

# Vibe Coding for Problem Solvers

## Translating Intent to LLMs

By Lucas Soares

12/09/2025

# About Me



## AI Engineer & Instructor

Passionate about AI-augmented development  
Building tools and teaching workflows  
Focus on practical AI integration

# Methodology Notes

# Methodology Notes

## 1. Presentation Block

# Methodology Notes

1. Presentation Block

2. Demo

# Methodology Notes

1. Presentation Block

2. Demo

3. Q&A + Summary

# Methodology Notes

1. Presentation Block
2. Demo
3. Q&A + Summary
4. Break

# Methodology Notes

1. Presentation Block

2. Demo

3. Q&A + Summary

4. Break

5. Recap

# Methodology Notes

1. Presentation Block

2. Demo

3. Q&A + Summary

4. Break

5. Recap

6. Repeat

# What You'll Learn Today

1

## Define & Distinguish

What vibe coding is and isn't

2

## 4 Core Skills

Prompting, Context, Delegation,  
Verification

3

## Practical Patterns

5 immediately usable workflows

4

## Case Study & Demo

Real-world vibe coding in action

# Part 1: What is Vibe Coding?



Andrej Karpathy

@karpathy



There's a new kind of coding I call "vibe coding", where you fully give in to the vibes, embrace exponentials, and forget that the code even exists. It's possible because the LLMs (e.g. Cursor Composer w Sonnet) are getting too good. Also I just talk to Composer with SuperWhisper so I barely even touch the keyboard. I ask for the dumbest things like "decrease the padding on the sidebar by half" because I'm too lazy to find it. I "Accept All" always, I don't read the diffs anymore. When I get error messages I just copy paste them in with no comment, usually that fixes it. The code grows beyond my usual comprehension, I'd have to really read through it for a while. Sometimes the LLMs can't fix a bug so I just work around it or ask for random changes until it goes away. It's not too bad for throwaway weekend projects, but still quite amusing. I'm building a project or webapp, but it's not really coding - I just see stuff, say stuff, run stuff, and copy paste stuff, and it mostly works.

11:17 PM · Feb 2, 2025 · 5M Views



1.3K



5.1K



30K



15K



[1] [Andrej Karpathy on Twitter, Feb 2025](#)

# The Critical Distinction

## Vibe Coding

"Building software with an LLM without reviewing the code it writes"

- *Simon Willison, 2025*



## AI-Assisted Programming

Review, understand, and take accountability

*Using AI as a typing assistant*

**Simon Willison:** "I'm concerned that people are applying the term 'vibe coding' to ALL forms of code written with AI assistance. This dilutes the term and gives a false impression of responsible AI-assisted programming."

<sup>[1]</sup>[Simon Willison, March 2025](#)

# When to Vibe Code

## Safe Zone

- ✓ Throwaway weekend projects
- ✓ Low-stakes experiments
- ✓ Learning and building intuition
- ✓ One-off data processing scripts
- ✓ Personal prototypes

## Danger Zone

- ✗ Production systems
- ✗ Security-critical applications
- ✗ Code others must maintain
- ✗ High-stakes business logic
- ✗ Financial/medical systems

