

Building an LLM from Zero

Chapter 2: What Is an LLM?

Module 1: Foundations

Truong (Jack) Luu · jackluu.io/book

Where we are

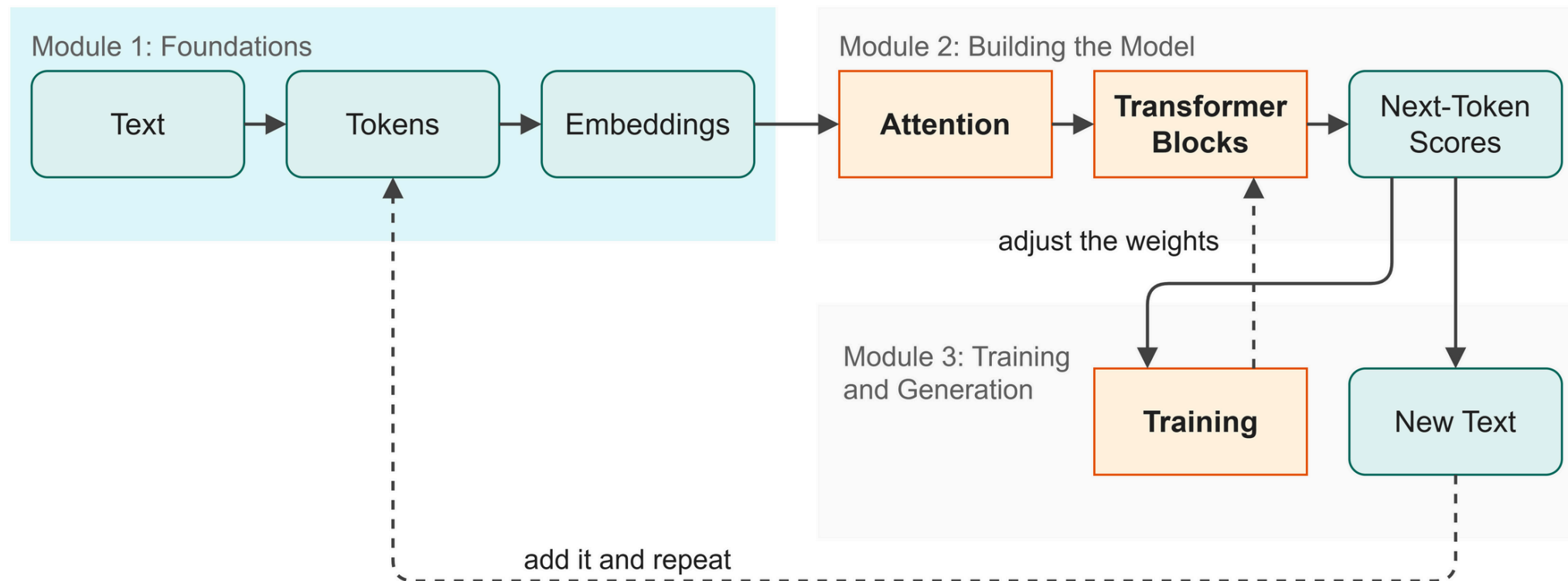


Figure 2.1: Before we zoom in, we look at what the whole system does.

Today is the one chapter with no code to build. We look at the whole chain first, so every later stage has a place on this map.

What this builds on (60-second recap)

- **Chapter 1** installed Python, Git and PyTorch (the math library we build with)
- It downloaded the **Tiny Shakespeare** text: 1,115,394 characters, 65 different ones
- The setup check printed **All checks passed!**

Next: a language model looks like magic. What does it actually do?

In this chapter you will

1. Learn the **single rule** that drives every language model
2. See how a model writes long answers **one step at a time**
3. Understand how reading data **replaces human teachers**

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

- **Token**: one small piece of text; in this book, a single character
- **Language model**: a system that predicts the next token in a sequence
- **Autoregressive**: using your own past outputs as inputs for the next step
- **Parameters**: the numbers inside the model that adjust during training

The next-token engine



Figure 2.2: The core task of a language model is guessing what comes next.

You type "I am so hungry, I could eat a" and your phone offers **horse**. Or **pizza**.

