

Building an LLM from Zero

Chapter 9: The Transformer Block

Module 2: Building the Model

Truong (Jack) Luu · jackluu.io/book

Where we are

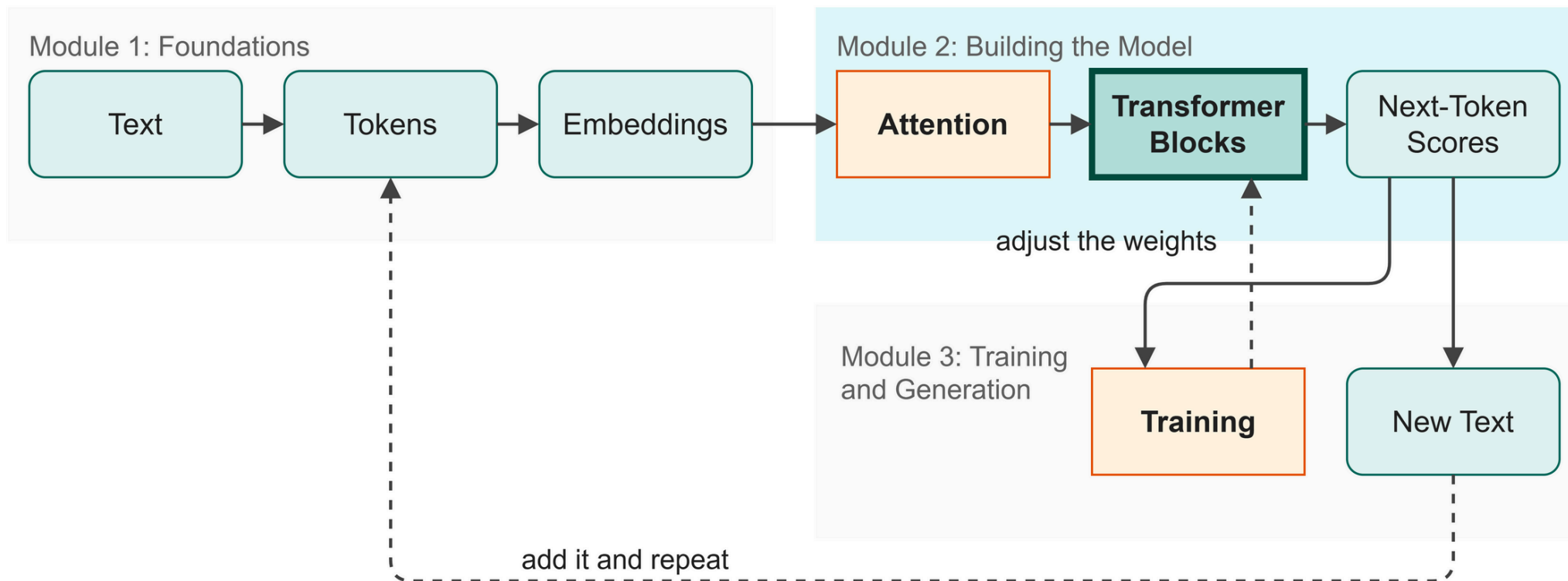


Figure 9.1: We combine our pieces into a reusable building block.

Today the Transformer Blocks stage gets its block.

What this builds on (60-second recap)

- **Chapter 8** built the **feed-forward** layer: expand 128 to 512, GELU, compress back to 128
- **LayerNorm** resets each token's numbers to mean 0 and spread 1, before the hard work
- Every piece so far takes `[B, T, 128]` in and gives `[B, T, 128]` out

Next: three ingredients, one reusable brick.