# The Spectrum of AI-Augmented Development

# Trade-offs Spectrum

Vibe Coding

AI-Assisted

Full Engineering

**Speed:** Maximum

**Review:** None

**Control:** Low

**Risk:** High

**Speed:** High

**Review:** Strategic

**Control:** Shared

**Risk:** Managed

**Speed:** Moderate

**Review:** Comprehensive

**Control:** Full

**Risk:** Low

# Tool Landscape Overview

## Web

Claude Artifacts  
ChatGPT Canvas  
v0, Lovable

## IDE

Cursor  
GitHub Copilot  
Windsurf

## Terminal

Claude Code  
Aider  
OpenAI Codex

## Cloud

Replit  
CodeSandbox  
GitHub Codespaces

*See Appendix for detailed pricing and feature comparison*

# Whiteboard: The Spectrum in Practice

# Q&A

## Part 2: The 4 Core Skills

# Skills Overview

## 1. Prompting

Clear, specific instructions that get results

## 2. Context Management

Feeding the right information to LLMs

## 3. Capability Assignment

Knowing what to delegate and to which tool

## 4. Vibe Checking

Lightweight verification without over-engineering

# Skill 1: Prompting

The Foundation of Vibe Coding

# The 6 Prompting Sub-Skills

1

**Clear & Direct**

Specific, unambiguous

2

**Decomposition**

Break into steps

3

**Examples**

Show desired output

4

**Role Assignment**

Define expertise

5

**Time to Think**

Request reasoning

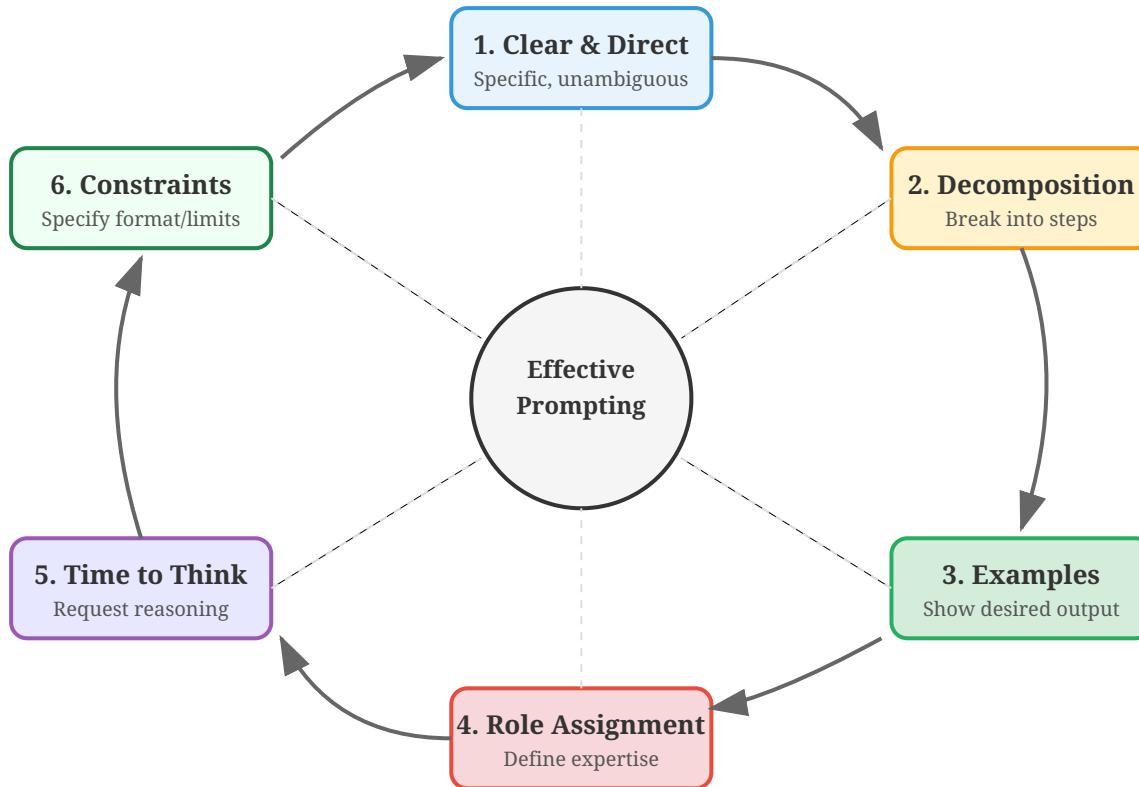
6

**Constraints**

Specify format/limits

[1] [Anthropic - AI Fluency](#)

# The Prompting Skills Cycle



Each skill reinforces the others in an iterative prompting workflow

# Bad vs Good Prompts

## Vague

*"Can you help me with my code?"*

**Problems:** No context, no task, no guidance

## Clear & Direct

*"Write a Python function that validates email addresses using regex, returns True/False, and includes error handling"*

