From Neural Networks to Agents

Zhiwei Li · AI Engineer

lzwjava@gmail.com · github.com/lzwjava · lzwjava.github.io

Ability Check and Factual Check

Before we start, a factual/trust check on this deck:

- **Made with Claude Code & Marp** — slides are AI-assisted, not hand-written
- Built from my public AI response notes — lzwjava.github.io/notes-en.html
- **Prompted by me**, **verified largely by me** — but I am the bottleneck, not an infallible reviewer
- I honestly understand the content in this deck 50% well
- Compared to **Andrej Karpathy**, my grasp of these topics is maybe **~30%** — he can build GPT from scratch on a whiteboard; I still reach for the code

> Treat this as a **learner's map**. Verify important things before you depend on it.

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.

Why This Talk — For 10-Year Engineers

For **senior application engineers** — backend, full-stack, mobile:

- **AI without the hype** — first-principles math & code, evaluate vendor claims
- **Code assistant skills on legacy code** — monoliths, tests, refactors (any tool your company allows)
- **Real numbers** — 1B tokens, H200 at $3.44/hr, where money actually goes
- **Agents on the terminal** — shell-level ops, internal tools, SSH chains
- **A clear path** — `y = wx` → MNIST → GPT → nanoGPT → your own agent
- **Bonus: reverse myopia, naturally** — 3-year self-experiment, 350 → 250 diopters

> Domain, judgment, production taste — AI multiplies, not replaces.

About Me

- **11 years** in software engineering — mobile, backend, full-stack, AI
- Currently at **a global IT services company** — contractor at **a global bank** — fullstack + AI-assisted development
- **~1 billion tokens** consumed by OpenRouter & other providers in past year
- Training GPT-2 with **nanoGPT** on **H200 GPU**
- Previously: contractor at DBS Bank & HSBC PayMe, startup founder (30k users)
- **AI/LLM:** PyTorch, nanoGPT, Claude Code, OpenRouter, Copilot, llama.cpp
- **Backend:** Java, Spring Boot, Python | **Cloud:** AWS, Azure | **Mobile:** iOS, Android
- **2011 NOIP Guangdong** — 1st prize (Guangzhou, round 1), advanced to round 2, ~top 300
- **Born 1995** | **IELTS 6.5** (Reading 8.5)

AI Journey

- Completed **ML Specialization** & **Deep Learning Specialization** (Andrew Ng)
- Training GPT-2 124M with **nanoGPT** on H200 GPU
- ~**1 billion tokens** consumed via OpenRouter & other providers
- Built custom agents, prompt pipelines, automation tools
- Experimented with llama.cpp, embeddings, MMLU benchmark
- Developing autonomous workflows with **Claude Code** & **OpenClaw**

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

GTY — Great Teacher Yin

Yin Wang (yinwang.org) — 140k followers on Weibo, independent thinker, CS researcher.

- Followed him since **2013** — he taught me how to think from first principles
- His professor: **Daniel P. Friedman** (Indiana University, born 1944)
- Yin Wang calls ML **"differentiable computing"** — it's just calculus
- His vision restoration method(learned from todd becker) inspired my **3-year myopia experiment**
- Subscribed to his Substack since 2022

> "He not only taught me knowledge, but truly showed me how to think. He passed on the technique of catching fish from Daniel P. Friedman to me."

Blog post: GTY - Great Teacher Yin

"Differentiable Computing"

> "Machine learning is really useful, one might even say beautiful theory, because it is simply **calculus after a makeover**! It is the old and great theory of Newton, Leibniz, in a simpler, elegant and powerful form."

> "There is no 'intelligence' in artificial intelligence, no 'neural' in neural network, no 'learning' in machine learning. What really works in this field is called **'calculus'**."

— **Yin Wang**

See through the buzzwords to the math underneath.

