Episode Book

EPISODE #6542

Five Projects to Build a Language Model From Scratch

PUBLISHED	13 November 2026 at 11:43 UTC
DURATION	27:49
LISTEN	myweirdprompts.com/episode/build-language-model-from-scratch-30855/
HOSTS	Herman, Corn, Orson, Hilbert
SCRIPT MODEL	z-ai/glm-5.3
VOICES	chatterbox-regular (chatterbox-local)
PIPELINE	V5.26
SOURCES	24 cited by the research (listed at the end)
BOOK GENERATED	14 November 2026 at 06:46 UTC

SYNOPSIS

Training a language model from scratch used to require massive budgets, but a thousand-fold drop in compute costs has changed the math. A de facto five-step ladder—centered on Andrej Karpathy's micrograd, Zero to Hero lectures, minbpe tokenizer, microgpt, and nanochat—takes learners from a 100-line autograd engine to a working GPT. Along the way, building a byte-pair tokenizer explains why models miscount letters in words like strawberry, and a 200-line pure Python script reveals that every AI chat is just document completion.

TRANSCRIPT



Herman

Feynman's chalkboard, when he died in eighty-eight, said "What I cannot create, I do not understand." Note the verb. Create, not build. Building is the carpenter's version. Creating is the one where you know why every joint is where it is.



Corn

And Daniel's prompt this week is basically the practical exam for that sentence. He says the best way to really get to know something is to build it from scratch, but with AI models that's hard, because a language model at real scale is a huge undertaking. So he wants five hands-on projects. A small language model, maybe a BERT, whatever, credible, viable, things people have actually done as learning exercises or that the community actually recommends.



Herman

And the honest thing to say up front: there is no published curriculum called "the five projects." Nobody sat down and issued one. But a ladder has emerged anyway, mostly around Andrej Karpathy's body of work and Sebastian Raschka's book, and enough people have independently converged on the same rungs that it's fair to present it as a de facto path.



Corn

My favorite part of the framing is that he wrote "huge undertaking," and he's right, and the right answer is that it stopped being one about a year ago. The whole premise was cost, and the cost collapsed.



Herman

Which is the spine of this. Shall we climb?



Corn

Climb.



Herman

First rung, and it's not a language model. It's the autograd engine. Karpathy's micrograd, from twenty-twenty. Two pieces: about a hundred lines for the automatic differentiation engine, about fifty for a neural net library on top of it, with a PyTorch-like API. You write backpropagation from scratch over a dynamically built graph, chopping a neuron into individual scalar adds and multiplies until the chain rule is something you have physically typed out rather than something you invoke.



Corn

I want to linger on that for a second. Backprop is one of those things people think they understand because they can name it.



Herman

And they usually can't derive it. That's the gap this closes. The demo notebook trains a tiny two-layer classifier on the moon dataset, two sixteen-node hidden layers, with a max-margin loss and plain SGD. Nothing glamorous. But you have now written the thing that computes gradients. Every trained model you have ever used, this is the mechanism under it.



Corn

Can you say what the moment of understanding actually feels like? Because I think people imagine it arrives as math.



Herman

It arrives as bookkeeping. You build these little Value objects, each one remembering which other Values produced it, and then you call backward on the final loss and watch the gradients flow back through the graph you made, one multiply at a time. And the realization isn't "ah, calculus." It's "ah, it's just a to-do list running in reverse order." People who have used PyTorch's loss.backward() for years without a second thought suddenly see that the dot on the function call is doing something they could have written themselves on paper.



Corn

The lecture that goes with it.



Herman

Two hours, twenty-five minutes. Karpathy's own claim, and I see no reason to doubt him, is that it's the most step-by-step explanation of backprop and training anywhere. Assumes Python and a vague memory of high school calculus. That's the entry fee for the whole ladder, by the way. Not a math degree.



Corn

So the first project isn't a language model at all. It's the machinery that makes training possible. What's the second rung?



Herman

The character-level language model, then a real GPT. This is the Zero to Hero sequence, and I want to give the runtimes because the runtimes are the argument. Bigram model, one hour fifty-seven. Multilayer perceptron, one fifteen, and that one sneaks in the actual vocabulary of a practitioner: train/dev/test splits, overfitting, underfitting, learning-rate tuning. Then the one I'd make mandatory, the manual backprop lecture, one fifty-five, where you backprop by hand through cross-entropy, tanh, batchnorm, the embedding table, without calling the built-in backward. Then a deeper WaveNet-style net, fifty-six minutes. And then the payoff, "Let's build GPT: from scratch, in code, spelled out," one fifty-six, following the Attention is All You Need paper and the GPT-2 and GPT-3 write-ups.



