XIP-1 Presentation: Transformers on Drones

State of the Art ML on Lightweight Devices

Agenda

1

Problem

the deep
learning gap

2

Solution

optimizing
implementations

3

Process

realizing
the dream

4

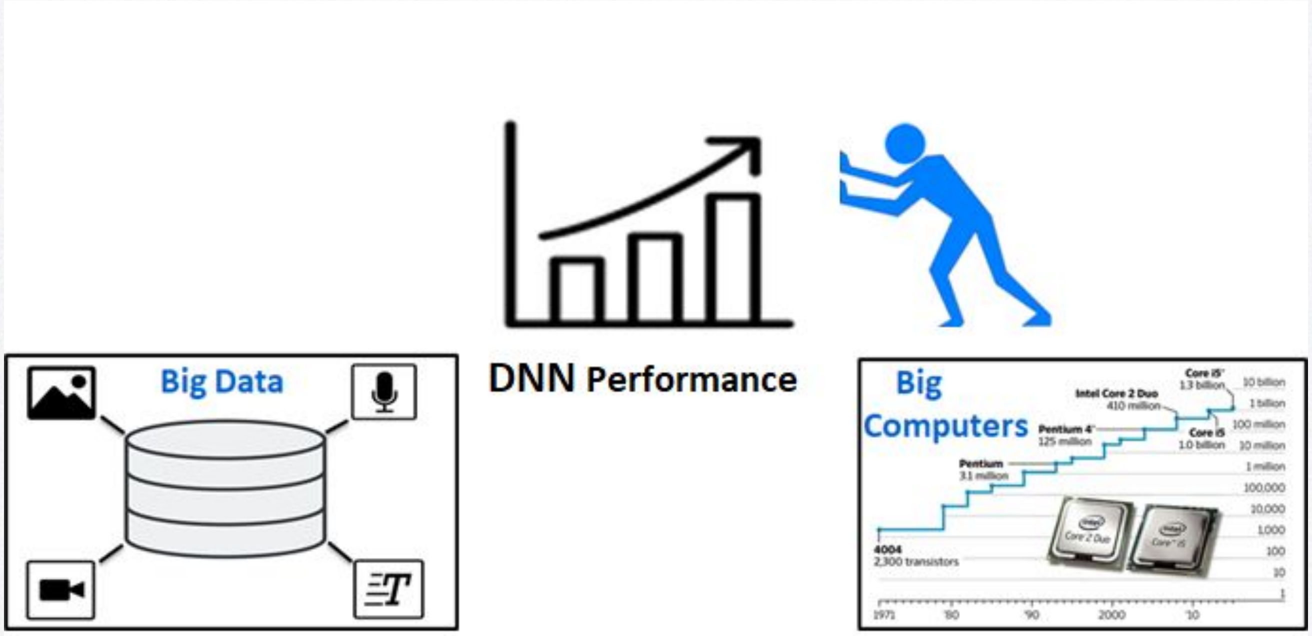
Impacts

the future is
now

1

Problem

the deep learning gap



Modern machine learning relies on enormous data and compute.

by conservative estimates, **GPT-3** used

153599846

books worth of data

40 lifetimes

worth of thinking by energy expended

ML is Impactful on Edge Devices

Personal Electronics

powerful but private
personal assistants

Robotic Devices

from disaster response
to autonomous vehicles

Internet of Things

smart homes
and smart cities

none of these devices can leverage state of the art models

2

Solution

optimizing implementations

Constraints

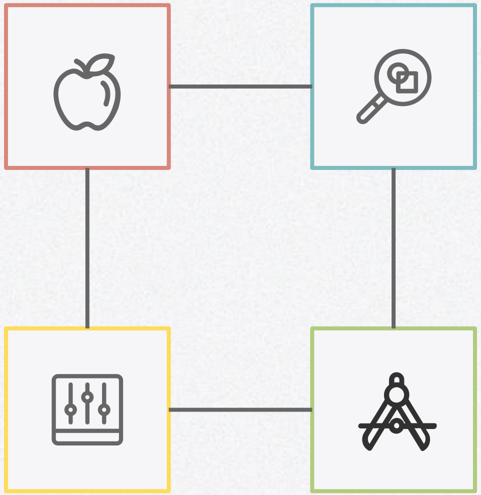
Hard
Memory Limit

IoT devices have up to tens of MB of memory and up to a few GB of swap, severely limiting the maximum model complexity.

Soft
Inference Time

The faster the better for real-time IoT tasks, meaning the tens of seconds needed for SoTA inference is unacceptable.

