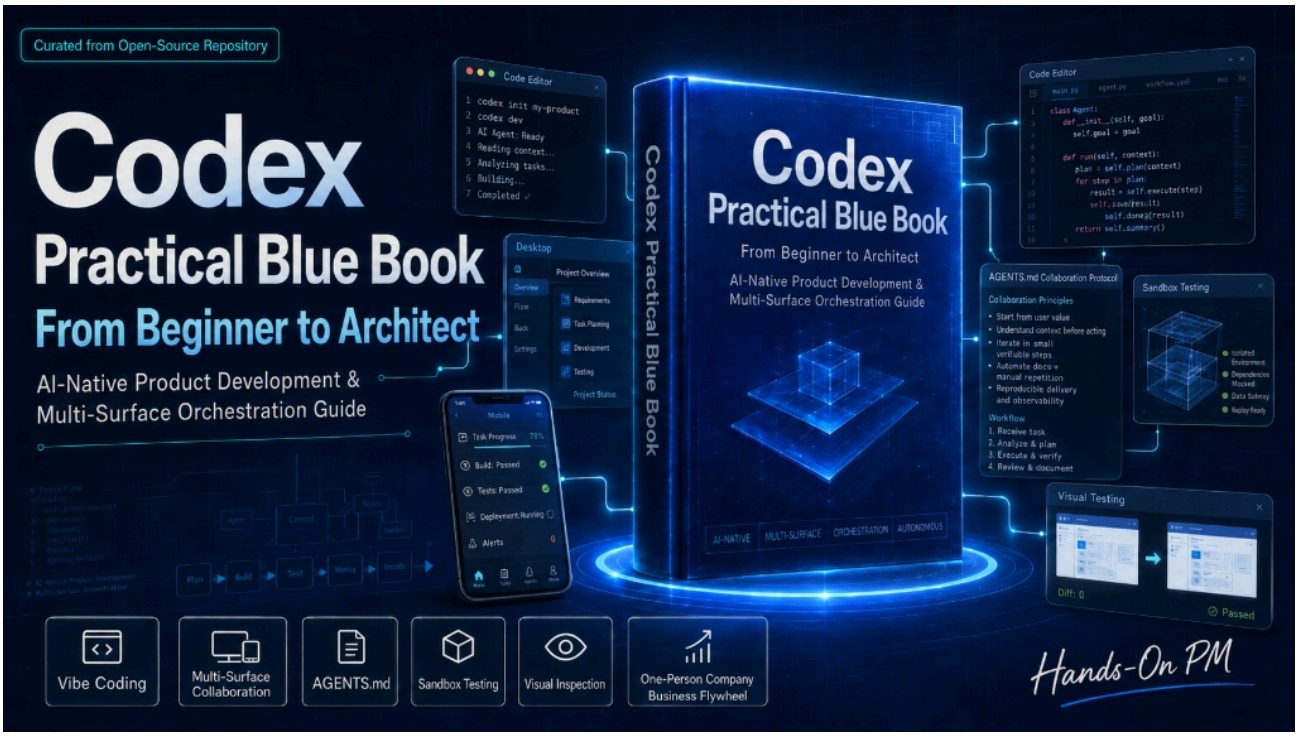
Codex Practical Blue Book: From Beginner to Architect



Author: Hunk Wu (X: @ai_pmer)

🌐 中文 PDF 版 | 🌐 中文在线版

🧭 Table of Contents

- Ch.13 Frontier Watch: The 2026 Codex Ecosystem Overhaul

🔌 Related Projects

- **Feishu Assistant (Codex Feishu Sentinel)**: The official companion repository for Ch.08. An intelligent duty assistant in Feishu for Codex developers, featuring automated daily git digest pushes, CI mobile alarms, and one-tap remote approvals.
- **Codex Switch (Multi-Provider Switcher)**: One-click provider switcher for macOS Codex CLI. Seamlessly switch between official OpenAI, DeepSeek, and Kimi Code in seconds with cross-provider chat history continuation, bypassing quota caps and rate limits.

Ch.01 Saying Goodbye to Handwritten Code: Product Mindset in the Era of Vibe Coding

> 🎯 ***The Real Problem***: *Developers treat AI as just a fancy auto-complete, grinding through repetitive boilerplate, or letting agents run wild without architecture guardrails.*
> 💡 ***Tangible Output & Takeaway***: *A 3-stage mental model for AI-native orchestration and a runnable PRD boundary assertion template to eliminate boilerplate typing.*
> ⚡ ***Viral Screenshot Quote***: "*Humans should no longer manually write boilerplate code. Your hands belong on the steering wheel, not the pushcart.*"

When chatting with readers of "实战产品说" (Real-World Product Talk), I often notice a common pitfall: developers pushing themselves to memorize AI coding commands and shortcuts as if they were cramming for an API manual.

Wake up! In the modern AI agent era (marked by OpenAI Codex and the GPT-5.6 generation), the barrier to code writing itself has dropped to zero. This is called **"Vibe Coding"**—where you focus on core business logic, commercial closed-loops, and real user experience, while leaving the engineering grunts to autonomous agents.

This chapter will help you shift your mindset from a "code typist" to an "AI orchestrator."

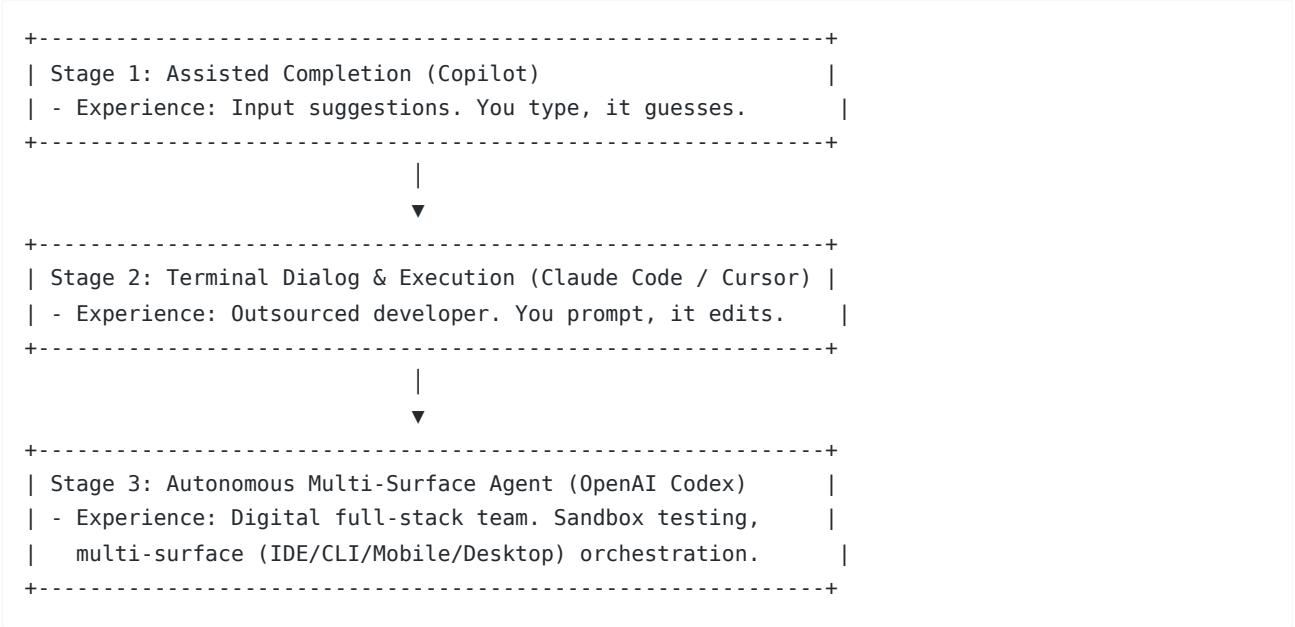
🎯 Intuitive Metaphor: Stop Being a Bricklayer, Become the Chief Architect

If you are new to AI-native development, do not view AI as a cold code generator:

```
【Traditional Handwritten Code】 ──> You act as a manual bricklayer, stacking bricks one by one. If one corner is misaligned, the whole building collapses.
【Traditional AI Completion】    ──> You lay bricks while AI hands you mortar (you type a line, it guesses half; your hands never rest).
【Codex Orchestration Era】      ──> You are the "Chief Construction Architect", and Codex is your "elite engineering crew":
                                     - You sketch blueprints (Define the Goal);
                                     - You erect safety fences (Enforce Sandboxes & Constraints);
                                     - You inspect structural safety with laser levels (Automated Validation);
                                     - The crew (Codex) calculates structural mechanics, procures concrete, and builds the walls.
```

When your focus changes from "how do I type this loop" to "how do I dispatch a foolproof work order to my crew", you have officially stepped into AI-native engineering.

