The Transformer Block

Lecture 09 — microgpt deep dive

Prof. Young Woo Choi

Department of Physics, Sogang University

A visual walkthrough of the architecture inside `gpt(token_id, pos_id, keys, values)`

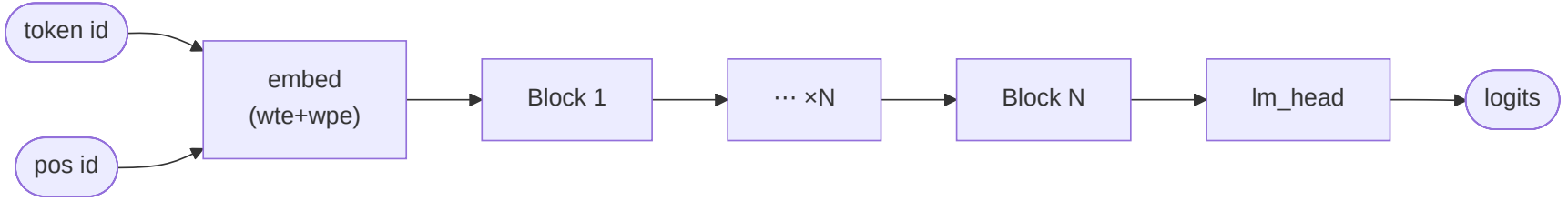
4/9/2026

The whole picture

Input: one token id t, one position id p · **Output:** logits over the vocabulary — "what comes next?"

The model is a **stateless function** (params + KV cache in, logits out):

```
gpt(token_id, pos_id, keys, values) -> logits
```



Everything else is just plumbing.

Why attention? — physics analogy

Many-body interaction

Each particle "feels" all others through a kernel:

$$\phi_i^{\text{new}} = \sum_j K(i,j)\,\phi_j$$

The **kernel** $K(i,j)$ encodes who-talks-to-whom.

Self-attention

Each token aggregates information from previous tokens, weighted by **learned similarity**:

$$x_i^{\text{new}} = \sum_{j \le i} a_{ij}\, v_j$$

where a_{ij} is computed on-the-fly from the data.

Key idea: attention = a *data-dependent* interaction kernel. The "couplings" are not parameters — they are computed from the inputs themselves.

Self-attention: Q, K, V

For each token, learn three projections:

Query $\mathbf{q}$

"What am I **looking for**?"

$\mathbf{q} = W_Q\,\mathbf{x}$

Key $\mathbf{k}$

"What do I **contain**?"

$\mathbf{k} = W_K\,\mathbf{x}$

Value $\mathbf{v}$

"What do I **offer** if matched?"

$\mathbf{v} = W_V\,\mathbf{x}$

Then:

$$a_{ij} = \mathrm{softmax}_j\left(\frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{d_{\text{head}}}}\right) \qquad \mathbf{x}_i^{\text{new}} = \sum_{j \le i} a_{ij}\,\mathbf{v}_j$$

Causal mask: the sum is only over $j \le i$ — a token at time t can only see the past. This is the **only** place tokens communicate.

Self-attention: a tiny worked example

Sequence: `e m m` at positions $0, 1, 2$. We compute the output at $t = 2$.

For each position i we already have $\mathbf{q}_i, \mathbf{k}_i, \mathbf{v}_i$. Then:

$$s_{2j} = \frac{\mathbf{q}_2 \cdot \mathbf{k}_j}{\sqrt{d_{\text{head}}}} \qquad j = 0, 1, 2$$

$$(a_{20}, a_{21}, a_{22}) = \text{softmax}(s_{20}, s_{21}, s_{22})$$

$$\mathbf{x}_2^{\text{new}} = a_{20}\,\mathbf{v}_0 + a_{21}\,\mathbf{v}_1 + a_{22}\,\mathbf{v}_2$$