That is **next-word prediction**: given everything typed so far, guess the most likely next piece. A **Large Language Model** (LLM, a very big next-token predictor) does exactly this, and nothing else.

The whole idea in one line: given a sequence of tokens, predict what comes next. Everything else in this book builds a machine that does this well.

How it talks back

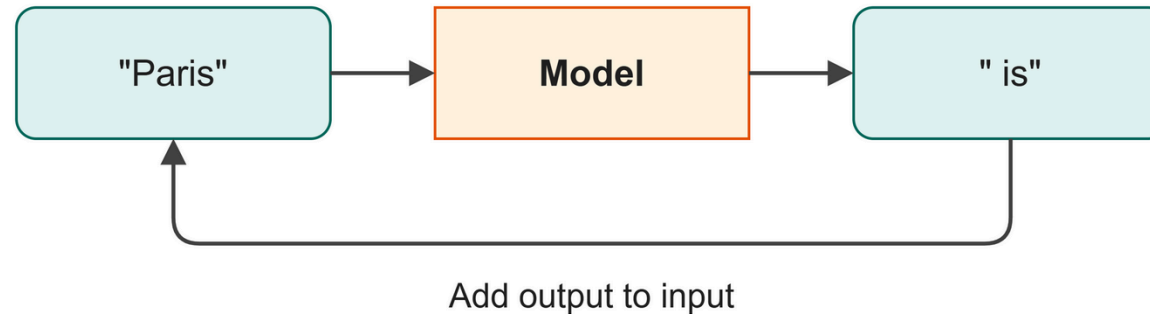


Figure 2.3: Each predicted output loops back to become the next input.

1. You type: What is the capital of France?
2. The model predicts the next token: Paris
3. That output is added to the input: What is the capital of France? Paris
4. It predicts the next token: is
5. It repeats until it decides to stop

Autoregressive (auto = self, regressive = feeding back): the model never plans the whole answer. It predicts one piece, appends it, predicts again.

What is a token?

A **token** is the smallest unit the model works with. It can be a whole word, a piece of a word, or a single character.

This book: characters

- Every letter, space and punctuation mark is one token
- The Shakespeare text has only **65** different characters
- You can see the whole vocabulary on one line

Real models: subwords

- Chunks such as `the`, `ing`, `Paris`
- Vocabularies of 50,000 or more
- Longer pieces, shorter sequences

Same math either way. We choose characters because the vocabulary is tiny and you can understand it instantly.

The text is its own answer key

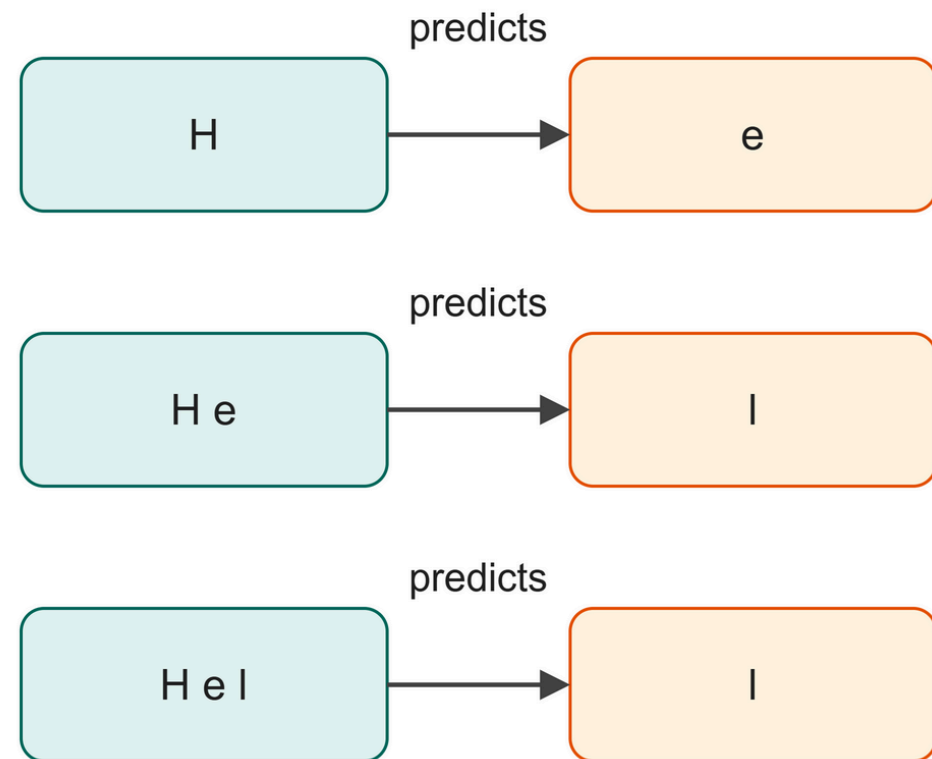
How does a model learn without a teacher grading its work?