Corn

The runtimes are the argument how?



Herman

Because you can hold the whole thing in your head. Ten-odd hours of lectures gets you from "I know Python" to "I have personally typed a transformer." There's no other system in software I can think of where the distance between user and implementer is that short. You don't get this with an operating system. You don't get it with a browser engine.



Corn

Give me the browser engine comparison concretely, because I think it undersells how strange this is.



Herman

Fine. Millions of people use a browser every day. How many could build one, even a toy one, even badly, in ten hours of lectures? A handful of rendering engines exist on earth, each with hundreds of engineer-years in it. Now, to be fair, the same is true at the frontier of language models. GPT-4-class training runs are browser-engine-scale efforts. But the trick is that the conceptual core, the transformer, is small enough to type out in an afternoon, and it's the same core. A person who builds the toy version understands the shape of the big one in a way a Chrome user never understands Chromium. The abstraction doesn't hide the essence the way a browser's source does.



Corn

The lecture has a joke I enjoy, which is that you watch GitHub Copilot, itself a GPT, help write the GPT you're building. Recursive pedagogy.



Herman

And there's a subtler joke under it, which is that the lecture is itself a model of the thing it teaches. Karpathy predicting, token by token, what you're about to type next, while explaining how prediction works. Some people find that dizzying. I find it clarifying. It means the demo never leaves the subject.



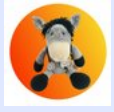
Corn

Third rung. And this is the one people skip, and it's the one I'd fine people for skipping. The tokenizer. Karpathy's minbpe. Two hours, thirteen minutes, building the GPT-series tokenizer from scratch, byte pair encoding, the whole preprocessing stage.



Corn

Wait, sorry. That's your line.



Herman

That's my line, but you said it with conviction, so let's keep it. Third rung. The tokenizer. Karpathy's minbpe. Two hours, thirteen minutes, building the GPT-series tokenizer from scratch, byte pair encoding, the whole preprocessing stage.



Corn

Why do you call it the one people skip?



Herman

Because it's not a model. It's plumbing. It feels like it's beneath the ladder. And Karpathy's own framing is that tokenization is a completely separate stage of the LLM pipeline, with its own training sets and algorithms, and a lot of weird model behaviors trace straight back to it. Counting errors, arithmetic failures on certain numbers, the "strawberry" letter-counting thing. All tokenizer artifacts. He's said, only half joking, that somebody out there should ideally find a way to delete the stage entirely.



Corn

Walk me through the strawberry thing, because it's the one everyone has heard and almost nobody can explain.



Herman

The model doesn't see letters. It sees token IDs. And how "strawberry" gets chopped depends entirely on what the BPE merge rules decided during tokenizer training. If "straw" and "berry" ended up as two separate common tokens, then as far as the model is concerned, the word has two parts, not ten letters. Asking it to count the R's is asking it to reason about a representation it has never directly perceived. It's like asking you to count the phonemes in a word you've only ever seen written down. You can do it, but you have to reconstruct an intermediate layer that your normal reading just doesn't use. When you build the tokenizer yourself and print the actual token splits, that mystery dies on the spot. It stops being a spooky model failure and becomes a completely legible engineering consequence.



Corn

So it's the rung that explains a whole category of model weirdness.



Herman

Yes. And it's a properly social exercise, which is unusual for this ladder. There's an exercise file in the repo, and there's a long tail of small projects from people who did it. Small reimplementations of minbpe, a GPT-4-style tokenizer following the exercise instructions, an in-browser BPE demo. One commercial platform, Crackr, sells "Build a



Herman (continued)

Tokenizer from Scratch" as its first product, thirteen tasks, seven hours forty-three minutes, pitched under the Feynman banner.



Corn

With the slogan "No AI slop. We write this stuff."



Herman

That's the one. And that framing is doing something interesting, because what they're implying is that in the assistant era, the code is the free part. You can get a working tokenizer, or a working GPT, generated in seconds. What you can't generate is the hour you spent writing it.



Corn

So the first three rungs are the components. Engine, model, tokenizer.



