

Building an LLM from Zero

Chapter 12: Dataset and DataLoader

Module 3: Training and Generation

Truong (Jack) Luu · jackluu.io/book

Where we are

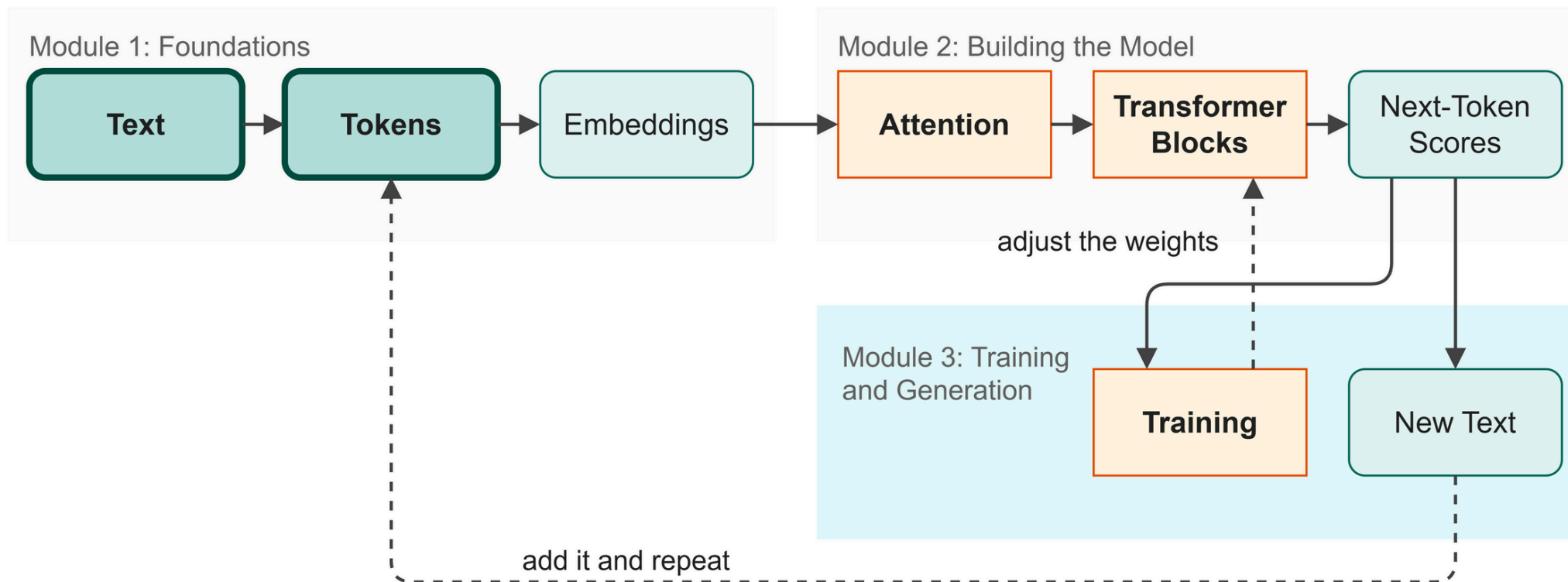


Figure 12.1: We begin the training module by preparing our data pipeline.

Module 3 starts here. The engine is built and has a goal; today we build the pipeline that feeds it thousands of examples at once.

What this builds on (60-second recap)

- **Chapter 11** gave the model its goal: predict the next token from the past only
- The **target** is the input shifted by one, so one sequence of length T gives T examples
- **Cross-entropy loss** measures surprise; blind guessing among 65 characters scores about **4.17**

Next: where do the sequences come from, and why feed 32 of them at a time?

