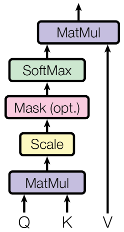
Vibe Coding: An Introduction

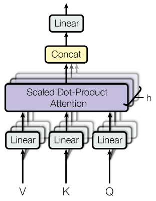
Ian J. Watson

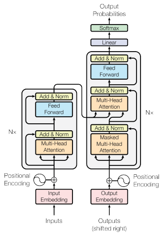
University of Seoul

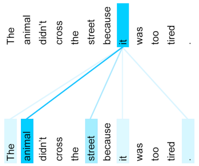
NSRI 2025 Workshop 2026-01-14

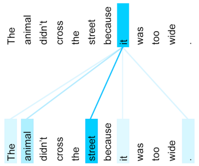
One Slide Intro to the Transformer Model








The animal didn't cross the street because it was too tired .
The animal didn't cross the street because it was too tired .


The animal didn't cross the street because it was too wide .
The animal didn't cross the street because it was too wide .

- The transformer model produces output sequence from input sequence
- Uses self-attention layers to model the dependencies between elements of the sequences: each word is trained to "attend" to related words
- The transformer model is at the heart of modern deep learning
    - This paradigm replaces the previous recurrent models which processed single tokens at a time to produce a single context vector
    - Ever since it was argued that Attention is All You Need in 2017
- The model for LLMs: you put in tokens representing some partial piece of writing, it predicts the next token, which is used to predict the token after, etc.

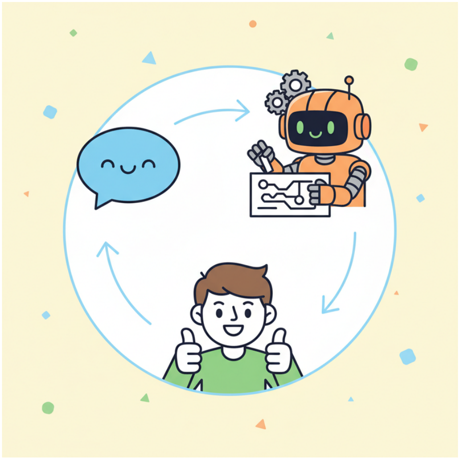
What is Vibe Coding?

- Coined by Andrej Karpathy (Feb 2025)
  - Also the author of a great series on how Transformers work
- "You fully give in to the vibes, embrace exponentials, and forget that the code even exists"
- Natural language → working code via LLM
- Not just autocomplete: AI writes entire functions, files, applications

https://x.com/karpathy/status/1886192184808149383
https://karpathy.ai/zero-to-hero.html

The Workflow

- Describe what you want in plain English
- AI generates code, you review/test
- Iterate: refine with follow-up prompts
- Tools: Claude Code (CLI), Codex, Gemini cli, etc.
    - In general, real vibe coding is done from the command line with an agentic cli
    - Agentic means that it can use tool use, and check its work as it goes



Why CLI Tools, Not Browser?

- **Browser chat** (ChatGPT, Claude web, Gemini): copy-paste ping-pong
  - You paste code → AI responds → you paste back → repeat
- **IDE**: → you'd just be distracted by the actual code, remember to vibe!
- **CLI tools** (Claude Code, Codex, Gemini CLI): direct access to files
  - Reads your codebase, writes changes in place
  - Runs tests, linters, builds—sees errors, fixes them
  - No manual copy-paste, no context lost in translation
- The agentic loop: generate → execute → observe → fix
  - AI becomes a collaborator *inside* your project, not outside it
- Maximal vibe coding: the Ralph Wiggum loop
  - Run in a loop: complete one task to reach a goal, use another agent to check the changes, fix the issues the agent found, repeat
  - A most dangerous game…
- Browser is fine for questions; CLI is for getting work done

`https://github.com/anthropics/claude-plugins-official/tree/main/plugins/ralph-loop`

The Major Players (Trust No One)

