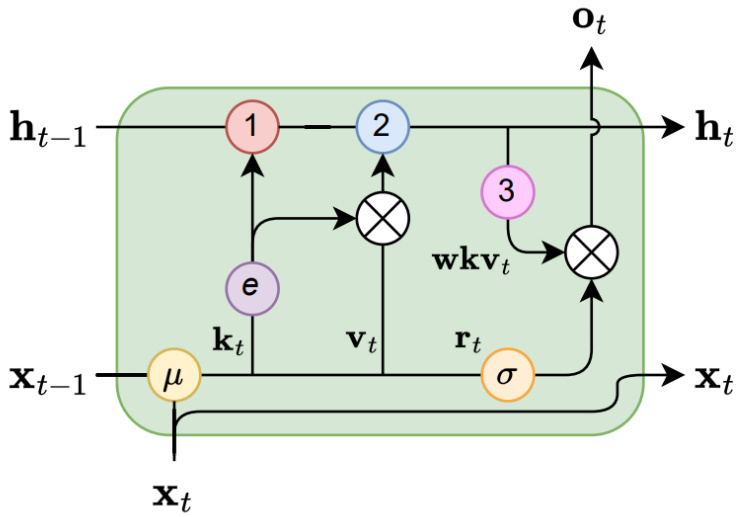
$$wkv_t = \frac{\sum_{i=1}^{t-1} e^{-(t-1-i)w+k_i} \odot v_i + e^{u+k_t} \odot v_t}{\sum_{i=1}^{t-1} e^{-(t-1-i)w+k_i} + e^{u+k_t}}.$$

$$a_0, b_0 = 0,$$

$$wkv_t = \frac{a_{t-1} + e^{u+k_t} \odot v_t}{b_{t-1} + e^{u+k_t}},$$

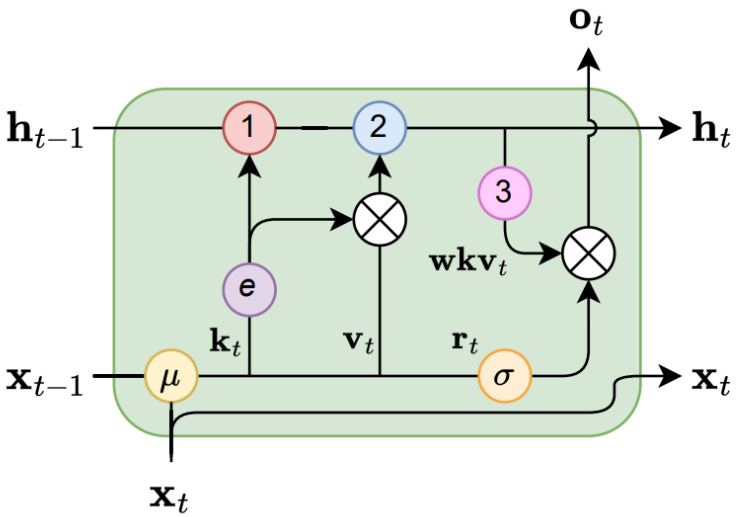
$$a_t = e^{-w} \odot a_{t-1} + e^{k_t} \odot v_t,$$