In this chapter you will

1. Use a **sliding window** to generate training examples
2. Understand how **batches** process multiple sequences in parallel
3. Build a PyTorch **Dataset** and **DataLoader**

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

- **Block size**: the maximum number of tokens the model can look at at one time (its context window)
- **Batching**: grouping multiple training examples and processing them at the same time
- **Dataset**: a PyTorch class that defines how to retrieve a single training example
- **DataLoader**: a PyTorch utility that groups examples into batches and shuffles them

The sliding window

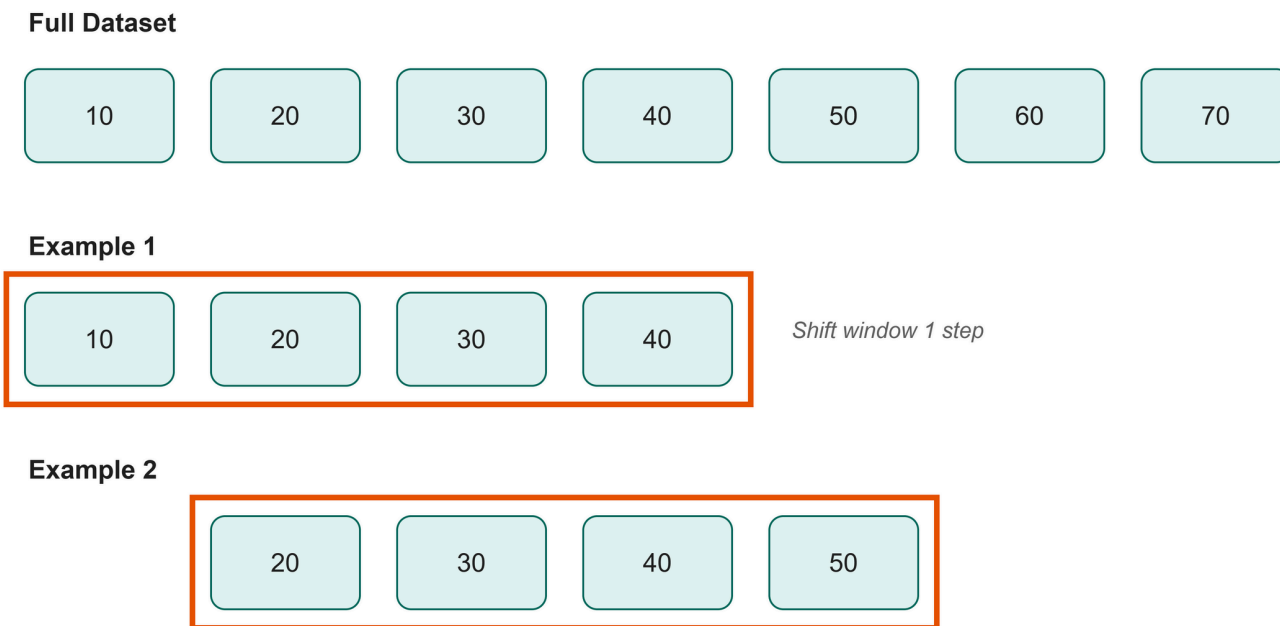


Figure 12.2: A sliding window creates multiple, overlapping examples from one long text.

A window sees only `block_size` characters at a time (128 in this book). Grab the first sequence, slide **one character** right, grab the next, repeat to the end.

One long text becomes about a million overlapping examples, without copying the text a million times.

Batches: time it, do not guess it

Feeding sequences one at a time would crawl. Computers are fast at doing the same math on many pieces of data at once. How much does it matter?

```
$ python src/examples/ch12_batch_speed.py
32 sequences, one at a time : 2.191 s
the same 32 as one batch    : 0.245 s
batching is 9.0x faster on this machine
```

- Nine times faster, on an ordinary laptop with no GPU
- One sequence at a time, the Chapter 13 training run would take roughly nine times as long
- On a GPU, where thousands of arithmetic units wait for work, the gap is wider still

The same 32 sequences, the same math, nine times faster. That is batching.