In this chapter you will

1. Combine the parts into a single brick called a **Transformer block**
2. Add **residual connections** so deep networks can learn effectively
3. **Stack** multiple blocks to build depth

Words to know (each is glossed again where it first appears):

- **Transformer block**: a reusable block of code containing attention, feed-forward and normalization
- **Residual connection**: a shortcut that lets information skip a step, keeping the original signal intact

Putting the pieces together

A single **Transformer block** (attention, feed-forward and LayerNorm in one unit) is two lines of code:

```
x = x + self.attn(self.ln1(x)) # "listen, then add to what I know"  
x = x + self.ff(self.ln2(x))   # "think, then update what I know"
```

- Line 1: normalize, let the tokens talk (attention), add the result to `x`
- Line 2: normalize, let each token think (feed-forward), add the result to `x`

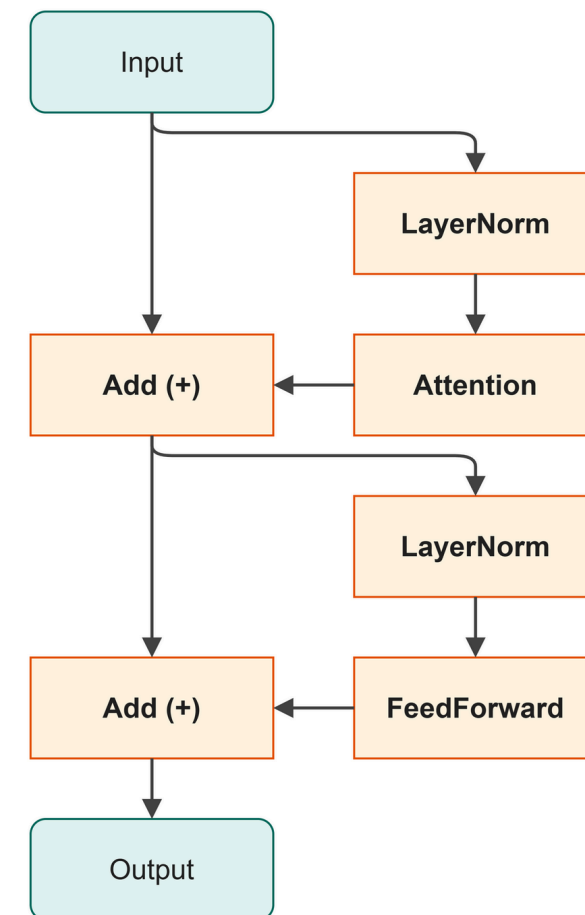
Notice the `x = x + ...` pattern. This is the secret to making deep neural networks work.

Trick 1: the residual connection

We **add** the attention output to the original data instead of replacing it: a **residual connection** (skip connection).

Like passing the written note alongside a game of telephone: the original survives.

The direct $+x$ highway lets the signal, and the training feedback, travel cleanly through dozens of stacked layers.



Trick 2: pre-layer norm

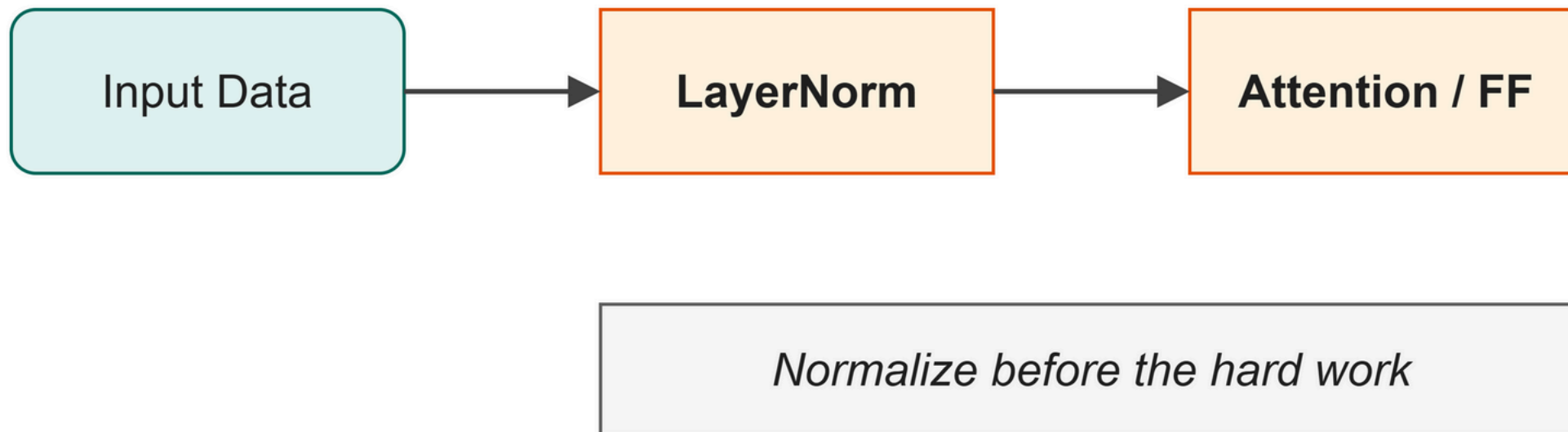


Figure 9.3: Normalize the data before the hard work.