Learning Philosophy

Inspired by **Yin Wang** and **Andrej Karpathy**

- **Print variables** to understand — not just read about it
- **Read real code** (nanoGPT, not just textbooks)
- **Train small models** yourself
- **Build systems**, not just theory
- Iterate like training an LLM — you don't get it right the first time

General Learning Tips

What actually works for me — cheap, repeatable, compounding:

- **Ask AI chatbots** — Claude, ChatGPT, Gemini
- **Use code assistants** — Claude Code, Copilot
- **Add logs with Claude Code** — run, observe, relearn
- **Write a mini / nano version** — a toy neural net, a tiny agent
- **Talk to people with similar passion** — communities, friends, coworkers who care
- **YouTube & podcasts** — Karpathy, StatQuest, Lex, Dwarkesh
- **Reflect and share** — take notes, write blog posts, follow tangents

> Stack these six. You'll be surprised how far you get in a year.

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.

The Simplest Neural Network — Guess a Number

One "neuron": $y = x \cdot w$. Input $x = 2$, target $t = 5$.

Step	w	$y = 2w$	Error $(y - t)$
Start	2.0	4.0	**–1.0** (too small)
Nudge up	2.2	4.4	–0.6
Again	2.5	5.0	**0** ✅

How do we know which way to nudge? Calculus.

- Loss $L = \frac{1}{2}(y - t)^2$
- Gradient $\frac{\partial L}{\partial w} = x \cdot (y - t) = 2 \cdot (-1) = -2$
- Update: $w \leftarrow w - \eta \cdot \frac{\partial L}{\partial w}$ → gradient is negative, so w goes **up**

> That's it. A real network is millions of these, stacked — but the idea is the same.

Neural Networks from First Principles

Reference: Michael Nielsen — Neural Networks and Deep Learning

Understand what a neural network really computes.

- Input: 784 pixels (28×28 handwritten digit) → Output: 10 neurons (digits 0–9)
- Each neuron: **weighted sum + bias + activation function**
- $$a^l = \sigma(w^l a^{l-1} + b^l)$$
- Sigmoid maps any value to 0–1: $\sigma(z) = \frac{1}{1+e^{-z}}$

For a 2-layer network: **784 × 10 weights + 10 biases** to learn.

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

Backpropagation — How Networks Learn

1. **Input:** set activation a^1 for input layer
2. **Feedforward:** compute $z^l = w^l a^{l-1} + b^l$, then $a^l = \sigma(z^l)$
3. **Output error:** $\delta^L = \nabla_a C \odot \sigma'(z^L)$
4. **Backpropagate:** $\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$
5. **Update:** $w \to w - \eta \frac{\partial C}{\partial w}$, $b \to b - \eta \frac{\partial C}{\partial b}$

Practice — Build a Neural Network from Scratch

Replicate the MNIST digit recognizer in pure Python + NumPy:

```
def __init__(self, sizes):          # e.g. [784, 30, 10]
    self.weights = [np.random.randn(y, x)
                    for x, y in zip(sizes[:-1], sizes[1:])]
    self.biases = [np.random.randn(y, 1) for y in sizes[1:]]
```

- Print every variable shape: `weights[0].shape = (30, 784)`
- Watch accuracy climb: 10% → 50% → 95% over epochs
- **If you can write it from scratch, you understand it**

After this, you understand every number inside a neural network.

Dimensions — The Key to Understanding

Know the **shape** of every variable in a neural network:

Variable	Shape	Meaning
Input `x`	`(784, 1)`	28×28 pixels flattened
`weights[0]`	`(30, 784)`	Layer 1 → Layer 2 connections
`biases[0]`	`(30, 1)`	One bias per hidden neuron
`z = w·a + b`	`(30, 1)`	Weighted sum before activation
`a = σ(z)`	`(30, 1)`	Activated hidden layer
`weights[1]`	`(10, 30)`	Layer 2 → Output connections
Output	`(10, 1)`	Confidence for digits 0–9

Matrix multiply: `(10, 30) × (30, 1) = (10, 1)` — dimensions must align.

> "It just does the differentiable computation. Know the dimensions of every

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.

Q&A — Checkpoint 1 / 3

Foundations: neural nets, backprop, gradient descent, dimensions.

Ask anything — before we jump into GPT and Transformers.

Language Modeling Fundamentals

Understand what GPT predicts.

- Tokenization (BPE), n-gram models
- RNN/LSTM intuition
- Next token prediction, cross-entropy for language

Practice: Build a character-level language model. Train on tiny Shakespeare. Generate text.

PyTorch Minimal Framework

Read any PyTorch model with confidence.

