Consuming Textbooks

Jarrod Connolly

May 28, 2026

Abstract

I never finished technical textbooks by reading them cover to cover. This article describes a study method that uses an LLM as a chapter-by-chapter partner, with room for side quests and real code along the way. The point is not to outsource understanding. It is to keep the doing in the loop so the material sticks.

Introduction

For years I could not finish a technical textbook cover to cover. Same story every time. I would buy the book, read a few chapters, hit a dense proof or a long exercise block, and quietly shelve it. Video courses and lessons were never really my thing. I would watch, nod along, and realize a week later that nothing had stuck. Passive reading felt virtuous, but I could not tell you much then either. Historically I learned by reading and doing. I always have to build something.

That changed when I stopped treating a textbook like a novel and started treating it like a project. I still read from the PDF. I still do the work. Now I talk through each section, code the examples in the same sitting, and take side quests when something will not let me go. It feels less like getting to the last page and more like building something real, one chapter at a time.

I wrote about the sci-fi roots of this mindset in Style Over Substance, and about building HAL as a RAG system over technical books. This article is the simpler, hands-on version. No custom server, no vector database, just a repeatable workflow that has helped me finish books faster and remember more of what I read.

Why Textbooks, Why Now

Everyone learns differently. Plenty of people swear by video courses, and that is fair. They have never been my path. Textbooks are where I have always gone for depth. That means edge cases, exercises, and the notation you need when you actually implement something. I still buy physical and digital books on purpose. I want a single curated path through a subject, not an algorithmic feed of hot takes.

The problem was never the books. It was how I consumed them. For a long time my choices were a textbook or a course, video or live. I always got more out of books than lectures. I got even more when I was actually building along with the material. The gap was that reading and building rarely happened in the same sitting. I would read a chapter, tell myself I would code later, and the thread would go cold.

That combination is the gem in this workflow. Not books instead of courses. Not reading today and building someday. Reading and building in the same loop, chapter by chapter.

The rules that hold it together sound contradictory until you live them. **Slow and steady** on the chapter you are in. **Side quests, always**, when something in the margin pulls you. The detours are not a failure of discipline. They are part of the deal.

That structure turned out to be a git repo, a folder of chapter Markdown, and those two rules written into how I work.

Setting Up the Book Project

When I commit to a subject, I start with a textbook I actually want to finish. Not the cheapest option. Not the one with the most stars. The one whose table of contents matches the gaps in my head.

From there the setup is mechanical:

1. **Get the book in PDF form** (purchase, legitimately).
2. **Convert it to Markdown and split by chapter.** I chunk the output so each chapter is its own file. That keeps context windows sane and mirrors how I already think about the material.
3. **Create a new project in Cursor** and initialize git. This repo is the learning space for the whole book, reading, coding, and building.
4. **Add a `book/` folder** and drop the chapter Markdown files inside (`ch01.md`, `ch02.md`, or whatever naming keeps you oriented).
5. **Keep the PDF open while you work.** I read the PDF, figures, equations, and all. The LLM follows along through the chapter Markdown files in `book/`.

I do not try to ingest the entire book into one giant chat. I open the chapter file, skim the headings, and start a conversation scoped to that chapter. When the chapter is done, I commit. The git history becomes a log of progress, which sounds small until you are on chapter nine and need proof you are not stuck on chapter three in your head.

The Chapter Loop

Every chapter follows the same loop. Slow and steady on the main path. Side quests whenever the book sparks a question you cannot let go. Both, every time.

I read the PDF section by section. As I go, I talk with the model about what I just read. We discuss each topic, dig deeper when something feels fuzzy, and stay with a section until it clicks. How deep I go is always case by case. Some paragraphs need one pass. Others need twenty minutes of back-and-forth.

When the book provides code, I stop chatting and start typing.

Code Along in `ch01`, `ch02`, `ch[n]`

For each chapter that includes listings, I create a matching folder (`ch01/`, `ch02/`, …). I retype or adapt the book's examples, run them, and break them on purpose when I am not sure I believe the explanation. "What happens if I change this hyperparameter?" is a better question than "looks fine."

This is where retention actually shows up. Reading about a tokenizer is one thing. Implementing a naive byte-pair encoding loop and watching vocabulary size creep up is another. The LLM is useful here as a pair programmer who has infinite patience for "why did this tensor shape explode?"

I do not paste entire chapters into prompts. I pull in the section I am working on, plus the file I am editing. Smaller context, focused answers.

Side Quests (On Purpose)

I named the rules earlier for a reason. The main path is the textbook. Side quests are the reason I stay interested.

When a paragraph mentions something I half know (Rotary embeddings, a paper name, a library I have never installed), I name the detour out loud as a **side quest**. I might:

- Pull a recent paper from arXiv and read the abstract and introduction.
- Skim blog posts, GitHub repos, or social posts from people who implemented the thing.
- Build just enough code to satisfy the curiosity, then stop.

The rule is to return. Side quests are not a new book. They are depth charges on the current chapter. I implement enough to feel the idea in my fingers, commit if it is worth keeping, and go back to the chapter file.

I borrowed the phrase “conversation side quests” from the Librarian in Stephenson’s *Snow Crash*, which I wrote about in the vibe-coding article. Textbook side quests use the same muscle. Fork, explore, merge back to the main thread.

Case Study: Build a Large Language Model (From Scratch)

The most recent book I ran through this way was Sebastian Raschka’s *Build a Large Language Model (From Scratch)*. It was a good stress test. Almost every chapter touched territory that was new to me.

I took a lot of side quests. Tokenization alone pulled me into long implementations from scratch (I have more to write about that in upcoming articles). Fine-tuning thinking led me to Unsloth and experiments with fine-tuning a small local model so it knew more about coding for a specific microcontroller. None of that was in the table of contents. All of it made the core chapters land harder when I came back.

What worked was not skipping the book’s exercises in favor of the shiny detour. It was letting the detour answer a question the book raised, then closing the loop on the chapter’s own code.

What This Is Not

This workflow is a heuristic, not a product. It assumes you already bought the book, you respect copyright, and you are using the LLM as a tutor, not as a shortcut to skip paying authors or doing exercises.

It also will not fit everyone. Some people learn better in a classroom. Some learn best from video. I am not claiming this replaces either. I am saying that for someone who already learns by reading and building, pairing a textbook with a patient conversational partner and a git repo changed my completion rate and my recall.

Conclusion

I used to collect textbooks like backlog items. Now I consume them like projects, with chapter files, PDF in one hand, Cursor in the other, code in `ch01/` through `chN/`, and side quests when curiosity spikes. Reading and building finally happen in one technique, which is more immersive than either alone ever was for me. I talk to the book, argue with it, and ship small commits that prove I was there.

If this resonates, start small. One book. One chapter. One repo. Slow and steady on the path. Side quests when something bites.

Coming up next on my side of the blog. The tokenizer rabbit holes that fell out of the Raschka book, with real code and the mistakes that taught me what “vocabulary” actually costs at runtime.