

Building an LLM from Zero

Chapter 17: Putting It All Together

Module 3: Training and Generation

Truong (Jack) Luu · jackluu.io/book

Where we are

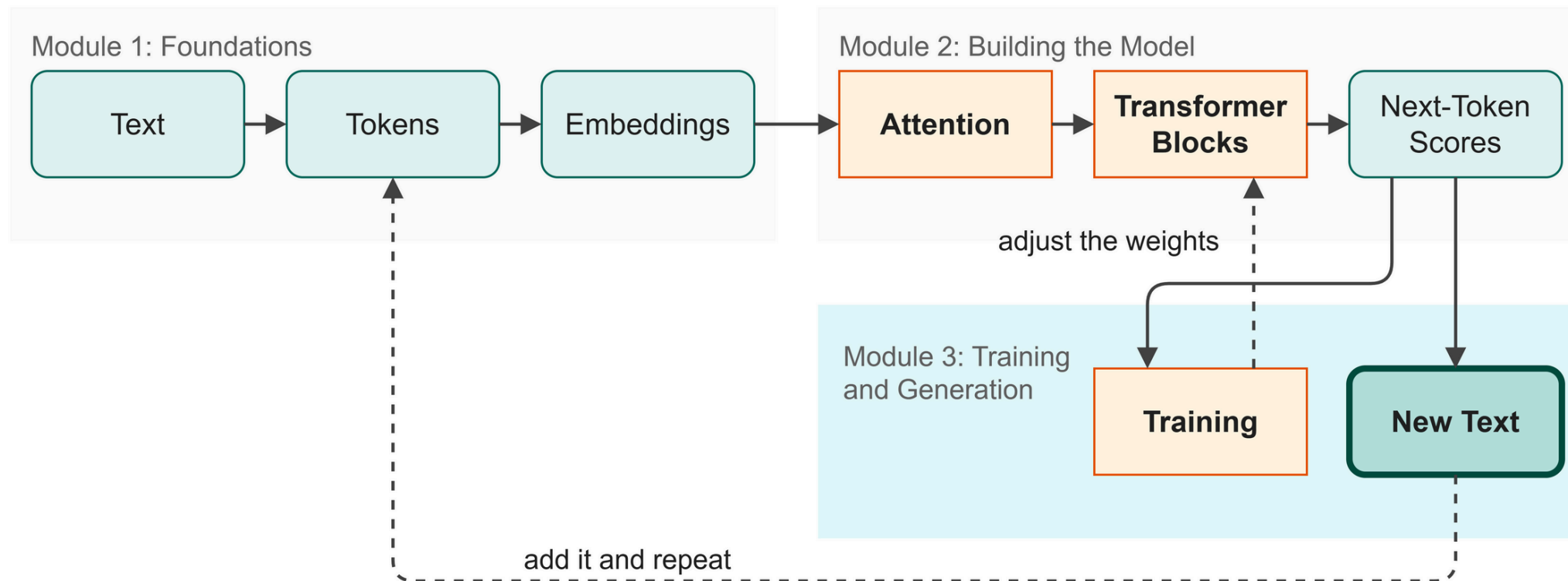


Figure 17.1: Where we are: completing the full pipeline from text to language model.

Every box on this map is now code you wrote. Today we run the whole map in one script and read what comes out.

What this builds on (60-second recap)

- **Chapter 16** added two dials to sampling: **temperature** reshapes the odds, **top-k** limits who can compete
- The order is temperature, then top-k, then softmax
- The sweet spot the book uses: temperature **0.8**, `top_k=40`

Next: the whole pipeline, end to end, and an honest look at the result.

In this chapter you will

1. Run the complete **end-to-end** pipeline in a single script
2. See the full architecture in action
3. Understand how your model relates to modern, production-grade LLMs

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

- **End to end**: a process that takes raw input (text) through every necessary step to the final output (a trained model and generated text) without manual intervention
- **Instruction tuning**: a post-training step that teaches the model to answer questions and follow instructions
- **RLHF**: Reinforcement Learning from Human Feedback, a method to train models to behave politely and align with human preferences

