Vibe Coding for Problem Solvers

Translating intent to LLMs

By Lucas Soares

08/05/2025

About Me

AI Engineer & Instructor

Passionate about AI-augmented development

Building tools and teaching workflows

Focus on practical AI integration

What You'll Learn

🎯 Core Concepts

Define vibe coding and understand when to use it

🧠 Vibe Coding Core Skills

Practice translating ideas into AI instructions

🛠️ Tools & Patterns

Learn effective workflows and techniques

🚀 Real Practice

Hands-on exercises and case studies

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

⚠️ CRITICAL DISTINCTION

Vibe Coding ≠ AI-Assisted Programming

Simon Willison (March 2025):

"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."

Vibe Coding x AI-Assisted Programming

🧪 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

When Vibe Coding is Appropriate

✅ Safe Vibe Coding Zone

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

❌ NOT for Production

- ✗ Complex, long-term systems
- ✗ Security-critical applications
- ✗ Code that others must maintain
- ✗ High-stakes business logic

The Middle Ground

Whiteboard - The Spectrum of AI-Augmented Development - Useful Definitions & Tools

Trade-offs and Characteristics

Vibe Coding

↓ Less systematicity
↓ Less precision
↓ Less review
↓ Less understanding
↓ Less control
↑ **Maximum speed**

AI-Assisted Programming

- ○ Balanced systematicity
- ○ Balanced precision
- ○ Strategic review
- ○ Guided understanding
- ○ Shared control
- ○ **True augmentation**

Software Engineering

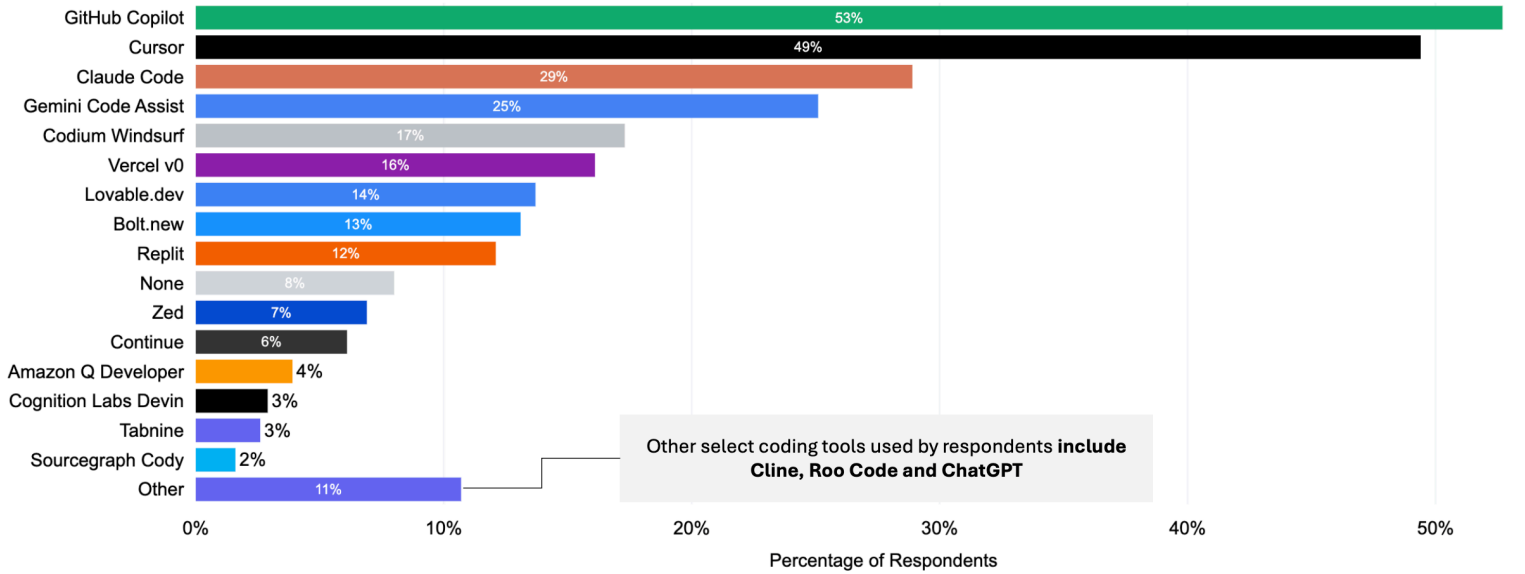
- ↑ More systematicity
- ↑ More precision
- ↑ More review
- ↑ More understanding
- ↑ More control
- ↓ **Slower process**