- Tensor fundamentals, autograd
- `nn.Module` design, optimizer mechanics
- Dataset and DataLoader

Practice: Rebuild MLP in PyTorch. Train a CIFAR classifier. Inspect gradients.

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.

GPT Architecture — The Big Picture

Reference: Andrej Karpathy — Let's build GPT: from scratch, in code

```
Input Text
  → Tokenizer (text → token IDs)
    → Token Embedding + Positional Encoding
      → N × Transformer Blocks
        → (Self-Attention → Feed-Forward → LayerNorm)
      → Linear → Softmax
        → Next Token Prediction
```

The entire model learns to predict: **given these tokens, what comes next?**

Tokenizer — Text to Numbers

Before the model sees anything, text must become numbers.

- **Character-level:** each character is a token — simple but slow
- **BPE (Byte Pair Encoding):** merge frequent character pairs iteratively
  - `"learning"` → `["learn", "ing"]` → `[4821, 278]`
- GPT-2 uses ~**50,257** tokens; GPT-4 uses ~**100k**
- Tokenizer is trained **separately** before the model

```
"The pizza came out" → [464, 13293, 1625, 503]
```

Different tokenizers produce different token IDs — changing the tokenizer changes everything.

Word Embedding — Tokens to Vectors

Each token ID maps to a **learned vector** (e.g., 768 dimensions in GPT-2).

- `token 464 ("The")` → `[0.12, -0.34, 0.56, ..., 0.08]`
- These vectors are **not hand-crafted** — they're learned during training
- Similar words end up with similar vectors

Positional encoding is added so the model knows word order:

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d})$$

$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d})$$

Without position info, "dog bites man" = "man bites dog" to the model.

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

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

Q, K, V — Intuition

Think of it like a **search engine** inside the model:

Component	Role	Analogy
Q (Query)	What am I looking for?	Your search query
K (Key)	What do I contain?	Page titles / descriptions
V (Value)	What's my actual content?	Page content returned

For the word "it":

- **Q** asks: "what noun am I referring to?"
- **K** of "pizza" answers: "I'm a noun, a food, the subject"
- **K** of "oven" answers: "I'm a noun, an appliance, inside a prepositional phrase"
- Attention score for "pizza" > "oven" → **V** of "pizza" contributes more

Q, K, V — Dimensions in Attention

For a sequence of **T** tokens with embedding size **d** and head size **d_k**:

Variable	Shape	How it's computed
Input `X`	`(T, d)`	Token embeddings, e.g. `(6, 768)`
`W_Q`	`(d, d_k)`	Learned query projection
`W_K`	`(d, d_k)`	Learned key projection
`W_V`	`(d, d_k)`	Learned value projection
`Q = X · W_Q`	`(T, d_k)`	e.g. `(6, 64)`
`K = X · W_K`	`(T, d_k)`	e.g. `(6, 64)`
`Q · K^T`	`(T, T)`	`(6, 6)` — every token attends to every token
`softmax(QK^T/√d_k) · V`	`(T, d_k)`	Weighted values output

GPT-2: `d=768`, `12 heads`, `d_k=64` per head. `12 × 64 = 768` — heads concat back

Multi-Head Attention & Transformer Block

Multi-head: run attention **multiple times in parallel** (e.g., 12 heads in GPT-2).

Each head can learn different relationships:

- Head 1: syntactic (subject-verb)
- Head 2: semantic (noun-pronoun)
- Head 3: positional (nearby words)

One Transformer Block:

```
Input
  → Multi-Head Self-Attention + Residual Connection
  → LayerNorm
  → Feed-Forward Network (MLP) + Residual Connection
  → LayerNorm
  → Output
```

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.

Generation — From Trained Model to Text

After training, generation is **autoregressive** — one token at a time:

```
Prompt:  "The meaning of life is"
Step 1:  → predict next token → "to"
Step 2:  "The meaning of life is to" → "find"
Step 3:  "The meaning of life is to find" → "purpose"
...
```

Sampling strategies:

- **Greedy:** always pick the highest probability token
- **Temperature:** lower = more deterministic, higher = more creative
- **Top-k:** sample from top k candidates only
- **Top-p (nucleus):** sample from smallest set covering p% probability

nanoGPT implements all of these in ~300 lines of Python.