Not a toy: the same building blocks as GPT-2

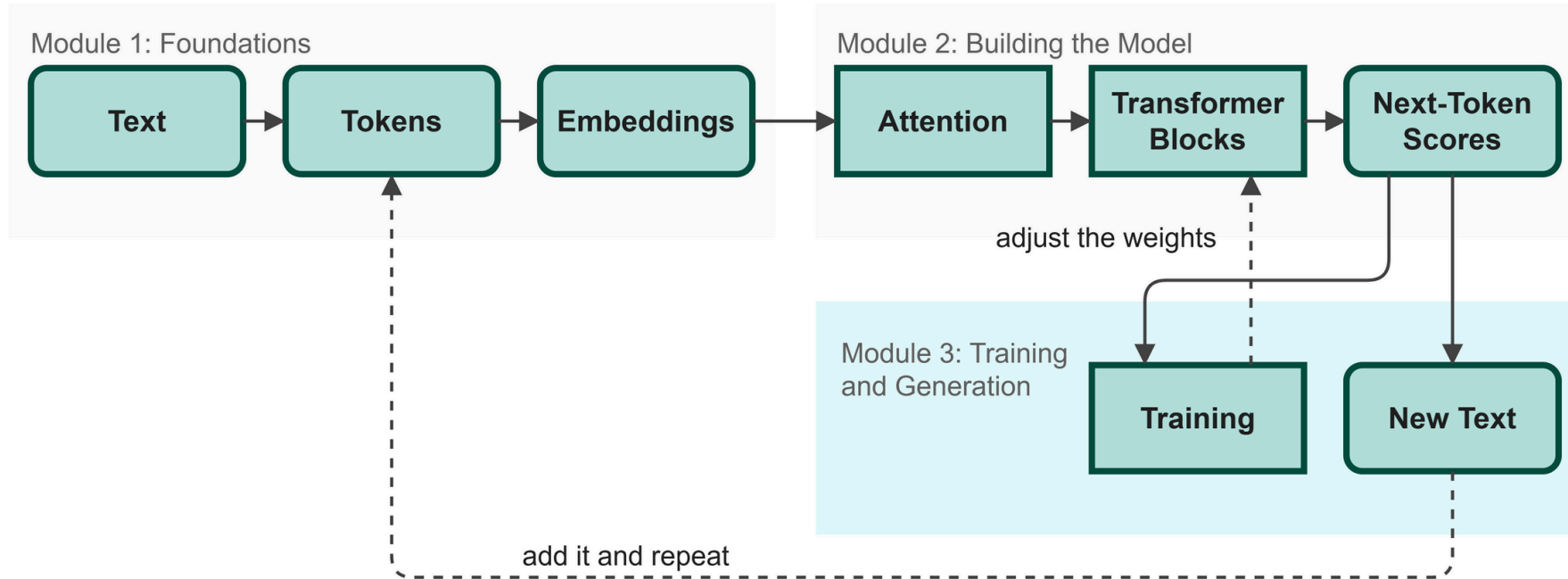


Figure 17.2: The complete system: from raw text to a trained language model.

- The same building blocks as **GPT-2**, the foundation of GPT-3, GPT-4 and many other modern LLMs
- Tokenize, embed, attend, stack blocks, train, generate: every stage is code you wrote

The design is the real one. What differs is scale.

What scale changes

	Your model	Production LLMs
Vocabulary	65 characters	50,000+ subword tokens
Size	128-dimensional embedding, 4 layers	thousands of dimensions, nearly a hundred layers
Data	1 MB of Shakespeare	terabytes of internet text
Hardware	a few minutes on a CPU	months on thousands of specialized GPUs

Every row is "more", not "different". The loop you built is the loop they run.

In business: the product lifecycle

For the house-style assistant, the end-to-end script is the full product lifecycle.

- Run it whenever the company archive changes
- Train a new model overnight on the updated documents
- Validate it, then deploy the new checkpoint to the marketing and compliance teams

One script, run on a schedule, is how a trained model stays current.

One script, seven stages

File: `src/ch16_full_pipeline.py` (source files are numbered one lower than chapters). It combines every piece from the previous chapters:

1. Download the data
2. Tokenize it
3. Create the DataLoaders
4. Build the **825,000-parameter** model
5. Train for **3,000** steps
6. Save a checkpoint
7. Generate a sample text

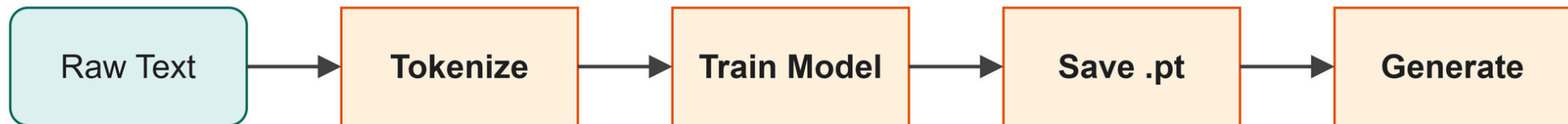


Figure 17.3: The full script executes every step in sequence without manual intervention.