The Vibe Coding Ecosystem

AI Coding Tools: GitHub Copilot and Cursor dominate the market as the most popular AI coding tools, with a significant lead over Claude Code and Gemini Code Assist

Demand for Coding Tools

Which AI tools are you using or considering using this year? N=955



Source: Artificial Analysis AI Adoption Survey – H1 2025

Artificial Analysis

Tool Categories by Use Case

🎨 Web Builders

Claude (artifacts) - $20/month

ChatGPT (canvas) - $20/month

Gemini (canvas) - $20/month
Safe sandbox prototyping

Vercel v0 - $20/month

Lovable - $25/month
React + production deployment

💻 IDE Integration

Cursor - $20/month
Full codebase understanding

GitHub Copilot - $10/month
Code completion workflows

Codium Windsurf - $50 - $250/month

Tool Landscape (Continued)

⚡ Terminal Agents

Claude Code - Max Plan: $17 - $200/month
Mobile coding, file access

Aider - API costs only
Git integration, autonomous

☁️ Cloud Environments

Replit - $7-25/month
Full dev environment

CodeSandbox - Free-$20/month
Instant web development

7 Skills → 12 Concrete Patterns

Vibe Coding Skills

✅ Basic

1. **Prompting**
2. **Context Management**
3. **Capability Assigment**
4. **Vibe Checking**

🆕 Advanced

5. **Strategic Cognitive Offloading**
6. **Personal Benchmarking**
7. **Agentic Task Orchestration**

Demo: Exemplifying the 7 Skills

Q&A & Break

Skills 1-4: The Foundation

1️⃣ Prompting

Bad: "How do I market my app?"
Good: "List 3 user personas for my app and suggest a marketing channel for each."

2️⃣ Context Management

30-page report + [prompt for dashboard app] + [language docs] → dashboard app

3️⃣ Capability Assignment

"Let the LLM write the code, but I'll review and test it myself."

4️⃣ Vibe Checking

"Does this answer feel right? If not, ask for clarification or a different approach."

Skills 5-7: Advanced Techniques

5️⃣ Strategic Cognitive Offloading

"Summarize these meeting notes while I focus on the next task."

6️⃣ Personal Benchmarking

"How does my solution compare to best practices or previous attempts?"

7️⃣ Agentic Task Orchestration

"Break this project into subtasks and assign each to the right AI tool or agent."

Deep Dive into the 7 Vibe Coding Skills

Skill 1: Prompting

The Foundation of Vibe Coding

🎯 The Core Insight

"The quality of your prompts directly determines the quality of your code"

Better prompts = Better vibes = Better results

The 6 Prompting Sub-Skills

1 Clear & Direct

Specific, unambiguous instructions

2 Decomposition

Break complex tasks into steps

3 Examples

Show desired output format

4 Role Assignment

Define AI's expertise context

5 Time to Think

Request reasoning process

6 Output Constraints

Specify format and limits

Bad vs Good: Clear & Direct Prompts

❌ Vague & Indirect

"Can you help me with my code?"

Problems: No context, no specific 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: Specific language, clear output, defined requirements

Decomposition: Breaking Down Complex Tasks

🧩 Instead of one massive prompt...

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

Decomposition: Breaking Down Complex Tasks

🧩 Instead of one 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 functionality"
4. **Step 4:** "Implement shopping cart logic"

Examples: Show, Don't Just Tell

💡 The Power of Examples

Before: "Write a React component"

Examples: Show, Don't Just Tell

💡 The Power of Examples

Before: "Write a React component"

After: "Write a React component like this structure:"

```
const MyComponent = ({ data }) => {
  return (


      {data.map(item => (
        {item.name}
      ))}


  );
};
```

Role Assignment: Context is King

🎭 Set the Stage

"Act as a senior Python developer with 10 years of experience in web APIs. Review this code and suggest improvements..."

🏗️ Domain Expertise

"You're a database architect. Design an efficient schema for a social media platform with users, posts, and relationships..."

🎯 Why This Works

Role assignment tells the AI which subset of knowledge to draw from, resulting in more specialized and accurate responses

Time to Think: Chain of Thought

🧠 Use Reasoning Models for Complex Tasks

Technique: "Before writing the code, walk me through your approach..."