Reading the result

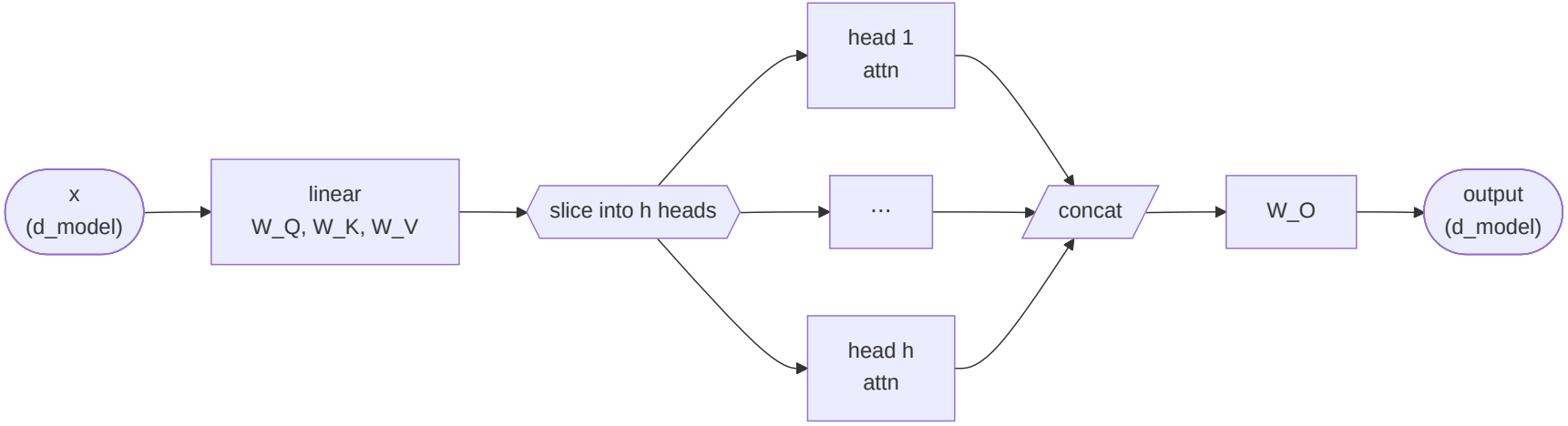
- $a_{2,0}$ tells you **how much** the position-2 token attends to "e"
- $a_{2,2}$ is self-attention to itself
- The output $\mathbf{x}_2^{\text{new}}$ is a **weighted blend of values** from positions 0…2

Why $\sqrt{d_{\text{head}}}$?

Without the scale, dot products grow with dimension and softmax saturates → vanishing gradients. The scale keeps logits at unit-variance.

Multi-head attention

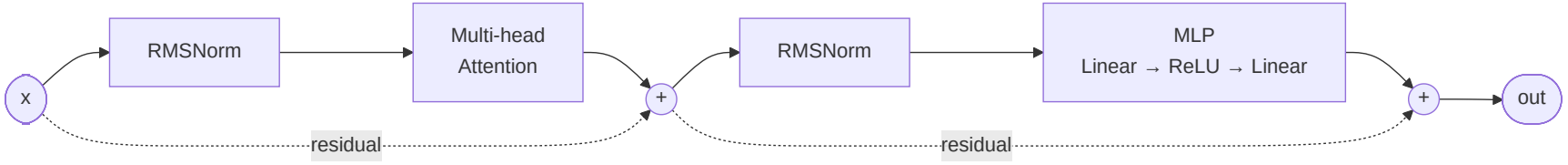
One head sees one kind of relationship. We want several in parallel. Q, K, V are computed **once** at full d_{model}, then **sliced** into h chunks of size d_{head}; each chunk runs an independent attention.



$d_{\text{model}} = h \cdot d_{\text{head}}$ · same parameter budget as one giant head, but lets the model learn several attention patterns in parallel — e.g. one head for nearby syntactic links, another for long-range coreference.

The Block: attention + MLP

Two sub-layers, same recipe: $y = x + \mathrm{SubLayer}(\mathrm{Norm}(x))$ (pre-norm — more stable than post-norm at depth).



Sub-layer	Role	Talks across tokens?
Attention	mix information across positions	yes
MLP	per-token feature transformation	no

> Transformer = **Communication** (attention) interspersed with **Computation** (MLP).

Residual connections & normalisation

Residual: the "highway"

$$\mathbf{x}_{\ell+1} = \mathbf{x}_\ell + f(\mathbf{x}_\ell)$$

- gradients flow **directly** through the `+`
- each block only has to learn a *correction*
- physics analogy: perturbative correction to a free propagator

Without residuals, gradients vanish exponentially with depth — we couldn't train deep stacks at all.

