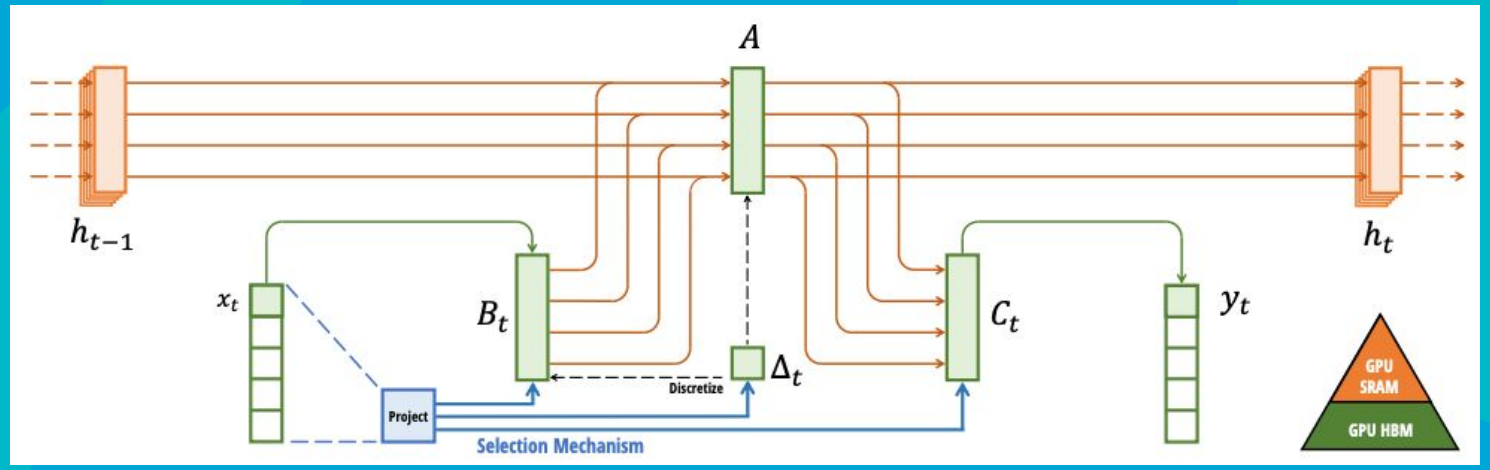
TUDelft

Fast Neural Networks

And how to find them…



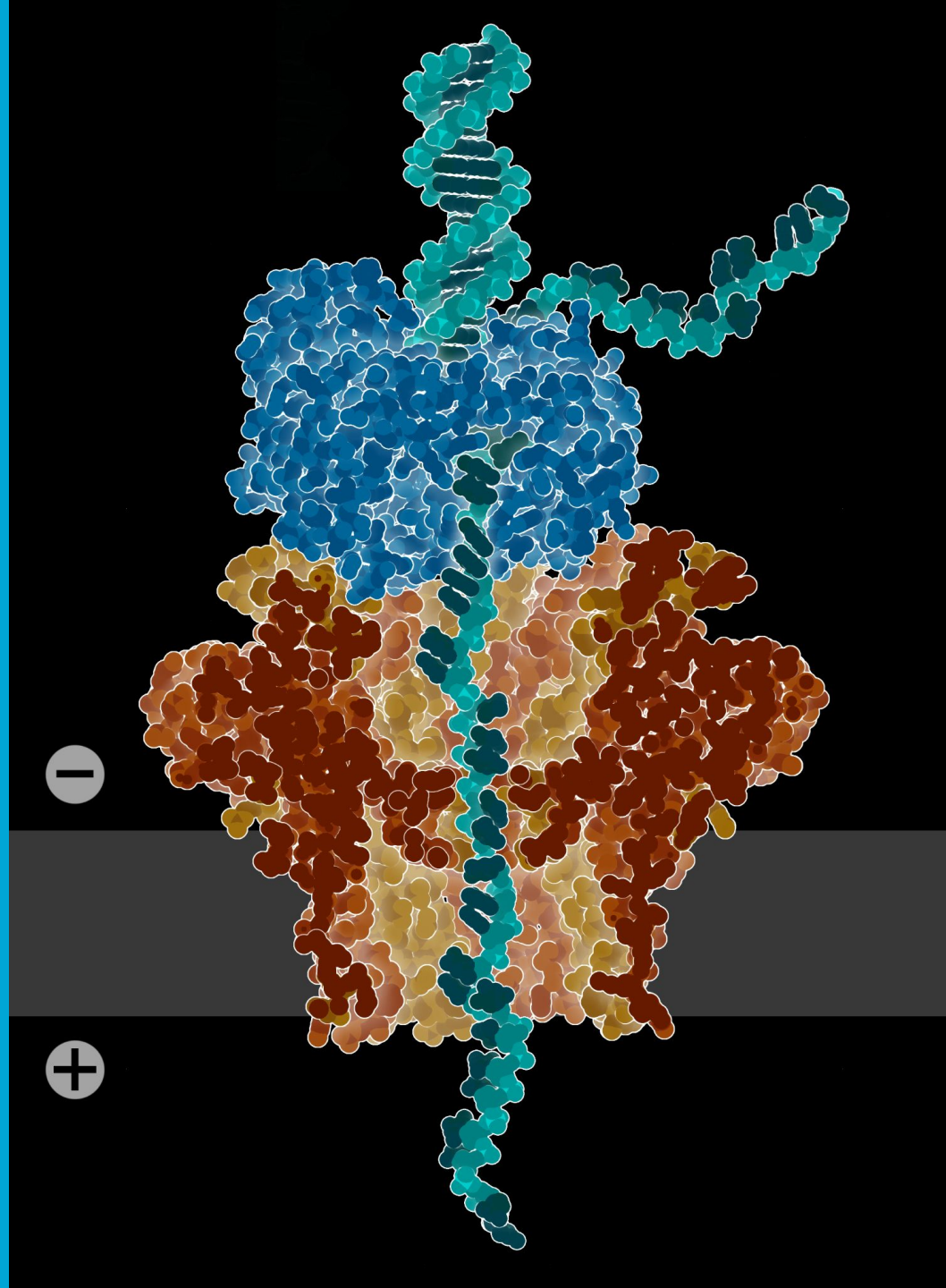