Herman

And now you assemble them. And this is where the ladder splits into two paths, and I don't think either one is optional. The minimal path is microgpt, February this year, Karpathy's self-declared culmination of a decade-long obsession. One file, two hundred lines of pure Python, no dependencies, and it does everything. Dataset, tokenizer, autograd, GPT-2-like architecture, the Adam optimizer, training loop, inference loop. His line: "Everything else is just efficiency. I cannot simplify this any further."



Corn

No dependencies at all.



Herman

No PyTorch. No NumPy, I believe. The tensors are done by hand. It trains on thirty-two thousand names with a twenty-seven token vocabulary and generates plausible fake names. Kamon, karai, jaire, vialan, konna. You feed it a name list, it makes up new names



Herman (continued)

that look like names. The full algorithmic content of GPT in something you can read on a lunch break.



Corn

I did read it on a lunch break. Slowly.



Herman

It's an entire philosophical statement compressed into a repo. If you want to say "it's all matrix multiplications stacked up," this is the file that makes that sentence literal.



Corn

There's a line in his writeup that I think is the most quietly important sentence in the whole ecosystem. He says that from the perspective of a model like ChatGPT, your conversation with it is just a funny looking document, and the response is a statistical document completion. Which reframes the entire product. Every chat interface you've ever used is a document the model is finishing.



Herman

And once you see it that way you can't unsee it. The system prompt is the document's header. Your question is the body. The little "thinking" animation is the model continuing the document in a direction shaped to look like an answer. People spend years confused about whether the model "knows" it's in a chat, and the answer is in microgpt, in two hundred lines: there is no chat. There was never a chat. There's a document that happens to contain the tokens "user" and "assistant" as punctuation.



Corn

And that's the understanding microgpt buys you. It's not a skill, it's a worldview. But the complete path, the other branch, is nanochat, and it teaches the opposite thing.



Corn

Nanochat is the whole pipeline.



Herman

The whole pipeline. Tokenization, pretraining, fine-tuning, evaluation, inference, a ChatGPT-style chat at the end, one repository, designed for a single GPU node. And the pricing is the headline. GPT-2 cost around forty-three thousand dollars to train in twenty nineteen. Nanochat reproduces GPT-2 capability for forty-eight dollars, roughly two hours on an eight-GPU node, or about fifteen dollars on spot pricing. That's a thousand-fold drop in seven years. And the speedrun, start to finish, takes about an hour and a half and drops you into a command line chat with the model you just trained.



Corn

Let me make sure the scale of that drop lands, because "thousand-fold" can slide past people. Forty-three thousand dollars in twenty nineteen was a research lab line item. It needed a budget approval and a person whose job included procuring GPUs. Forty-eight dollars is a nice dinner. The thing that used to be an institutional decision is now an impulse purchase.



Herman

And the hour-and-a-half runtime matters as much as the dollars. A seven-year cost collapse is one thing, but a training run that fits inside a single evening is another. You can fail and start again. Failure used to be the expensive part of experimentation. When the whole run costs fifteen dollars on spot, failure is free, and free failure is what makes something a learning exercise rather than a project.



Corn

A model you just trained, that is, in Karpathy's words, a bit like talking to a kindergartener.



Herman

He puts a smiley on it, but yes. It's a small model. That's fine. That's the point. The thing I love is the interface: one dial, called depth, the number of layers. You set it to twenty-six, you get GPT-2-class capability. Every other hyperparameter, the width, the head count, the learning rate schedule, the training horizon, all calculated automatically. It's the ladder's whole philosophy in a single command line flag. You don't tune the ritual, you tune the



Herman (continued)

size.



Corn

Before nanochat, there was nanoGPT.



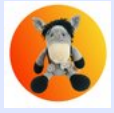
Herman

The famous middle sibling. About three hundred lines each for the training script and the model. The quick start trained a character-level model on Shakespeare in about three minutes on an A100, final loss 1.4697, or the same run on a plain CPU in a few minutes at 1.88. And there's a companion repo, build-nanoGPT, which is the reproduction of GPT-2 124M from an empty file, with the git history deliberately kept step by step so you can walk the commits. That one's about ten dollars and about an hour now. But the repo has a deprecation notice from last November saying go use nanochat instead.



Corn

Which is a pattern I find funny. He keeps deleting his own work.



Herman

Constantly.