LayerNorm comes **before** each step: `attention(LayerNorm(x))`. This is the **pre-norm** convention. Attention and feed-forward always get well-behaved numbers, which stabilizes early training.

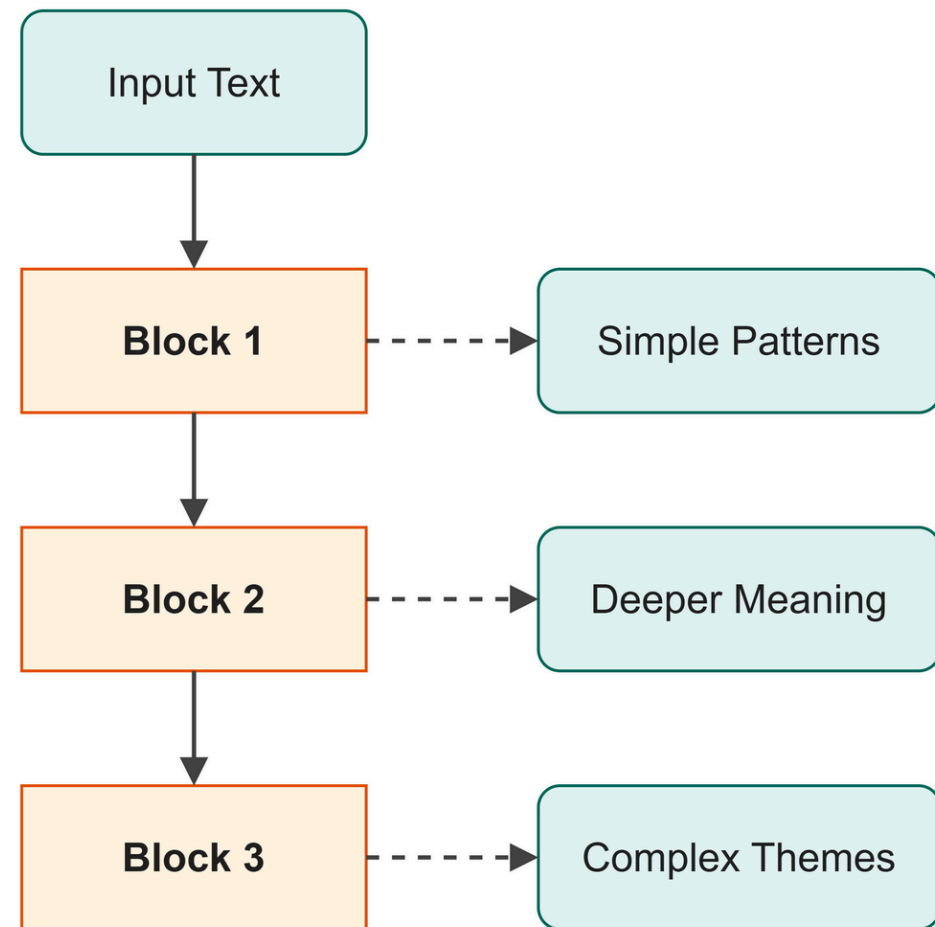
Normalize first, then do the hard work.

Stacking blocks

One block is not enough. We stack blocks in sequence; our model uses **4**.

Like reading a complex document three times: first the characters, then their motivations, then the themes.

Earlier blocks capture simple patterns, later blocks capture deeper meaning. This is why deeper models perform better.