Optimization Methodologies



Model Distillation

Teach a smaller model using a larger one

Architecture Search

Optimize model shape for hardware

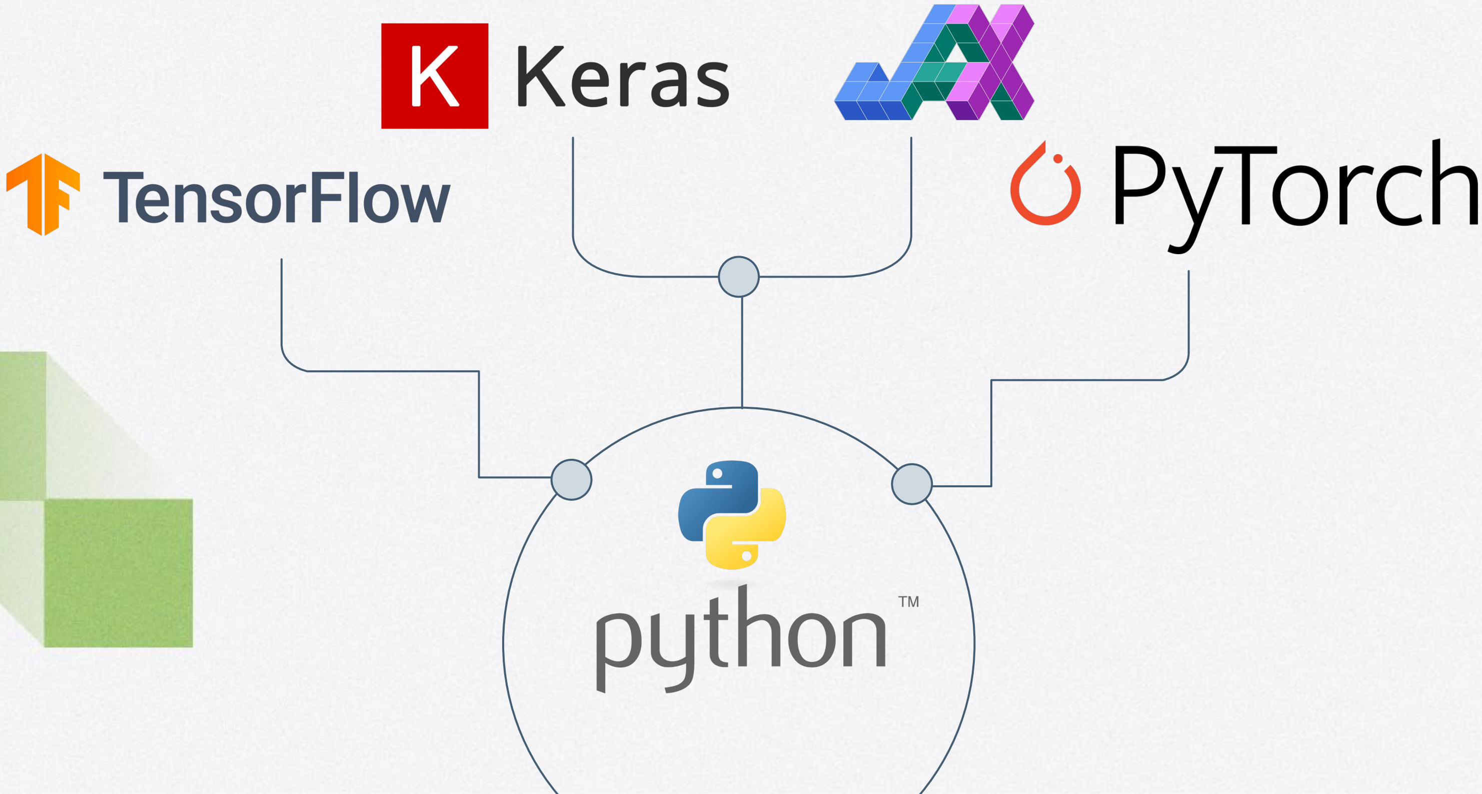
Weight Quantization

Approximate weights with less granularity

Hardware Acceleration

Design custom hardware around existing models

Why so slow?