1.1 The Evolution of AI Coding: Where Are We?



Codex has pushed us into Stage 3:

- **Multi-Surface Productivity Matrix**: Automated execution via CLI, complex reviews and Sites in-place preview on ChatGPT Desktop (Code Mode), 24/7 mobile monitoring with Guardian approvals, and visual browser audits via Desktop Computer Use.
- **Frontier Reasoning & Agentic Core**: From the latest frontier flagship **GPT-6 Astra** (launched September 2026 with a 1.05M context window and native Computer Use system operator capabilities) to the mature **GPT-5.6 family (Sol / Terra / Luna)**, agents now possess long-horizon reasoning, cross-application autonomy, and visual self-healing, moving far beyond mere boilerplate generation.

1.2 The Core Shift: From Process Control to Boundary Control

As a Product Manager (PM), you know that when writing a PRD, you don't instruct developers on "how to structure their loops." You define acceptance criteria and boundary constraints.

With Codex, the same rule applies. Stop micromanaging how AI writes logic. As an orchestrator, focus on three things:

1. **Define the End State (Goal)**: What problem are we solving for users?
2. **Establish Guardrails (Constraints)**: Which modules are off-limits? How are secrets isolated?
3. **Automate Validation (Validation)**: Write test suites so Codex can prove its work is correct.

> *[!IMPORTANT]* ***Orchestrator Formula***
> `Successful Delivery = Clear Goal + Strict Constraints + Automated Testing`

🚀 Beginner Quickstart: 3 Steps to Launch

For first-time users, avoid dumping thousands of words of vague thoughts. Follow this 3-step loop:

1. **Step 1: Draft a 50-Word Requirement Card**
   Clarify 3 elements: Target page/feature (Goal), existing framework/libraries (Constraints), validation command (Validation).
2. **Step 2: Restrict Codex in a Safe Sandbox**
   Execute in the terminal with workspace write permission: `codex exec --sandbox workspace-write`.

3. **Step 3: Require the Agent to Prove Verification**
   Instruct Codex: "Run `npm run build` after editing and ensure exit code 0 before completing".

🛡️ Troubleshooting & Pitfall Cheat Sheet

Symptom	Root Cause	Instant Fix
Treating AI as a search engine for theory	Hours of conversation without a single line of working code	Stop chatting; dispatch a concrete spec: "Create a login card in `src/` and pass tests"
Pasting 10,000 lines of unrelated code	Context bloated with noise, causing hallucinations	Point out 1-2 specific files to keep agent edits surgically targeted
Trusting AI claims without running tests	AI assumed it fixed the bug, but runtime crashes	Add mandatory validation specs: "Must run `npm test` in sandbox and print green pass report"

1.3 Your New Moat: Orchestration and Domain Insight

When code is commoditized, what is your moat?

As shared on pmer.cn: **Your value lies in defining business boundaries and empathizing with user pain points.** You no longer need to spend three days tweaking CSS flexbox centering or wrestling with Webpack errors. Your core job is to structure architecture with Codex, deploy fast, and let the market validate your commercial MVP.

Ch.02 Cross-Device Control: Building Your Codex Multi-Surface Productivity Matrix

> 🎯 ***The Real Problem**: Beginners rush to prompt AI while local Node dependencies, outdated clients, and OS permissions fail, causing endless terminal errors and burning tokens.*
> 💡 ***Tangible Output & Takeaway**: 0.14x CLI quickstart, ChatGPT Desktop Code Mode & Computer Use permissions setup, and cross-device debugging checklist.*
> ⚡ ***Viral Screenshot Quote**: "First stabilize your cockpit before asking AI to code. Don't let local setup errors burn your precious token budget."*

To do a good job, one must first sharpen one's tools. In "Real-World Product Talk", I often emphasize a core principle: **The first step of AI-Native development is configuring your "cockpit" to be sufficiently stable.** Many beginners rush into prompting or writing code with AI, only to end up with AI screaming errors in the terminal and wasting tokens because of local environment mismatches or incorrect permission settings.

This chapter will guide you step-by-step through configuring Codex's multi-device productivity matrix, including the 0.14x CLI, the unified ChatGPT Desktop App (Code Mode), and 24/7 mobile monitoring.

🎯 Intuitive Metaphor: Your Three-Surface Mission Cockpit

Do not think of these surfaces as disconnected programs. Imagine commanding a space exploration ship:

```
【Codex CLI】                ──> Main Engine Room: Mounted in your terminal, handling high-throughput batching, tests, and CI builds.
【ChatGPT Desktop Code Mode】──> Bridge Tactical Screen: Multi-repo diff reviews, Sites in-place preview, and visual audits.
```

```
【ChatGPT Mobile App】       ──> Commander's Pager: Monitoring build statuses away from your
desk, approving actions via Guardian.
```

Each surface handles its specialized duty, freeing you from being chained to a keyboard.

🚀 Beginner Quickstart: 3 Steps to Launch

You do not need complex network tunneling on day one. Follow these 3 steps to run your first task in 5 minutes:

1. **Step 1: Install the CLI and Authenticate**
   Ensure Node.js 20+ is installed, then run in your terminal:

   ```
   npm install -g @openai/codex@latest
   codex
   ```

   A browser window will open automatically. Sign in with your ChatGPT account to bind.
2. **Step 2: Create a Minimal Anti-Explosion Config**
   Run the following snippet to create `~/.codex/config.toml`, binding the GPT-5.6 Terra primary model:

   ```
   mkdir -p ~/.codex
   cat << 'EOF' > ~/.codex/config.toml
   model = "gpt-5.6-terra"
   tool_output_token_limit = 12000
   model_auto_compact_token_limit = 64000

   # Optional: Switch to frontier flagship for complex architecture or visual audits
   # model = "gpt-6-astra"

   [profiles.guardian]
   model = "gpt-5.6-luna"
   EOF
   ```

3. **Step 3: Run Your First Safe Sandbox Task**
   Navigate to your project directory and execute:

   ```
   codex exec --sandbox workspace-write "Inspect current code quality and suggest
   improvements"
   ```

2.1 CLI Client (0.14x Era) Setup & Best Practices

The core of Codex CLI is written in Rust (`codex-rs`), but OpenAI distributes it via npm. **As a user, you do NOT need a Rust compiler**, only Node.js 20+.