Batches: what the model sees

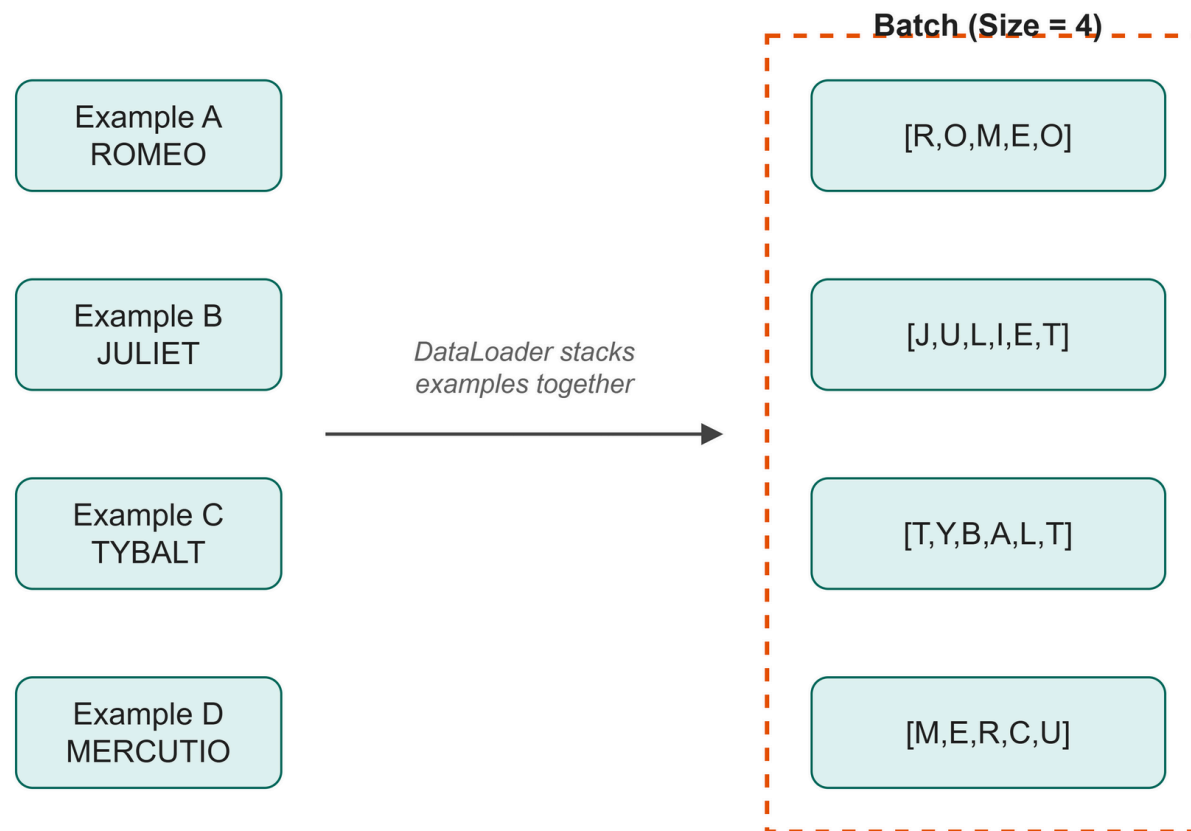


Figure 12.3: A batch stacks multiple independent examples into a single block.

Several unrelated pieces of text, stacked into one block of numbers.

A batch of 32

A **batch** stacks several independent sequences into one block.

- With a batch size of **32**, the model gets 32 sequences in one go
- It processes them in parallel
- It calculates the loss for all 32 and **averages** it

A batch is 32 unrelated pieces of Shakespeare, handled as one block of numbers.

Why 32, and not 1 or 1,000

Speed is half the reason. The other half is the **quality of each step** the model takes.