Nathan Ordonez

TU Delft

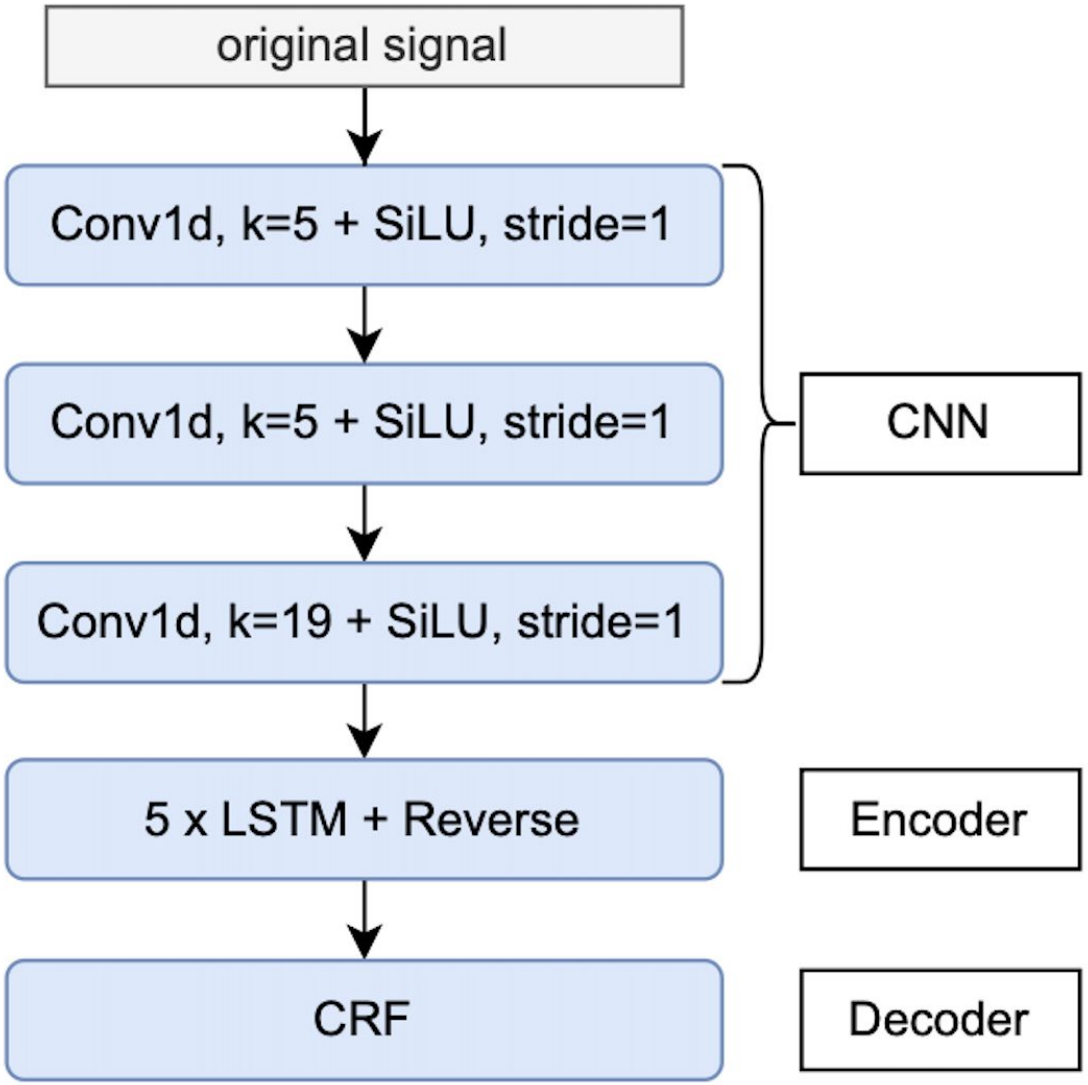
Thesis: Towards faster sequence-to-sequence models for basecalling