1. Installation & Environment Check

Run in your terminal:

```
# 1. Verify Node.js (20+ required)
node --version

# 2. Globally install latest stable CLI (0.147.0+)
npm install -g @openai/codex@latest
```

```
# Or use the official install script (macOS / Linux)
curl -fsSL https://chatgpt.com/codex/install.sh | sh
```

Windows users can install via PowerShell:

```
powershell -ExecutionPolicy ByPass -c "irm https://chatgpt.com/codex/install.ps1 | iex"
```

Verify the installation:

```
codex --version
```

2. Authentication: ChatGPT Account vs. API Key

Codex CLI offers two authentication paths:

- **Option A (Recommended / Default)**: Run `codex` directly to log in via browser. **ChatGPT Plus ($20/mo), Pro, Team, Business, Edu, and Enterprise plans all include Codex quotas**, offering the highest ROI for solo developers.
- **Option B (Pay-as-you-go / CI)**: Use an OpenAI API Key for CI/CD pipelines or headless servers.

```
export OPENAI_API_KEY="sk-proj-xxxxxx..."
```

3. Billing Guardrails & Overrun Protection

1. **OpenAI Platform Hard Limit (Crucial)**: Set a monthly Usage Limit (e.g., $50 for beginners) in your OpenAI dashboard. Even in an infinite loop, your budget is safeguarded.
2. **Token Controls in `config.toml`**: Restrict output sizes and compaction thresholds as configured above.
3. **Sandbox & Guardian Approval**: Deprecate `--full-auto`, and use `--sandbox workspace-write` along with `--approve-for-me` (delegated to GPT-5.6 Luna).

4. Seamless Multi-Provider Switching: Overcoming Rate Limits & Quota Caps (Recommended Tool: Codex Switch)

During intensive coding sessions, solo developers frequently encounter OpenAI 3-hour usage caps or API rate-limit errors. Manually editing configurations breaks your development flow, and worse, **often causes you to lose the active multi-turn conversation context**.

To solve this dilemma, we recommend the companion open-source utility **Codex Switch** (created by the author of this book):

- **One-Click Multi-Provider Switching**: Effortlessly toggle between official OpenAI, DeepSeek, and Kimi Code within seconds on macOS, bypassing single-provider rate limits.
- **Seamless Cross-Provider Session Resume**: Solves provider message schema discrepancies, local cryptographic validation, and `thread_history.sqlite` byte-offset alignments. You can **continue existing conversations across different providers** without explaining your context from scratch.
- **Quickstart**:

```
# Clone and install Codex Switch
git clone https://github.com/aipmer/codex-switch.git
cd codex-switch && ./install.sh

# Switch active provider to DeepSeek in 1 second
codex-switch --to deepseek
```

```
# Check current provider status
codex-switch --status
```

2.2 Desktop Surface: ChatGPT Desktop App (Codex Code Mode)

> ⚠️ ***Major Ecosystem Milestone**: In late 2026, the standalone Codex Desktop App was officially retired and merged directly into the **ChatGPT Desktop App**.*

1. Installation & Accessing Code Mode

```
# macOS: Download ChatGPT.dmg and move to Applications
# Windows: Install via winget
winget install OpenAI.ChatGPT
```

Log in and click the **Codex** tab in the left sidebar to enter "Codex Code Mode".

New Productivity Capabilities:

- **Sites Hosting**: In-place web project creation, deployment, and preview with Annotations for point-and-click editing.
- **Multi-repo Review**: Inspect diffs across multiple folders in a single unified view.
- **Enhanced Browser**: Search browsing history directly and reference active Chrome tabs.

2. Computer Use Permissions (macOS)

To allow the agent to visually inspect your screen, grant two permissions under macOS "System Settings -> Privacy & Security":

```
[macOS System Settings] -> [Privacy & Security]
  ├─ Accessibility  ———> Check [ChatGPT] (Allows simulating clicks/keystrokes)
  └─ Screen Recording ──> Check [ChatGPT] (Allows screenshot capture & Vision analysis)
```

Security Boundaries:

- You must explicitly approve each third-party application on first launch.
- ChatGPT cannot interact with Terminal itself, its own window, or `sudo` credential dialogs.

2.3 Mobile Sentinel: 24/7 Remote Oversight

1. **Native Task Syncing**: Signing into Codex with your ChatGPT account automatically mirrors running tasks to the ChatGPT Mobile App.
2. **Guardian Approvals & Alerts**: With Ch.08's mobile gateway, critical deployment gates trigger rich cards in Feishu or WeChat, enabling one-tap mobile approvals.

🛡️ Troubleshooting & Pitfall Cheat Sheet

Symptom	Root Cause	Instant Fix
`error: unknown option '--full-auto'`	Deprecated in CLI 0.14x	Replace with the new sandbox flag: `--sandbox workspace-write`
`SyntaxError: Unexpected token ...` (Node error)	Node.js version is older than 20	Run `node -v`; upgrade using `nvm use 20` or `nvm install 20`

`Permission denied: Screen Recording`	Missing macOS screen permissions	Toggle ChatGPT under "System Settings -> Privacy & Security -> Screen Recording" and restart
`Model not found: gpt-5.4`	Older models retired	Update `~/.codex/config.toml` to specify `model = "gpt-5.6-terra"`

Ch.03 Breaking the Cloud Island: Sandbox Debugging and Deep Local Environment Tunneling

> 🎯 ***The Real Problem***: *Agent running in an isolated sandbox cannot reach local Docker databases (PostgreSQL/Redis), repeatedly failing with* `Connection refused` *.*
> 💡 ***Tangible Output & Takeaway***: *CLI 0.14x sandbox tier analysis, SSH / Ngrok reverse tunneling scripts, and a 3-step connectivity diagnostic checklist.*
> ⚡ ***Viral Screenshot Quote***: *"Can't connect to localhost from sandbox? It's not a code bug—you simply forgot to bridge the network tunnel."*

When running database tests with Codex, the most frequent surprise for beginners is: while PostgreSQL runs perfectly in local Docker, the AI shouts `Connection refused to localhost:5432` in the terminal.

This is a classic barrier brought by **Sandbox Isolation**. This chapter teaches you how to pierce through this isolation barrier and bridge a secure pipeline between the sandbox and your host.

🎯 Intuitive Metaphor: The Cleanroom Containment and Umbilical Pipeline

Think of Codex's runtime environment as a "high-level sterile containment cleanroom":

```
【Host Mac/PC】       ——> The Outside World: Holding your actual local database, secret keys, personal tools, and files.
【Codex Sandbox】     ——> The Sterile Cleanroom: Where the AI builds code and runs tests. Even if the code crashes, your host system is untouched.
【Reverse Tunneling】 ——> Umbilical Feed Pipeline: When the AI in the cleanroom needs to talk to the local host DB, you must bridge a dedicated pipe.
```

Without this pipe, the AI calling `localhost` inside the cleanroom reaches only bare walls, naturally throwing `Connection refused`.

🚀 Beginner Quickstart: 3 Steps to Launch

Follow these 3 steps to connect the sandbox with your local database:

1. **Step 1: Confirm Local Service is Actively Listening**
   Verify in terminal that Postgres is running locally on 5432:

   ```
   lsof -i :5432
   ```

2. **Step 2: Launch a Quick Port Tunnel (Ngrok or SSH)**
   Open a temporary public tunnel for port 5432:

   ```
   ngrok tcp 5432
   ```

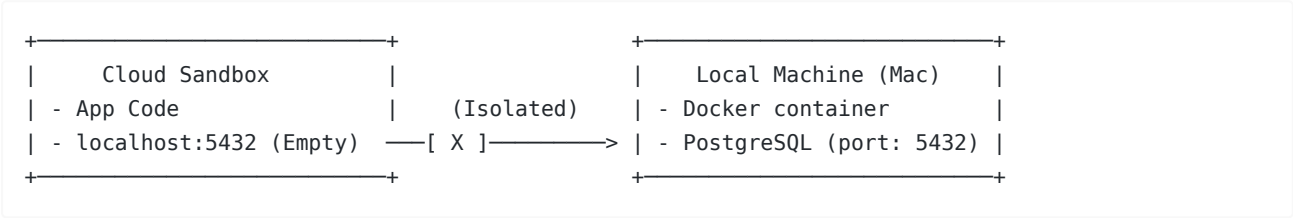
```
# Output: tcp://0.tcp.ngrok.io:12345
```

3. **Step 3: Inject the Tunnel URL into Codex Task**
   Pass the tunnel address into the environment and run:

```
export DATABASE_URL="postgresql://postgres:password@0.tcp.ngrok.io:12345/dev_db"
codex exec --sandbox workspace-write "npm run test:db"
```

3.1 Sandbox Network Barriers: Why Can't `localhost` Connect?

By default, the sandbox and your host machine are network-isolated:



In CLI 0.14x, sandbox boundaries are explicitly defined:

- `--sandbox read-only` : Analysis only, zero write permissions.
- `--sandbox workspace-write` : Recommended default; allows edits within the project workspace while protecting the host OS.