Batch size	What happens to each step
1	The direction comes from one stretch of text: a stage direction, a run of dialogue, a line of spaces. The model lurches after each one.
32	Averaging over 32 independent sequences cancels most of that noise. Each step points closer to the truth for the whole text.
1,000	The whole batch must fit in memory, the first limit on a laptop. And the returns fade: only slightly truer, thirty times the cost.

32 is not sacred; other books use 16 or 64. It is large enough to steady the direction and small enough to fit.

How the code flows

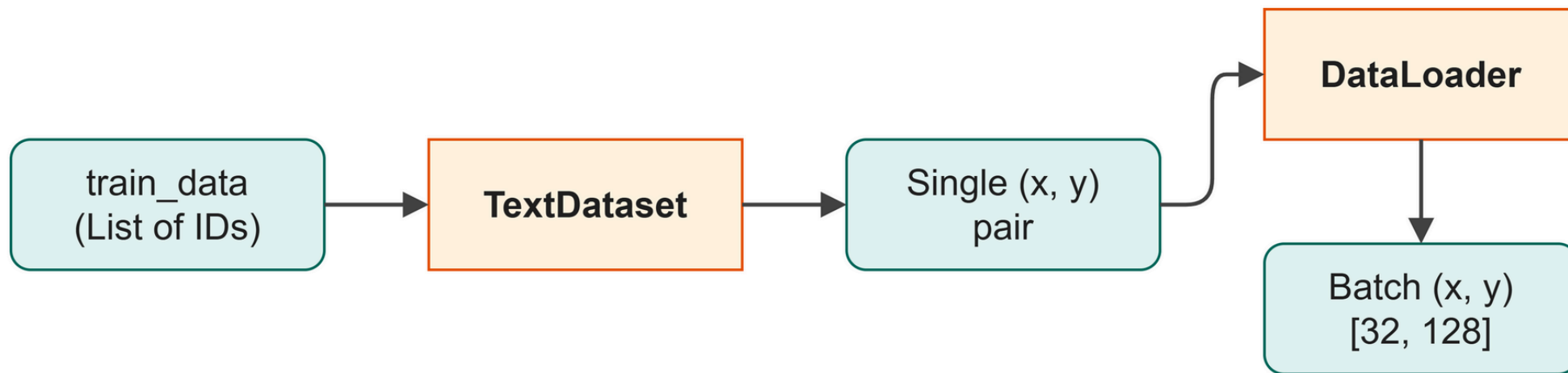


Figure 12.4: The Dataset handles the sliding window, and the DataLoader stacks the examples into a batch.

Two PyTorch tools, one job each: the Dataset cuts examples, the DataLoader stacks them.

Code: Dataset and DataLoader

File: `src/ch11_data_loader.py` (source files are numbered one lower than chapters)

```
def __len__(self):
    # We need block_size + 1 tokens to form one (input, target) pair
    return len(self.data) - self.block_size

def __getitem__(self, idx):
    # Input is a block of text
    x = self.data[idx : idx + self.block_size]
    # Target is the same text, shifted one character to the right
    y = self.data[idx + 1 : idx + self.block_size + 1]
    return x, y
```

- Line 3 counts the starting positions: the text length minus one window
- Lines 7 and 9 are the sliding window and the shifted target, one slice each

Run it

```
$ python src/ch11_data_loader.py
...
Dataset sizes:
  Train examples: 1,003,726
  Val   examples: 111,412

DataLoader config:
  Batch size   : 32
  Train batches per epoch: 31,367
...
x_batch shape: torch.Size([32, 128]) (batch_size, block_size)
y_batch shape: torch.Size([32, 128]) (batch_size, block_size)
```

What just happened

- The 1,003,854 training tokens gave **1,003,726** starting positions (the length minus one window of 128)
- The **DataLoader** cut them into batches of 32: **31,367** batches per pass over the text (an epoch)
- One batch is 32 rows of 128 input characters, and 32 rows of 128 targets

One batch up close

The script prints the first example of the batch it drew:

First example in batch:

```
x (input)  : 'ness! serious vanity!\nMis-shapen chaos of well-seeming for...  
y (target) : 'ess! serious vanity!\nMis-shapen chaos of well-seeming form...  
(y is x shifted by 1 character)
```

- `x` starts with `ness!`; `y` starts with `ess!`
- Every character of `y` is the character that follows it in `x`

Chapter 11's shifted pair, now cut from real Shakespeare by the sliding window.