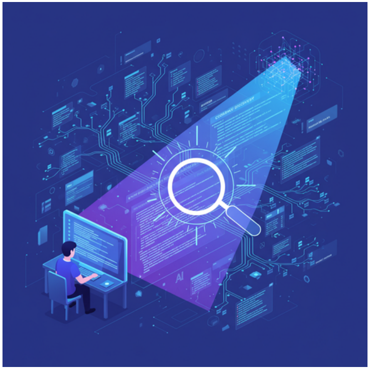
- **OpenAI** (GPT, Codex): started "open" and safety-focused, now closed-source and profit-driven
- **Anthropic** (Claude): safety-focused spinoff from OpenAI, but now courting military/defense contracts
- **Google** (Gemini): powerful models (but Opus 4.5 still best for vibes), but Google kills products constantly—will this last?
- **xAI** (Grok): Elon's venture—enough said
- All have your data, all have incentives misaligned with yours
- Use them, but don't depend on any single one
  - Its a fight for the future and not everyone should survive

"I Tried It Before, It Was Useless"

- If you tried AI coding in 2023–2024 and gave up: try again
- The latest models (basically Nov. 2025 as the inflection point) have undergone a phase transition
- Previous generations: autocomplete on steroids, hallucinated constantly
- Current generation: actually understands context, writes working code
- Agentic tools (Claude Code, Cursor Agent): can run commands, fix their own errors
- Still not perfect, but now genuinely useful for real work

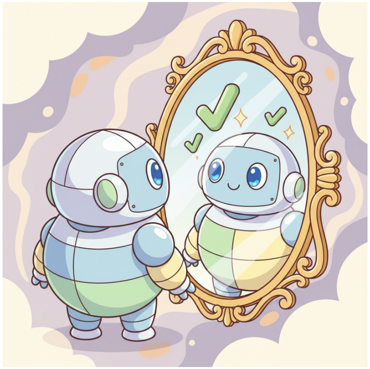
Warmup Use Case: Understanding Large Codebases

- AI CLI tools excel at navigating unfamiliar code
- Example: CMSSW (CMS experiment software, millions of lines)
  - Asked: "How is the Zero Bias skim implemented?"
  - AI searched autonomously, summarized the trigger logic
  - Discovered: despite the name, Zero Bias is *biased*
- Later confirmed via bunch crossing ID distributions
- Needed correction in our background analysis



The Testing Loop

- Vibe coding works best when the AI can check its own work
- Give it concrete ways to verify: tests, type checks, linters, access to browser
- The AI can build these checks itself, then use them to iterate
- Tight feedback loop: generate → test → fix → repeat
- Without this, you're just generating code and hoping
- AI is very good at both generating large swathes of uneeded code, and removing working code in order to "pass tests"
    - One must be vigilant for these failure modes, even when vibing



Managing Context

- Each session starts with a amnesiac LLM, which you fill with **context**
- Context window = AI's working memory (limited!)
- **Clear context** when starting a new task—old context can poison new work
- **Get relevant files in early**: the AI can only work with what it sees
- **Keep irrelevant info out**: wrong context $\rightarrow$ wrong answers
- For long tasks: summarize progress, break into sessions

Project Memory: CLAUDE.md / AGENTS.md

- Config files that persist across sessions
- Include: project structure, conventions, key commands
- Can reference other files—AI figures out when to read them
- Don't include: session-specific details, obvious things
- AI tends to over-fill these—review and trim regularly

Skills and MCP

- **Skills**: reusable process knowledge (SKILL.md files)
  - "How to deploy", "how to run tests", "how to make a PDF"
  - Claude has built-in discovery: "make a talk" $\rightarrow$ finds the skill
  - Gemini/Codex can use them too, but need explicit pointers
- **MCP** (Model Context Protocol): connect external data sources
  - Databases, APIs, services the AI can't otherwise access
  - Useful, but not everything needs to be an MCP server
- Skills = how to do things; MCP = access to things

Simon Willison: `https://simonwillison.net/2025/Oct/16/claude-skills/`

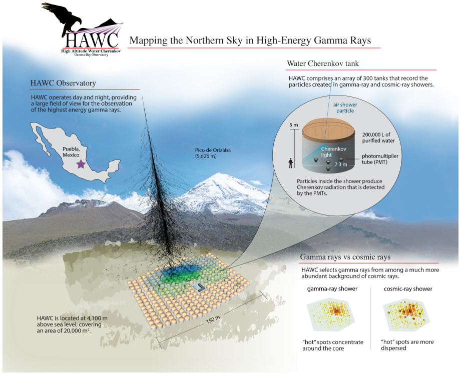
When Does It Work Well?