Run it

```
$ python src/ch16_full_pipeline.py
[1/7] Downloading dataset...
...
[4/7] Building model...
...
[5/7] Training for 3000 steps...
...
step      1 | train: 4.2726 | val: 4.1660 | elapsed: 2s | ETA: 6148s
...
step  3000 | train: 1.7323 | val: 1.7411 | elapsed: 474s | ETA: 0s
...
[7/7] Generating text...
```

What just happened

- The model trained and the loss steadily decreased, from **4.2726** at step 1 to **1.7323** at step 3,000
- The seven stages ran without a hand touching the keyboard

What it wrote

ROMEO:

Wen a lame, and made thou have to the well.

Strenater:

My lord, there as I will strant we in thus face

His 'pperer come to the somen you; and seather

Figure 17.4: The model generates new text.

Read it closely

Generated with temperature 0.8 and `top_k=40` :

ROMEO:

Praviour soul to shall that that are the and not,

```
$ python diagrams/charts/ch17_generated_text.py
```

Saved chart: ch17-generated-text.png

- The character names, line structure and vocabulary mimic the training data
- The logic and some of the grammar do not

The shape of Shakespeare is there. The sense is not.

Shape check

Tensor	Shape	What it means
<code>model.parameters</code>	824,832	The total number of weights the model learned during training

The same count as Chapter 10. Training changed the values, not the shape.

Try it: treat the model like a deployed service

Open `src/examples/ch17_deploy.py`. It loads the final checkpoint (or falls back to `model.pt`) and answers a prompt.

```
if not os.path.exists(checkpoint_path):  
    # Fall back to model.pt if model_final.pt does not exist  
    checkpoint_path = os.path.join(  
        os.path.dirname(__file__), "..", "..", "checkpoints", "model.pt"  
    )
```

```
$ python src/examples/ch17_deploy.py  
--- Prompt: 'USER: What is the news?\n' ---  
USER: What is the news?  
  
FROKENTER:  
And with is there?
```

Ask it a question and it answers in the only shape it knows: a play.

Discussion

"Praviour soul to shall that that are the and not,"

In pairs, 3 minutes:

1. List what the model learned: which characters follow which, how long words run, where lines break, what a speaker's name looks like.
2. Chapter 13 measured it: after training, the model is wrong about the next character roughly **four times in five**. Reconcile that with the text on the screen.

Take two answers from the room.

Recap: today in 4 points

1. You built the complete GPT architecture from scratch, and ran it end to end.
2. Tokenization, embeddings, self-attention, transformer blocks and gradient descent are the core of all modern LLMs.
3. Larger models scale these same components with more data, more parameters and better hardware.
4. Refinements such as **instruction tuning** and **RLHF** are applied after the pretraining you did.

You can now read a polished paragraph from a chatbot and know what produced it.

Check your understanding

1. What is the difference between the model we built and GPT-2?
2. Why does the model output sometimes contain spelling or logic errors despite being fully trained?
3. What is the purpose of an end-to-end pipeline in a business setting?

Where to go from here

The architecture stays. Modern models add refinements on top of what you built:

- **Rotary positional embeddings** instead of simple position lookups, to understand relative distances between words
- **Grouped-query attention** instead of standard multi-head attention, to save memory and speed up generation
- **Instruction tuning** and **RLHF** after pretraining, so the model answers questions politely instead of only predicting the next word

Pretraining is what you did. The rest is built on it.

References

Book and figures

- Luu, T. (2026). *Building an LLM from Zero*, Chapter 17. <https://jackluu.io/book/>
- Figures 17.1 to 17.3 are the book's own diagrams (draw.io sources in the book repository, `diagrams/`); Figure 17.4 is rendered by `diagrams/charts/ch17_generated_text.py`.
- Terminal output: `src/ch16_full_pipeline.py` and `src/examples/ch17_deploy.py`.

Further reading (from the chapter)

- Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., ... Amodei, D. (2020). *Language models are few-shot learners* (arXiv:2005.14165). <https://doi.org/10.48550/arXiv.2005.14165>
- Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., ... Amodei, D. (2020). *Scaling laws for neural language models* (arXiv:2001.08361). <https://doi.org/10.48550/arXiv.2001.08361>
- Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C. L., Mishkin, P., ... Lowe, R. (2022). *Training language models to follow instructions with human feedback* (arXiv:2203.02155). <https://doi.org/10.48550/arXiv.2203.02155>

Dataset

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