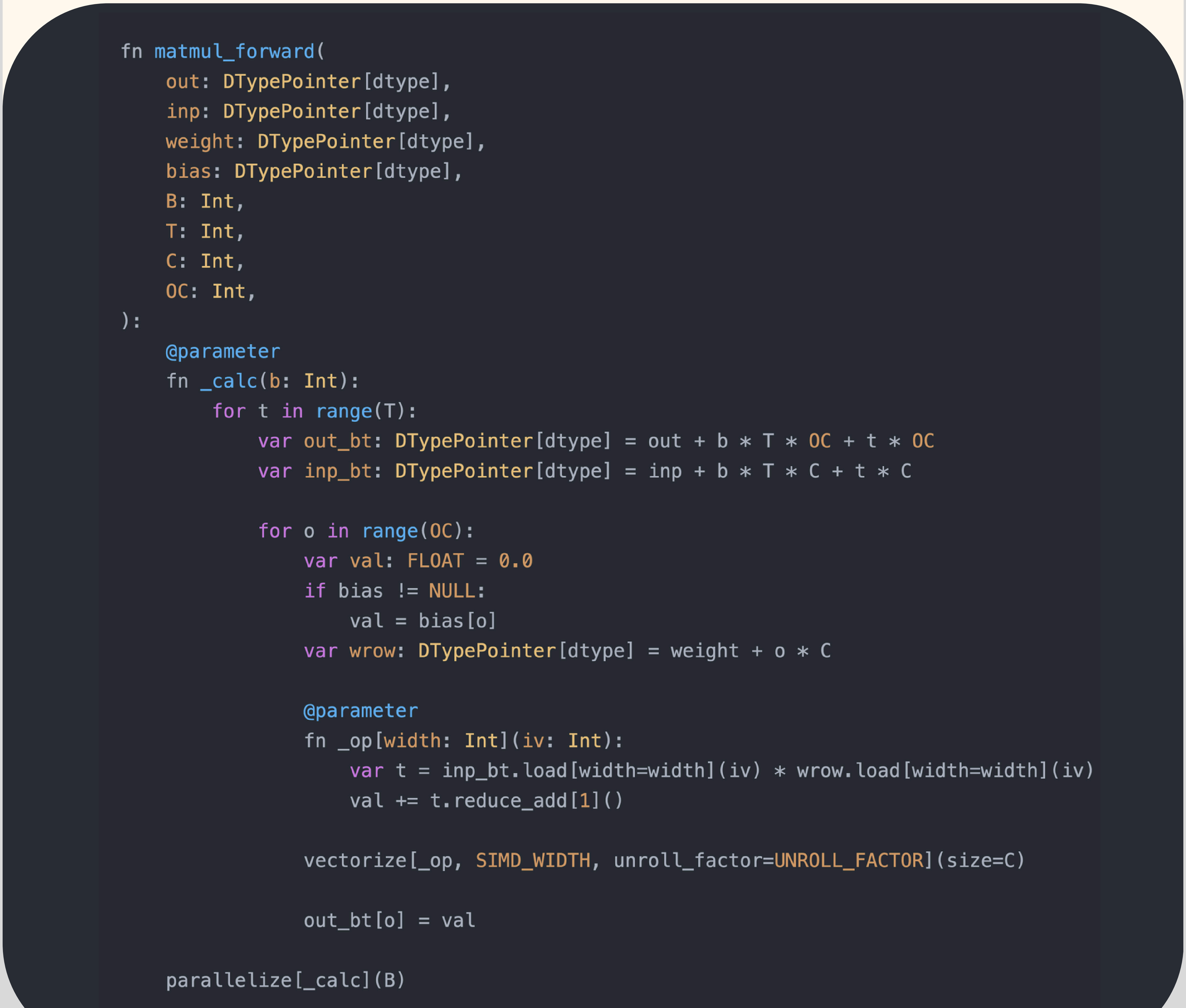
LLM.🔥

GPT-2 written in **Mojo**

1. **Matrix Multiplication**

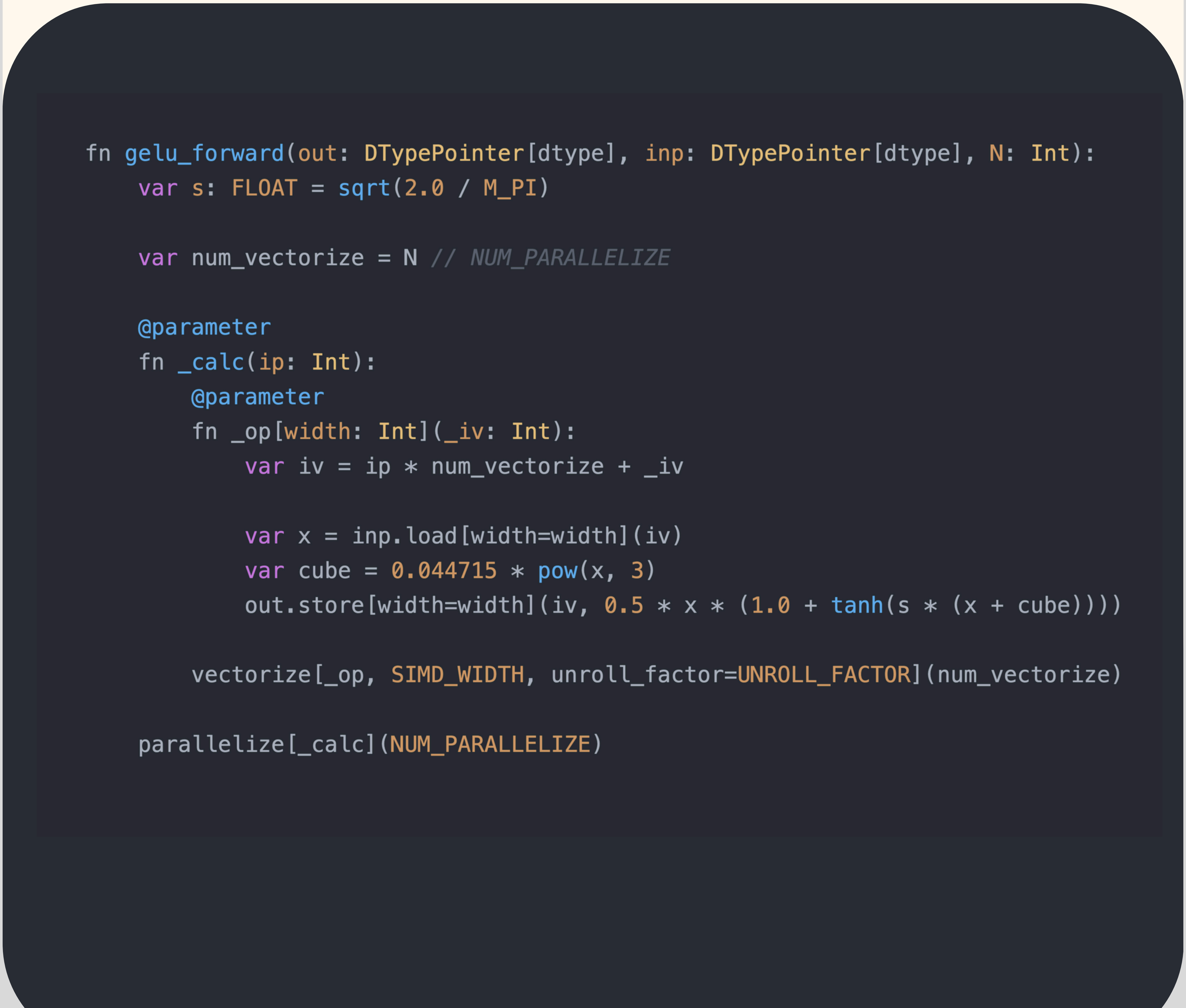


```
fn matmul_forward(
    out: DTypePointer[dtype],
    inp: DTypePointer[dtype],
    weight: DTypePointer[dtype],
    bias: DTypePointer[dtype],
    B: Int,
    T: Int,
    C: Int,
    OC: Int,
):
    @parameter
    fn _calc(b: Int):
        for t in range(T):
            var out_bt: DTypePointer[dtype] = out + b * T * OC + t * OC
            var inp_bt: DTypePointer[dtype] = inp + b * T * C + t * C

            for o in range(OC):
                var val: FLOAT = 0.0
                if bias != NULL:
                    val = bias[o]
                var wrow: DTypePointer[dtype] = weight + o * C

                @parameter
                fn _op[width: Int](iv: Int):
                    var t = inp_bt.load[width=width](iv) * wrow.load[width=width](iv)
                    val += t.reduce_add[1]()

                vectorize[_op, SIMD_WIDTH, unroll_factor=UNROLL_FACTOR](size=C)

                out_bt[o] = val

    parallelize[_calc](B)
```