RMSNorm: keep activations sane

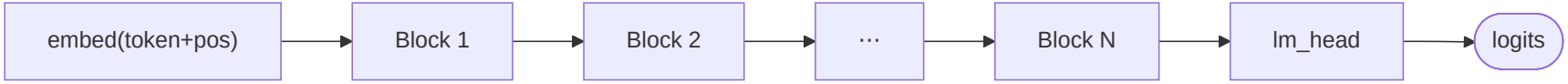
$$\mathrm{RMSNorm}(\mathbf{x}) = \frac{\mathbf{x}}{\sqrt{\frac{1}{d}\sum_i x_i^2 + \varepsilon}}$$

- rescales each token's activation to **unit RMS**
- prevents activations from blowing up or collapsing across depth
- simpler than LayerNorm (no mean subtraction, no learned bias) — same job

```
def rmsnorm(x):
    ms = sum(xi*xi for xi in x) / len(x)
    scale = (ms + 1e-5) ** -0.5
    return [xi * scale for xi in x]
```

Together: residuals make depth *trainable*, normalisation makes depth *stable*. You need both.

Stacking blocks



The "residual stream" view

Think of $\mathbf{x}$ as a **shared bus** that every block reads from and writes to (via the $+$). Early blocks: surface features. Middle blocks: syntactic / structural relations. Late blocks: task-specific predictions.

Scaling laws — in one line

> **bigger N, bigger d_{model}, more data → better.**

- GPT-2 small: $N = 12$, $d_{\text{model}} = 768$, 12 heads
- GPT-3: $N = 96$, $d_{\text{model}} = 12288$, 96 heads

The *recipe* is the same. Only the dimensions change.

Map back to the microgpt code

```
def gpt(token_id, pos_id, keys, values):
    tok_emb = state_dict['wte'][token_id]                    # ← embedding
    pos_emb = state_dict['wpe'][pos_id]                      # ← positional
    x = [t + p for t, p in zip(tok_emb, pos_emb)]
    x = rmsnorm(x)

    for li in range(n_layer):                                # ← stack N blocks
        # ── Attention sub-layer ──────────────────────────
        x_residual = x
        x = rmsnorm(x)                                       # ← pre-norm
        q = linear(x, ...wq); k = linear(x, ...wk); v = linear(x, ...wv)   # ← Q,K,V
        keys[li].append(k); values[li].append(v)             # ← KV cache
        for h in range(n_head):                              # ← multi-head
            ...  attn_logits / √d_head  →  softmax  →  weighted v
        x = linear(x_attn, ...wo)
        x = [a + b for a, b in zip(x, x_residual)]           # ← residual

        # ── MLP sub-layer ────────────────────────────────
        x_residual = x
        x = rmsnorm(x)                                       # ← pre-norm
        x = linear(x, ...fc1); x = [xi.relu() for xi in x]; x = linear(x, ...fc2)
        x = [a + b for a, b in zip(x, x_residual)]           # ← residual

    logits = linear(x, state_dict['lm_head'])                # ← unembedding
    return logits
```

Recap — four ideas

1. Self-attention

Data-dependent interaction kernel. Q asks, K answers, V is what flows. Causal mask: only past → present.

2. Multi-head

Same parameter budget, several attention patterns in parallel. Slice d_{model} into h chunks of d_{head}.

3. Residual + RMSNorm

Highway for gradients + activation rescaling. Makes deep stacks both **trainable** and **stable**.

4. Block stack

Communication (attn) ↔ Computation (MLP), repeated N times on a shared residual stream.

Next: we resume the microgpt walkthrough from the training loop and watch these pieces actually learn.

Further reading

- **Karpathy**, *Let's build GPT from scratch* — github.com/karpathy/ng-video-lecture
- **Karpathy**, *microgpt* — karpathy.ai/microgpt.html (this is what we're walking through)
- **Jay Alammar**, *The Illustrated Transformer* — jalammar.github.io/illustrated-transformer
- **3Blue1Brown**, *Attention in transformers* — 3blue1brown.com/lessons/attention
- **poloclub**, *Transformer Explainer* — poloclub.github.io/transformer-explainer (live GPT-2 in the browser)
- **Vaswani et al. (2017)**, *Attention Is All You Need* — arxiv.org/abs/1706.03762