Or: "Think step by step about how to solve this problem..."

Or: "Let's break this down: What are the main components we need?"

Output Constraints: Be Specific About Format

📏 Length & Structure

"Write a Python function that is max 15 lines, uses type hints, and follows PEP 8"

"Generate exactly 3 alternative solutions"

🎨 Style & Format

"Return only the code, no explanations"

"Include detailed comments for each step"

"Format as a markdown code block"

🎯 Vibe Coding Power Move

Constraint + Context: "Generate a complete HTML page with inline CSS and JavaScript. No external dependencies. Mobile-first responsive design. File should be under 500 lines total."

Real Vibe Coding Prompts in Action

📱 Simon Willison's Open Sauce App Prompts

For OpenAI 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"

Real Vibe Coding Prompts in Action

📱 Simon Willison's Open Sauce App Prompts

For OpenAI 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."

The Prompting Hierarchy for Vibe Coding

🏆 **Expert Level: Combine All 6 Techniques**

🎯 **Intermediate: Use 3-4 Techniques per Prompt**

🚀 **Beginner: Start with Clear & Direct + Examples**

The Prompting Hierarchy for Vibe Coding

🏆 **Expert Level: Combine All 6 Techniques**

🎯 **Intermediate: Use 3-4 Techniques per Prompt**

🚀 **Beginner: Start with Clear & Direct + Examples**

💡 Pro Tip

If your first prompt doesn't work: Ask the AI to help improve your prompt! Meta-prompting is a legitimate strategy.

"The previous response wasn't quite what I needed. Help me refine my prompt to get better results."

Skill 2: Context Management

Feeding Context to LLMs

🧠 Context Engineering

Filling the context window with **just the right information**

The quality of output is proportional to context quality

The 4 Core Context Strategies

📝 Write Context

Save information outside the context window

Example: External memory systems, documentation files

🎯 Select Context

Pull relevant information when needed

Example: RAG systems, smart file selection

🗜️ Compress Context

Retain only essential tokens through summarization

Example: Checkpoint summaries, sliding windows

🏗️ Isolate Context

Split context across different agents

Example: Multi-agent architectures, specialized contexts

LangChain Blog Post - Context Engineering

LLMs.txt - Input to LLM Ready

```
# Title

> Optional description goes here

Optional details go here

## Section name

- [Link title](https://link_url): Optional link details

## Optional

- [Link title](https://link_url)
```

Source: llms-txt

📄 Repository to Context

gitingest.com - Turn any GitHub repo into LLM-ready context

repomix - Pack entire repository contents into a single file

files-to-prompt - CLI tool to concatenate files for LLM prompts

Repo Prompt - Generate prompts from repository structure

🔍 Specialized Extractors

gitdiagram.com - Visualize and extract repo architecture

r.jina.ai - Reader API for clean web content extraction

arxiv-txt - Convert arXiv papers to clean text

Vibe Coding Context Patterns

✅ Effective Context Loading:

Error messages + logs - Full error traces when debugging

API documentation - Relevant sections for current task

Test outputs - Results when tests fail

Conversation history - Key decisions and context

❌ Context Pollution:

Including irrelevant files or outdated information

Letting context "rot" with conflicting information

Overwhelming context window with noise

Memory Management Strategies

🔄 Sliding Window Technique

Process text in overlapping segments:

```
Segment 1: tokens 1-1000
Segment 2: tokens 501-1500
Segment 3: tokens 1001-2000
```

📋 Checkpoint Summaries

Regular summaries of current state:

"What's been accomplished + current state"

Use as seeds for fresh context in new sessions

Context Refresh Anti-Pattern

⚠️ 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

Strategically dividing work between **human and AI**

Problem Awareness + Platform Awareness + Task Delegation

[Anthropic - AI Fluency Delegation]

Model Selection Framework

📅 **Knowledge Cutoffs (2024-2025):**

GPT-4.1:	**Claude 4 Sonnet:**	**Gemini 2.5 Pro:**
June 2024	January 2025	January 2025

Model Selection Framework

📅 **Knowledge Cutoffs (2024-2025):**

GPT-4.1:	Claude 4 Sonnet:	Gemini 2.5 Pro:
June 2024	January 2025	January 2025

What Model is Good For What?

🏆 **Use Leaderboards to Guide Model Choice**

- LM Arena Leaderboard – Community-driven, multi-metric LLM rankings
- Artificial Analysis – Up-to-date model comparisons and benchmarks
- LM Arena Leaderboard – Community-driven, multi-metric LLM rankings

Check these resources before choosing a model for your task—what was "best" last month may not be best today!