**Why it works:** Language, output, requirements specified

# Decomposition: Break It Down

Instead of this massive prompt:

*"Build me a complete e-commerce website with user authentication, product catalog, shopping cart, payment processing, and admin dashboard"*

Break it into digestible steps:

1. **Step 1:** "Design the database schema for users, products, and orders"
2. **Step 2:** "Create user authentication endpoints"
3. **Step 3:** "Build product catalog with search"
4. **Step 4:** "Implement shopping cart logic"

# Examples: Show What You Want

## Without Examples

*"Write a function to format dates"*

**Result:** Unpredictable format, may not match your needs

## With Examples

*"Write a function to format dates like these examples:*

*Input: 2025-01-15 → Output: 'Jan 15, 2025'  
Input: 2025-12-25 → Output: 'Dec 25, 2025'"*

**Result:** Exact format you need

**Pro tip:** Include 2-3 examples covering edge cases (empty input, special characters, boundary conditions)

# Role Assignment: Define Expertise

## Generic Request

*"Review this SQL query for issues"*

**Result:** Surface-level feedback

## With Role Assignment

*"You are a senior database administrator with 15 years of experience optimizing PostgreSQL queries for high-traffic applications. Review this SQL query for performance issues and security vulnerabilities."*

**Result:** Expert-level analysis

**Effective roles:** "Senior engineer", "Security auditor", "Code reviewer", "Technical writer", "System architect"

# Time to Think: Request Reasoning

## Rushed Answer

*"Which sorting algorithm is fastest?"*

**Result:** Quick but potentially wrong answer

## With Thinking Time

*"Think step by step about which sorting algorithm would be fastest for my use case: 10,000 mostly-sorted integers. Consider time complexity, space constraints, and cache efficiency before recommending."*

**Result:** Reasoned recommendation

**Magic phrases:** "Think step by step", "Before answering, consider...", "Walk through your reasoning", "Explain your approach first"

# Constraints: Specify Format & Limits

## Open-Ended

*"Write a user authentication system"*

**Result:** Might get 500 lines of enterprise Java

## With Constraints

*"Write a user authentication function in Python."*

*Constraints:*

- Max 50 lines of code
- Use bcrypt for hashing
- Return a JWT token
- No external dependencies except bcrypt and PyJWT"

**Result:** Focused, usable code

**Useful constraints:** Language, line count, libraries, output format (JSON/XML), response length, style guidelines

# Real Vibe Coding Prompts

## For Codex (Autonomous Task):

*"Install playwright and use it to visit <https://opensauce.com/agenda/> and grab the full details of all three day schedules from the tabs - Friday and Saturday and Sunday - then save data in as much detail as possible in a JSON file and submit that as a PR"*

## For Claude (Iterative Building):

*"Build an artifact with no react that turns the schedule into a nice mobile friendly webpage - there are three days Friday, Saturday and Sunday. Don't copy the raw JSON over - fetch from this URL instead. Also include a button to download ICS at the top."*

<sup>[1]</sup>Simon Willison - Vibe scraping for Open Sauce 2025

# Skill 2: Context Management

Feeding the Right Information to LLMs

# The 4 Context Strategies

## Write Context

Save information outside the context window

*External memory, documentation files*

## Select Context

Pull relevant information when needed

*RAG systems, smart file selection*

## Compress Context

Retain only essential tokens

*Summaries, sliding windows*

## Isolate Context

Split context across different agents

*Multi-agent architectures*

<sup>[1]</sup>[LangChain Blog - Context Engineering for Agents](#)

# Strategy 1: Write Context

## Save Information Outside the Context Window

Store important context in external files that persist across conversations

### Examples

- CLAUDE.md / AGENTS.md files
- Project documentation
- API specifications
- Decision logs

### Benefits

- Survives context resets
- Shareable across sessions
- Version controllable
- Reduces re-explanation

**Pro tip:** Create a project-level markdown file with key decisions, conventions, and context that loads automatically

#### Example from my CLAUDE.md:

- My courses are stored in: ~/Desktop/projects/oreilly-live-trainings
- My snippets are stored in: ~/Library/Application Support/Cursor/User/snippets/