Nathan Ordonez

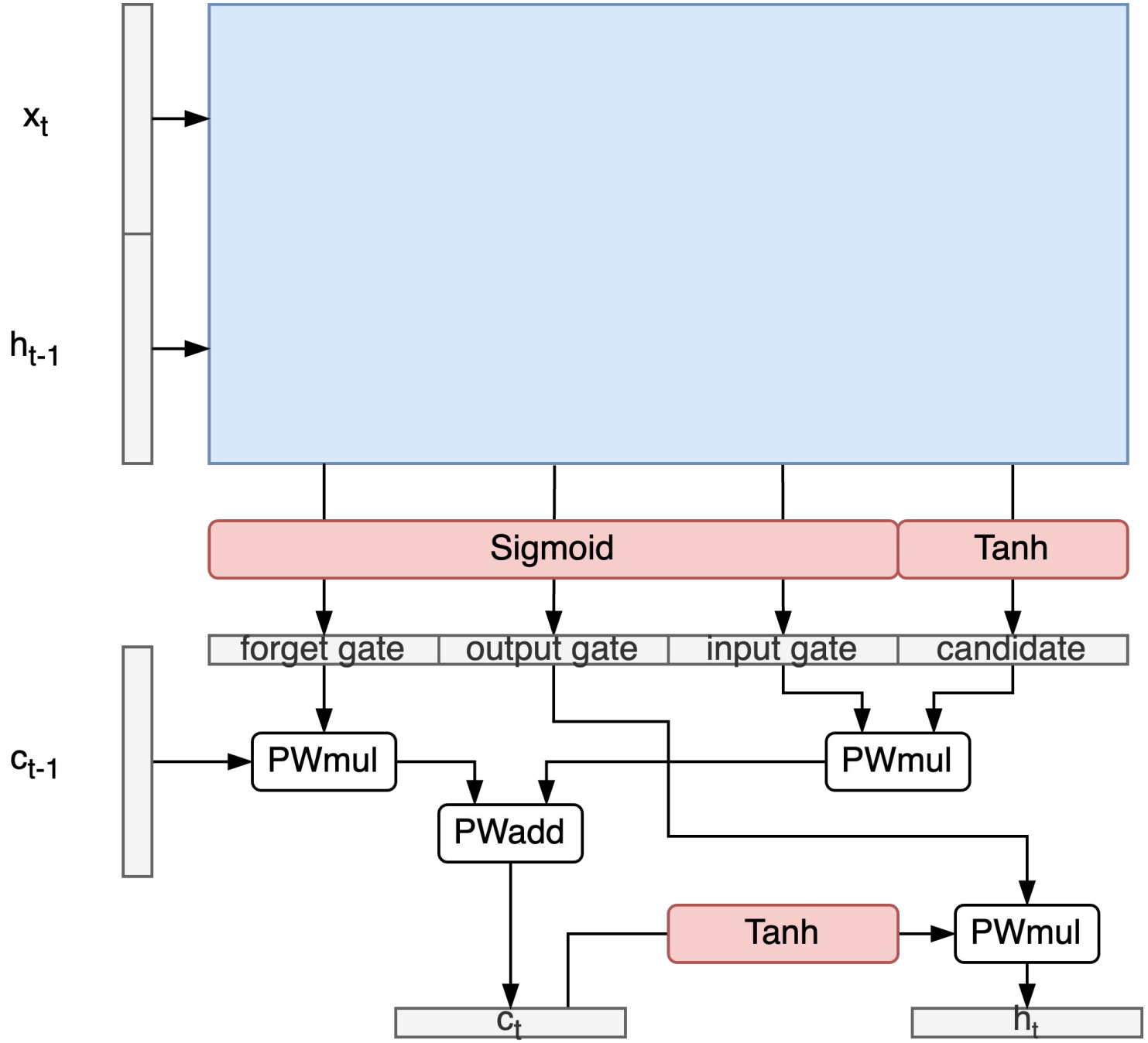
4-11-2024



Bonito


original signal
Conv1d, k=5 + SiLU, stride=1
Conv1d, k=5 + SiLU, stride=1
Conv1d, k=19 + SiLU, stride=1
CNN
5 x LSTM + Reverse
Encoder
CRF
Decoder

The LSTM "Compute -flow"



LSTM implementation

```
# Initialize hidden and cell states to zero. There will be one hidden
# and cell state for each input, so they will have shape of (N, H)
h0 = torch.zeros(N, H, device=x.device)
c0 = torch.zeros(N, H, device=x.device)

# Define a list to store outputs. We will then stack them.
y = []

ht_1 = h0
ct_1 = c0
for t in range(T):
    # LSTM update rule
    # xh = torch.addmm(self.bias_xh, x[t], self.weight_xh)
    # hh = torch.addmm(self.bias_hh, ht_1, self.weight_hh)
    gates = x[t] @ self.weight_ih.t() + self.bias_ih + ht_1 @ self.weight_hh.t() + self.bias_hh
    it, ft, gt, ot = gates.chunk(4, 1)
    # add_res = xh + hh
    it = torch.sigmoid(it)
    ft = torch.sigmoid(ft)
    gt = torch.tanh(gt)
    ot = torch.sigmoid(ot)
    ct = ft * ct_1 + it * gt
    ht = ot * torch.tanh(ct)

    # Store output
    y.append(ht)

    # For the next iteration c(t-1) and h(t-1) will be current ct and ht
    ct_1 = ct
    ht_1 = ht
```

LSTM implementation

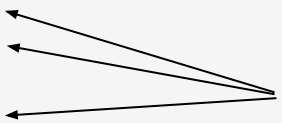
```python
# Initialize hidden and cell states to zero. There will be one hidden
# and cell state for each input, so they will have shape of (N, H)
h0 = torch.zeros(N, H, device=x.device)
c0 = torch.zeros(N, H, device=x.device)

# Define a list to store outputs. We will then stack them.
y = []

ht_1 = h0
ct_1 = c0
for t in range(T):
    # LSTM update rule
    # xh = torch.addmm(self.bias_xh, x[t], self.weight_xh)
    # hh = torch.addmm(self.bias_hh, ht_1, self.weight_hh)
    gates = x[t] @ self.weight_ih.t() + self.bias_ih + ht_1 @ self.weight_hh.t() + self.bias_hh
    it, ft, gt, ot = gates.chunk(4, 1)
    # add_res = xh + hh
    it = torch.sigmoid(it)
    ft = torch.sigmoid(ft)
    gt = torch.tanh(gt)
    ot = torch.sigmoid(ot)
    ct = ft * ct_1 + it * gt
    ht = ot * torch.tanh(ct)

    # Store output
    y.append(ht)

    # For the next iteration c(t-1) and h(t-1) will be current ct and ht
    ct_1 = ct
    ht_1 = ht
```



stack then sigmoid?