$$b_t = e^{-w} \odot b_{t-1} + e^{k_t}.$$

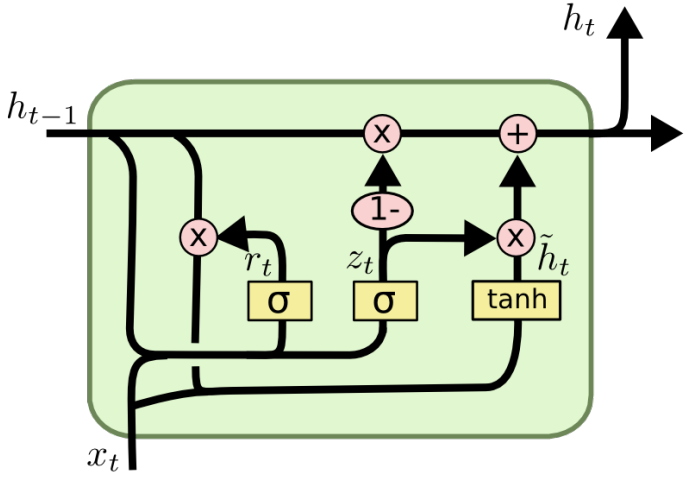


- (B, L, H) batch size, seq length, hidden size

WKV vs LSTM

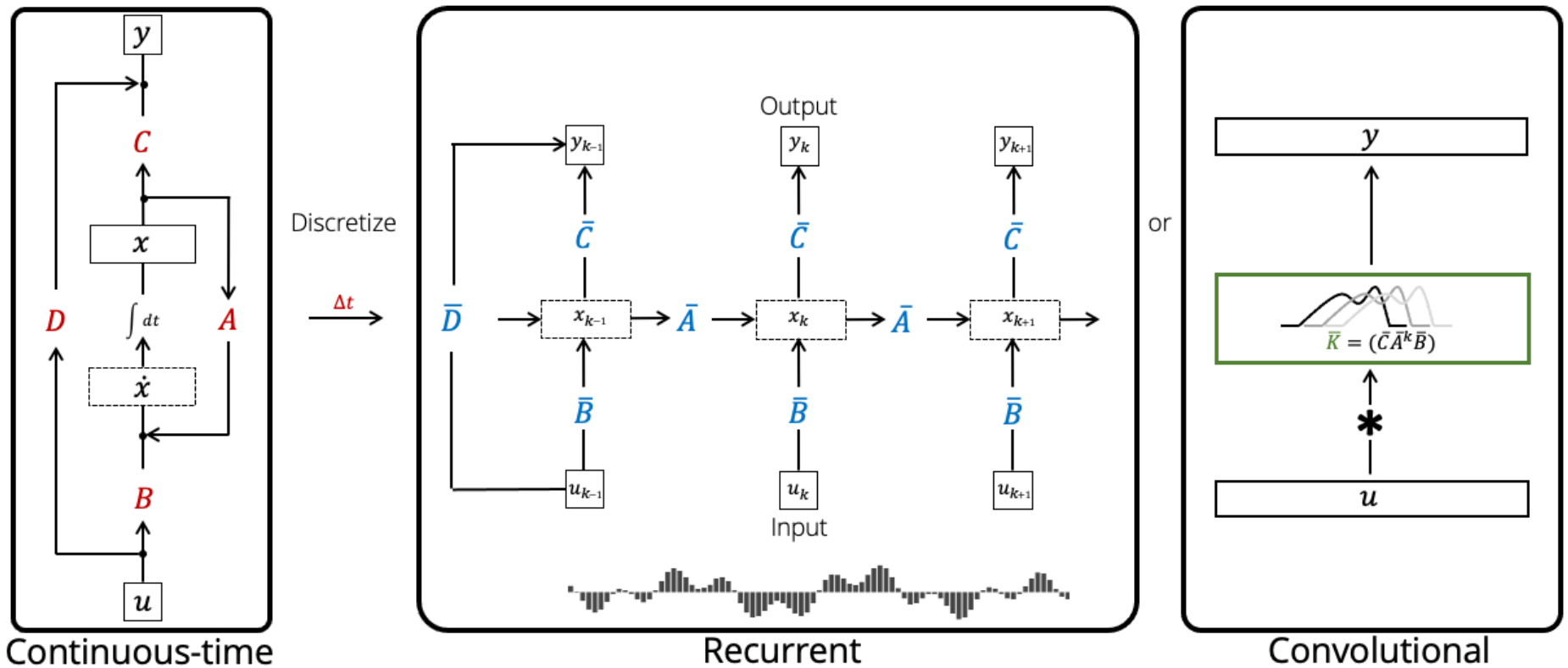


$$
\begin{aligned}
q' &:= \max(p_{t-1} - w, k_t), \\
a'_t &= e^{p_{t-1}-w-q'} \odot a'_{t-1} + e^{k_t-q'} \odot v_t, \\
b'_t &= e^{p_{t-1}-w-q'} \odot b'_{t-1} + e^{k_t-q'}, \\
p_t &= q'.
\end{aligned}
$$



$$
\begin{aligned}
z_t &= \sigma\left(W_z \cdot [h_{t-1}, x_t]\right) \\
r_t &= \sigma\left(W_r \cdot [h_{t-1}, x_t]\right) \\
\tilde{h}_t &= \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right) \\
h_t &= (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t
\end{aligned}
$$