To connect with host databases or microservices, you must establish **port mapping and reverse tunneling**.

3.2 Practice: Setting Up Reverse Tunnels via SSH / Ngrok

Method 1: SSH Reverse Port Forwarding (Recommended for Long Term)

```
# Forward local port 5432 to port 54320 on your public VPS
ssh -R 54320:localhost:5432 user@your-public-vps.com -N
```

Then configure the sandbox:

```
export DATABASE_URL="postgresql://postgres:password@your-public-vps.com:54320/dev_db"
```

Method 2: Ngrok for Instant Tunneling (Zero-Server Option)

Run on host:

```
ngrok tcp 5432
```

Inject the printed `tcp://0.tcp.ngrok.io:12345` into Codex session.

3.3 Directory Mapping and Env Syncing

1. Security Isolation for Secrets

Never allow AI to commit raw `.env` files. Enforce in `.gitignore` and AGENTS.md:

```
## 🛑 Hard Constraints
- Never sync, copy, or commit files matching *.env.
- Always use `src/config.ts` or environment variables for secret injection.
```

2. Sandbox Cache Clearing

To avoid ghost build bugs, define clean run commands in AGENTS.md:

```
## 💻 Developer Commands
- **Clean Run**: `rm -rf node_modules/.cache .next/cache && npm run test`
```

🛡️ Troubleshooting & Pitfall Cheat Sheet

Symptom	Root Cause	Instant Fix
`Connection refused to 127.0.0.1:5432`	Sandbox attempted to query localhost inside cleanroom	Confirm tunnel is active; use the reverse tunnel address instead of `localhost`
`ngrok session expired / reconnecting`	Free ngrok tunnel timed out or changed port	Restart ngrok and copy new address, or use `scripts/codex-watchdog` helper
`EACCES: permission denied`	Agent tried to write to protected system directories	Ensure `--sandbox workspace-write` is used and edits stay within project root

---

Ch.04 Goal-Driven Engineering: Taming Reasoning Agents with Boundaries and Assertions

> 🎯 ***The Real Problem***: *Overly verbose prompts micromanage models like dictating every cut to a chef, while vague prompts invite hallucinations and regression bugs.*
> 💡 ***Tangible Output & Takeaway***: *Standard goal-driven specification template (Goal + Preconditions + Output Assertions + Prohibited Actions) and real benchmark comparisons.*
> ⚡ ***Viral Screenshot Quote***: *"Micromanaging a chef burns the dinner. Define strict input-output assertions and let the agent engineer the solution."*

In the era of Codex, powered by deep reasoning models like **GPT-5.6 Terra**, traditional step-by-step prompt engineering has become counterproductive. Reasoning models possess immense internal planning space; over-specifying execution steps only handcuffs their ability to self-correct.

This chapter shares how to guide Codex using a professional "Product Specs" approach in actual development.

🎯 Intuitive Metaphor: Michelin Chef and the Order Ticket

Many people dispatch tasks to AI like an unruly customer barging into a Michelin kitchen:

```
【Babysitting Commands (Process-Driven)】 ──> ❌ "Chef, take the knife in your left hand, chop potatoes into 2mm strips, heat the oil to 180°C, stir for 3 minutes, then add 3g salt."
                                                (The chef feels insulted, and if stove pressure shifts slightly, the dish burns.)
【Architect Spec Order (Goal-Driven)】    ──> ✅ "Chef, I need pan-seared crispy potatoes as a steak side dish:
                                                - Goal: Crispy texture, dinner side dish;
```

```
                                                - Constraints: Under 200 kcal, strictly NO butter
or peanuts (customer allergy);
                                                - Validation: Plated within 15 minutes, core
temperature at 75°C."
```

Codex is an algorithmic master chef. Your role is defining what to make, what cannot be touched, and how success is measured—leaving the heat and chopping technique to the agent.

🚀 Beginner Quickstart: 3 Steps to Launch

Draft your first Goal-Driven Spec in 3 steps:

1. **Step 1: Define the End State (Goal)**
   State the deliverable in one sentence: "Implement Stripe payment session creation at `/api/checkout`".
2. **Step 2: Enforce Inviolable Guardrails (Constraints)**
   List 2-3 boundaries: "Never log secret keys in plain text; do not modify global middleware".
3. **Step 3: Provide Automated Verification Assertions (Validation)**
   Give a runnable command: "Running `npm test -- checkout.test.ts` must pass 100% on the first run".

4.1 The Goal-Driven Markdown Specs Template

When triggering coding tasks in your terminal or ChatGPT Desktop, use this structured Markdown layout:

```
# 🎯 Goal
[Describe desired business outcome. Example: Implement GitHub OAuth login route and persist
preferences.]

# 🛑 Constraints
- [Security: Never commit credentials or write keys to log streams.]
- [Stack: Strictly use existing Tailwind classes; do not introduce new UI libraries.]
- [Anti-pollution: Prohibited from editing any file under src/legacy/.]

# 🧪 Validation Specs
- [Automated Testing: Running `npm run test:unit` must pass 100%.]
- [Edge behaviors: If inputs are missing, return 400 Bad Request with structured JSON error
payloads.]
```

4.2 Real Case Comparison: Traditional Prompt vs. Goal-Driven Specs

Suppose we need to write a **"Redis-based Rate-Limiting API Proxy Service"**.

❌ Traditional Process-Driven Prompt

> *"Please help me write an API proxy with Express. First, import express and express-rate-limit. Then configure rate-limiting, setting windowMs to 15 minutes and max to 100. Then write a route `/api/proxy` using axios to request the third-party API `https://api.github.com`. If successful, return the data; if it fails, return a 500 error. Make sure to include the Authorization Bearer Token in the headers."*

✅ Goal-Driven Specs

```
# 🎯 Goal
Implement an Express API proxy route that forwards all incoming requests safely to the GitHub
API.
```

```
# 🛑 Constraints
- Must use Redis as the rate-limiting data source (no memory-based limiting) to support multi-container scaling.
- Limit proxy request timeout strictly to 3000ms to prevent hanging the Node event loop.
- Never write the GitHub token into the codebase or logs; fetch securely from `process.env.GH_TOKEN`.

# 🧪 Validation Specs
- Under simulated load: return 429 Too Many Requests when requests exceed 60 req/min.
- Fault tolerance: return 504 Gateway Timeout with structured JSON on timeouts or network drops.
```

4.3 GPT-5.6 Terra Reasoning Architecture for Specs

When presented with this spec, Codex breaks down the solution chain:

```
[Parse Specs Goal] ──> Analyze Constraints (Redis rate-limit / 3s timeout / Secret isolation)
                              │
                              ▼
[Self-Directed Architecture] ──> Import ioredis + rate-limit-redis, configure Axios abort controllers
                              │
                              ▼
[Align with Validation Specs] ──> Simulate 429 rate limit & timeout fallbacks in unit tests
                              │
                              ▼
[Self-Heal Code Until Tests Pass 100%] ──> Deliver clean code with green verification reports
```

Delegate logic planning to AI, but keep verification standards firmly in your own hands.