- Prototyping and quick tools
- Boilerplate-heavy tasks (e.g. Deep Learning!)
- Learning new frameworks/languages
- Tasks with clear specifications
- Glue code, scripts, utilities
- Can do more complex things, but the more complex/specialized the task, the more the human must be in the loop

Why Scientists Should Care

- We write a lot of "one-off" code
  - Data processing scripts
  - Plotting and visualization
  - Automation of tedious tasks
- Often not our primary skill
- Time spent coding $\neq$ time spent on science

Examples from HAWC: High Altitude Water Cherenkov Observatory


HAWC
High Altitude Water Cherenkov Gamma-Ray Observatory
Mapping the Northern Sky in High-Energy Gamma Rays
HAWC Observatory
HAWC operates day and night, providing a large field of view for the observation of the highest energy gamma rays.
Puebla, Mexico
Pico de Orizaba (5,626 m)
Water Cherenkov tank
HAWC comprises an array of 300 tanks that record the particles created in gamma-ray and cosmic-ray showers.
air shower particle
5 m
200,000 L of purified water
Cherenkov light
photomultiplier tube (PMT)
7.3 m
Particles inside the shower produce Cherenkov radiation that is detected by the PMTs.
Gamma rays vs cosmic rays
HAWC selects gamma rays from among a much more abundant background of cosmic rays.
gamma-ray shower
cosmic-ray shower
"hot" spots concentrate around the core
"hot" spots are more dispersed
HAWC is located at 4,100 m above sea level, covering an area of 20,000 m².
150 m

- Gamma-ray observatory in Mexico at 4100m altitude
- 300 water tanks, each with 4 PMTs
- Detects Cherenkov light from air shower particles
- Reconstructs energy, direction, and particle type of incoming gamma rays

Example: Modernizing a Legacy Codebase

- **Aerie**: HAWC's core software framework (~960k lines of code)
  - Data I/O, reconstruction, map making, significance calculations
- Problem: depends on Python 2 and ROOT 5—increasingly hard to build
- Vibe-coded a modernization to pixi (modern package manager)
  - Replaced CMake files, rebuilt modules incrementally
  - My role: check it compiles, say "keep going with the next module"
  - Done over a day or two with minimal intervention
- 2383 files, 150 CMake files—all updated by Claude

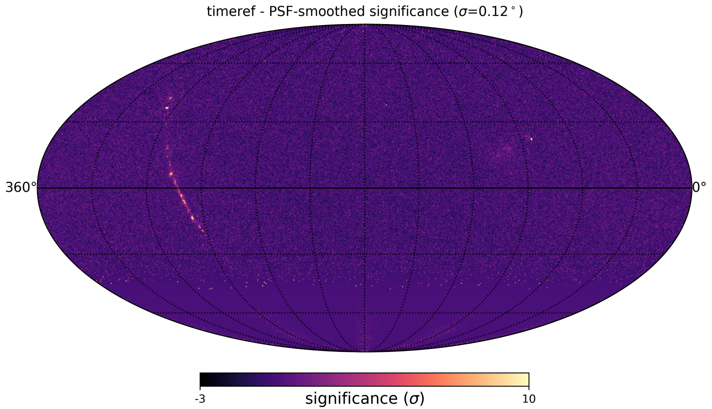
Example: Deep Learning Training TUI

- Training DeepHAWC models for gamma-ray event reconstruction
- Multiple training runs with different hyperparameters
- Built a TUI (terminal UI) to compare runs:
    - View loss curves across runs
    - Drill into model architecture and training config for each run
    - Quickly identify best performing models

Example: Batch Job Tracker

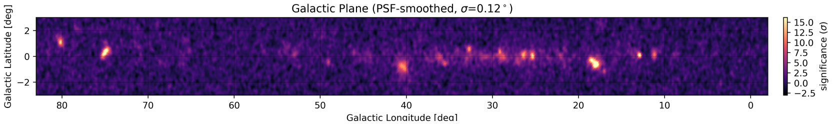
- Running inference over HAWC data: thousands of batch jobs
- Needed to manage jobs without overwhelming the batch system
- Vibe-coded a bespoke tracker with:
  - SQLite database tracking job status
  - Tools to query: how many jobs complete? which dates covered?
  - Automatic parsing of output for event counts
  - Live sky map plotting to check results as jobs finish