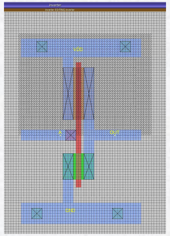


Development Trade Off

PyTorch
TensorFlow
python

C
C++

SystemVerilog
VHDL



flexible **efficient**

historically, edge-level hardware solutions are

500x faster

Project Methodology

Stage 1

Learn about hardware efficient transformers

Stage 2

Implement GPT-2 in vanilla Python

Stage 3

Rewrite impl in C-style High Level Synthesis

Stage 4

Generate hardware description using HLS

Stage 5

Deploy code to FPGAs on Drones

3

Process

realizing the dream

Project Methodology

Stage 1

Learn about hardware efficient transformers

Stage 2

Implement GPT-2 using low-level NumPy

Stage 3

Rewrite impl in C-style High Level Synthesis

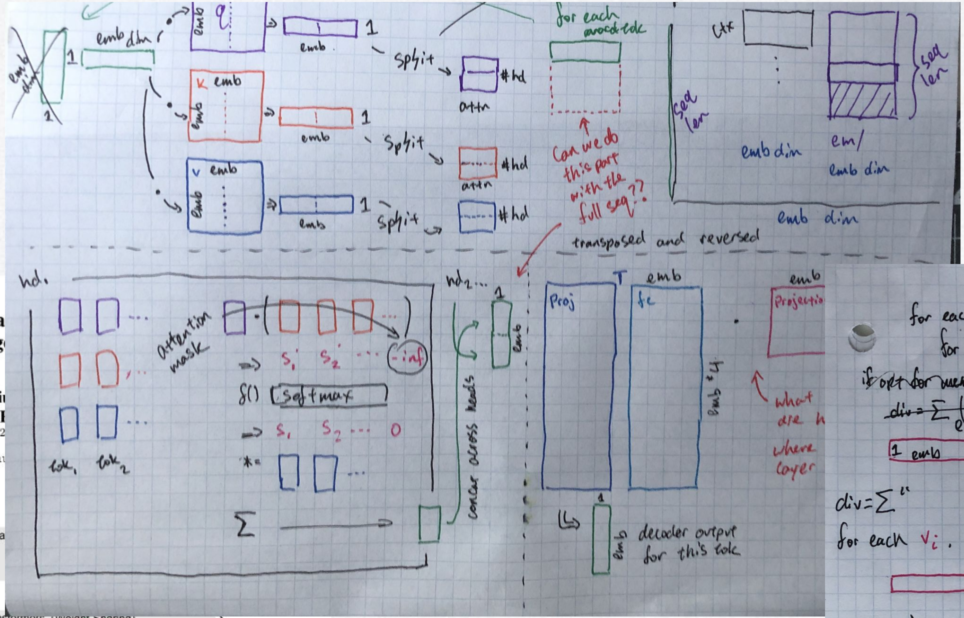
Stage 4

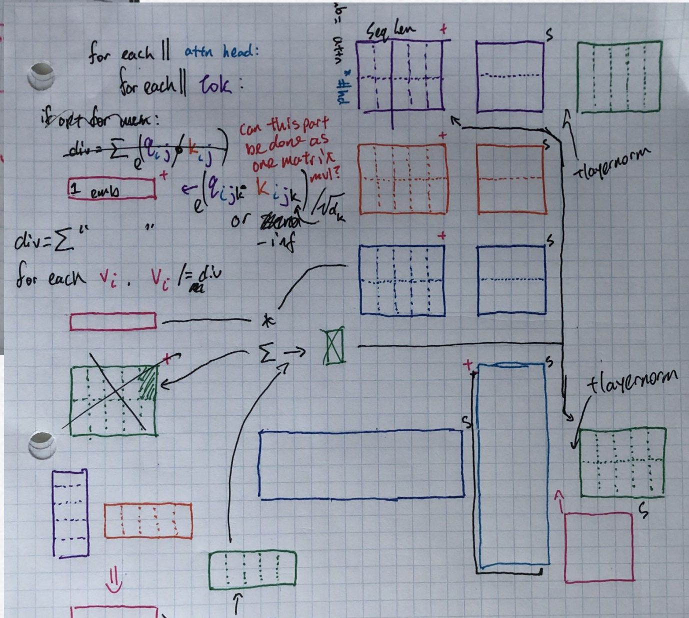
Generate hardware description using HLS

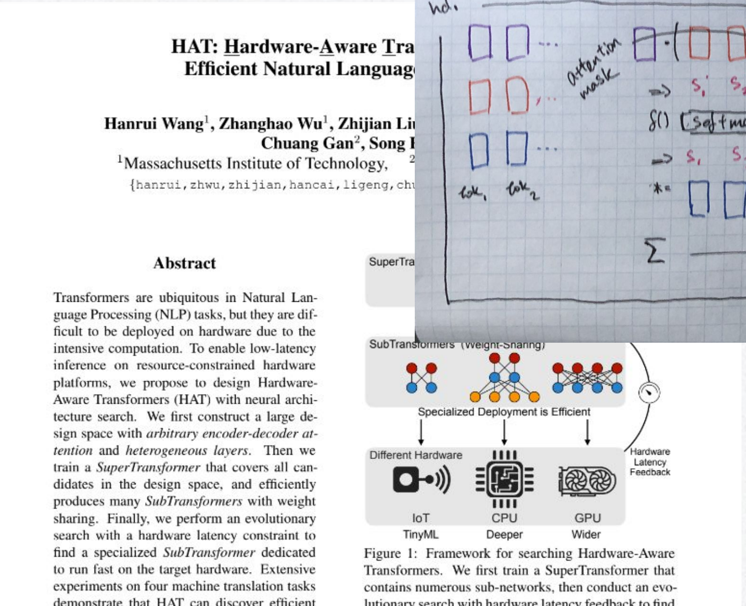
Stage 5

Deploy code to FPGAs on drones

Challenge: Transformers






HAT: Hardware-Aware Tra
Efficient Natural Languag

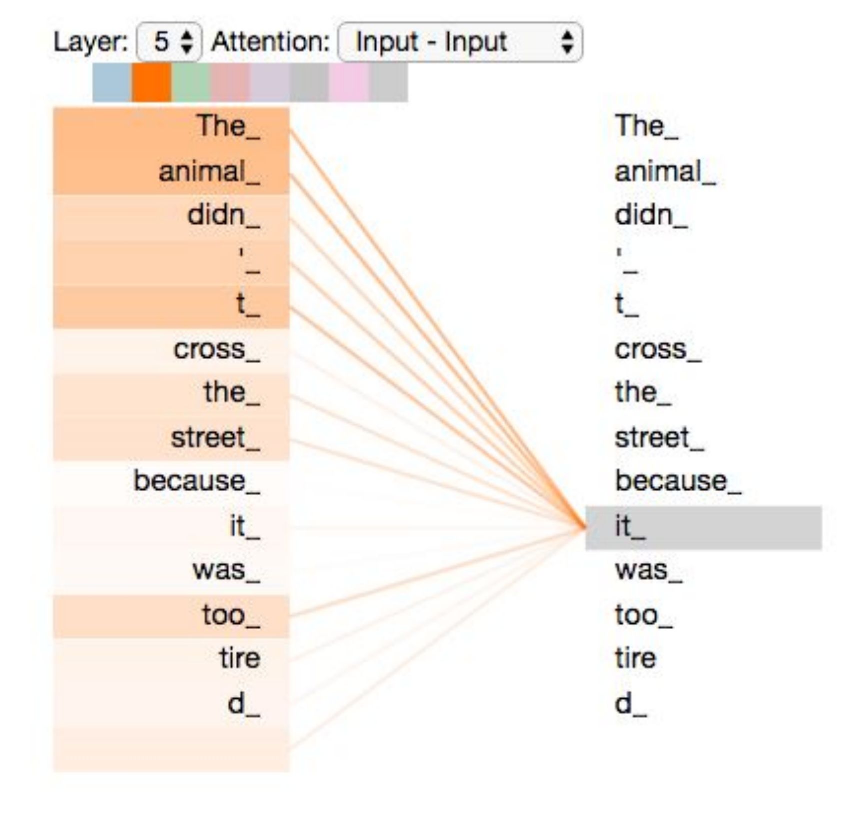
Hanrui Wang[1], Zhanghao Wu[1], Zhijian Liu
Chuang Gan[2], Song
[1]Massachusetts Institute of Technology, [2]
{hanrui,zhwu,zhijian,hancai,ligeng,chu