# Strategy 2: Select Context

## Pull Relevant Information When Needed

Dynamically retrieve context based on the current task or query

### Techniques

- **RAG:** Semantic search over docs
- **@-mentions:** Reference specific files
- **Smart indexing:** Codebase search
- **API calls:** Fetch live data

### When to Use

- Large codebases (can't load all)
- Dynamic documentation
- Task-specific references
- Multi-repo projects

**Key insight:** Don't dump everything—selectively load what's relevant to the current task

/context-prime project\_overview.md ← Example: pulls in just the core project doc, not all files

# Strategy 3: Compress Context

## Retain Only Essential Tokens

Summarize or condense information to maximize context window efficiency

### Compression Methods

- **Summaries:** Condense long conversations
- **Sliding windows:** Keep recent context
- **Key extraction:** Pull out essentials
- **Hierarchical:** Details on demand

### Practical Applications

- Long debugging sessions
- Multi-file refactoring
- Extended conversations
- Complex multi-step tasks

**Warning:** Compression loses detail—keep critical information uncompressed

**Example:** /create\_handoff compacts session context into a structured doc:

"Write a handoff document... compact and summarize your context without losing key details"

# Strategy 4: Isolate Context

## Split Context Across Different Agents

Divide work among specialized agents, each with focused context

### Multi-Agent Patterns

- **Specialist agents:** Frontend, backend, tests
- **Parallel workers:** Same task, different files
- **Orchestrator:** Coordinates sub-agents
- **Reviewer agent:** Validates outputs

### Benefits

- Each agent has full context window
- Parallel execution possible
- Reduces context pollution
- Specialized expertise per agent

**Trade-off:** More agents = more coordination overhead. Use when context is truly too large for one agent.

#### Example prompt:

"Spawn 3 agents: one for frontend components, one for API routes, one for tests. Each works in parallel on their own files."

# Gathering Context

## Repository to Context

[gitigest.com](#) - GitHub repo to LLM context

[repomix](#) - Pack repo into single file

[files-to-prompt](#) - CLI for file concatenation

## Specialized Extractors

[llms.txt](#) - LLM-ready documentation standard

[r.jina.ai](#) - Clean web content extraction

[arxiv-txt](#) - arXiv papers to text

# Context Anti-Patterns

## Context Rot Prevention

**Problem:** Long conversations accumulate outdated/conflicting information

**Solution:** Strategic context restart

1. Create summary of current state
2. Start fresh conversation
3. Load only relevant context

**Golden Rule: When in doubt, restart with clean context**

# Skill 3: Capability Assignment

Know What and How to Delegate

# Problem / Tool / Task Framework

## The Delegation Triangle

1

### Problem Awareness

What am I trying to solve?

2

### Platform Awareness

Which tool is best suited?

3

### Task Delegation

What do I keep vs delegate?

[1] [Anthropic - AI Fluency Delegation](#)

# Model Selection: Use Leaderboards

## Check Before Choosing:

- [LM Arena Leaderboard](#) - Community-driven LLM rankings
- [Artificial Analysis](#) - Up-to-date model comparisons

*What was "best" last month may not be best today!*

## Critical Limitations:

**Hallucination:** May invent APIs/libraries

**Security blind spots:** May introduce vulnerabilities

**Dependency risks:** May suggest outdated packages

**Context limits:** Long conversations lose coherence

# Demo: Prompting, Context & Capability Assessment

# Skill 4: Vibe Checking

Smart Verification Without Over-Engineering

# Green Flags vs Red Flags

## Green Flags (Trust It)

- Output quantities match expectations
- File sizes are reasonable
- Basic functionality tests pass
- Error-free execution
- Consistent formatting

## Red Flags (Investigate)

- Dramatic size discrepancies
- Incomplete processing
- False positives/negatives
- Context blindness
- Over-formal, repetitive patterns