Understanding Tool Limitations

⚠️ **Critical Limitations to Remember:**

Hallucination: May generate non-existent APIs or libraries

Security blind spots: May introduce vulnerabilities

Dependency risks: May suggest outdated or compromised packages

Context limits: Long conversations lose coherence

Understanding Tool Limitations

⚠️ **Critical Limitations to Remember:**

Hallucination: May generate non-existent APIs or libraries

Security blind spots: May introduce vulnerabilities

Dependency risks: May suggest outdated or compromised packages

Context limits: Long conversations lose coherence

💡 **Key Insight**

Orchestrate the right combination for each task, context, and risk level, and build intuition for when to delegate and when to retain control.

Skill 4: Vibe Checking

Smart Verification Without Over-Engineering

🔍 The Core Concept

Lightweight verification that's **simpler than the original task**

Validate AI outputs without reverting to full code reviews

What is Vibe Checking?

Traditional QA

Comprehensive testing and code review

Too slow for vibe coding

≠

Vibe Checking

Strategic verification points

Speed + confidence balance

What is Vibe Checking?

Traditional QA

Comprehensive testing and code review

Too slow for vibe coding

≠

Vibe Checking

Strategic verification points

Speed + confidence balance

🎯 **The Sweet Spot**

Catch major issues while preserving development velocity

Green Flags vs. Red Flags

✅ Green Flags (Good Signs)

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

❌ Red Flags (Warning Signs)

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

When Vibe Checks Aren't Enough

✅ **Vibe Checks Work For:**

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

❌ **Full Review Required:**

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

Building Your Vibe Check Intuition

📈 **Developing Your "Spidey Sense":**

> **Practice daily:** Run vibe checks on every AI-generated solution
>
> **Track patterns:** Note what types of issues you catch most often
>
> **False alarms:** Learn when your instincts are overcautious
>
> **Near misses:** Document issues that slipped through

Building Your Vibe Check Intuition

📈 **Developing Your "Spidey Sense":**

Practice daily: Run vibe checks on every AI-generated solution

Track patterns: Note what types of issues you catch most often

False alarms: Learn when your instincts are overcautious

Near misses: Document issues that slipped through

💡 **Key Insight**

Vibe checking bridges the gap between reckless speed and paralyzed perfectionism

Demo - Practicing Basic Vibe Coding Skills

Q&A & Break

Skill 5: Strategic Cognitive Offloading

Smart Mental Load Management

🧠 The Core Concept

Thoughtful delegation of cognitive tasks to AI while **maintaining critical thinking**

Reduce mental effort on routine tasks, amplify focus on high-value work

Strategic vs. Non-Strategic Offloading

✅ **Strategic (Beneficial)**

- Using AI for boilerplate and prototyping
- Leveraging AI for information retrieval
- Maintaining regular "AI-free" practice
- Critical evaluation of AI outputs

≠

❌ **Non-Strategic (Problematic)**

- Complete problem-solving delegation
- Over-reliance for debugging
- Accepting code without comprehension
- Using AI as learning replacement

The 70-30 Rule

Cognitive Load Distribution

70% AI-Assisted

Routine tasks, boilerplate, research

30% Independent

Core skills, critical thinking, design

The 70-30 Rule

📊 Cognitive Load Distribution

70% AI-Assisted

Routine tasks, boilerplate, research

30% Independent

Core skills, critical thinking, design

💡 **Key Insight**

Reserve 30% of coding time for independent work to maintain skills

What to Offload vs. What to Keep

📤 Tasks Suitable 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

The Risks: Cognitive Skill Atrophy

⚠️ Research Findings

Significant negative correlation between frequent AI tool usage and critical thinking abilities