Abstract

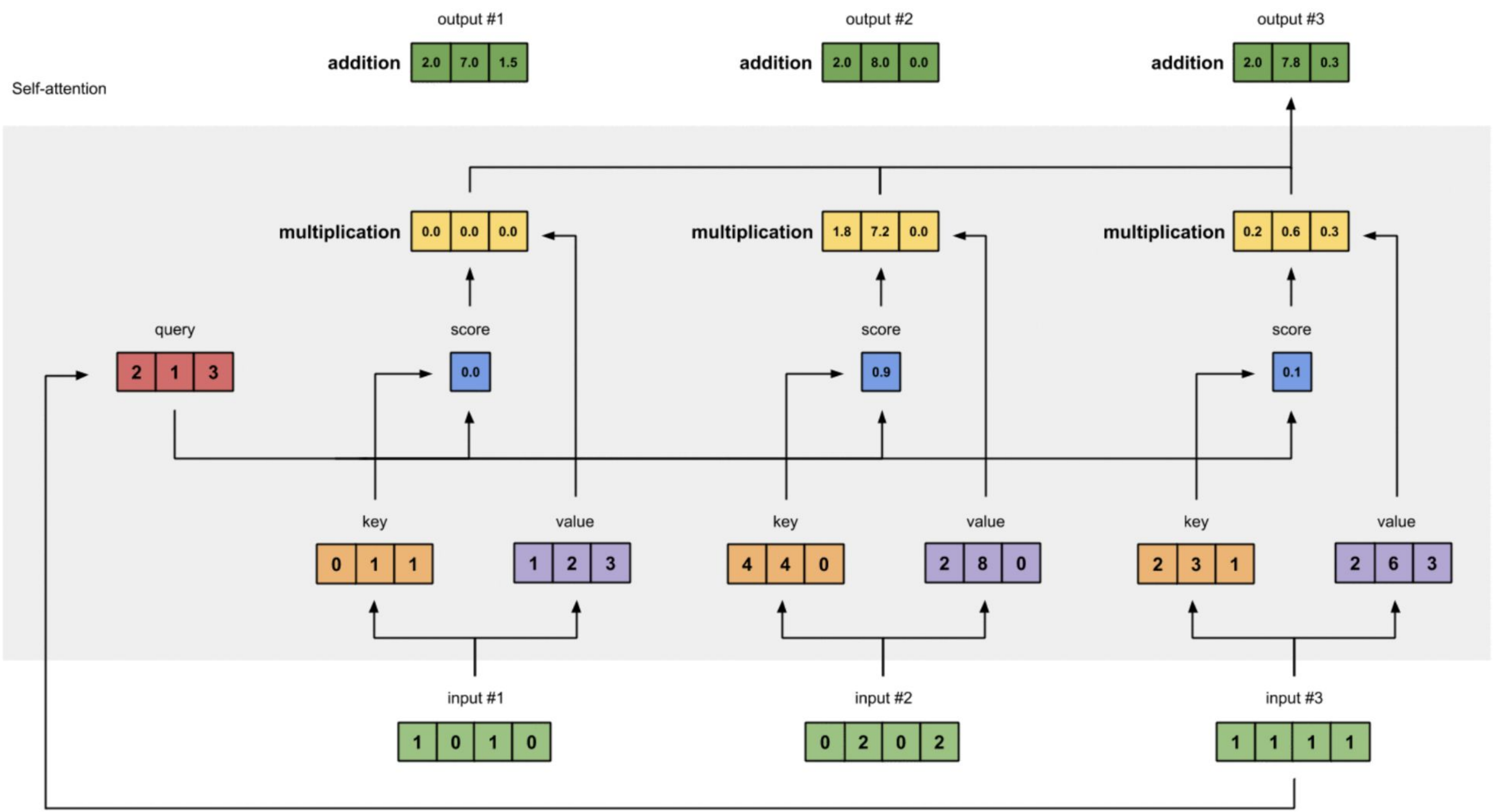
Transformers are ubiquitous in Natural Language Processing (NLP) tasks, but they are difficult to be deployed on hardware due to the intensive computation. To enable low-latency inference on resource-constrained hardware platforms, we propose to design Hardware-Aware Transformers (HAT) with neural architecture search. We first construct a large design space with *arbitrary encoder-decoder attention* and *heterogeneous layers*. Then we train a *SuperTransformer* that covers all candidates in the design space, and efficiently produces many *SubTransformers* with weight sharing. Finally, we perform an evolutionary search with a hardware latency constraint to find a specialized *SubTransformer* dedicated to run fast on the target hardware. Extensive experiments on four machine translation tasks demonstrate that HAT can discover efficient

Figure 1: Framework for searching Hardware-Aware Transformers. We first train a SuperTransformer that contains numerous sub-networks, then conduct an evolutionary search with hardware latency feedback to find