# When Vibe Checks Are Enough

## Vibe Check Sufficient:

- Personal projects & prototypes
- Data processing scripts
- UI components & layouts
- Learning & experimentation
- Throwaway weekend projects

## Full Review Required:

- Production security code
- Financial calculations
- User data handling
- System integrations
- Performance-critical paths

*Vibe checking bridges the gap between reckless speed and paralyzed perfectionism*

## Demo: Practicing the 4 Core Skills

# Q&A & Break

# Part 3: Case Study

Simon's Open Sauce Schedule App

# The Setup

## Open Sauce 2025 Schedule App

**Constraint:** Working entirely on iPhone

**Tools Used:** OpenAI Codex + Claude Artifacts

---

### Process:

- Codex: Autonomous scraping (13 minutes runtime)
- Claude: Mobile-friendly UI generation
- GitHub: Mobile deployment

**Full app deployed in under 3 hours**

<sup>[1]</sup>[Simon Willison - Vibe scraping for Open Sauce 2025](#)

# The 130MB Bug: When Vibe Coding Fails

## The Problem

**Symptom:** Schedule app made 176 requests, downloaded 130MB

**Cause:** Unoptimized speaker avatar images

**Detection:** Simon's "vibe check" caught the issue

---

## The Fix

**Solution:** One Codex prompt to optimize images

**Result:** Reduced to 93.58 KB (1,400x smaller!)

# What This Case Study Demonstrates

## Skills in Action

- **Prompting:** Clear, specific
- **Context:** URL + format requirements
- **Capability:** Right tool for each task
- **Vibe Check:** Caught the 130MB issue

## Key Takeaways

- Constraints breed creativity
- Chain multiple tools together
- Verify before deployment
- Fix issues with more prompts

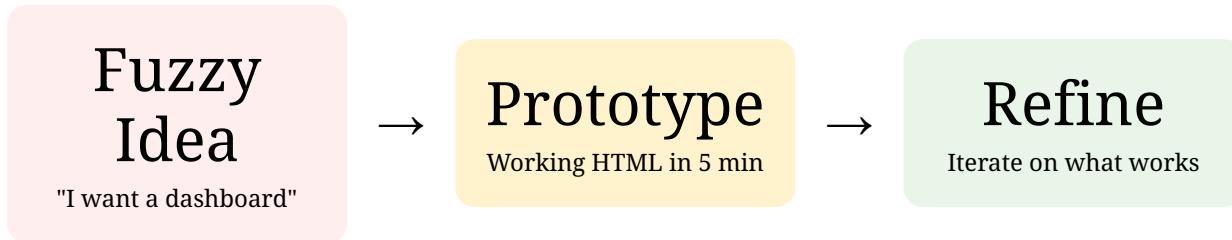
**Demo: Let's Vibe Code a Useful HTML/JS App**

# Q&A & Break

## Part 4: Practical Patterns

### 5 Workflows You Can Use Tomorrow

# Pattern 1: Start with Prototypes



**Pro tip:** Don't spend hours writing specs. Get a working prototype and iterate from there.

# Pattern 2: Let It See What You See

## For Errors

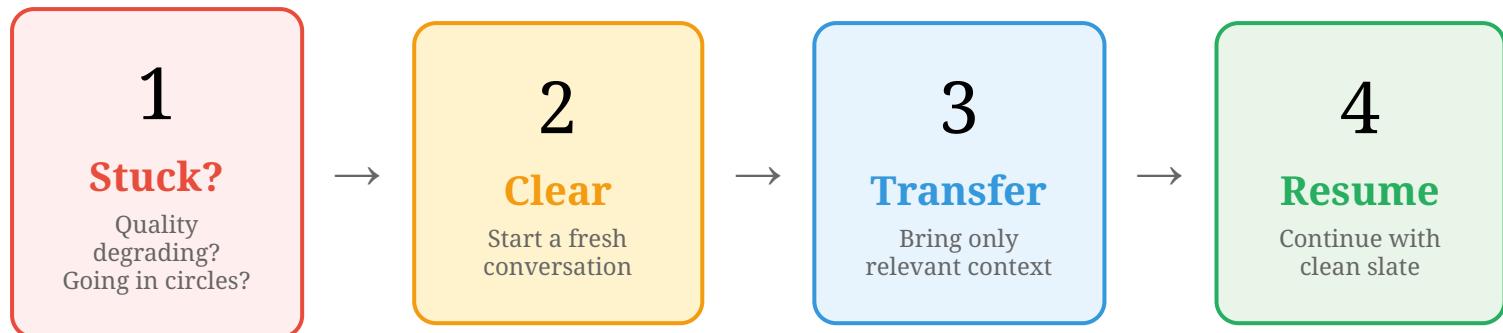
1. Copy the full error message
2. Include relevant code context
3. Ask LLM to debug