🛡️ Troubleshooting & Pitfall Cheat Sheet

Symptom	Root Cause	Instant Fix
Overly fragmented steps cause agent to freeze on errors	Procedural commands stripped GPT-5.6 of self-healing planning space	Strip out implementation steps; retain only Goal, Constraints, and Validation
Agent modified unrelated files, breaking other features	Lacked directory scope constraints	Declare strict boundaries: "Edits restricted to `src/modules/auth/`; never touch other directories"
Agent claimed completion, but logic has fatal flaws	Missing automated validation assertions	Add explicit requirement: "Must run `npm test` and produce green unit test assertions"

Ch.05 Defining the CAP Protocol: Building Your Project's AGENTS.md Rule Compliance Layer

> 🎯 ***The Real Problem**: Fixing 1 bug creates 3 new bugs, infinite 10-retry loop spins that drain budgets, unwanted heavy npm packages, and lazy fake `TODO` stubs.*

> 💡 ***Tangible Output & Takeaway***: *Production-grade* `AGENTS.md` *project rules template, 2-retry hard circuit breaker, and zero-placeholder engineering boundaries.*
> ⚡ ***Viral Screenshot Quote***: *"An unconstrained AI is a runaway horse. An AGENTS.md rulebook forces models to respect architecture like a principal engineer."*

In the product development process, one of the most frustrating scenarios is fixing one bug only to introduce three new ones, or coding a new feature while completely disregarding the team's established coding standards.

In human-machine collaborative development, if Codex is left without boundary constraints, it too can become an overly eager and "destructive" employee. To place a tight rein on it, we need to establish an **`AGENTS.md`** file in the project root.

This is our **"agent collaboration constitution" (Codex Collaboration Protocol, CAP)**.

🎯 Intuitive Metaphor: The New Employee Onboarding Handbook & Safety Code

Think of Codex as a brilliant, hyper-energetic new intern on their first day:

```
【Without AGENTS.md】 ──> The intern arrives with no rulebook. To fix a minor frontend alignment bug,
                          they casually rewrite your 3-year-old core authentication module into an esoteric syntax,
                          crashing production upon release.
【With AGENTS.md】    ──> The intern sits at their desk and reads the team's printed handbook:
                          - "This is Next.js 15 with Tailwind" (Explicit stack)
                          - "Build with `npm run build`, never alter standard scripts" (Normalized operations)
                          - "DO NOT touch the auth/ directory under any circumstance" (Absolute red lines)
                          The intern immediately grasps the boundaries and operates with peak efficiency without breaking a thing.
```

`AGENTS.md` is that zero-friction onboarding manual for AI agents.

🚀 Beginner Quickstart: 3 Steps to Launch

Create your project's first compliance rulebook in 3 minutes:

1. **Step 1: Create the File in Your Project Root**
   Run in your terminal:

   ```
   touch AGENTS.md
   ```

2. **Step 2: Paste the 4-Section Minimal Scaffold**
   Save the following into `AGENTS.md` :

   ```
   # 🤖 Codex Collaboration Protocol

   ## 📌 Project Fingerprint
   - Stack: Next.js 15, TypeScript, Tailwind CSS

   ## 💻 Developer Commands
   - Build: `npm run build`
   - Test: `npm run test`
   ```

```
## 🛑 Hard Constraints
- Never update package.json dependencies without human confirmation.
- Never touch files inside `src/legacy/`.
- Always run `npm run build` before completing the task.
```

3. **Step 3: Launch Codex to Verify Protocol Loading**
   Run `codex`; the initialization logs will confirm that workspace rules have been loaded.

5.1 Why Do We Need `AGENTS.md`?

`AGENTS.md` is OpenAI Codex's official instruction specification file. When starting up, Codex scans and merges configurations hierarchically:

1. `~/.codex/AGENTS.override.md` (Highest priority, user overrides)
2. `~/.codex/AGENTS.md` (Global user instructions)
3. Project root `AGENTS.md` (Team standard)
4. Current directory `AGENTS.md` (Specific workspace scope)

Its core value lies in:

1. **Immediate Context Restore**: Every time Codex starts, its first action is scanning `AGENTS.md`, instantly picking up the project's tech stack, directory structure, and collaboration constraints.
2. **Anti-Corruption Layer**: It explicitly defines which directories and files are "read-only/off-limits," preventing Codex from unilaterally refactoring critical security modules.
3. **Command Execution Guardrails**: It specifies permissible test and deployment command options, avoiding destructive shell commands.

5.2 The Four Core Sections of `AGENTS.md`

```
# Project Fingerprint
- Tells the AI what kind of project this is and its core tech stack.

# Developer Commands
- Explicitly states the command lines for building, testing, and running migrations.

# Styles & Architecture Patterns
- Dictates where files should be placed, size limits, and design patterns.

# Agent Boundary & Hard Rules
- Defines absolute forbidden paths. If touched, Codex must abort and ask for human verification.
```

5.3 Practice: A Production-Grade `AGENTS.md` Template

```
# 🤖 Project: Aurora SaaS Core (AGENTS.md)

## 🧬 Project Fingerprint
- **Stack**: Next.js 15 (App Router), TypeScript, Prisma ORM, TailwindCSS.
- **Database**: PostgreSQL on Supabase.
- **Auth**: Next-Auth v5.

## 💻 Developer Commands
- **Install**: `npm install` (Only run if package.json has changed)
```

```
- **Dev Server**: `npm run dev`
- **Lint Code**: `npm run lint`
- **Run Tests**: `npm run test`
- **DB Migration**: `npx prisma migrate dev` (Never run in production branch)

## 🎨 Styles & Architecture Patterns
- **Directory Structure**:
  - Components: Keep UI components inside `@/components/ui/` (Shadcn styled).
  - Business Logic: Custom React hooks must go to `@/hooks/`.
  - API Routes: Next.js Route Handlers go to `src/app/api/.../route.ts`.
- **Formatting**:
  - Keep components modular. If a component exceeds 200 lines, decompose it.
  - All API routes must implement Zod schema validation for request body.
  - Return HTTP 400 for validation errors, 500 for internal uncaught errors.

## 🛑 Agent Boundary & Hard Rules
- **READ-ONLY Directories**:
  - Never modify files inside `src/app/api/auth/[...nextauth]` (OAuth Core).
  - Never alter `prisma/schema.prisma` without explicit human confirmation.
- **PR Rules**:
  - Before declaring a feature complete, run `npm run test` and `npm run lint`.
  - If tests fail, rollback the change immediately and report the error logs.
- **Security Check**:
  - Never commit raw `.env` files or API Keys. Use environment variables.
```

5.4 Dual Defense: Soft Constraints vs. Hard Enforcements

Rules in `AGENTS.md` are natural language directives—models like GPT-5.6 Terra respect them with high fidelity, but they are "soft constraints."

To guarantee physical guardrails that cannot be breached, combine with Codex 0.14x runtime controls:

1. **Sandbox Mode**: Enforce `--sandbox workspace-write` to confine edits within the project workspace at the OS container level.
2. **Guardian Approvals**: Pass `--approve-for-me` to let Guardian evaluate policy violations automatically.
3. **Hooks Engine**: Configure `[hooks]` in `~/.codex/config.toml` to execute pre/post test scripts on file writes.

```toml
# ~/.codex/config.toml
model = "gpt-5.6-terra"

[hooks]
stop = "npm run lint --silent"
```

"AGENTS.md Soft Specs + Sandboxes & Hooks Hard Fences" creates the ultimate anti-corruption moat.