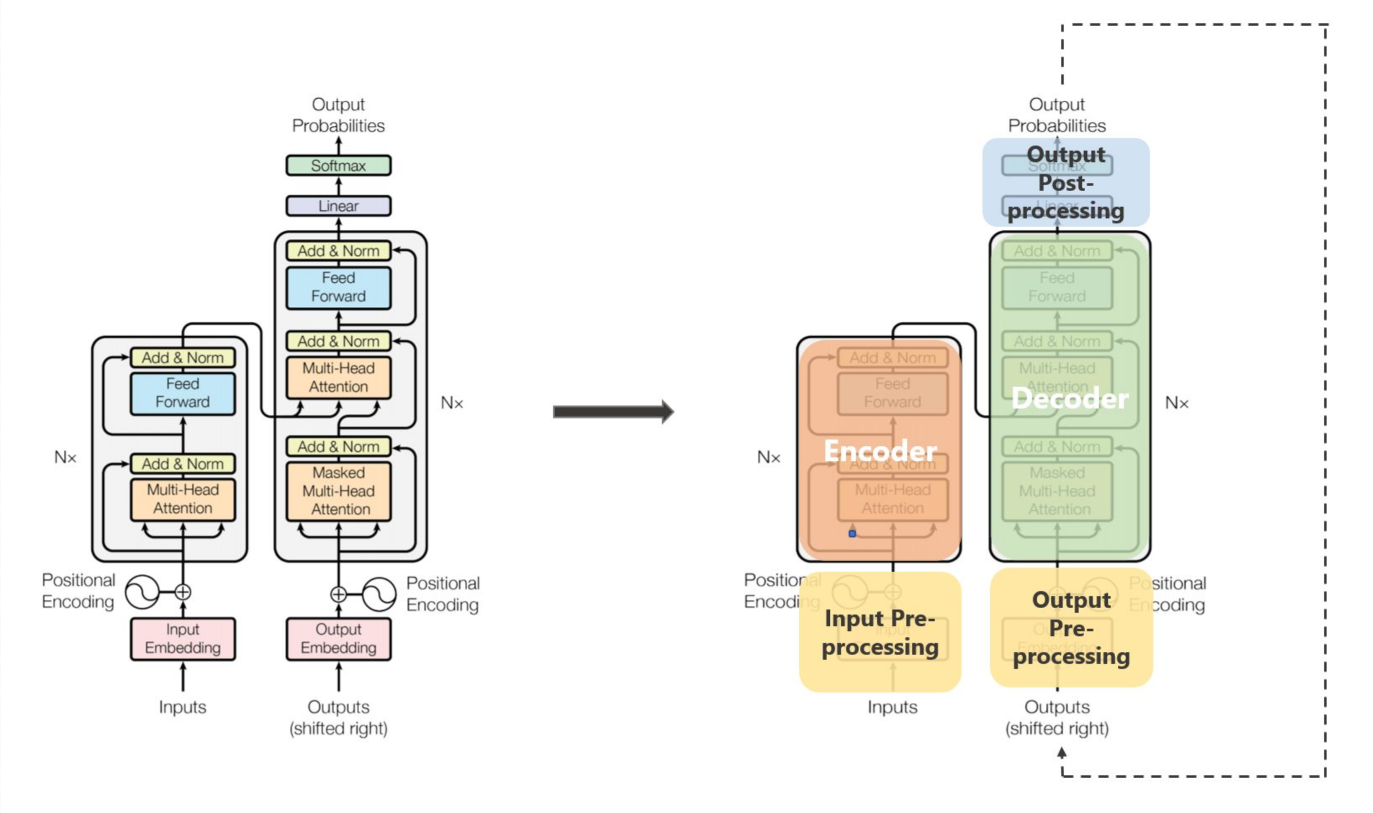
Attention



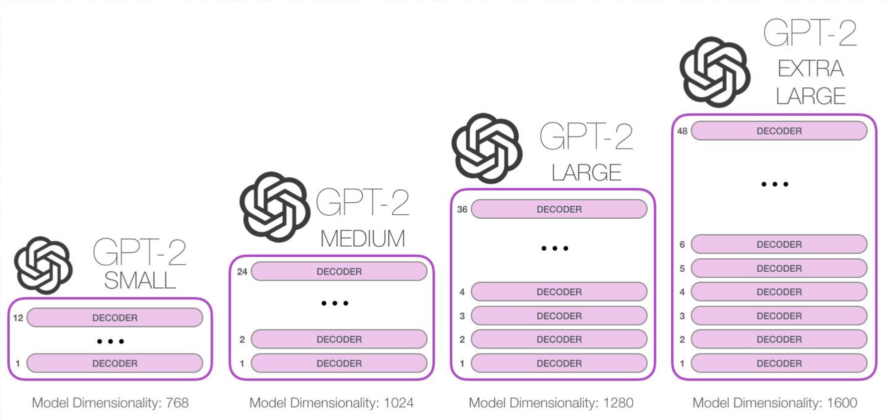
Layer: 5
Attention: Input - Input
The_
animal_
didn_
'_
t_
cross_
the_
street_
because_
it_
was_
too_
tire
d_
The_
animal_
didn_
'_
t_
cross_
the_
street_
because_
it_
was_
too_
tire
d_

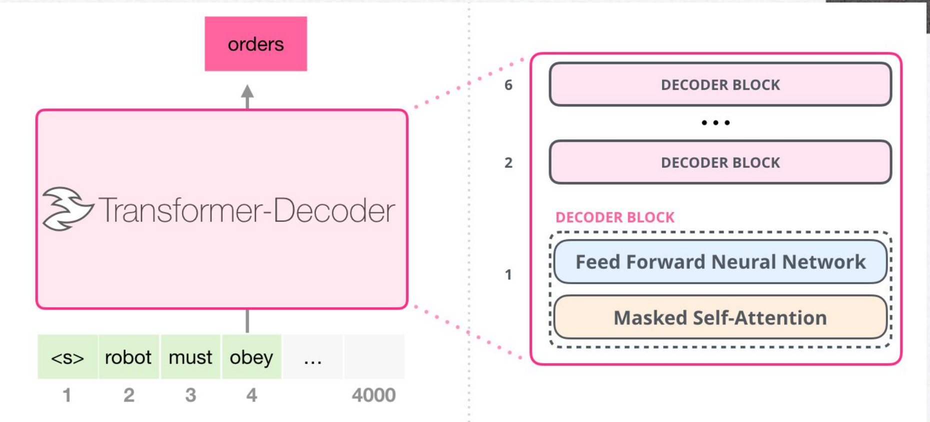

Self-attention
output #1
addition
2.0 7.0 1.5
output #2
addition
2.0 8.0 0.0
output #3
addition
2.0 7.8 0.3
multiplication
0.0 0.0 0.0
multiplication
1.8 7.2 0.0
multiplication
0.2 0.6 0.3
query
2 1 3
score
0.0
score
0.9
score
0.1
key
0 1 1
value
1 2 3
key
4 4 0
value
2 8 0
key
2 3 1
value
2 6 3
input #1
1 0 1 0
input #2
0 2 0 2
input #3
1 1 1 1

Challenge: Transformers

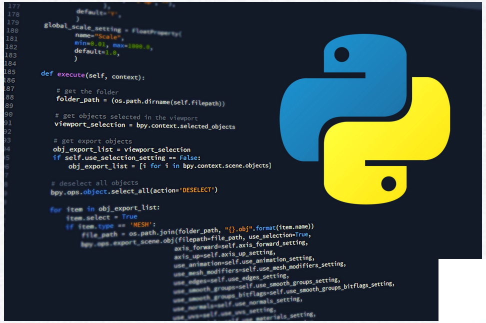


The GPT-2 Model





Coding GPT-2 in Vanilla Python



```python
def self_attention(attn, x):
    # Adds the mask so decoder cannot see values after current word
    def mask(x):
        new_mat = x
        for i in range(seq_len):
            for j in range(i+1, seq_len):
                new_mat[i, j] = -math.inf
        return new_mat

    assert(x.shape == (config['size']['emb_dim'], seq_len))

    adim = config['size']['emb_dim'] // config['size']['attn_heads']

    norm_x = layernorm(x, attn['ln1_w'], attn['ln1_b'])

    # NTFS: new memory initialized
    at = attn['attn_w'].dot(norm_x) + attn['attn_b']
    gq, gk, gv = np.vsplit(at, 3)

    for hdi in range(12):#range(config['size']['attn_heads']):     # NTFS: loop can be parallelized
        bq, bk, bv = gq[hdi*adim:(hdi+1)*adim], gk[hdi*adim:(hdi+1)*adim], gv[hdi*adim:(hdi+1)*adim]
        scalars = bk.T.dot(bq) / np.sqrt(adim)
        scalars = np.transpose(casually_masked_softmax(np.transpose(scalars)))
        gq[hdi*adim:(hdi+1)*adim] = bv.dot(scalars)

    return attn['proj_w'].dot(gq) + attn['proj_b'] + x
```

Coding GPT-2 in C

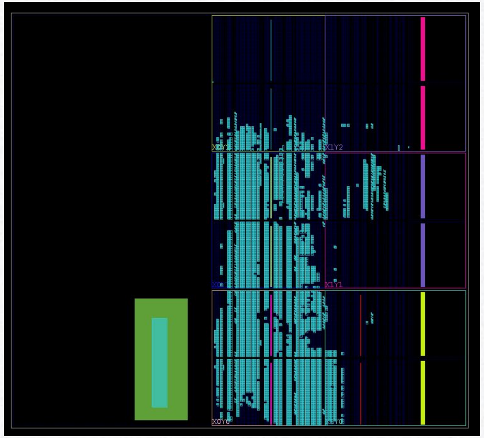
$$
\begin{array}{c|cccc}
 & \color{red}{1} & \color{red}{2} & \color{red}{\dots} & \color{red}{n} \\
\hline
\color{green}{1} & a_{11} & a_{12} & \dots & a_{1n} \\
\color{green}{2} & a_{21} & a_{22} & \dots & a_{2n} \\
\color{green}{3} & a_{31} & a_{32} & \dots & a_{3n} \\
\color{green}{\vdots} & \vdots & \vdots & \vdots & \vdots \\
\color{green}{m} & a_{m1} & a_{m2} & \dots & a_{mn}
\end{array}
$$