Self-supervised learning (the data supplies its own answers): every position in the text is a practice question, and the next character is the answer.

Training text `Hello world` gives, for free:

- See `H`, predict `e`
- See `He`, predict `l`
- See `Hel`, predict `l`

No human labels are needed, so you can train on millions of pages of raw text.



What makes it "large"

Parameters (the numbers inside the model that adjust during training) work like dials. Training turns the dials until the output improves.

Model	Parameters	Trained on
The one you will build	about 825,000	1 MB of Shakespeare, on a laptop CPU
Production LLMs	billions	terabytes of text, on thousands of GPUs

Bigger means more dials and more text, not a different idea. The loop is the same.

The numbers game: is it really just prediction?

If you see the letters `the` , what comes next in Shakespeare? Count every character that follows it in the training text:

```
$ python src/examples/ch02_next_char.py
What follows 'the ' in Shakespeare?
's': 573 times
'w': 459 times
'c': 448 times
'p': 426 times
'm': 376 times
```

This is exactly what the model learns to do. It does not count by hand; it uses math to estimate these probabilities from the text.

In business: a house-style writing assistant

A company wants an assistant that drafts emails in its own voice.

- The model trains on the company's **archive of past writing**
- It learns what typically follows what **in that archive**: terms, tone, sign-offs
- Nobody writes rules for the house style; the archive is the answer key

The concept is identical to the Shakespeare model. Only the text changes.

Watch out: "it understands"

It is easy to say the model **understands** the text.

It does not. It is a math engine calculating the most probable next token from patterns in its training data.

- A confident, fluent answer can still be wrong
- The loop never asks whether an answer is **true**, only whether it **fits the pattern**

Fluent is not the same as correct. You will see your own model write Shakespeare that no one ever wrote.

Discussion

Is it really just predicting the next word?

In pairs, 3 minutes:

1. Name one thing a chatbot did for you this week that felt like understanding.
2. Explain it as next-token prediction. Where does the explanation feel thin?

Take two answers from the room.

Recap: today in 4 points

1. A **language model** predicts the next token. That is the whole task.
2. **Autoregressive generation** feeds each prediction back as input, one piece at a time.
3. **Tokens** are the pieces: characters here, subwords in production models.
4. **Self-supervised learning** uses the raw text as its own answer key, so no labels are needed.

Next, we zoom in on the first stage of the map: the containers that hold the numbers.

Check your understanding

1. What does a language model actually predict?
2. Why is it called "autoregressive"?
3. How does self-supervised learning differ from having a teacher grade the work?

Bonus: the count for **the** put **s** first. Give two Shakespeare words that would produce that.

Next up: Chapter 3, Tensors and PyTorch

- The **tensor** (a multi-dimensional array of numbers) is the container every stage of the map passes along
- You will run your first PyTorch script and read its shapes

Before next class: finish the Chapter 1 setup and run `python src/ch00_setup_check.py` until it prints **All checks passed!**

References

Book and figures

- Luu, T. (2026). *Building an LLM from Zero*, Chapter 2. <https://jackluu.io/book/>
- Figures 2.1 to 2.4 are the book's own diagrams (draw.io sources in the book repository, `diagrams/`).
- Terminal output: `src/examples/ch02_next_char.py` , run on the Tiny Shakespeare text.

Further reading (from the chapter)

- Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). *Language models are unsupervised multitask learners*. OpenAI technical report.
- Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8), 1735–1780.

Dataset

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