Shape check

Variable	Shape	Meaning
x_batch	[32, 128]	32 sequences, each 128 input characters
y_batch	[32, 128]	32 sequences, each 128 target characters

Batch size by block size. The model's `[B, T]` input from Chapter 10 now has real numbers: 32 by 128.

Try it: the window on seven numbers

Run `src/examples/ch12_batching_demo.py` on a tiny dataset:

```
$ python src/examples/ch12_batching_demo.py
Data: [10, 20, 30, 40, 50, 60, 70]
Block size: 3

Example 1:
  Input  : [10, 20, 30]
  Target : [20, 30, 40]
Example 2:
  Input  : [20, 30, 40]
  Target : [30, 40, 50]
...
```

Seven numbers and a window of 3 give 4 examples. Then change `block_size` to 4 and run again: the chapter says the number of examples decreases.

Fewer examples with a wider window: the count is always the length minus the block size.

In business: reading 32 emails at once

The house-style assistant learns from an archive of 100,000 documents.

- You would not train it by showing it one word at a time
- Batching is like reviewing **32 different emails at the same time**, learning from all of them at once
- It is the key to training efficiently at scale

Same idea as the Shakespeare batch: many independent examples, one pass.

Watch out: the last window

Be careful with `__len__`. With 100 characters and a `block_size` of 10, you get **90** starting positions, not 100.

- One example needs 11 characters: 10 for the input and 1 more for the target
- Start any later and the target runs past the end of the text

```
return len(self.data) - self.block_size
```

The subtraction is not a detail. Without it, the last examples would read past the end of the data.

Discussion

The DataLoader shuffles the training examples.

In pairs, 3 minutes:

1. What could go wrong if the model read the text in order, from the first line to the last?
2. Does shuffling change *what* the model learns, or only *how* it gets there?

Take two answers from the room.

Recap: today in 4 points

1. A **sliding window** extracts overlapping sequences from one continuous text.
2. **Batching** processes many independent sequences in parallel: 9 times faster on the author's laptop, and each step is steadier.
3. PyTorch's **Dataset** defines how to grab one example: the window and its shifted target.
4. PyTorch's **DataLoader** groups examples into batches of 32 and shuffles them.

The data is batched and ready. Next, the loop that turns batches into a trained model.

Check your understanding

1. If you have a sequence of 1,000 tokens and a block size of 100, how many examples can a sliding window extract?
2. Why is batching important for training speed?
3. Why do we shuffle the training data?

Bonus: the run reported 31,367 batches per epoch. Show where that number comes from.

Next up: Chapter 13, The Training Loop

- Everything built so far comes together in a loop of four steps: **forward, loss, backward, step**
- You will watch the loss fall from blind guessing to real prediction, on your own laptop

Before next class: make sure `python src/ch11_data_loader.py` runs on your machine and prints the batch shapes. Chapter 13 starts a training run at the beginning of class.

References

Book and figures

- Luu, T. (2026). *Building an LLM from Zero*, Chapter 12. <https://jackluu.io/book/>
- Figures 12.1 to 12.4 are the book's own diagrams (draw.io sources in the book repository, `diagrams/`).
- Terminal output: `src/ch11_data_loader.py` , `src/examples/ch12_batch_speed.py` and `src/examples/ch12_batching_demo.py` .

Further reading (from the chapter)

- Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., & Amodei, D. (2020). *Scaling laws for neural language models* (arXiv:2001.08361).

Dataset

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