Audience Check — Type `1` or `0` in Chat

Help me calibrate — reply `1` (yes) or `0` (no) in Zoom chat:

- Run **OpenClaw or Hermes** on your laptop or VPS?
- Know **how an MNIST neural net works** — forward pass, backprop, SGD?
- Understand **how a Transformer works** — Q, K, V, multi-head attention?
- Written code for **GPU inference or training** small models yourself?
- Know **Yin Wang (王垠)** — compiler expert and programming KOL in China?
- English ability **≥ IELTS 6.5**?

> No judgment either way. Just calibrating where to spend time vs. skim.

My Path to Understanding Transformers

- First read about **K, Q, V** mechanism around end of **2023** — didn't understand much
- Read *The Illustrated Transformer*, watched Karpathy, Umar Jamil, StatQuest
- Replicated neural network for handwritten digits **from scratch** — that's where real understanding began
- By April-**2026**, transformers finally clicked — after **~3 years** of mulling
- Key insight: you don't get it the first time — the brain does its work over time

> "If one can write it from scratch without copying any code, one understands very well."

Blog post: Neural Network, Transformer and GPT

Learning Path Overview

Foundations (1–4): Neural net from scratch → PyTorch → language model

GPT (5–7): Transformers → train nanoGPT → scale up training

Agents (8–10): Chat model → coding agent → personal AI system

From math to GPT to AI system. That's the path.

Read Alongside — neuralnetworksanddeeplearning.com

Open the site in your browser and read **chapter by chapter**. This talk walks alongside it.

- **Ch 1:** Using neural nets to recognize handwritten digits
- **Ch 2:** How the backpropagation algorithm works
- **Ch 3:** Improving the way neural networks learn
- **Ch 4:** A visual proof that neural nets can compute any function
- **Ch 5:** Why are deep neural networks hard to train?
- **Ch 6:** Deep learning

Free online. Interactive diagrams. Python code on GitHub: mnielsen/neural-networks-and-deep-learning

From Neural Networks to Deep Learning

How deep learning actually trains.

- Gradient descent, learning rate, convergence
- Overfitting vs generalization
- Regularization, dropout, batch/mini-batch/SGD

Practice: Train a 3-layer classifier. Visualize the loss curve. Implement dropout manually.

Read Alongside — The Illustrated Transformer

Jay Alammar — jalammar.github.io/illustrated-transformer

The best visual walkthrough of the Transformer on the internet. We'll scroll through together:

- **Encoder / Decoder stack** — the macro picture
- **Self-attention step by step** — Q, K, V vectors drawn as colored boxes
- **Score → softmax → weighted sum** — animated across every token
- **Multi-head attention** — 8 heads in parallel, concatenated
- **Positional encoding** — why sin/cos waves?
- **Residual connections & LayerNorm** — the glue between layers

> If Karpathy's video is the code, Alammar's post is the **picture**. Read both.

Self-Attention — How Words Relate

> "The pizza came out of the oven and **it** tasted good."

How does the model know "it" refers to "pizza" and not "oven"?

Self-attention computes a similarity score between every pair of tokens:

1. Each token produces a **Query**, **Key**, and **Value** vector
2. Dot-product Query × Key = attention score (how relevant?)
3. Softmax normalizes scores to weights (0 to 1)
4. Weighted sum of Values = output for that token

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Training — Data, Loss, and Gradient Accumulation

Data: next-token prediction on massive text corpora.

- Input: `[The, pizza, came]` → Target: `[pizza, came, out]`
- Loss: **cross-entropy** between predicted and actual next token

Gradient accumulation simulates larger batch sizes on limited GPU memory:

- Instead of 1 batch of 64, do 8 mini-batches of 8
- Accumulate gradients, then update weights once
- Same math, fits in GPU memory

Mixed precision (fp16/bf16): halves memory, doubles speed.

Checkpointing: save model weights periodically to resume if training crashes.

nanoGPT Deep Dive

The key lesson. Train your own GPT.

- nanoGPT training loop, model architecture
- Weight initialization, data pipeline, sampling

Practice: Train nanoGPT on a small dataset. Modify model size. Train on a Chinese corpus. Change the tokenizer.

The 21-Second Iteration — Random Disk Access

Each iteration was **21 seconds** on H200 — way too slow.

- Effective batch: `16 × 1024 × 64 (grad_accum) = ~1M tokens/iter`
- With `gradient_accumulation_steps = 64`, the trainer does **64 random disk reads per iter**
- Data on boot disk, not scratch NVMe → **I/O bottleneck, not compute**
- MFU reported >100% — artifact (nanoGPT's MFU baseline is A100, not H200)

Fix:

- `grad_accum: 64 → 4`, `batch_size: 16 → 256` (same effective batch, **16× fewer disk reads**)
- Move `train.bin` to local NVMe scratch (`/mnt/scratch`)
- Target after fix: **<1 sec/iter, >100k tokens/sec**

LLM Agents (Claude Code / OpenClaw)

Build an OpenClaw-style system.