Source Code Exploration — nanoGPT Locally

`git clone https://github.com/karpathy/nanoGPT` — a whole GPT in ~**600 lines** of Python.

File	What to read	Lines
`model.py`	`CausalSelfAttention`, `Block`, `GPT` — full architecture	~300
`train.py`	training loop, gradient accumulation, AdamW, DDP	~350
`sample.py`	autoregressive generation, temperature, top-k	~90
`data/shakespeare_char/prepare.py`	char-level tokenizer, train/val split	~40
`config/train_shakespeare_char.py`	hyperparameters: `n_layer`, `n_head`, `n_embd`	~30

- Read `model.py` **top to bottom** with pen and paper — draw the tensor shapes
- Set a breakpoint in `forward()` — inspect `Q`, `K`, `V` shapes match the slide
- Change `n_layer: 6 → 2` and retrain — watch loss get worse

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.

GPT Training Engineering

Run real GPT training at scale.

- GPU memory optimization, gradient accumulation
- Mixed precision, checkpointing
- Multi-GPU training, dataset scaling

Practice: Train a 100M parameter model. Resume training. Fine-tune a model.

H200 Trial — DigitalOcean GPU Droplet

Config	H200 ×1	H200 ×8
VRAM	141 GB	1.1 TB
vCPU / RAM	24 / 240 GB	192 / 1920 GB
NVMe Scratch	5 TB	40 TB
Price	**$3.44/hr**	**$27.52/hr**

- **Linear scaling, no bulk discount** — ×8 is just 8 × ×1
- Rule: **×1** for experiments & fine-tuning, **×8** only for distributed training
- At $82/day per GPU, every idle hour hurts

Training Setup — FineWeb 60GB

- **Dataset:** FineWeb (HuggingFace) — 60 GB text, ~10B token sample (full = 1.4T)
- **VPS → GPU Droplet transfer:** 5 hours over internet → **5 minutes on private network** (450 MB/s internal)
- **Preprocessing:** 12 parallel CPU workers, ~40M tokens/sec, done in ~20 min
- **Training target:** GPT-2 124M → 1.5B with nanoGPT
- For a 1.5B model to be decent: needs ~100B–1T tokens

> Lesson: **never transfer big datasets over public internet** — use the provider's private network.

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**

Why Runpod > DigitalOcean for Training

- **Per-second billing** — DO bills per hour (pay for idle minutes)
- **30+ GPU types** — RTX 4090, A100, H100, H200, B200 — DO has only a few
- **Community Cloud** pricing much cheaper than Secure Cloud / DO
- **Bring your own Docker** — no fighting with pre-baked images
- **Sub-minute provisioning**, FlashBoot serverless endpoints
- NVMe local scratch by default — fewer I/O surprises

> For a solo researcher doing trial runs, Runpod's flexibility + price usually wins.

Q&A — Checkpoint 2 / 3

GPT internals: tokenizer, Q/K/V, attention, training, generation, nanoGPT.

Questions on Transformers or GPU training — before we shift to agents.

Benefits for Senior Engineers

Why invest time in neural networks and agents?

Area	Without AI	With AI Agents
Legacy code	Weeks to months to understand & modify	Days to weeks with AI-assisted exploration
Boilerplate	Hours writing CRUD, tests, docs	Minutes to generate, you review & refine
Debugging	Manual grep, print, trace	Agent probes the system at machine speed
Research	Read docs line by line	Ask and get code examples instantly
Documentation	Tedious, always outdated	Generate draft from code, keep in sync
Career	AI is optional	AI is becoming table stakes

For bank engineers: Your value isn't in typing code — it's in judgment, architecture, and domain knowledge. AI handles the boilerplate so you focus on decisions.

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.

Claude Code Tips — Prompting

- **input + model = output** — the only equation that matters
- **Precise input** — vague prompts produce vague code; be specific about files, signatures, constraints
- **Paste, don't describe** — copy the code snippet, HTML, or error message directly into the prompt
- **One-shot or few-shot** — show an example output when the pattern is non-obvious

> Garbage in, garbage out. The model is fixed — you control the input.