Sky Maps from the Batch Tracker



Sky map generated incrementally as batch jobs completed

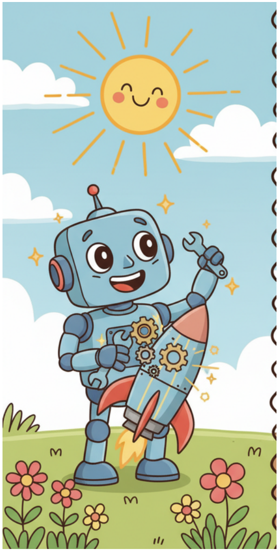
The Plotting Was Also Vibe Coded



- Claude set up astropy/matplotlib/healpy in a uv toml
- Learned our output data format, filled HEALPix maps
- Wrote simple background estimate + significance calculator
- GBs of data $\rightarrow$ map in minutes (full HAWC pipeline: days + specialized IRFs)

The Good

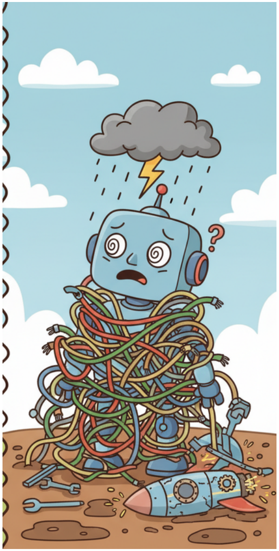
- Dramatically speeds up prototyping
- Great for exploring unfamiliar libraries
- Handles boilerplate you'd otherwise copy-paste
- Can explain code as it writes
    - If you are learning, do this constantly, ask it why its making the changes it is, if there is a line of code that doesn't make sense to you, or feature you haven't used, make it explain it



The Bad

- Hallucinations: confident but wrong
  - Non-existent APIs, incorrect syntax
- "They lie, but not on purpose—they just want to make you happy"
- Security blind spots, unnecessary dependencies
- Can produce subtly buggy code
- Struggles with complex, multi-step requests
  - Less true of the more recent models

`https://refact.ai/blog/2025/top-10-tips-for-conscious-vibe-coding/`



Trust But Verify

- Always review generated code
- Test before deploying
- Be especially careful with:
  - Security-sensitive code
  - Numerical/scientific calculations
  - Code that modifies data
- The AI is a junior collaborator, not an expert
- "If you can't verify it, don't use it"
- Andrew Ng: telling beginners "don't learn to code, AI will do it" is "some of the worst career advice ever given" `https://en.wikipedia.org/wiki/Vibe_coding`

Let's Build Something

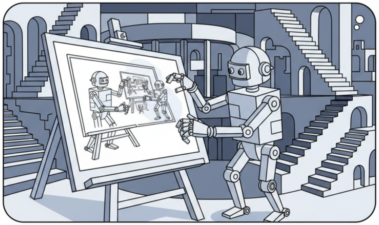
- Janggi (Korean Chess) in the browser
- Goal: playable game from scratch
- Watch the vibe coding workflow in action

Demo

Live Demo

This Talk Made Itself

- The tools used to build this talk were themselves vibe-coded:
  - Skill for creating new talks (directory structure, templates)
  - Skill for generating illustrations (Gemini MCP integration)
  - CLAUDE.md with project conventions
- Even these slides were built in a vibe coding session
  - But then polished by hand, human-AI collaboration
- It's turtles all the way down



The Buck Stops With You

- Vibe coding: describe → generate → iterate
- Excellent for scientific tooling and prototypes
- Verify everything, especially calculations
- Great force multiplier, not a replacement for understanding
- "The AI did it" is not an excuse
- "I trust Claude/Gemini/Codex" is not verification
- When correctness matters: **understand the code**
- When it doesn't: feel the vibes

YOU are responsible for your code

The AI is a tool. You are the scientist.