LSTM implementation

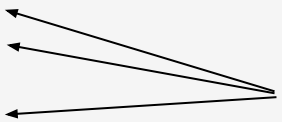
```
# Initialize hidden and cell states to zero. There will be one hidden
# and cell state for each input, so they will have shape of (N, H)
h0 = torch.zeros(N, H, device=x.device)
c0 = torch.zeros(N, H, device=x.device)

# Define a list to store outputs. We will then stack them.
y = []

ht_1 = h0
ct_1 = c0
for t in range(T):
    # LSTM update rule
    # xh = torch.addmm(self.bias_xh, x[t], self.weight_xh)
    # hh = torch.addmm(self.bias_hh, ht_1, self.weight_hh)
    gates = x[t] @ self.weight_ih.t() + self.bias_ih + ht_1 @ self.weight_hh.t() + self.bias_hh
    it, ft, gt, ot = gates.chunk(4, 1)
    # add_res = xh + hh
    it = torch.sigmoid(it)
    ft = torch.sigmoid(ft)
    gt = torch.tanh(gt)
    ot = torch.sigmoid(ot)
    ct = ft * ct_1 + it * gt
    ht = ot * torch.tanh(ct)

    # Store output
    y.append(ht)

    # For the next iteration c(t-1) and h(t-1) will be current ct and ht
    ct_1 = ct
    ht_1 = ht
```

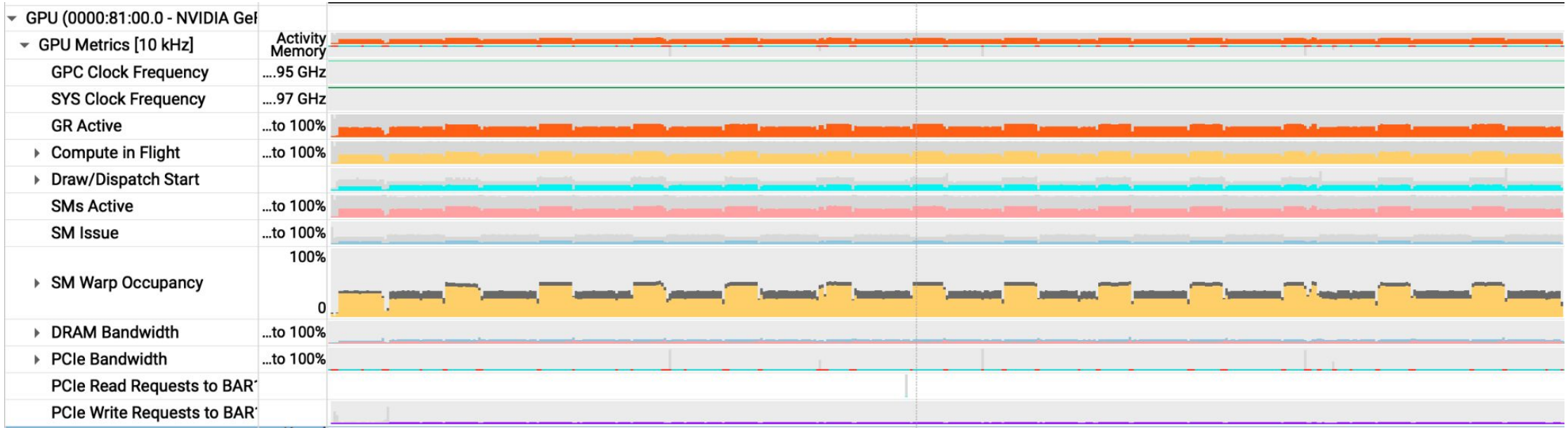


stack then sigmoid?
Would that help?