Metacognitive laziness: Diminished self-regulatory processes

Source: Your Brain on ChatGPT - MIT Media Lab

Benefits of Strategic Offloading

💪 **Productivity Enhancement**

Reduces time on repetitive tasks, allows focus on high-level architecture

Benefits of Strategic Offloading

💪 **Productivity Enhancement**

Reduces time on repetitive tasks, allows focus on high-level architecture

🧘 **Mental Health**

Reduces cognitive overload and burnout by automating grunt work

Benefits of Strategic Offloading

💪 **Productivity Enhancement**

Reduces time on repetitive tasks, allows focus on high-level architecture

🧘 **Mental Health**

Reduces cognitive overload and burnout by automating grunt work

🎯 **Context Preservation**

Eliminates time spent searching documentation and understanding legacy code

Benefits of Strategic Offloading

💪 **Productivity Enhancement**

Reduces time on repetitive tasks, allows focus on high-level architecture

🧘 **Mental Health**

Reduces cognitive overload and burnout by automating grunt work

🎯 **Context Preservation**

Eliminates time spent searching documentation and understanding legacy code

⚡ **Accelerated Development**

Ideas that once took days can be prototyped in hours

Best Practices for Strategic Offloading

🎓 **Use AI for Learning:**

Ask follow-up questions to understand the reasoning behind AI suggestions

Approach: "Why did you choose this approach?" or "What are the trade-offs?"

Best Practices for Strategic Offloading

🎓 **Use AI for Learning:**

Ask follow-up questions to understand the reasoning behind AI suggestions

Approach: "Why did you choose this approach?" or "What are the trade-offs?"

🏋️ **Practice Deliberate Engagement:**

Regularly solve problems without AI to maintain core skills

Schedule: Weekly "AI-free" coding sessions

Practical Implementation Examples

✅ **Good Strategic Offloading:**

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

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

Pair Programming: AI accelerates iteration while you maintain oversight

Practical Implementation Examples

✅ **Good Strategic Offloading:**

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

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

Pair Programming: AI accelerates iteration while you maintain oversight

❌ **Poor Offloading Examples:**

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

Building Sustainable AI Habits

Daily Practices

- Start projects with AI assistance
- Review and understand all generated code
- Ask "why" to learn from AI
- Maintain explanation ability

Weekly Practices

- Solve one problem without AI
- Review AI collaboration patterns
- Practice core skills independently
- Track progress

Building Sustainable AI Habits

📅 Daily Practices

- Start projects with AI assistance
- Review and understand all generated code
- Ask "why" to learn from AI
- Maintain explanation ability

📅 Weekly Practices

- Solve one problem without AI
- Review AI collaboration patterns
- Practice core skills independently
- Track progress

💡 Key Insight

Strategic cognitive offloading is about amplifying human intelligence, not replacing it

Skill 6: Personal Benchmarking

Measuring Your AI Collaboration Effectiveness

The Core Concept

Creating **custom evaluation frameworks** to track AI collaboration effectiveness

Move beyond standardized benchmarks to measure your specific use cases

Why Personal Benchmarking Matters

⚠️ **The Productivity Paradox:**

> Recent research shows experienced developers may actually be **19% slower** with AI tools
>
> Despite feeling **20% more productive**

Why Personal Benchmarking Matters

⚠️ **The Productivity Paradox:**

Recent research shows experienced developers may actually be **19% slower** with AI tools

Despite feeling **20% more productive**

💡 **The Solution:**

Objective measurement alongside subjective assessments

Track both *how you feel* and *actual performance metrics*

The 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

The Long-Term Goal: Skill Development Trajectory

🎯 Ultimate 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?

💡 **Key Insight**

Personal benchmarking reveals the difference between feeling productive and being productive

Skill 7: Agentic Task Orchestration

Coordinating AI Agent Clusters

🤖 The Core Concept

Coordinating **multiple AI agents** working together autonomously

From individual coding to AI orchestration as a core developer skill

What is Agentic Task 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

What is Agentic Task 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 Shift**

From writing code to orchestrating intelligent agent clusters

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

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

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

When to Use 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

Best Practices for Agent Orchestration

📋 **Explicit Delegation Instructions:**

- Provide clear objectives, output formats, and task boundaries
- Specify which tools and sources agents should use
- Define clear boundaries to prevent overlap and gaps

Best Practices for Agent Orchestration

📋 **Explicit Delegation Instructions:**

- Provide clear objectives, output formats, and task boundaries
- Specify which tools and sources agents should use
- Define clear boundaries to prevent overlap and gaps

🔬 **Research-First Approach:**

- Have agents research and plan before implementation
- Significantly improves performance for complex problems
- Prevents premature optimization and ensures comprehensive solutions