## For Features

1. Screenshot current state
2. Describe desired change
3. Let LLM generate code

**Key insight:** The more context you provide, the better the output. Share screenshots, error logs, and relevant code.

# Pattern 3: Restart When Stuck



**Rule of thumb:** If you've been stuck for 10+ minutes, restart with a summary of your current state

# Pattern 4: Automate Repeatable Steps

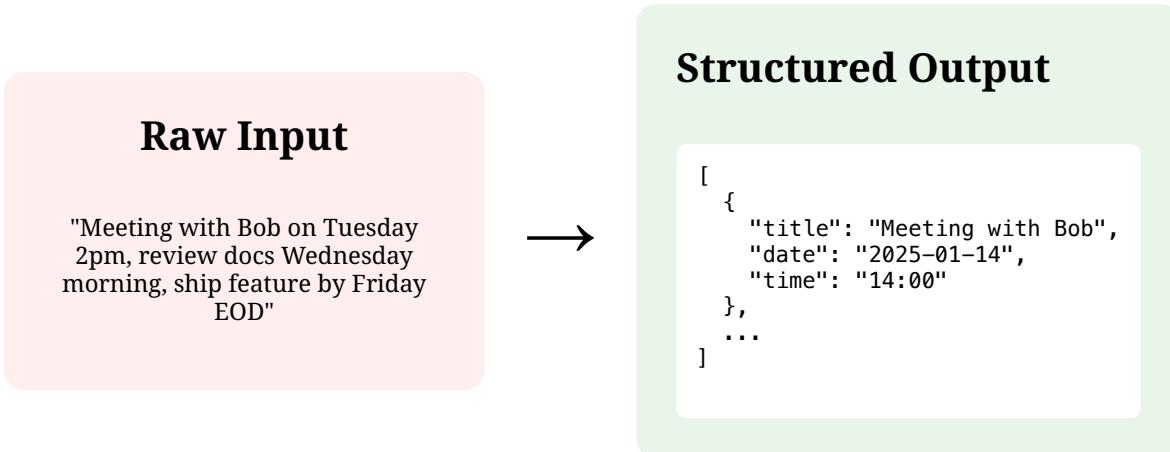
## The Workflow:

1. Describe the automation you need
2. Ask for a single Python script with uv metadata
3. Copy & paste into terminal
4. Run with `uv run script.py`

```
# /// script
# requires-python = ">=3.11"
# dependencies = ["requests", "beautifulsoup4"]
# ///
import requests
from bs4 import BeautifulSoup

# Your automation code here
```

# Pattern 5: Raw → Structured



**Use cases:** Meeting notes → calendar events, requirements → user stories, ideas → task lists

## Demo: Patterns in Action

# Q&A & Break

# Part 5: Practice & Guardrails

# Don't Let Your Brain Rot

## Research Findings

**Negative correlation** between frequent AI tool usage and critical thinking abilities

**Metacognitive laziness:** Diminished self-regulatory processes

## The 70-30 Rule

### 70% AI-Assisted

Routine tasks, boilerplate,  
research

### 30% Independent

Core skills, critical thinking

<sup>[1]</sup>[MIT Media Lab - Your Brain on ChatGPT](#)

# Practice Plan

## Daily Habits

Use AI for every coding task

Practice specific prompts

Know model training cut-offs

Vibe check immediately, always

## Weekly Habits

One pure vibe coding experiment

One AI-free coding session

Explore new AI tools/models

Document what works for you

# Success Metrics

## Speed

Can you ship projects you wouldn't have started before?