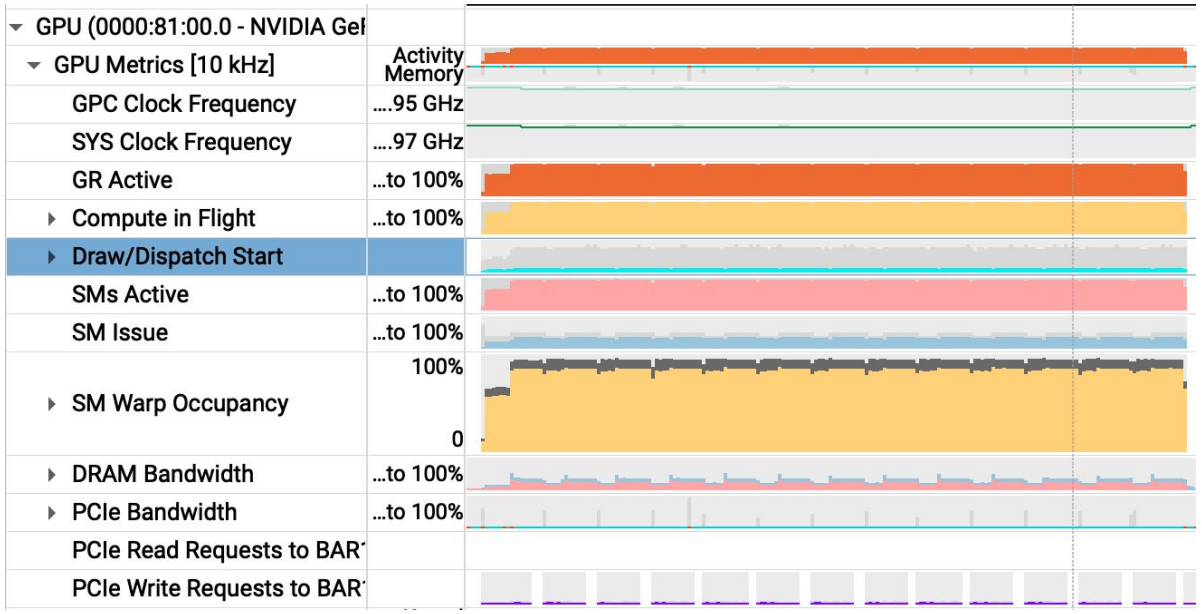
. . .

Are we doomed to timed benchmarks?

Accelerator metrics!



Accelerator metrics!

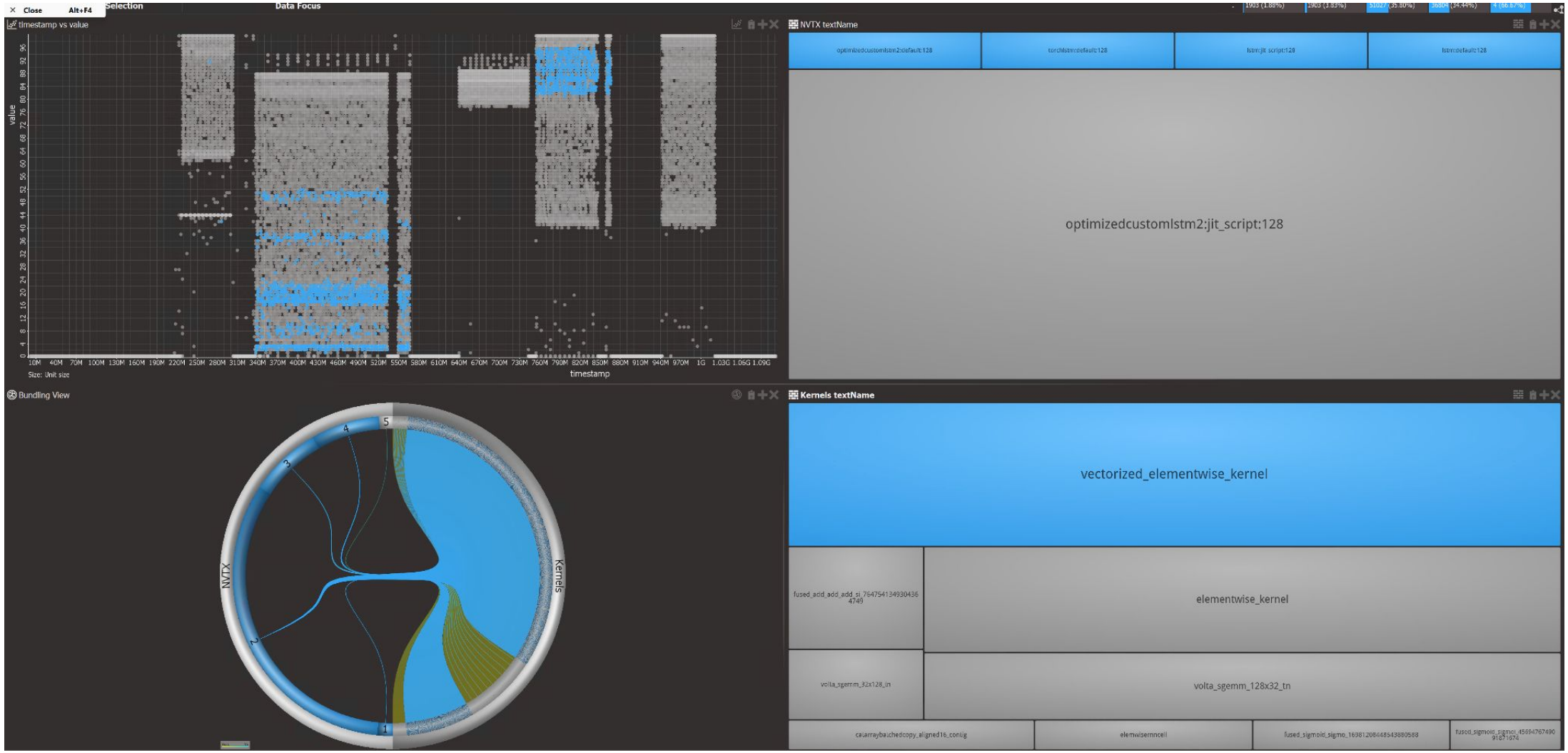



```python
for t in range(seq_len):
    x_t = x[:, t, :]
    combined = torch.cat((x_t, h_t), dim=1)
    gates = self.combined_layer(combined)
    i_t, f_t, g_t, o_t = gates.chunk(4, dim=1)
    i_t = torch.sigmoid(i_t)
    f_t = torch.sigmoid(f_t)
    g_t = torch.tanh(g_t)
    o_t = torch.sigmoid(o_t)
    c_t = f_t * c_t + i_t * g_t
    h_t = o_t * torch.tanh(c_t)
    outputs.append(h_t)
```