LLM.🔥

GPT-2 written in **Mojo**

2. **GeLU** Activation Function

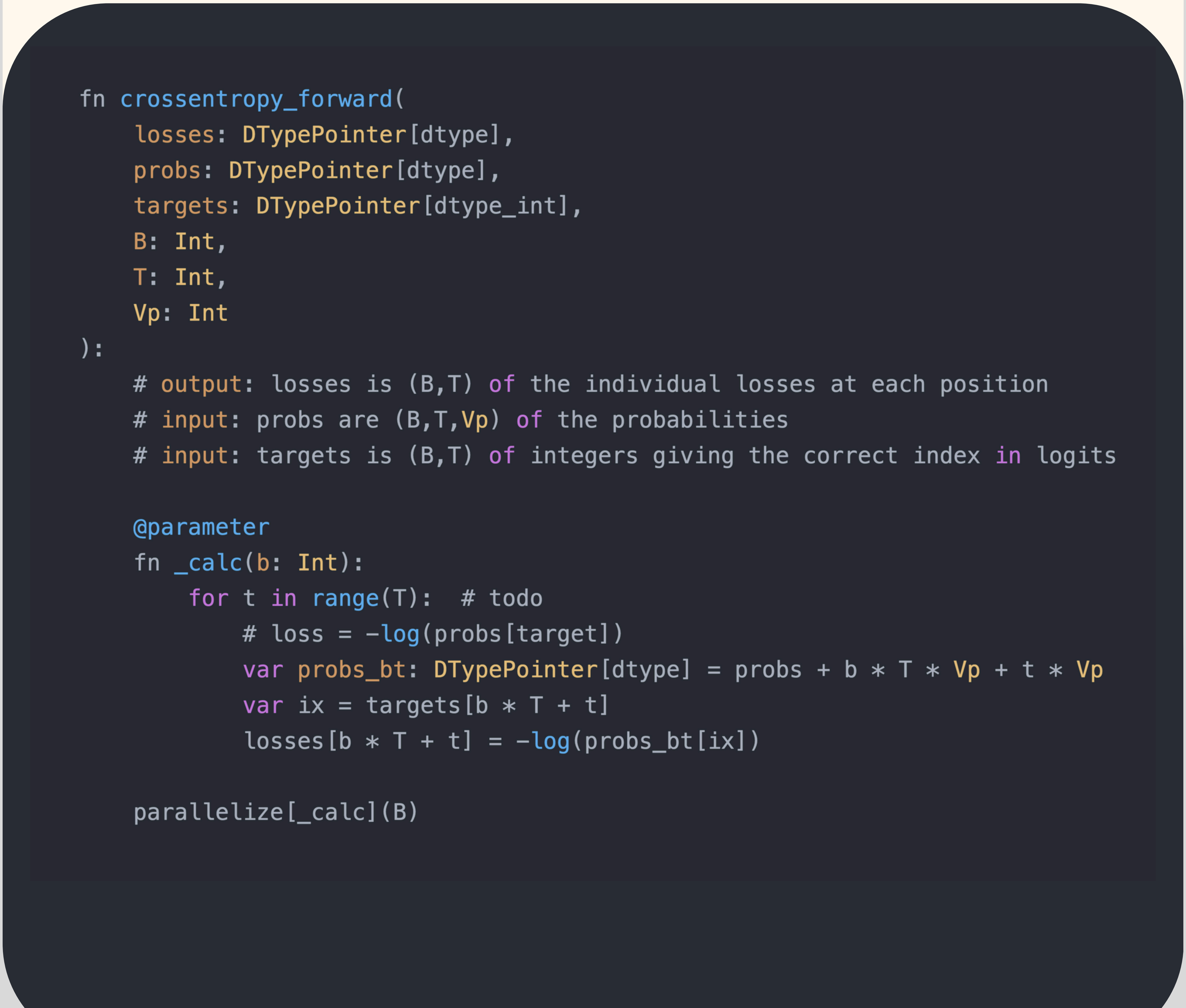


```
fn gelu_forward(out: DTypePointer[dtype], inp: DTypePointer[dtype], N: Int):
    var s: FLOAT = sqrt(2.0 / M_PI)

    var num_vectorize = N // NUM_PARALLELIZE

    @parameter
    fn _calc(ip: Int):
        @parameter
        fn _op[width: Int](_iv: Int):
            var iv = ip * num_vectorize + _iv

            var x = inp.load[width=width](iv)
            var cube = 0.044715 * pow(x, 3)
            out.store[width=width](iv, 0.5 * x * (1.0 + tanh(s * (x + cube))))

        vectorize[_op, SIMD_WIDTH, unroll_factor=UNROLL_FACTOR](num_vectorize)

    parallelize[_calc](NUM_PARALLELIZE)
```

LLM.🔥

GPT-2 written in **Mojo**

3. **Cross Entropy Loss**



```
fn crossentropy_forward(
    losses: DTypePointer[dtype],
    probs: DTypePointer[dtype],
    targets: DTypePointer[dtype_int],
    B: Int,
    T: Int,
    Vp: Int
):
    # output: losses is (B,T) of the individual losses at each position
    # input: probs are (B,T,Vp) of the probabilities
    # input: targets is (B,T) of integers giving the correct index in logits

    @parameter
    fn _calc(b: Int):
        for t in range(T):  # todo
            # loss = -log(probs[target])
            var probs_bt: DTypePointer[dtype] = probs + b * T * Vp + t * Vp
            var ix = targets[b * T + t]
            losses[b * T + t] = -log(probs_bt[ix])

    parallelize[_calc](B)
```