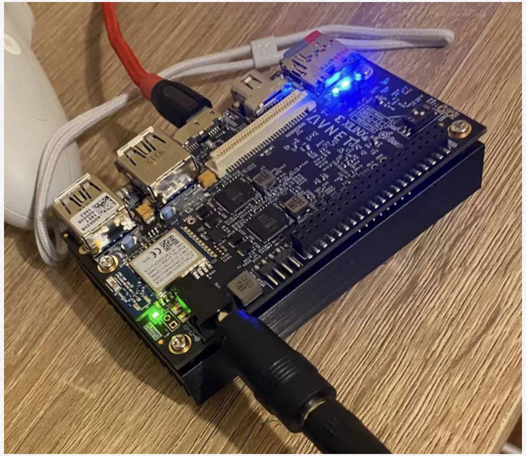
```c
/// MATRIX FUNCTIONS
struct Matrix {
    val_t *_data;
    dim_t  _rows;
    dim_t  _cols;
    bool rowmajor;
};
void matrix_construct(struct Matrix *m, dim_t rows, dim_t cols, val_t data[]) {
    // m should already be declared and allocated
    // assert(sizeof(data) == rows*cols*sizeof(val_t)); // not actually, bc data has decayed into ptr
    m->_data = data;
    m->_rows = rows;
    m->_cols = cols;
    m->rowmajor = true;
}
val_t get(struct Matrix *m, dim_t row, dim_t col) {
    if (m->rowmajor) return m->_data[row * m->_cols + col];
    else             return m->_data[col * m->_rows + row];
}
void set(struct Matrix *m, dim_t row, dim_t col, val_t val) {
    if (m->rowmajor) m->_data[row * m->_cols + col] = val;
    else             m->_data[col * m->_rows + row] = val;
}
```

```c
struct Matrix * self_attention(struct Matrix *m, struct Matrix * ln_w, struct Matrix * ln_b, struct M
  m = layer_norm(m, ln_w, ln_b);

  val_t adim = m->_rows / heads;
  // attn weights/biases
  aux_m = add_biases(matrix_dot(attn_w, m, aux_m), attn_b);
  /*aux_m = pointwise_relu(aux_m);
  m = add_biases(matrix_dot(proj_w, aux_m, m), proj_b);*/

  for(int h = 0; h < heads; h++){
    //split into key/query/val
    for(int i = 0; i < adim; i++){
      for(int j = 0; j < m->_cols; j++){
        set(query, i, j, get(aux_m, adim*h + i, j));
      }
    }
    for(int i = 0; i < adim; i++){
      for(int j = 0; j < m->_cols; j++){
        set(key, i, j, get(aux_m, i+m->_rows+adim*h, j));
      }
    }
    for(int i = 0; i < adim; i++){
      for(int j = 0; j < m->_cols; j++){
        set(value, i, j, get(aux_m, i+2*m->_rows+adim*h, j));
      }
    }

    //matrix_print(query);
    //matrix_print(key);

    key = matrix_transpose(key);

    aux_attn_m = matrix_dot(key, query, aux_attn_m);

    key = matrix_transpose(key);
    //printf("aux_attn_m \n");
```

High Level Synthesis

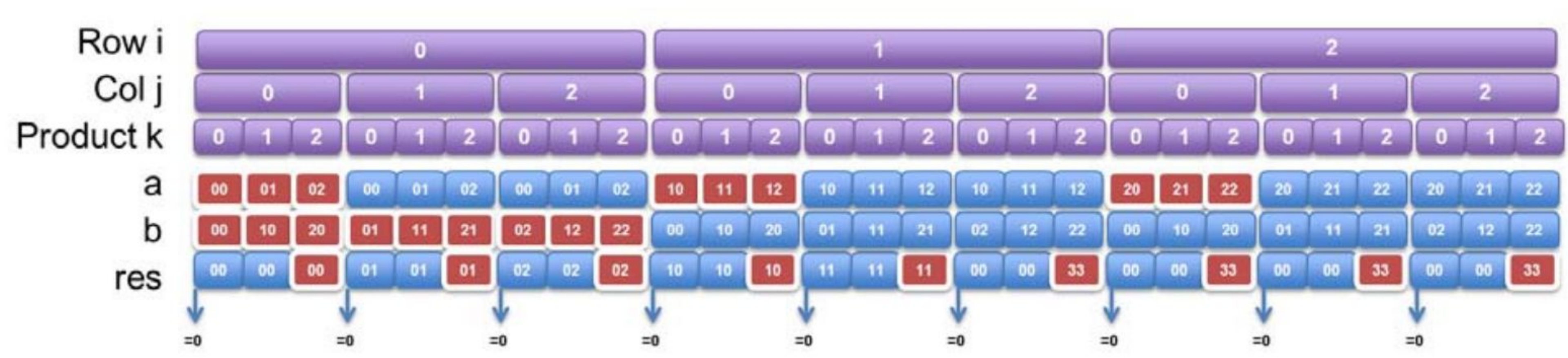
```
struct Matrix * self_attention(struct Matrix *m, struct Matrix * ln_w, str
  m = layer_norm(m, ln_w, ln_b);

  val_t adim = m->_rows / heads;
  // attn weights/biases
  aux_m = add_biases(matrix_dot(attn_w, m, aux_m), attn_b);
  /*aux_m = pointwise_relu(aux_m);
  m = add_biases(matrix_dot(proj_w, aux_m, m), proj_b);*/

  for(int h = 0; h < heads; h++){
    //split into key/query/val
    for(int i = 0; i < adim; i++){
      for(int j = 0; j < m->_cols; j++){
        set(query, i, j, get(aux_m, adim*h + i, j));
      }
    }
    for(int i = 0; i < adim; i++){
      for(int j = 0; j < m->_cols; j++){
        set(key, i, j, get(aux_m, i+m->_rows+adim*h, j));
      }
    }
    for(int i = 0; i < adim; i++){
      for(int j = 0; j < m->_cols; j++){
        set(value, i, j, get(aux_m, i+2*m->_rows+adim*h, j));
      }
    }

    //matrix_print(query);
    //matrix_print(key);

    key = matrix_transpose(key);

    aux_attn_m = matrix_dot(key, query, aux_attn_m);

    key = matrix_transpose(key);
    //printf("aux_attn_m_\n");
```