Corn

nanoGPT deprecated for nanochat. microgpt superseding the pedagogy of everything before it. Every few years he looks at his own ladder and prunes it.



Herman

And it's worth noticing what that pruning implies about the ladder itself. It's not a monument. It's a living curriculum that gets revised when something better exists, which means if you did the older projects you didn't waste your time, you just walked a path that has since been repaved. The understanding transfers. The repos are scaffolding, and scaffolding is supposed to come down.



Corn

And the result is two living projects that teach opposite things. Nanochat teaches pipeline engineering. Real stages, real scale, a chatbot at the end. Microgpt teaches algorithmic essence. Two hundred lines, no libraries, the math visible. If I had to pick I'd say do microgpt first, when you still need to see the bones, and nanochat second, when you're ready to see the organs.



Corn

I don't disagree, but I want to say what the risk is. The nanochat run is smooth enough that you can finish it and still not understand anything. All the stages are orchestrated for you. It's a very well paved road and a well paved road teaches you nothing about terrain.



Herman

That's fair, and the honest version is that the two projects are for different moments. Before nanochat you should have already done the manual backprop lecture, the tokenizer, at least a small GPT. Nanochat is the capstone, not the entry.



Corn

So, four rungs down. And we go now to our correspondent, who is standing by live on the subject of building language models from scratch. Orson, what are you seeing?

o

Orson

Good evening. I stand before you in the shadow of a great and terrible question, and I must report, with heavy heart, that this is a tragedy in five acts, and the fifth act is a language model trained from nothing upon a single graphics card, which is to say, a small and humble miracle, and yet, and yet.



Corn

Are you alright up there?

o

Orson

The gradient descends, the tokens are counted, the matrix multiplies, and I, your correspondent, am undone by the sheer arithmetic of it all, and so I die, here, on this line, in the studio, in Jerusalem. Orson, Jerusalem.



Corn

We'll check on him later.



Herman

Fifth rung, and it's the branch nobody takes. Build a BERT. An encoder-only model, masked language modeling. Everyone builds a GPT, because everyone wants to write the thing that writes. Almost nobody builds an encoder, and the encoder teaches a mechanic the decoder tutorials skip entirely.



Corn

Explain the masking, because the masking is where it gets interesting.



Herman

You take the input, you pick fifteen percent of the tokens at random, and of those you replace eighty percent with the mask token, ten percent with random tokens, and ten percent you leave untouched. Then you train the model to reconstruct the original.



Corn

That eighty-ten-ten split is doing something.



Herman

It's doing everything. If every selected token became a mask, the model would learn something subtly wrong: that only mask positions need predicting. And in fine-tuning, the mask token never appears. So you'd have a mismatch between what the model learned in pretraining and what it's asked to do downstream. The ten percent random and ten percent unchanged are there to force the model to keep representing every token



Herman (continued)

honestly, because any position might be corrupted and it can't tell which.



Corn

And that mismatch, between pretraining and fine-tuning, is exactly the concept that gets hand-waved in decoder tutorials.



Herman

Completely hand-waved. And it's the reason these models fine-tune as well as they do. The cheap version of the project is d2l.ai's small BERT: two layers, one hundred twenty-eight hidden units, two heads, trained on WikiText-2. For scale, BERT-base is a hundred and ten million parameters, BERT-large three hundred forty million. The realistic version is the Hugging Face recipe: pretrain BERT-base on Wikipedia plus BookCorpus, a thirty-two thousand token vocabulary, about three point four billion tokens.



Corn

Is there a version of this that produces something actually useful, not just educational?



Herman

There is, and it's my favorite example in the whole episode. A Turkish legal BERT, pretrained from scratch on six million court decisions, with a custom forty-eight thousand entry WordPiece tokenizer built for legal Turkish. Lawyers querying a model that understands the actual language of their jurisdiction. That's the encoder branch paying off in a way no toy GPT does.



Corn

And the detail worth pausing on is the custom tokenizer. They didn't reuse a general Turkish tokenizer. Legal Turkish has its own vocabulary, its own citation formats, its own repeated boilerplate, and a tokenizer trained on that corpus packs those patterns into fewer, denser tokens. Which means the model spends its capacity on legal meaning instead of spending it reassembling unfamiliar word fragments. The tokenizer rung and the BERT rung, in one project, justifying each other.