Security and Quality 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

Security and Quality 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

The Paradigm Shift

Traditional Developer

Writes individual lines of code

Manages single context

Linear problem-solving

→

Orchestration Developer

Orchestrates intelligent agent clusters

Manages multiple contexts

Parallel system coordination

The Paradigm Shift

Traditional Developer

Writes individual lines of code

Manages single context

Linear problem-solving

→

Orchestration Developer

Orchestrates intelligent agent clusters

Manages multiple contexts

Parallel system coordination

💡 **The Transition**

Similar to moving from individual contributor to engineering management

New skills: coordination, strategy, and system design

Building Orchestration Skills

📚 Learn to Design

- Effective agent architectures
- Clear delegation strategies
- Robust quality control systems
- Complex multi-agent workflows

🏋️ Practice Daily

- Start with simple two-agent workflows
- Experiment with Claude Code's orchestration
- Build custom workflow templates
- Scale complexity gradually

Safety First: The Sandbox Principle

🔒 What's a Sandbox?

Safe execution of untrusted code in isolated environments

Examples: Claude Artifacts, ChatGPT Canvas/Code Interpreter

Learn more about sandboxes →

The Core Insight: Intent Translation

🧠 Mental Model Shift

Vibe coding = **translating human intent**

Into AI-executable instructions

The Core Insight: Intent Translation

🧠 Mental Model Shift

Vibe coding = **translating human intent**

Into AI-executable instructions

Vague Intent

"Can you help me with my code?"

→

Refined Request

"Write a function that reverses a string."

→

Actionable Output

```
def reverse_string(s):
    return s[::-1]
```

Case Study

🎯 Building a Simple HTML/JS App Entirely on Mobile

Simon scraped a conference website and built a mobile-friendly schedule app working entirely on his iPhone.

Vibe scraping and vibe coding a schedule app for Open Sauce 2025 entirely on my phone

📱 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

Outcome: Full app deployed in under 3 hours

Vibe scraping and vibe coding a schedule app for Open Sauce 2025 entirely on my phone

⚠️ Critical Lesson: When Vibe Coding Fails

🚨 The 130MB Bug

Problem: Schedule app made 176 requests, downloaded 130MB

Cause: Unoptimized speaker avatar images

Simon's Response: "Vibe check" caught the issue

Lesson: Even low-stakes projects need verification

Fix: One Codex prompt reduced to 93.58 KB (1,400x smaller!)

Vibe scraping and vibe coding a schedule app for Open Sauce 2025 entirely on my phone

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

Q&A & Break

12 Patterns for Vibing with LLMs

🎯 Start with Prototypes

Fuzzy Instruction - Prototype - Refine

🧪 Vibe checking checklist

LLM Output - Check - custom checklist (works? valid?...)

📝 Provide Clear Specs

Prompt for Spec Doc - Reasoning LLM Feedback - Refined Spec Doc

🔄 Restart When Stuck

Stuck or low quality output? - clear context - transfer only relevant context - prompt again

🖥️ Let It See What You See

error - Paste code + error - ask LLM to debug

Screenshot - ask LLM for feature - refine

🔁 Automate Repeatable Steps

prompt for automation - llm output - refine as single python script with uv metadata - copy & paste in terminal

📈 Build Personal Benchmark

LLM produces wrong output - save example + prompt in table - re-test later with new

🔗 Store Formatting Prompts

prompt to transform X to Y - store - re-use with shortcut

📊 Generate Synthetic Data

complex data analysis - llm outputs sample data - test with sample data - refine - implement in real data

🗣️ Voice Workflows

- Use voice workflows to communicate initial long instructions

🛠️ Context-Based Coding

- Write code using the context from official up to date docs

🗣️ Raw to Structured

Raw input (e.g tasks for the week) - LLM - structured events for calendar

Demo: Exemplifying Patterns for Vibe Coding

Q&A & Break

Prompting Modes

🔍 Exploration Mode

"What are the ways to do X?"

For research and options to get to a spec doc.

👑 Execution Mode

"Write this function, add tests, refactor"

To prototype a solution.

🎯 Real World Examples - Execution Mode

For Codex (One-shot tasks):

> *"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."*

🐌 Traditional Approach

- Manual research and documentation reading
- Writing every line from scratch
- Linear problem-solving processes
- Limited to personal knowledge

🚀 AI-Augmented Approach

- Instant exploration of options
- Rapid prototyping and iteration
- Parallel processing of multiple solutions
- Access to vast knowledge synthesis

Demo - Building Personal Workflows

Q&A & Break

Practice Plan

Daily AI Habits

✓ Use AI for every coding task

✓ Practice extremely specific prompts

✓ Know your model's training cut-off

✓ Test/Vibe check immediately, always

Weekly Skill Building

✓ Try one pure vibe coding experiment

✓ Practice vibe checking workflows

✓ Explore new AI tools/models

✓ Document what works for you

Critical Success Metrics (Updated):

Critical Success Metrics (Updated):

- **Speed**: Can you ship projects you wouldn't have started before?