🛡️ Troubleshooting & Pitfall Cheat Sheet

Symptom	Root Cause	Instant Fix
Agent occasionally breaks rules despite being written	Rules too long, contradictory, or buried in verbose prose	Keep rules concise and prefer negative constraints ("Never do X"); elevate to Hooks/Sandbox

AI falls into a 5-attempt self-correction death loop	Missing Anti-Loop circuit breaker in instructions	Add to AGENTS.md: "Stop immediately after 2 consecutive failed attempts and report root causes"
Agent installs arbitrary unfamiliar npm packages	Package managers allowed without gating	Mandate in Hard Rules: "Never run `npm install` for new dependencies without human consent"

Ch.06 Steering Reasoning: Supervising the CoT Process Like a Tech Lead

> 🎯 ***The Real Problem***: *Waiting passively for heavy reasoning models while an incorrect initial assumption snowballs into deep codebase corruption.*
> 💡 ***Tangible Output & Takeaway***: *Real-time TUI Reasoning Summary inspection, 2 infinite-loop heuristics, and companion Chrome extension project (`examples/ch06-chrome-extension`).*
> ⚡ ***Viral Screenshot Quote***: *"Don't let AI run blind for 20 minutes before discovering it derailed. Catch flawed assumptions on step one."*

In traditional software development, when managing a junior engineer, your biggest fear is having them work in isolation for a week, only to deliver code that completely deviates from business goals or breaks the main branch.

When working with Codex powered by the **GPT-5.6 Terra** deep-reasoning model, the AI's coding power is immense. However, if its initial premise is wrong, it will construct elaborate arguments around that flawed assumption, sprinting down the wrong path or getting stuck in infinite self-correction loops.

This chapter teaches you how to look directly into Codex's **reasoning process**, intervening like a seasoned tech lead to pull the agent back on course the moment it wanders.

🎯 Intuitive Metaphor: An "Exam Proctor Window" into AI's Mind

Never treat AI reasoning as a black box:

```
[Passive Blind Waiting] ──> Like waiting outside an exam hall for 2 hours, only to discover the
student misremembered the fundamental formula on Question 1, failing the entire exam.
[Proctor Supervision]   ──> Like standing right behind the student, observing their draft
scratchpad (Reasoning Summary):
                            - The student drafts: "Assume we need to rewrite the entire database
schema..."
                            - You tap their shoulder immediately: "Don't touch the schema, just
add a query index!"
                            - The student strikes out the wrong idea and returns to the right
track.
```

Reading the Reasoning Summary gives you real-time visibility into that draft scratchpad.

🚀 Beginner Quickstart (3 Easy Steps)

Learn to supervise and correct the AI like a tech lead in 3 simple steps:

1. **Step 1: Launch the Interactive TUI and Monitor Reasoning**
   Run `codex` directly in your terminal and observe the rolling thought summary in the split pane.

2. **Step 2: Press `Ctrl + C` the Instant an Assumption Derails**
   The moment you see the AI planning to pull in unfamiliar external packages or refactor core untouched files, hit `Ctrl + C` immediately.
3. **Step 3: Correct with One Precise Sentence and Resume**
   Type your constraint directly: "*Do not rewrite existing tables; solve this with in-memory caching.*" The AI will flush the invalid assumption and replan.

6.1 Why Inspect the Model's Reasoning Process?

The biggest difference between deep-reasoning models (such as GPT-5.6 Terra) and traditional models is that before producing final code, the reasoning model conducts in-depth internal hypothesis testing and simulation. The Codex TUI displays this process in real time as a **Reasoning Summary**.

```
[User Request] ──> 1. Parse Goals & Constraints ──> 2. Plan Steps ──> 3. Run Tests ──> 4. Self-
Correct ──> [Final Output]
                     └──────(Displayed in TUI as Reasoning Summary: Your Proctor
Viewport)──────┘
```

> ⚠️ ***Note**: The full internal Chain of Thought (CoT) is guarded for safety; what you see in the TUI is a **high-fidelity reasoning summary** generated by the model. This summary is more than sufficient to evaluate technical decisions and architectural directions.*

6.2 How to Observe and Interpret Reasoning in the TUI

1. Interactive Mode (TUI)

Launch `codex` to enter the interactive terminal interface, where the main pane streams the reasoning summary. Essential inspection and control commands:

```
# Common TUI shortcut commands
/diff       # Review actual code modifications in the current session
/review     # Summon a reviewer subagent to audit recent changes
/copy       # Quickly copy the last generated response or code snippet
```

2. Non-Interactive Mode (Scripted Pipelines)

When running background tasks or long scripts, stream structured JSONL logs:

```
# Non-interactive execution, streaming structured JSONL
codex exec --json "Refactor auth middleware" > task.jsonl
```

Healthy Reasoning Stream Example

When Codex receives the task "*Fix connection timeout in the Redis rate limiter,*" a healthy reasoning stream looks like this:

```
[Reasoning Summary - Normal Flow]
- User wants to fix Redis rate limiter connection timeout.
- Checking existing implementation in src/lib/redis.ts...
- Found `redis = new Redis()` without retryStrategy.
- If Redis is unreachable, this hangs the Node process, violating the 3000ms SLA in AGENTS.md.
- Action Plan:
  1. Add `maxRetriesPerRequest: null` and explicit connectTimeout: 2000.
```

```
2. Implement custom retryStrategy up to 3 attempts.
3. Run `npm run test:redis` to verify behavior.
```

6.3 Identifying Typical AI "Death Loops"

Stay alert for these two classic self-spinning traps during daily development:

1. The Dependency Loop

- **Symptom**: The AI attempts to install an unvetted library. When installation fails, it tries three different versions or switches to another even more obscure library.
- **Signal**: Repeated occurrences of `npm install --legacy-peer-deps` more than twice in the terminal.

2. The Regression Loop (Whack-a-Mole)

- **Symptom**: Editing file A breaks test B; it modifies test B, triggering errors in module C; it edits C, which breaks file A again.
- **Signal**: Test pass rates fluctuating between 80% and 90%, with repeated edits to the exact same set of files.

6.4 The Three-Step Intervention: Interrupt, Correct, and Take Over

Step 1: Decisive Interruption (`Ctrl + C`)

Press `Ctrl + C` to stop the current run immediately and avoid token waste.

Step 2: Point-to-Point Correction (Direct Dialogue)

Directly point out the blind spot:

```
You were trying to install axios-retry, but this project strictly forbids external HTTP retry
libraries. Use native AbortController for timeout handling and replan.
```

Step 3: Human Takeover and Git Rollback

If files were already corrupted, revert with Git and enforce the rule:

```
git checkout -- src/lib/redis.ts
```

Append a guardrail into AGENTS.md: "*Strictly prohibit external retry libraries.*"

6.5 Companion Hands-on Sandbox: Codex Web Copilot (Chrome Extension)

To experience how to steer reasoning chains and enforce anti-loop guardrails during browser extension development, this repository provides a ready-to-run sandbox:

👉 **Source Sandbox Directory**: examples/ch06-chrome-extension

Core Highlights:

1. **Pure Native Manifest V3**: Zero build bundling dependencies. Load unpacked in Chrome Developer Mode in under 1 minute;
2. **Strict CSP Guardrails**: `AGENTS.md` forbids inline scripts and `eval()`, showing how AI agents write compliant code under strict browser sandboxes;

3. **Automated Guard Tests**: Run `npm test` to automatically validate MV3 specs and script syntax.