Challenge: Hardware

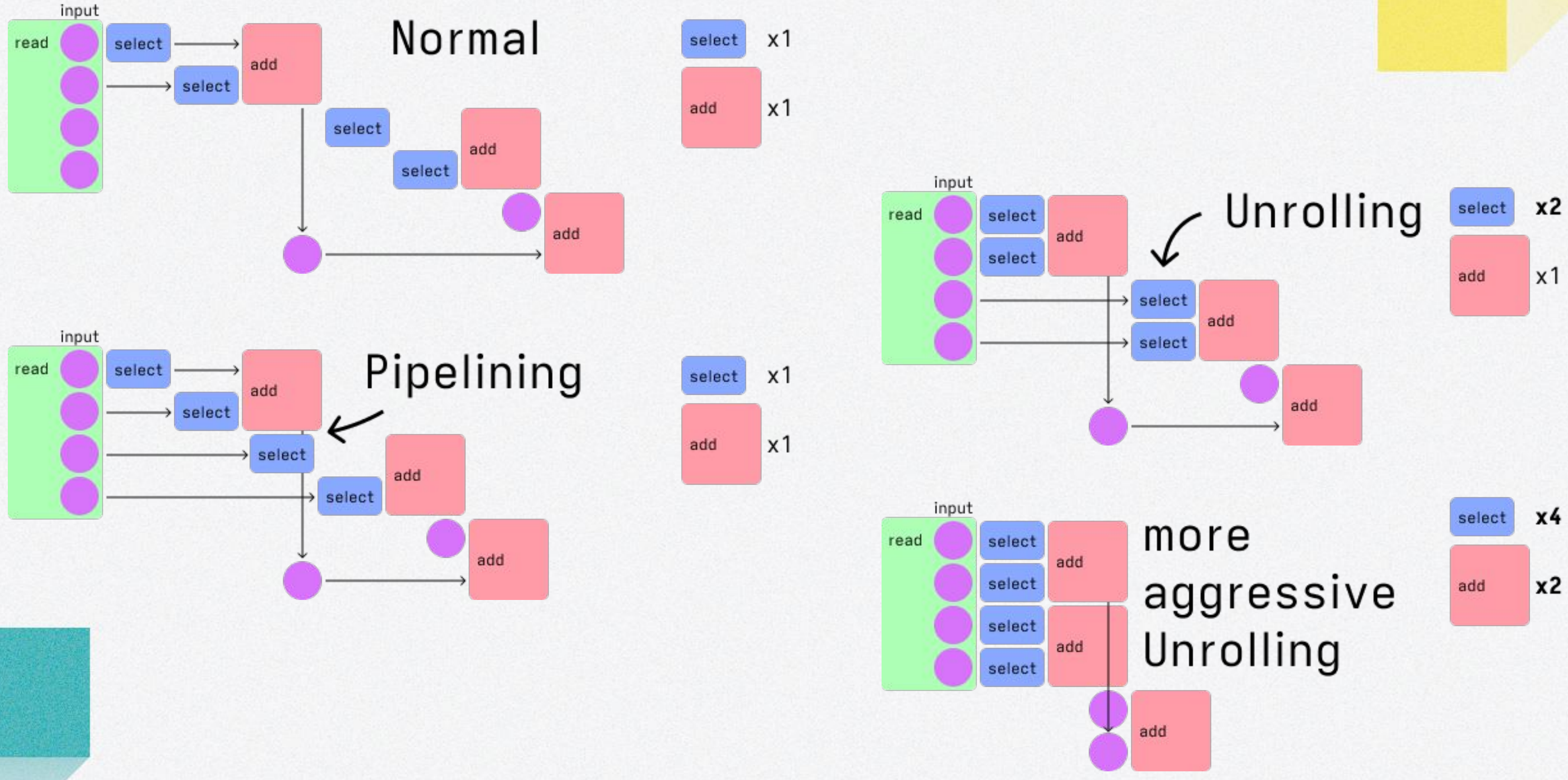
- No dynamic memory allocation
- No syscalls
- Arbitrary precision
- Hardware messiness