RWKV as CNN



https://zhuanlan.zhihu.com/p/614311961?utm_id=0

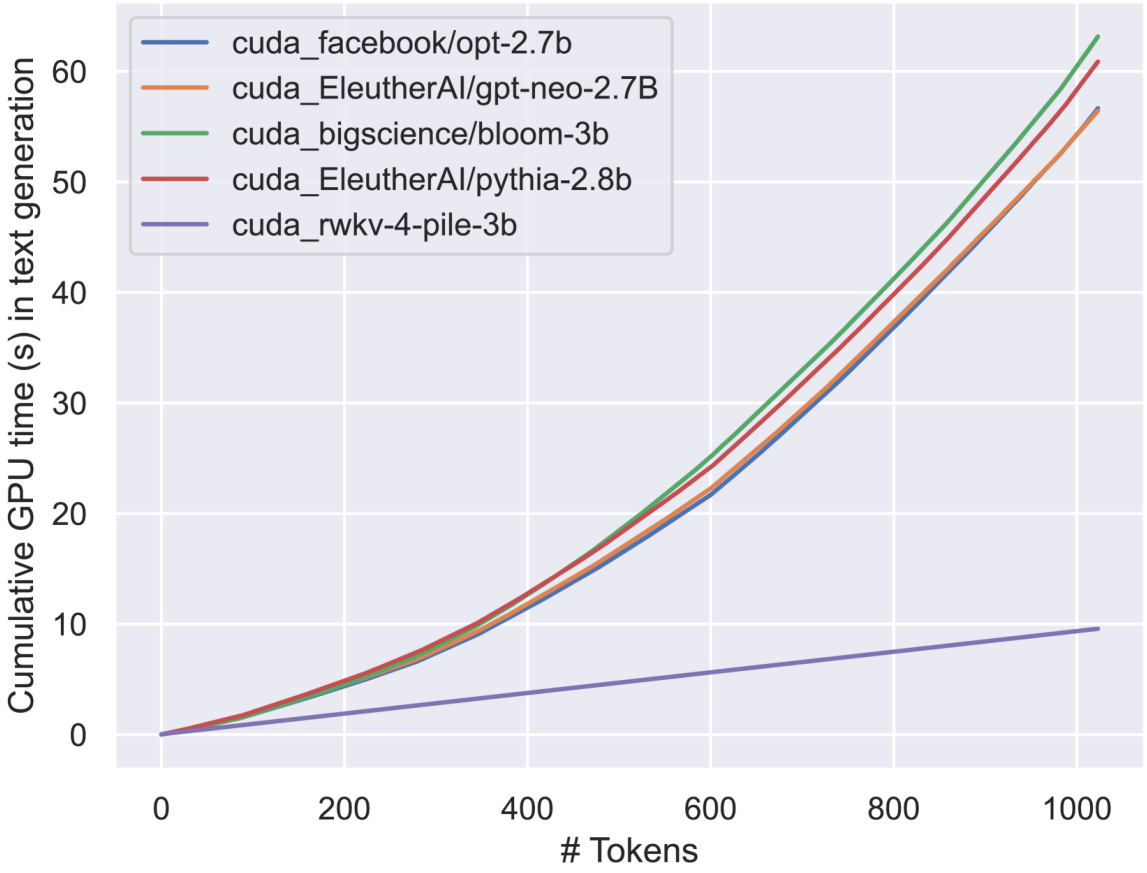
复杂度优化

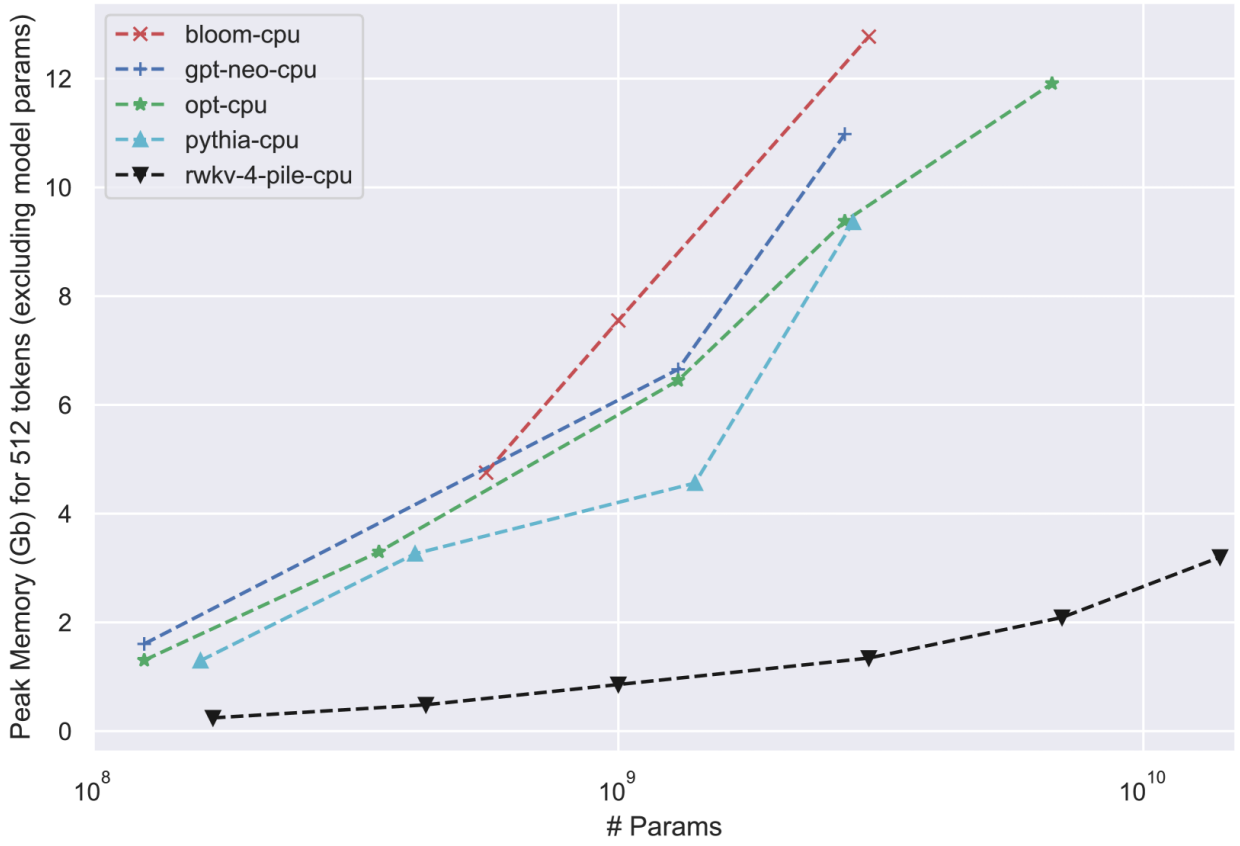
Model	Time	Space
Transformer	$O(T^2 d)$	$O(T^2 + Td)$
Reformer	$O(T \log T d)$	$O(T \log T + Td)$
Performer	$O(Td^2 \log d)$	$O(Td \log d + d^2 \log d)$
Linear Transformers	$O(Td^2)$	$O(Td + d^2)$
AFT-full	$O(T^2 d)$	$O(Td)$
AFT-local	$O(Tsd)$	$O(Td)$
MEGA	$O(cTd)$	$O(cd)$
RWKV (ours)	$O(\mathbf{Td})$	$O(\mathbf{d})$