- Tool calling, agent loop
- Planning and execution
- Memory systems, CLI agent design

Practice: Build a coding agent. Build a CLI automation agent. Build a multi-step reasoning agent.

Hermes — Single Agent, Persistent Memory

One agent. Learns across sessions. Writes its own skills.

- **MEMORY.md** — long-term knowledge persisted between runs
- **Camoufox browser** — anti-detect scraping (e.g., Hacker News)
- **Auto-generated skills** — after a task, the agent writes the skill
- **Easier to reason about** than multi-agent — one brain, one log
- **Tool groups:** `browser_*`, `exec`, `file_*`, `memory`, `web_search`, `clarify`

> OpenClaw = **breadth** (channels, ecosystem). Hermes = **depth** (one agent that gets better over time).

lzwjava.github.io — My Blog

A Jekyll blog with **~400 original posts**, enhanced with AI-powered tools:

- **LLM Translation** — auto-translated to **9 languages** (EN, ZH, JA, FR, DE, etc.)
- **Google Cloud TTS** — audio versions of posts
- **XeLaTeX** — PDF generation for offline reading
- **EPUB** — ebook export for all post collections
- **GitHub Actions** — automated build, test, translate, deploy
- **MathJax** — renders math in technical posts
- **Night mode**, RSS feed, bilingual content

Plus **8000+ notes**, **323 Python scripts**.

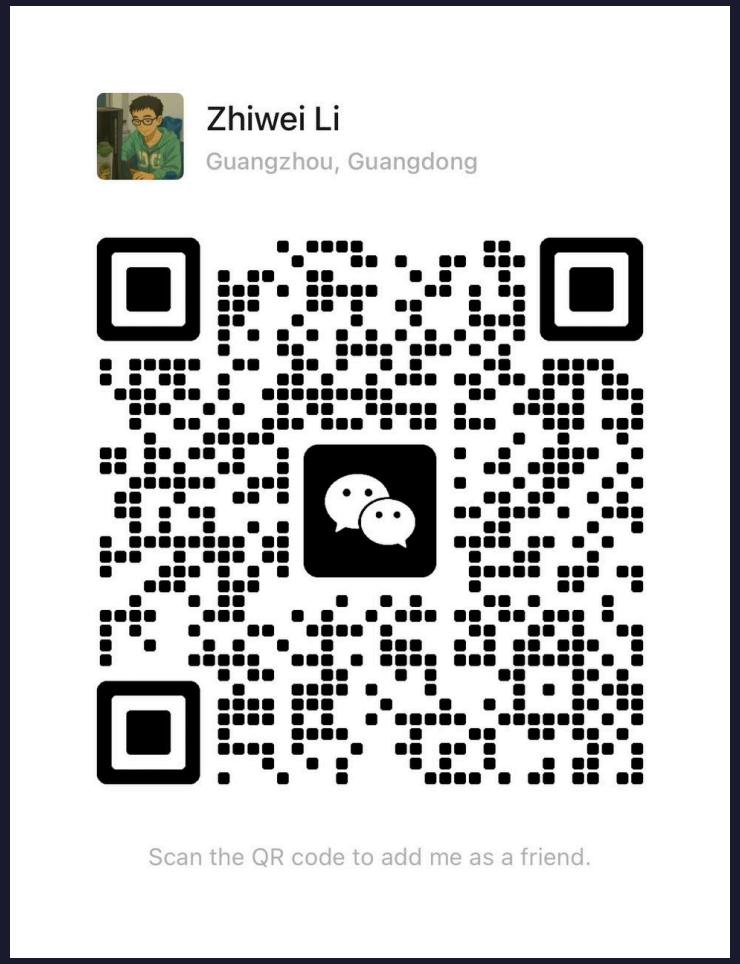
Reversing Myopia — 3 Years of Self-Experimentation

- Inspired by **Todd Becker** and **Yin Wang**. 3 papers published. 3 years of data.
- Myopia worsens because we use **full-prescription glasses** for close-up work
- Core principle: **"Just barely clear"** — wear glasses ~1.50D below full prescription
- Left eye: **350 → 250** (2022–2024), Right eye improving too

> Categorization: separate glasses for **near** (phone, laptop) and **far** (driving) — two glasses for two major scenarios.

Blog post: Vision Tips

WeChat


Zhiwei Li
Guangzhou, Guangdong
Scan the QR code to add me as a friend.