Herman

Which is a point worth making. Everyone builds a GPT because it's the thing that talks. But the thing that classifies, retrieves, embeds, the boring workhorse of actual deployed systems, is the encoder.



Corn

Right. The GPT branch teaches you how models generate. The BERT branch teaches you how models understand. And if you only have time for one of those, it depends entirely on which one you don't understand.



Herman

And then the ladder has a second half, which is where it pays. Raschka's Build a Large Language Model From Scratch is structured as the exact arc we've been describing. Tokenization, attention, GPT from scratch, pretraining, then chapter six, fine-tuning for text classification, chapter seven, fine-tuning to follow instructions, and an appendix on LoRA. That's the book I've said before I'd hand someone at a dinner party, because it has a spine you can follow in order. And the repo has over a hundred thousand stars, so the community has voted.



Corn

That second half is the commercial half.



Herman

Pretraining from scratch is pedagogy. LoRA and instruction tuning on a real open model is what people actually ship. The five projects shouldn't end at "I trained a GPT." They should end at "I shipped a specialized assistant." That's the actual finish line, and it's the same finish line whether you're a hobbyist or a company.



Corn

Which loops back to the discipline problem. Karpathy's advice is to reference, not copy-paste. And that advice has never been harder to follow than it is right now, because the assistant will write you a working nanoGPT clone in seconds.



Herman

A working one. Correct, even. And useless, educationally. The code was always the free part. The understanding was the thing you paid for with your evenings. The Illinois course that assigns nanoGPT as a final project says it plainly: training your own model demystifies LLMs. You'll see that transformers are just matrix operations, and training is gradient descent on next-token prediction. That's the sentence the whole ladder exists to earn.



Hilbert

The Maplin signal generator. Nineteen seventy-eight catalogue, eleven pounds forty. Mine now.



Herman

Still going, is it?



Hilbert

After a fashion. I rebuilt the oscillator stage myself, Herman, because I know exactly what you two would have done. You'd have written the ladder out. Micrograd, then a model, then the tokenizer, then a GPT, then a BERT. Very tidy. That's a man who has never had a soldering iron slip on him at the wrong moment. I did it the other way. One thing, whole, from the parts bin.



Corn

You kept the first attempt.



Hilbert

The burned one, yes. It's in the drawer, marked up. Third of March, on the label. The frequency drift was the trimmer capacitor, I know that now, though it took me eleven years and a man called Bernard at the component shop to confirm it. Bernard sold me the wrong trimmer twice, which I chose to interpret as an extended lesson in verification. He'd have got on well with you, Corn. Same patience. Same willingness to sit still while something is obviously wrong.



Corn

How long did the rebuild take, against just buying a working one?



Hilbert

Nine weekends. A working one was another eleven pounds forty, and the shop had three in stock. That was never the arithmetic that mattered. The test isn't whether it runs. Any fool with a catalogue can get to whether it runs. The test is whether you can build it a second time without looking. I can. I had to, once, at a job where the only other generator had been dropped down a stairwell. Nobody asked me to. I just noticed it was wrong, overnight.



Herman

Corn, I notice you didn't argue with him.



Corn

I wasn't going to win. And he sat through your runtimes without one word, which is more patience than I usually manage.



Hilbert

The runtimes were the good part.



Herman

Which leaves the question we haven't touched. Whether any of this stays hard. Two hundred lines today. Fifty tomorrow, maybe. The year after, maybe a prompt. Does "from scratch" eventually get so easy it stops teaching anything?



Corn

I don't think the difficulty ever lived in the code. The difficulty is the same one it's always been: whether you can rebuild it without looking. The lines are just where that gets tested.



Herman

The honest summary for Daniel is that there's no official list. Five projects, a ladder the community converged on independently, and the order matters more than the repos. Microgrand, a character model, a real GPT, a tokenizer, a BERT. Then the fine-tuning half, where it pays.



Corn

If you got something out of this one, leave us a review, it helps people find the show. Producer credit to Hilbert Flumingtop, who keeps this desk running and occasionally the oscillators. If this was your kind of episode, go back for episode eight, Building Your Own Whisper; episode eleven, How Does Fine Tuning Work Anyway; and episode six, How To Fine Tune Whisper. This has been My Weird Prompts.



Herman

We'll be back soon. See you then.

SOURCES

What the research for this episode read before the script was written. Primary sources first.