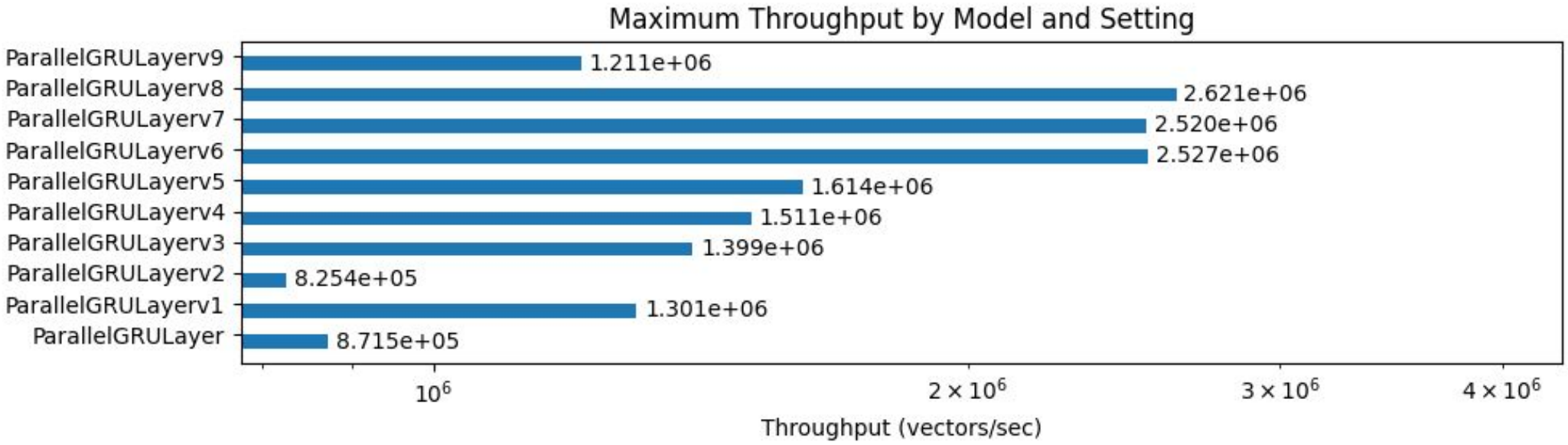
Too many things to test…

- Model compilers: JIT, Cudagraphs, Inductor, TensorRT, MLIR…
- Batch sizes
- Input dimension order: batch first? Sequence length first?
- Precisions? (sparsity, quantisation…)
- Interchangeable operations: manual matmul, PyTorch “Linear layer”...
- Accelerator components & their interactions
- Distributed???

ModelView: Get more details, with less noise



ModelView: Get more details, with less noise



TU Delft

Interested? Contact us:

Thank you for listening!

Nathan Ordonez - nathanaxcan@gmail.com