Claude Code Tips — Workflow

- **Clone the repo down and ask** — single source of truth beats screenshots and pasted fragments
- **Categorize errors** — fix one big class at a time (all type errors, then all lint, then all tests)
- **Save inputs and outputs** (prompts and responses) — review the good ones later; your input library compounds
- Treat inputs like code: refine, reuse, version them

OpenClaw — Multi-Agent Orchestration

Self-hosted AI runtime — "AI employee" for system tasks. Model-agnostic: bring your own key (Claude, GPT, Kimi, local).

- **Channels:** Telegram, WeChat, WhatsApp, Nextcloud Talk
- **Agent loop:** observe → call tools → exec → reflect → repeat
- **Skills ecosystem** — 13k+ community skills (one can be ~20 lines)
- **Distributed** — chain across hosts A → B → C via SSH / HTTP
- **My setup:** ~29 active sessions, browser + shell + scheduled heartbeats

> Think "AI operator" for channels and infra — not just coding.

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).

Why Agents Click — My "Aha" Insights

Understanding deepens the longer you live with them. Some takeaways from daily use:

- **Exec as first-class citizen** — the essence of Lobster (OpenClaw 🦞). Full shell, full sudo, full control of the machine
- **Dozens to hundreds of tool calls per minute** — the loop runs at machine speed, not human speed
- **Rapid iteration = experimentation** — hypothesis → tool call → result → next hypothesis, in seconds
- **The agent probes, not reads** — `ls`, `cat`, `grep`, `python -c ...` — like a dev at a terminal
- **You stop being the I/O bottleneck** — you become the bottleneck on intent and review

> These insights come from *using* the agent, not reading about it. Living with it for

Agent Tips — Lessons From Running Both

- **PRD first** — write the spec before the agent starts. Review every 10–15 min on long tasks
- **Cost is real** — autonomous loops burn tokens fast; one user hit **$3.6k** in month one. Set budgets
- **Security is not abstract** — prompt injection, leaky `allowedOrigins`, unrestricted exec. Run `security audit --deep`
- **Behavior files are the guardrail** — `AGENTS.md` (rules), `SOUL.md` (values), `TOOLS.md` (tool use)
- **Vet community skills** — powerful, but malware / exfiltration / injection live there too
- **Debug** — hang? SSH in, check CPU, restart gateway. Failed queue: `~/.openclaw/delivery-queue/failed/`

> These are my personal learning setups — production has more edges.

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**.

ww — Cross-Platform CLI Toolkit

github.com/lzwjava/ww — A Python CLI for developer productivity with LLM-powered helpers.

Group	Examples
Git	AI commit messages, squash, diff-tree, classify commits
Note	Create notes with git integration, obfuscate data
PDF	Markdown → PDF, batch pipeline, code → PDF
Image	Crop, remove background, compress, screenshots
Search	Multi-engine web search (Bing, DuckDuckGo, Ecosia)
Copilot	OAuth auth, model listing, chat with Copilot API
System	macOS/Linux utils, network scan, process management
Sync	Claude settings, bashrc, ssh configs

iclaw — AI Coding in Restricted Environments

github.com/lzwjava/iclaw — A minimal terminal REPL for AI-assisted coding via GitHub Copilot.

Built for **enterprise-constrained environments** — no browser extensions, no IDE plugins, just a Python CLI.

- **Multi-turn conversations** with Copilot (GPT-5.2 default)
- **Native tool calling** — web search, shell execution, file editing
- **Multiple search providers** — DuckDuckGo, Startpage, Bing, Tavily
- **GitHub OAuth** device flow authentication
- Slash commands: `/model`, `/search`, `/compact`, `/export`, `/copy`

Disclaimer: Built for learning purposes. All tests with Copilot were performed using my personal account.

Beyond Engineering

- **Self-taught researcher** — 3 papers on natural vision restoration
- **~400 blog posts** — translated to Chinese with LLMs, ~8000 AI answer notes
- **Life hacker** — hundreds of small innovative practices
- **Traveler** — USA (twice), Hong Kong, Macao, half of China
- English learned from **60+ Filipino teachers** online
- **Crypto & US stock investor** since 2018
- Owns **hundreds of gadgets** — EMF meter, telescope, etc.