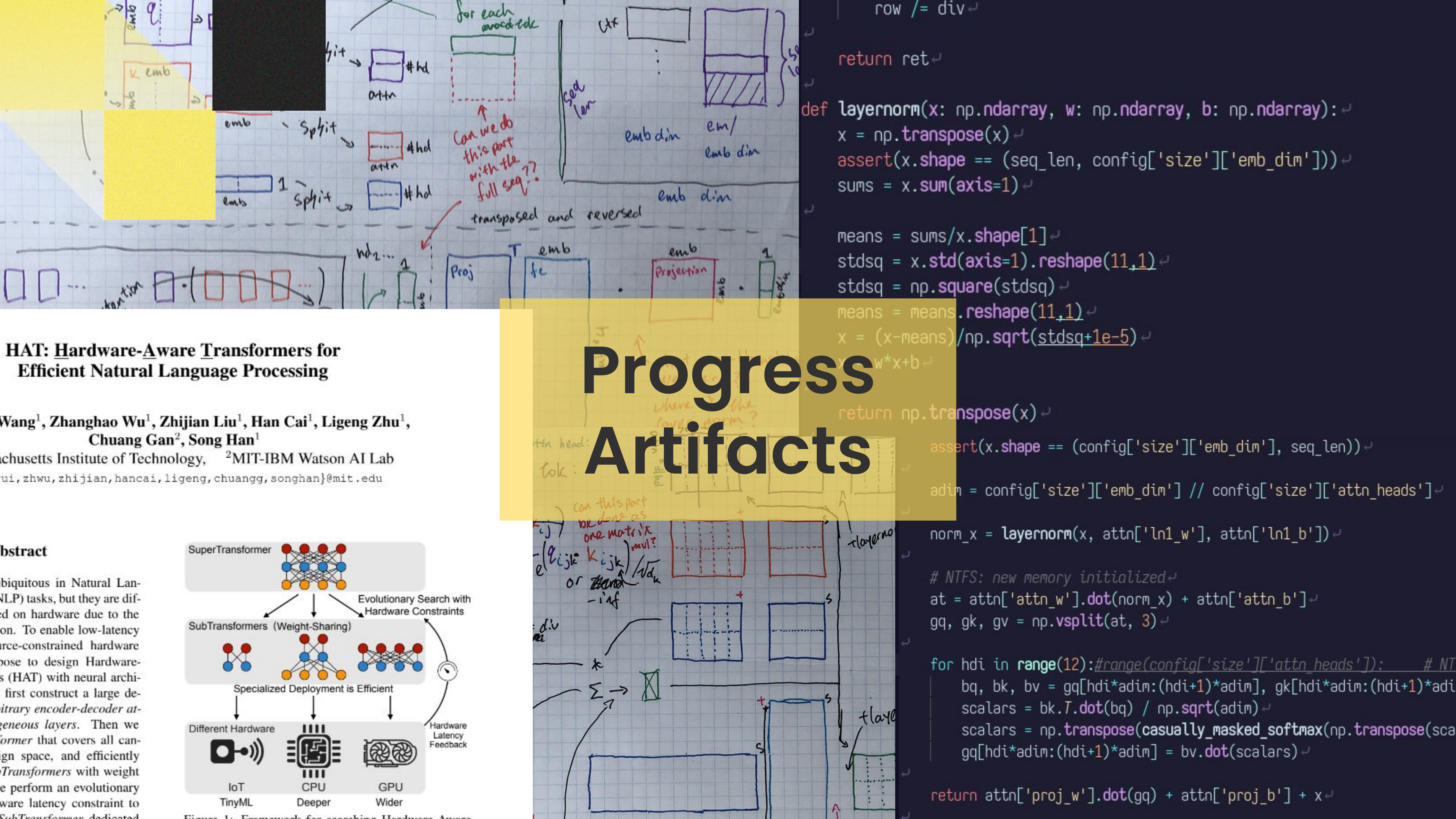


Re-coding GPT-2 in C

```c
/// MATRIX FUNCTIONS
struct Matrix {
    val_t *_data;
    dim_t  _rows;
    dim_t  _cols;
    bool rowmajor;
};
void matrix_construct(struct Matrix *m, dim_t rows, dim_t cols, val_t data[]) {
    // m should already be declared and allocated
    // assert(sizeof(data) == rows*cols*sizeof(val_t)); // not actually, bc data has decayed into ptr
    m->_data = data;
    m->_rows = rows;
    m->_cols = cols;
    m->rowmajor = true;
}
val_t get(struct Matrix *m, dim_t row, dim_t col) {
    if (m->rowmajor) return m->_data[row * m->_cols + col];
    else             return m->_data[col * m->_rows + row];
}
void set(struct Matrix *m, dim_t row, dim_t col, val_t val) {
    if (m->rowmajor) m->_data[row * m->_cols + col] = val;
    else             m->_data[col * m->_rows + row] = val;
}
```

```c
val_t get(val_t *dat, int rowmajor, dim_t rows, dim_t cols, dim_t row, dim_t col) {
    if (rowmajor) return dat[row * cols + col];
    else return dat[col * rows + row];
}
void set(val_t *dat, int rowmajor, dim_t rows, dim_t cols, dim_t row, dim_t col, val_t val) {
    if (rowmajor) dat[row * cols + col] = val;
    else          dat[col * rows + row] = val;
}
```


Normal
input
read
select
select
add
select
select
add
add
select x1
add x1
Pipelining
input
read
select
select
add
select
select
add
add
select x1
add x1
Unrolling
input
read
select
select
add
select
select
add
add
select x2
add x1
more aggressive Unrolling
input
read
select
select
select
select
add
add
add
select x4
add x2


Progress
Artifacts
HAT: Hardware-Aware Transformers for
Efficient Natural Language Processing
Zhanghao Wu, Zhijian Liu, Han Cai, Ligeng Zhu,
Chuang Gan, Song Han
MIT-IBM Watson AI Lab
SuperTransformer
Evolutionary Search with
Hardware Constraints
SubTransformers (Weight-Sharing)
Specialized Deployment is Efficient
Different Hardware
Hardware
Latency
Feedback
IoT
TinyML
CPU
Deeper
GPU
Wider
return ret
def layernorm(x: np.ndarray, w: np.ndarray, b: np.ndarray):
x = np.transpose(x)
assert(x.shape == (seq_len, config['size']['emb_dim']))
sums = x.sum(axis=1)
means = sums/x.shape[1]
stdsq = x.std(axis=1).reshape(11,1)
stdsq = np.square(stdsq)
means = means.reshape(11,1)
x = (x-means)/np.sqrt(stdsq+1e-5)
return np.transpose(x)
assert(x.shape == (config['size']['emb_dim'], seq_len))
norm_x = layernorm(x, attn['ln1_w'], attn['ln1_b'])
# NTFS: new memory initialized
at = attn['attn_w'].dot(norm_x) + attn['attn_b']
gq, gk, gv = np.vsplit(at, 3)
scalars = bk.T.dot(bq) / np.sqrt(adim)
gq[hdi*adim:(hdi+1)*adim] = bv.dot(scalars)
return attn['proj_w'].dot(gq) + attn['proj_b'] + x

4

Impacts

the future is now

Impacts

Drones + Robots

Disaster response robots must react quickly to novel situations and survive days on one charge.

Always-on Devices

Mobile assistants activation triggers like “Hey, Siri” are primarily limited by power consumption.

Self-driving Cars

The range and comfort of self-driving cars can be significantly improved through hardware optimization.

Datacenters

Google’s 2015 TPU v1 increased inference throughput by 71X while using 25% less power, by accelerating large matrix multiplication alone.

Future Directions

Model Architectures

GPT-2 was chosen for simplicity, but application-specific architectures exist.

Application Constraints

Depending on the final application, circuits can be optimized for speed or efficiency.

Joint Evolution

Coevolution of model and hardware architectures will open new doors.

Special Thanks To

EIC Lab @ Rice
X-Camp Internship Program

Thanks

Please reach out with any questions!