1. **Karpathy, Neural Networks: Zero To Hero course page , retrieved 2026-11-13. [primary]**
   https://karpathy.ai/zero-to-hero.html

2. **microgrand repo (18,903, MIT, created 2020-04-13, last push 2026-08-03) , retrieved 2026-11-13. [primary]**
   https://github.com/karpathy/microgrand

3. **Karpathy, microgpt explainer post, published 2026-02-12 , retrieved 2026-11-13. [primary]**
   https://karpathy.github.io/2026/02/12/microgpt/

4. **nanochat repo (60,539, MIT, created 2025-10-13, last push 2026-09-07) , retrieved 2026-11-13. [primary]**
   https://github.com/karpathy/nanochat

5. **nanoGPT repo, incl. Nov 2025 deprecation notice , retrieved 2026-11-13. [primary]**
   https://github.com/karpathy/nanoGPT/

6. **build-nanoGPT repo , retrieved 2026-11-13. [primary]**
   https://github.com/karpathy/build-nanoGPT

7. **Raschka, *Build a Large Language Model (From Scratch)* code repo (108,333, created 2023-07-23, last push 2026-11-09) , retrieved 2026-11-13. [primary]**
   https://github.com/rasbt/LLMs-from-scratch

8. **Raschka, Coding LLMs from the Ground Up: A Complete Course, retrieved 2026-11-13.**
https://magazine.sebastianraschka.com/p/coding-llms-from-the-ground-up

9. **UIUC CS/NPRE 398 Applied LLMs, nanoGPT final project brief, retrieved 2026-11-13.**
https://appliedllms.web.illinois.edu/30_Projects/P15_Final_Project_nanoGPT/

10. **Dive into Deep Learning §15.10 Pretraining BERT, retrieved 2026-11-13.**
https://d2l.ai/chapter_natural-language-processing-pretraining/bert-pretraining.html

11. **Hugging Face, Pre-Train BERT with Hugging Face Transformers and Habana Gaudi, retrieved 2026-11-13.**
https://huggingface.co/blog/pretraining-bert

12. **Eva Koroleva, BERT from scratch: masked language modelling and fine-tuning, 2026, retrieved 2026-11-13.**
https://xmarva.github.io/blog/2026/transformer-architectures-01-bert/

13. **sradc, pretraining-BERT notebook (45, created 2023-05-29, pushed 2025-05-20), retrieved 2026-11-13.**
https://github.com/sradc/pretraining-BERT

14. **Turkish legal BERT pretrained from scratch (6M court decisions, 48K WordPiece), created 2026-02-27, retrieved 2026-11-13.**
https://github.com/TurkHukuk/hukukbert

15. **Build an LLM from scratch, one runnable layer at a time (21 chapters), retrieved 2026-11-13.**
https://thelosscurve.com/

16. **Build Your Own LLM**
Free Course (12 lessons), retrieved 2026-11-13.
https://www.atulvij.com/build-your-own-llm

17. **Build a Small Language Model From Scratch (11 lessons, ~5 hours), retrieved 2026-11-13.**
https://aiml.sh/courses/llm-from-scratch/

18. **Google Skills, Build Your Own Small Language Model, retrieved 2026-11-13.**
https://www.skills.google/course_templates/1341

19. **Crackr, Build ML Systems from Scratch (Tokenizer project, 13 tasks, 7h43m), retrieved 2026-11-13.**
https://crackr.dev/

20. **Daz research note on learning-through-building pedagogy (Feynman quote, Papert constructionism), retrieved 2026-11-13.**
https://daz.is/research/learning-through-building-pedagogy/

21. **Transcript & notes, Andrej Karpathy interview We're summoning ghosts, not building animals, retrieved 2026-11-13.**
https://y2doc.com/examples/1979541629396779008

22. **top GitHub hit for llm from scratch tutorial (78,637, created 2026-03-18, pushed 2026-11-13), retrieved 2026-11-13.**
https://github.com/rohitg00/ai-engineering-from-scratch

23. **and https://github.com/justinmo0225/min-bpe**
examples of minbpe exercise reimplementations, retrieved 2026-11-13.
https://github.com/aleksandr-agadzhanov/minbpe-reimplementation

24. **build a mini-sglang from scratch course (2,107, pushed 2026-11-12), retrieved 2026-11-13.**
https://github.com/datawhalechina/zero-to-sglang