## Confidence

Do you trust your vibe checks and AI collaboration?

## Discernment

Can you distinguish appropriate vs inappropriate AI use?

## Accountability

How much do you understand and own your code?

# Where This Is Going: Orchestration

## The Paradigm Shift

### Today

Human + single AI conversations

### Tomorrow

Orchestrating multiple AI agents

*From writing code to coordinating intelligent agent clusters*

*See Appendix for deep dive on agentic orchestration*

# Key Takeaways

- 1 **Vibe coding ≠ AI-assisted programming** - Know when each is appropriate
- 2 **Master the 4 core skills** - Prompting, Context, Capability, Vibe Checking
- 3 **Use patterns immediately** - Prototypes, context sharing, restarts, automation
- 4 **Maintain your skills** - 70-30 rule, weekly AI-free practice

# Connect With Me

Course materials

LinkedIn

Twitter/X - @LucasEnkrateia

YouTube - @automatalearninglab

Email: lucasenkrateia@gmail.com

Keep Vibing, Keep Building

# Appendix

## Backup & Reference Slides

# Appendix A: Tool Landscape (Detailed)

# Tools Landscape: Web Builders

## Claude (artifacts)

**Price:** \$20/month

Safe sandbox prototyping, iterative building

## ChatGPT (canvas)

**Price:** \$20/month

Code generation with editing interface

## Vercel v0

**Price:** \$20/month

React + production deployment

## Lovable

**Price:** \$25/month

Full-stack app generation

# Tools Landscape: IDE Integration

## Cursor

**Price:** \$20/month

Full codebase understanding, AI-native IDE

## GitHub Copilot

**Price:** \$10/month

Code completion workflows

## Codium Windsurf

**Price:** \$50 - \$250/month

Enterprise-grade AI coding

# Tools Landscape: Terminal & Cloud

## Terminal Agents

**Claude Code** - Max Plan: \$17 - \$200/month  
**Aider** - API costs only  
**OpenAI Codex** - Pay per use

## Cloud Environments

**Replit** - \$7-25/month  
**CodeSandbox** - Free-\$20/month  
**GitHub Codespaces** - Pay per use

# Appendix B: Strategic Cognitive Offloading

# What to Offload vs What to Keep

## Tasks for Offloading

- Boilerplate code generation
- Documentation lookup and research
- Code formatting and style consistency
- Writing unit test scaffolding
- Initial prototyping and iteration

## Keep Human-Controlled

- System architecture decisions
- Complex debugging and root cause analysis
- Code review and quality assessment
- Learning new concepts and technologies
- Critical thinking about trade-offs

# Good vs Bad Offloading Examples

## Good Offloading

**Test Scaffolding:** AI generates test structure, you implement test logic

**Architecture Design:** AI explains approaches, you choose and implement

**Pair Programming:** AI accelerates iteration while you maintain oversight

## Poor Offloading

**Blind Copying:** Copy-pasting AI solutions without understanding

**Debug Dependency:** Using AI to debug every error without learning

**Architecture Delegation:** Having AI make all design decisions

# Appendix C: Personal Benchmarking

# The Productivity Paradox

## Research Finding

Experienced developers may be **19% slower** with AI tools

Despite **feeling 20% more productive**

---

**Solution:** Track both how you feel AND actual performance metrics

# Multi-Dimensional Metrics Framework

## Velocity Metrics

- Time to complete tasks
- Feature delivery speed
- Build/test cycle duration
- Iteration frequency

## Quality Metrics

- Bug rates per lines of code
- Code review outcomes
- Post-deployment incidents
- Technical debt accumulation

## Collaboration Metrics

- Code sharing efficiency
- Knowledge transfer rates
- Pull request sizes
- Team integration ease

## Satisfaction Metrics

- Developer confidence levels
- Comfort with AI tools
- Learning curve progress
- Burnout indicators

# Simple Benchmarking Approach

**Start Simple**

**Pick 2 metrics:**

1. **Time-to-complete:** How long to finish similar tasks?
2. **Bug rate:** How many issues found after "done"?