Code: the block

```
class TransformerBlock(nn.Module):
    def __init__(self):
        super().__init__()
        self.attn = MultiHeadAttention()
        self.ff    = FeedForward()
        self.ln1   = nn.LayerNorm(config.n_embd)
        self.ln2   = nn.LayerNorm(config.n_embd)

    def forward(self, x):
        x = x + self.attn(self.ln1(x))
        x = x + self.ff(self.ln2(x))
        return x
```

- Lines 4 to 7 build the block from attention, feed-forward and two LayerNorms
- Lines 10 to 11 are the two residual steps from the first slide

The file is `src/ch08_transformer_block.py`. (Source files are numbered one lower than chapters.)

Run it

```
$ python src/ch08_transformer_block.py
One TransformerBlock parameters: 197,888
  MultiHeadAttention : 65,664
  FeedForward        : 131,712
  LayerNorms (x2)    : 512
...
Original token norm : 11.470
Attention output norm: 4.599
After residual norm : 12.903 (combined)
...
Total params       : 791,552 (4 x 197,888)
Output shape: torch.Size([2, 10, 128]) (unchanged after 4 blocks)
```

What just happened: one block holds 197,888 parameters, most of them in the feed-forward layer. The residual step combined the original token (norm 11.470) with the attention output (4.599). Four stacked blocks gave 791,552 parameters and the same output shape.

Shape check

Tensor	Shape	Meaning
Input to TransformerBlock	[Batch, Time, 128]	One vector of 128 numbers per token.
Output of TransformerBlock	[Batch, Time, 128]	Identical shape.
Output after 4 blocks	[Batch, Time, 128]	Still identical.

The block is **shape-preserving**. Identical input and output shapes are exactly what let us stack blocks endlessly, like bricks.

Try it: a deeper stack

1. Open `src/ch08_transformer_block.py`
2. Change the number of layers in the stack from `config.n_layers` to `12`
3. Run the script again

What the chapter says you will see: the total parameter count grows, but the output shape stays exactly the same.

Depth changes how many numbers the model has. It never changes the shape of what flows through.

In business: a modular unit

Building a house-style writing assistant, you want modular, scalable processes.

- A Transformer block is the ultimate modular unit
- If the assistant is not smart enough, you do not invent a new architecture
- You stack more blocks and train it longer

Same brick, more of them. That is how these systems scale.

Discussion

Why add the new information to the original instead of replacing it?

In pairs, 3 minutes:

1. In the telephone game, what does "passing the written message alongside" protect against?
2. The run showed the original token (11.470) and the attention output (4.599). What would the block lose if it kept only the 4.599?

Take two answers from the room.

Recap: today in 4 points

1. A **Transformer block** combines LayerNorm, attention and feed-forward layers.
2. **Residual connections** ($x + \dots$) create a direct highway for data, so deep models can learn.
3. **Pre-norm** applies LayerNorm before the hard work, which stabilizes the math.
4. Blocks **preserve the shape** of the data, so we can stack them: 4 blocks, 791,552 parameters.

Next: embeddings in, blocks in the middle, and a final layer that turns 128 numbers into 65 scores.

Check your understanding

1. Why do we add the attention output to the original input (`x + attention`)?
2. What does "pre-norm" mean?
3. Why does the Transformer block output exactly the same shape it took in?

Next up: Chapter 10, The Full GPT Architecture

- Everything you have built gets assembled end to end
- You will see the parameter breakdown of the whole model and meet **logits**, the raw scores for the next character

Before next class: run `python src/ch08_transformer_block.py` and check that the total for 4 blocks is 791,552.

References

Book and figures

- Luu, T. (2026). *Building an LLM from Zero*, Chapter 9. <https://jackluu.io/book/>
- Figures 9.1 to 9.4 are the book's own diagrams (draw.io sources in the book repository, `diagrams/`).
- Terminal output: `src/ch08_transformer_block.py`.

Further reading (from the chapter)

- He, K., Zhang, X., Ren, S., & Sun, J. (2015). *Deep residual learning for image recognition* (arXiv:1512.03385).
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). *Attention is all you need* (arXiv:1706.03762).

Dataset

- Tiny Shakespeare, from Andrej Karpathy's char-rnn project: <https://github.com/karpathy/char-rnn>