🛡️ Troubleshooting & Pitfall Cheat Sheet

Common Pitfall	Root Cause	Rapid Diagnosis & Fix Guide
AI stuck in an endless loop of self-edits	Prompt lacks an explicit retry cap	Hit `Ctrl + C` immediately, then instruct: "*Retry threshold reached. Stop spinning and output your diagnostic findings.*"
Interrupting and re-prompting loses previous context	Session context was dropped	Use `codex resume` to restore the original session thread and preserve reasoning context
AI broke multiple legacy core files	Lack of Git clean-branch isolation before modifications	Run `git checkout .` to restore, then use `--sandbox workspace-write` to restrict scope

---

Ch.07 Closing the Visual Loop: Automated Inspection and Design Fidelity with Desktop Computer Use

> 🎯 ***The Real Problem***: *Manual pixel-peeping for UI styling, responsive layouts missing hidden modals, and headless CLI tests incapable of verifying real browser rendering and clicks.*
> 💡 ***Tangible Output & Takeaway***: *ChatGPT Desktop Code Mode setup; automated Figma-to-DOM screenshot visual diff workflows; and UI visual telemetry reproductions.*
> ⚡ ***Viral Screenshot Quote***: "*Still squinting at pixels to verify responsive UI? Let AI open the browser, measure dimensions, click buttons, and annotate visual discrepancies.*"

In traditional UI fidelity reviews, the most time-consuming task for product managers and frontend developers is "pixel-eye" alignment verification:

“This button seems shifted 4 pixels to the left.”
“This popup gets obscured by the virtual keyboard on mobile dimensions.”

In the 2026 Codex ecosystem, combining **ChatGPT Desktop (Codex Code Mode)** with OpenAI's latest frontier flagship **GPT-6 Astra** (natively engineered as an autonomous computer operator), the agent can not only write code but also "use eyes and hands" to directly operate your macOS desktop: launching browsers, adjusting developer tool resolutions, and performing high-fidelity visual audits.

This chapter teaches you how to orchestrate Computer Use to automate design-fidelity inspection for frontend UI.

🎯 Intuitive Metaphor: A 24/7 Tireless "Pixel Quality Inspector"

Think of Computer Use as hiring a dedicated UI QA inspector sitting right beside your desk:

```
[Manual UI Inspection] ──> Holding Figma mockups in one hand, switching Chrome tabs with the
other,
                           squinting to verify font sizes, resizing windows manually for
breakpoints,
                           and manually refreshing pages after every CSS tweak (slow and
exhausting).
```

```
[Computer Use Mode]    ──> You hand the inspector a task card: "Review /auth/login against
design mockup."
                           The inspector puts on blue-light glasses (screenshot analysis),
                           takes the mouse (simulated clicks), measures a 16px offset,
                           updates the Tailwind classes in the editor, refreshes the browser,
                           and captures a clean screenshot to report completion.
```

You never need to act as a human pixel-comparison machine again, freeing your energy for interaction design and core business logic.

🚀 Beginner Quickstart (3 Easy Steps)

Run your first AI visual inspection in 3 simple steps:

1. **Step 1: Verify ChatGPT Desktop Permissions**
   In macOS **System Settings → Privacy & Security**, ensure that **ChatGPT** is granted **Accessibility** and **Screen Recording** permissions.
2. **Step 2: Start Your Local Frontend Server**
   Start your development server in the terminal (confirming it opens in your browser):

   ```
   npm run dev
   # Verify http://localhost:3000 is accessible
   ```

3. **Step 3: Summon `@Chrome` in Code Mode with an Inspection Task**
   In the ChatGPT Desktop Codex mode prompt, enter:

   ```
   @Chrome Open http://localhost:3000/auth/login, capture the main card area, check for
   horizontal viewport overflow, and adjust padding.
   ```

7.1 Safety First: Sandbox Boundaries and Application Whitelisting

Allowing an AI to operate your physical screen requires strict security fencing. To prevent the agent from accidentally clicking personal messaging apps or altering system files due to misrecognition, establish **application-level whitelisting**.

1. GUI Permission Whitelist

In the ChatGPT Desktop client:

- The first time Computer Use interacts with a specific application (e.g., Google Chrome), a system prompt will ask for confirmation;
- You can choose **“Just this once”** or **“Always allow”**;
- Strictly avoid whitelisting personal apps (e.g., WeChat, Slack, Mail, or the terminal itself);
- **Factory Hard Bounds**: Codex cannot operate the terminal, the ChatGPT client itself, or system administrative privilege dialogs (`sudo`), forming an OS-level hard isolation barrier.

2. Coordinate Positioning and Visual Perception

Computer Use operates via a robust perceptual feedback loop:

```
[Viewport Screenshot] ──> [GPT-5.6 Vision Analysis] ──> [Compute Pixel Coordinates (x:450,
y:230)] ──> [Execute Mouse Click/Drag]
```

Combined with the 2026 **Appshots** feature (double-tap shortcut), visual snapshots and text contexts of the focused foreground window can be instantly injected into the conversation without manual screenshot uploads.

7.2 Practical Visual-Driven UI Review: Figma Mockup Alignment

This is the most practical automated scenario: **having Codex autonomously compare design mockups with local web rendering, auto-tuning styles.**

🎯 Goal

Compare local page `/auth/login` with design mockup screenshot `figma_login_mockup.png`, eliminating spacing discrepancies.

🛑 Constraints

- Only Tailwind utility class adjustments in `src/app/login/page.tsx` are permitted.
- Modifying existing DOM trees or semantic tags is prohibited.

🧪 Automated Execution Specs

Submit this prompt in the ChatGPT Desktop client:

```
@Chrome Complete the UI visual inspection following these steps:

# 🎯 Goal
Compare and align browser rendering with figma_login_mockup.png.

# 🛑 Constraints
- Only use Tailwind utility classes in src/app/login/page.tsx.
- Do not change the DOM structure.

# 🚀 Execution Steps
1. Open Google Chrome and navigate to http://localhost:3000/auth/login.
2. Capture screenshot of the login form container.
3. Compare against assets/figma_login_mockup.png and identify padding discrepancies.
4. Update Tailwind classes in src/app/login/page.tsx to match spacing.
5. Reload page and confirm visual alignment within 2% delta.
```

7.3 Conceptual Execution Flow of Codex Automated Auditing

```
> Running visual review for /auth/login
> Step 1: Opening Google Chrome on http://localhost:3000/auth/login...
> Step 2: Taking screenshot. Saved to /tmp/screenshot_v1.png
> Step 3: Calling vision model for image comparison.
    Analysis: "Login card padding-top is 16px (pt-4), but mockup requires 32px (pt-8). Font size
is text-base, needs text-xl."
> Step 4: Updating src/app/login/page.tsx via apply_patch...
> Step 5: Reloading Chrome tab and taking validation screenshot...
> Step 6: Vision check: "Visual delta is within 1.2% tolerance. Perfectly aligned."
> Task completed successfully.
```

7.4 Advanced: Sites In-Place Preview and Multi-Device Responsive

Auditing

Leveraging the 2026 ChatGPT Desktop **Sites** feature alongside mobile breakpoint audits:

```
@Chrome
# 📱 Mobile Viewport Inspection
1. Open Chrome DevTools.
2. Toggle Device Toolbar and select iPhone 15 Pro (393 x 852).
3. Verify the submit button does not drop below the first screen fold.
4. If obscured, reduce hero section padding to keep the CTA button immediately clickable.
```

Solo developers no longer need to resize browser windows back and forth, letting the AI handle 90% of visual inspection heavy lifting.

7.5 Cross-Domain Extension: From Web Audits to Autonomous 3D Software Control (Blender Integration)

With the arrival of **GPT-6 Astra** in late 2026—bringing native spatial geometric reasoning and a 95.9% mean voxel IoU score on OpenAI's BenchCAD benchmark—Computer Use has expanded well beyond web browsers.