Critical Success Metrics (Updated):

- **Speed**: Can you ship projects you wouldn't have started before?
- **Confidence**: Do you trust your vibe checks?

Critical Success Metrics (Updated):

- **Speed**: Can you ship projects you wouldn't have started before?
- **Confidence**: Do you trust your vibe checks?
- **Discernment**: Can you distinguish appropriate vs inappropriate vibe coding scenarios?

Critical Success Metrics (Updated):

- **Speed**: Can you ship projects you wouldn't have started before?
- **Confidence**: Do you trust your vibe checks?
- **Discernment**: Can you distinguish appropriate vs inappropriate vibe coding scenarios?
- **Accountability**: How much do you understand and own your code?

Critical Success Metrics (Updated):

- **Speed**: Can you ship projects you wouldn't have started before?
- **Confidence**: Do you trust your vibe checks?
- **Discernment**: Can you distinguish appropriate vs inappropriate vibe coding scenarios?
- **Accountability**: How much do you understand and own your code?

Key Setup Principles:

1. **Start with safer tools** (Claude Artifacts, ChatGPT Code Interpreter)

Critical Success Metrics (Updated):

- **Speed**: Can you ship projects you wouldn't have started before?
- **Confidence**: Do you trust your vibe checks?
- **Discernment**: Can you distinguish appropriate vs inappropriate vibe coding scenarios?
- **Accountability**: How much do you understand and own your code?

Key Setup Principles:

1. **Start with safer tools** (Claude Artifacts, ChatGPT Code Interpreter)
2. **Learn model training cut-offs** (OpenAI: Oct 2023/May 2024)

Critical Success Metrics (Updated):

- **Speed**: Can you ship projects you wouldn't have started before?
- **Confidence**: Do you trust your vibe checks?
- **Discernment**: Can you distinguish appropriate vs inappropriate vibe coding scenarios?
- **Accountability**: How much do you understand and own your code?

Key Setup Principles:

1. **Start with safer tools** (Claude Artifacts, ChatGPT Code Interpreter)
2. **Learn model training cut-offs** (OpenAI: Oct 2023/May 2024)
3. **Practice constantly**

Critical Success Metrics (Updated):

- **Speed**: Can you ship projects you wouldn't have started before?
- **Confidence**: Do you trust your vibe checks?
- **Discernment**: Can you distinguish appropriate vs inappropriate vibe coding scenarios?
- **Accountability**: How much do you understand and own your code?

Key Setup Principles:

1. **Start with safer tools** (Claude Artifacts, ChatGPT Code Interpreter)
2. **Learn model training cut-offs** (OpenAI: Oct 2023/May 2024)
3. **Practice constantly**
4. **Build your discernment** on how to use AI effectively

Beyond Individual Practice

🎯 Bonus Skill: Cross-Domain Translation

"Apply a concept from biology to team management"

Use AI as an analogical engine for transductive thinking

Beyond Individual Practice

🎯 Bonus Skill: Cross-Domain Translation

"Apply a concept from biology to team management"

Use AI as an analogical engine for transductive thinking

The Meta-Skill - "Vibe coding for thought"

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

A Paradigm Shift

🎲 Strategic Vibe Coding

Know when to embrace the vibes for rapid experimentation and learning

✅ AI Augmentation

Review, understand, and take accountability for AI-generated solutions

A Paradigm Shift

🎲 Strategic Vibe Coding

Know when to embrace the vibes for rapid experimentation and learning

✅ AI Augmentation

Review, understand, and take accountability for AI-generated solutions

"We need to turn a solitary interaction with an AI into a shared journey with a community, and to move them towards learning the important lessons about engineering. We do not want to have a generation of developers held captive by companies building vibe-coding tools with little incentive for their users to break from those shackles."

— Armin Ronacher, "Welcoming The Next Generation of Programmers"

Connect With Me

 Course materials

 LinkedIn

 Twitter/X - @LucasEnkrateia

 YouTube - @automatalearninglab

 Email: lucasenkrateia@gmail.com

Keep Vibing, Keep Building