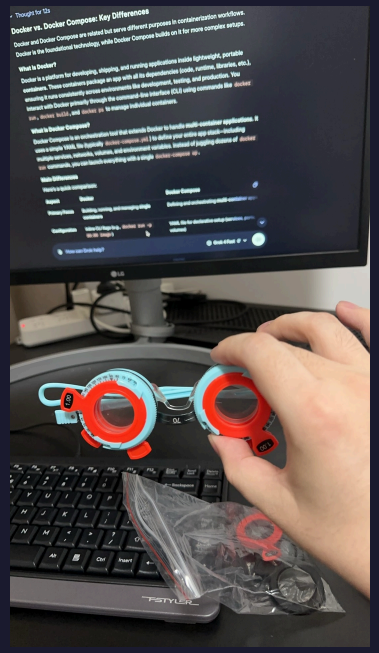
Reversing Myopia — 3 Years of Self-Experimentation

- 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

Myopia Experiment — Photos



Q&A — Checkpoint 3 / 3

Agents, OpenClaw, Hermes, Claude Code, blog, CLI tools — anything from the whole talk.

Final questions before we wrap up.

Let's Connect

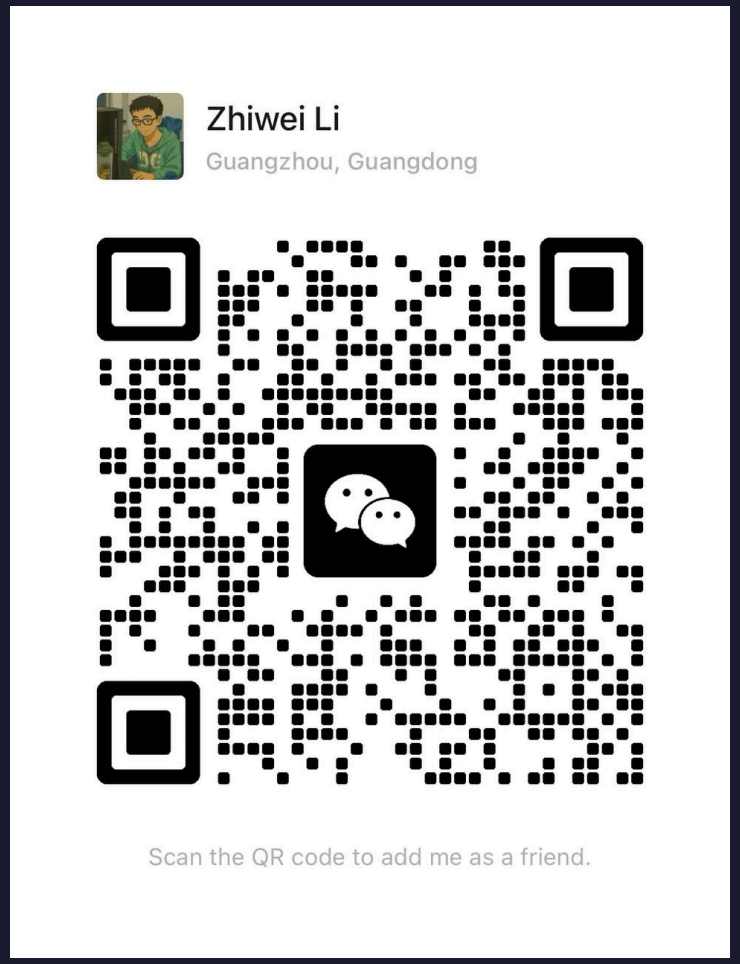
Email: lzwjava@gmail.com

GitHub: github.com/lzwjava

Blog: lzwjava.github.io

WeChat: lzwjava

WeChat


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

LinkedIn

linkedin.com/in/lzwjava

or search **Zhiwei Li AI Engineer**

Thanks

(unordered, no particular ranking)

Family · Qi Zhang · Yin Wang · Andrej Karpathy · Ming
Shawn Shao · F Team · Gary Ma · Lin Zhen · Liezun Xiao
Shuming Liang · Steve Chen · Raymond · Gaven

and many more

Thank You

lzwjava@gmail.com · lzwjava.github.io