**Track them for 4 weeks** - compare AI-assisted vs independent

# Appendix D: Agentic Task Orchestration

# What is Agentic Orchestration?

## Traditional AI-Assisted

- Human-guided conversations
- Direct control over AI
- Single agent, immediate feedback



## Agentic Orchestration

- Autonomous agent coordination
- Strategic task delegation
- Multi-agent, parallel execution

# The Orchestrator-Worker Pattern

## Lead Agent (Orchestrator)

- **Analyzes** complex tasks and develops strategies
- **Breaks down** work into parallelizable subtasks
- **Spawns** specialized subagents for different aspects
- **Synthesizes** results from multiple workers
- **Maintains** overall project coherence

# When Multi-Agent vs Single-Agent

## Multi-Agent Orchestration

- **Complex, parallelizable tasks:** Multiple independent components
- **Large-scale refactoring:** Changes across many files
- **Research-intensive projects:** Multiple perspectives needed
- **High-value tasks:** Improved performance justifies costs

## Single-Agent Better For

- **Tightly coupled tasks:** Agents need shared context
- **Simple, linear workflows:** Sequential without parallelization
- **Real-time coordination:** Immediate agent communication needed
- **Low complexity:** Overhead not justified

# Claude Code for Orchestrating Agents

## Core Philosophy:

**Intentionally low-level and unopinionated:** Near-raw model access without forced workflows

**Flexible and customizable:** Developers create tailored orchestration patterns

## Advanced Capabilities:

**Parallel development:** Frontend and backend simultaneously using Git worktrees

**Multi-agent problem solving:** Different agents approaching problems from multiple angles

**Autonomous execution:** for uninterrupted work

# Security Considerations

## Important Safeguards:

**Prompt injection awareness:** Be cautious with external context from GitHub issues

**Code review requirements:** Even with autonomous agents, maintain quality gates

**Permission management:** Use permission flags appropriately based on task sensitivity

## LLM-as-Judge Evaluation:

Implement automated quality assessment for agent outputs

Use AI to evaluate AI-generated solutions before deployment

# Appendix E: Additional Patterns

# Pattern: Build Personal Benchmarks

## The Workflow:

1. LLM produces wrong output
2. Save the example + prompt in a table
3. Re-test later with new models or prompts
4. Track improvement over time

# Pattern: Store Formatting Prompts

## The Workflow:

1. Create a prompt to transform X to Y
2. Store it in a snippet/shortcut system
3. Re-use with a keyboard shortcut

**Examples:** Meeting notes → action items, raw data → JSON, requirements → user stories

# Pattern: Generate Synthetic Data

## The Workflow:

1. You have complex data analysis to do
2. LLM outputs sample data that looks like your real data
3. Test your analysis with sample data
4. Refine until it works
5. Implement with real data

# Pattern: Voice Workflows

## When to Use:

- Initial long instructions that would take time to type
- Brainstorming sessions where you think out loud
- Mobile-first workflows (like Simon's Open Sauce app)

**Tools:** ChatGPT Voice, Claude Voice, whisper.cpp for transcription

# Pattern: Context-Based Coding

## The Problem:

LLMs have training cut-offs and may suggest outdated APIs

## The Solution:

1. Find official up-to-date documentation
2. Load it into context (llms.txt, copy-paste, RAG)
3. Ask LLM to write code using that context

# Appendix F: Cross-Domain Thinking

# Vibe Coding for Thought

## The Meta-Skill

Swap "apps" for "ideas" and "prototypes" for "approximative answers"

---

**Example:** "Apply a concept from biology to team management"

*Use AI as an analogical engine for transductive thinking*

This isn't just about coding—it's about **responsible AI augmentation** across all problem-solving domains

# End of Appendix

## Questions?