Accuracy对比

	params	LAMBADA	AVERAGE	LAMBADA	PIQA	StoryCloze16	Hellaswag	WinoGrande	arc_challeng	arc_easy	headQA	openbookQA	sciq	triviaQA	ReCoRD	COPA
	B	ppl	%	acc	acc	acc	acc_norm	acc	acc_norm	acc	acc_norm	acc_norm	acc	acc	em	acc
World	0.19	25.51	44.35%	36.29%	62.68%	60.34%	31.38%	51.30%	22.61%	45.41%	26.33%	29.40%	75.00%	1.58%	62.28%	72.00%
RWKV-4	0.17	29.33	44.13%	32.99%	65.07%	58.79%	32.26%	50.83%	24.15%	47.47%	25.78%	29.60%	77.50%	1.26%	62.03%	66.00%
Pythia	0.16	24.38	44.14%	38.97%	62.68%	58.47%	31.63%	52.01%	23.81%	45.12%	25.82%	29.20%	76.50%	1.31%	66.32%	62.00%
GPT-Neo	0.16	30.27	43.42%	37.36%	63.06%	58.26%	30.42%	50.43%	23.12%	43.73%	25.16%	26.20%	76.60%	1.18%	64.92%	64.00%
	params	LAMBADA	AVERAGE	LAMBADA	PIQA	StoryCloze16	Hellaswag	WinoGrande	arc_challeng	arc_easy	headQA	openbookQA	sciq	triviaQA	ReCoRD	COPA
World	0.46	11.10	48.13%	48.65%	65.94%	65.85%	38.94%	50.12%	23.98%	50.17%	27.35%	32.00%	80.80%	2.36%	70.49%	69.00%
RWKV-4	0.43	13.04	48.04%	45.16%	67.52%	63.87%	40.90%	51.14%	25.17%	52.86%	27.32%	32.40%	80.30%	2.35%	70.48%	65.00%
Pythia	0.4	11.58	48.39%	50.44%	66.70%	62.64%	39.10%	53.35%	25.77%	50.38%	25.09%	30.00%	81.50%	2.03%	75.05%	67.00%
GPT-Neo	0.4	13.88	47.25%	47.29%	65.07%	61.04%	37.64%	51.14%	25.34%	48.91%	26.00%	30.60%	81.10%	1.38%	73.79%	65.00%
	params	LAMBADA	AVERAGE	LAMBADA	PIQA	StoryCloze16	Hellaswag	WinoGrande	arc_challeng	arc_easy	headQA	openbookQA	sciq	triviaQA	ReCoRD	COPA
World	1.6	6.13	54.58%	59.83%	71.11%	71.30%	51.52%	55.64%	30.12%	59.97%	27.90%	35.00%	85.40%	8.08%	77.63%	76.00%
RWKV-4	1.5	7.04	53.91%	56.43%	72.36%	68.73%	52.48%	54.62%	29.44%	60.48%	27.64%	34.00%	85.00%	5.65%	76.97%	77.00%
Pythia	1.4	6.58	53.55%	60.43%	71.11%	67.66%	50.82%	56.51%	28.58%	57.74%	27.02%	30.80%	85.50%	5.52%	81.43%	73.00%
GPT-Neo	1.4	7.50	52.64%	57.25%	71.16%	67.72%	48.94%	54.93%	25.85%	56.19%	27.86%	33.60%	86.00%	5.24%	80.62%	69.00%
	params	LAMBADA	AVERAGE	LAMBADA	PIQA	StoryCloze16	Hellaswag	WinoGrande	arc_challeng	arc_easy	headQA	openbookQA	sciq	triviaQA	ReCoRD	COPA
World	3.1	4.68	58.33%	65.86%	73.01%	73.92%	58.70%	58.88%	34.47%	64.65%	27.68%	36.60%	87.50%	11.18%	81.89%	84.00%
RWKV-4	3 ctx4k	5.25	57.93%	63.96%	74.16%	70.71%	59.89%	59.59%	33.11%	65.19%	28.45%	37.00%	86.50%	11.68%	80.87%	82.00%
Pythia	2.8	4.93	57.64%	65.36%	73.83%	70.71%	59.46%	61.25%	32.25%	62.84%	28.96%	35.20%	87.70%	9.63%	85.10%	77.00%
GPT-Neo	2.8	5.63	55.92%	62.22%	72.14%	69.54%	55.82%	57.62%	30.20%	61.07%	27.17%	33.20%	89.30%	4.82%	83.80%	80.00%
	params	LAMBADA	AVERAGE	LAMBADA	PIQA	StoryCloze16	Hellaswag	WinoGrande	arc_challeng	arc_easy	headQA	openbookQA	sciq	triviaQA	ReCoRD	COPA
World	7.5	3.93	61.76%	70.15%	75.24%	75.63%	65.23%	62.19%	36.52%	68.18%	31.22%	40.00%	91.20%	16.74%	84.60%	86.00%
RWKV-4	7.4	4.38	61.20%	67.18%	76.06%	73.44%	65.51%	61.01%	37.46%	67.80%	31.22%	40.20%	88.80%	18.30%	83.68%	85.00%
Pythia	6.9	4.30	60.44%	67.98%	74.54%	72.96%	63.92%	61.01%	35.07%	66.79%	28.59%	38.00%	90.00%	15.42%	86.44%	85.00%
GPT-J	6.1	4.10	61.34%	68.31%	75.41%	74.02%	66.25%	64.09%	36.60%	66.92%	28.67%	38.20%	91.50%	16.74%	87.71%	83.00%

推理速度和内存占用


cuda_facebook/opt-2.7b
cuda_EleutherAI/gpt-neo-2.7B
cuda_bigscience/bloom-3b
cuda_EleutherAI/pythia-2.8b
cuda_rwkv-4-pile-3b
Cumulative GPU time (s) in text generation
0
10
20
30
40
50
60
0
200
400
600
800
1000
Tokens


bloom-cpu
gpt-neo-cpu
opt-cpu
pythia-cpu
rwkv-4-pile-cpu
Peak Memory (Gb) for 512 tokens (excluding model params)
0
2
4
6
8
10
12
10^8
10^9
10^{10}
Params

WKV cuda kernel

$$
\begin{aligned}
q' &:= \max(p_{t-1} - w, k_t), \\
a'_t &= e^{p_{t-1}-w-q'} \odot a'_{t-1} + e^{k_t-q'} \odot v_t, \\
b'_t &= e^{p_{t-1}-w-q'} \odot b'_{t-1} + e^{k_t-q'}, \\
p_t &= q'.
\end{aligned}
$$

```
template <typename F>
__global__ void kernel_forward(const int B, const int T, const int C,
                               const F *__restrict__ const _w, const F *__res
                               F *__restrict__ const _y) {
    const int idx = blockIdx.x * blockDim.x + threadIdx.x;
    const int _b = idx / C;
    const int _c = idx % C;
    const int _offset = _b * T * C + _c;

    F u = _u[_c];
    F w = _w[_c];
    const F *__restrict__ const k = _k + _offset;
    const F *__restrict__ const v = _v + _offset;
    F *__restrict__ const y = _y + _offset;

    F p = 0, q = 0, o = MIN_VALUE;
    // p and q are running sums divided by exp(o) (to avoid overflows)
    for (int i = 0; i < T; i++) {
        const int ii = i * C;

        F no = max(o, u + k[ii]);
        F A = exp(o - no);
        F B = exp(u + k[ii] - no);
        y[ii] = (A * p + B * v[ii]) / (A * q + B);

        no = max(w + o, k[ii]);
        A = exp(w + o - no);
        B = exp(k[ii] - no);
        p = A * p + B * v[ii];
        q = A * q + B;
        o = no;
    }
}
```

MindSpore custom operator

```python
def load_wkv_cuda_kernel(func_name, context_length):
    """load wkv cuda kernel"""
    device_target = mindspore.get_context('device_target')
    if device_target != 'GPU':
        raise RuntimeError('WKV operator only support GPU currently.')

    logger.info(f"Loading CUDA kernel for RWKV at context length of {context_length}.")

    from ...kernel_utils import compile_kernel
    so_path = compile_kernel(Tmax=context_length)
    wkv_op = ops.Custom(
        str(so_path) + ':' + func_name,
        out_shape=WKV_SHAPE_INFER[func_name],
        out_dtype=WKV_DTYPE_INFER[func_name],
        func_type='aot'
    )
    wkv_op.add_prim_attr('primitive_target', device_target)
    return wkv_op
```

Huggingface-like demo