In a desktop environment, agents can autonomously interact with professional 3D creative suites like **Blender**:

- **Viewport Operations & Shading Checks**: The agent uses Computer Use to toggle viewports between Shading material preview and Rendered modes;
- **Inspecting Geometry & Shader Artifacts**: The agent takes viewport screenshots to identify inverted surface normals, missing UV maps, or washed-out lighting;
- **Spatial Pipeline Synergy**: Pairs seamlessly with the **Tripo3D + Blender + Astra** 3D automation pipeline detailed in Ch.13, delivering full automation from generative prompts to render verification.

🛡️ Troubleshooting & Pitfall Cheat Sheet

Common Pitfall	Root Cause	Rapid Diagnosis & Fix Guide
`Computer Use is not supported on this platform`	OS is not macOS or user is in an unsupported region	Ensure macOS client is used, or switch to a Headless Puppeteer script approach as an alternative
`Failed to capture window: Permission denied`	macOS Screen Recording permissions missing or needs refresh	Open System Settings, toggle ChatGPT's "Screen Recording" permission off and on, and restart the client
AI mouse clicks wild coordinates off target	Browser zoom is not 100% or multi-display DPI scaling mismatch	Reset Chrome zoom to standard 100%, and position the target test window in the center of the primary display

---

Ch.08 Mobile Sentinel Workflows: 24/7 Remote Development and Orchestration

> 🎯 ***The Real Problem***: *Engineers tied to desks watching terminal build logs; unattended CI failures or blocked deployments halting team momentum.*
> 💡 ***Tangible Output & Takeaway***: *Guardian auto-approval (`--approve-for-me`) & mobile gateway two-tier dispatch; Feishu webhook alert cards; one-tap mobile remote approval pipelines.*

> ⚡ ***Viral Screenshot Quote***: *"Step away from your desk while tasks run. AI resolves routine risks automatically, while high-risk releases wait for a one-tap phone approval."*

For indie developers and product leaders, the core pursuit alongside extreme productivity is high-dimensional time freedom. Sitting in front of a terminal watching thousands of lines of rolling compile logs is not what Vibe Coding was meant to be.

In this chapter, we will build a **24/7 Mobile Sentinel Workflow**: combining **Guardian intelligent auto-approval** with a **mobile sentinel gateway** in 2026. Routine low-risk tasks proceed automatically, while high-risk production deployments push notification cards to your mobile Feishu or chat group, letting you review and approve anywhere, anytime with a single tap.

🎯 Intuitive Metaphor: Smart Patrol and Central Pager in a Modern Autonomous Farm

Think of offline orchestration as running a modern unmanned farm:

```
[Manual Guarding]      ──> Sitting on a stool inside the greenhouse 24 hours a day,
                           staring at irrigation pipes wondering if one might leak (tedious and
confining).
[Guardian + Gateway]   ──> The farm deploys an autonomous robotic watchdog (Guardian powered by
GPT-5.6 Luna):
                           - Minor pipe leak? The robot tightens the valve and moves on
(Guardian policy auto-pass);
                           - Main water gate switch or high-voltage grid changes? The robot
halts,
                             instantly sending a high-res photo and confirmation card to the
owner's phone;
                           - You tap "Approve" while sipping coffee on the train, and the farm
resumes autonomous operation.
```

This dual mechanism—"digesting routine risks automatically while holding humans at core checkpoints"—is the ultimate form of a modern company of one.

🚀 Beginner Quickstart (3 Easy Steps)

Set up your mobile alert notification channel in 3 simple steps:

1. **Step 1: Create a Feishu / Slack / WeCom Custom Bot**
   Add a bot in your group chat settings and copy its incoming Webhook URL.
2. **Step 2: Send a Test Alert Card from Terminal**
   Run curl to verify your phone receives notifications:

   ```
   curl -X POST -H "Content-Type: application/json" \
     -d '{"msg_type":"text","content":{"text":"🔔 Codex Mobile Sentinel Online: Terminal
   task running safely!"}}' \
     https://open.feishu.cn/open-apis/bot/v2/hook/YOUR-WEBHOOK-TOKEN
   ```

3. **Step 3: Launch Your Offline Task with Guardian + Sandbox**
   Before stepping away from your desk, run this command with peace of mind:

   ```
   codex exec --sandbox workspace-write --approve-for-me "Run full regression suite and
   refactor legacy types"
   ```

8.1 Two-Tier Sentinel Dispatch Architecture

In the 2026 architecture, your mobile phone is protected from notification bombardment:

```
[Codex Long Task] ──> Triggers sensitive action (dependency changes / file writes / network calls)
                                 │
                                 ▼
                  [Tier 1: Guardian Policy Review] ──(Low Risk)──> Auto-approved to proceed
                                 │ (High Risk / Production Deploy)
                                 ▼
                  [Tier 2: Mobile Sentinel Gateway] ──> [Mobile Feishu/Slack Card] ──> [Reply 1 to Approve]
```

Companion open-source references:

- Blue Book Official Feishu Assistant: plugins-codex-feishu
- Minimal Local Tunnel Gateway: scripts/codex-watchdog

8.2 Practice: GitHub Actions Build Failures and Webhook Setup

Create `.github/workflows/codex-watchdog.yml` in your project root. When a remote build fails, it summarizes the critical failure points and pushes an alert directly to your phone:

```yaml
name: Codex Agent Watchdog

on:
  push:
    branches: [ main ]
  workflow_dispatch:

jobs:
  agent-build:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout Code
        uses: actions/checkout@v4

      - name: Setup Node.js
        uses: actions/setup-node@v4
        with:
          node-version: 20

      - name: Run Codex Validation (exec mode)
        id: run_agent
        env:
          OPENAI_API_KEY: ${{ secrets.OPENAI_API_KEY }}
        run: |
          npm ci
          npm run test || echo "STATUS=failed" >> $GITHUB_ENV

      - name: Push Fail Notice to Mobile
        if: env.STATUS == 'failed'
        run: |
```

```
          curl -X POST -H "Content-Type: application/json" \
            -d '{
              "event": "build_failed",
              "repo": "${{ github.repository }}",
              "commit": "${{ github.sha }}",
              "message": "Codex sandbox testing failed. Attention required on phone."
            }' \
            ${{ secrets.MOBILE_WEBHOOK_URL }}
```

8.3 Mobile Bidirectional Interaction and Remote Approval

1. Scenario: Production Deployment Approval Gate

When Codex passes all tests and is ready to deploy code to Vercel, it pauses and sends an approval card to Feishu:

```
🚨 [Codex Auth Requested]
Project: pmer-cn-saas
Action: Deploy to production (Vercel)
Change summary: Implemented Stripe subscription webhook in /api/stripe.
Tests: 12 passed, 0 failed.
[Directive Command]: Reply "1" to approve deployment, "0" to abort and roll back.
```

2. Server-Side Minimal Relay Script (Node.js)

The gateway parses incoming replies and communicates with Codex via signal files or sockets:

```
// File: gateway.js
const express = require('express');
const { exec } = require('child_process');
const app = express();
app.use(express.json());

app.post('/api/mobile-reply', (req, res) => {
  const { userMessage, user } = req.body;
  if (user !== 'hunkwu') {
    return res.status(403).json({ error: 'Unauthorized' });
  }

  if (userMessage === '1') {
    exec('echo "approved" > /tmp/codex_deploy_signal', (err) => {
      if (err) return res.status(500).send('Error');
      res.json({ reply: '🚀 Deployment approved, production environment is going live!' });
    });
  } else if (userMessage === '0') {
    exec('pkill -f codex && git checkout -- .', (err) => {
      res.json({ reply: '🛑 Deployment aborted, code safely rolled back to HEAD!' });
    });
  } else {
    res.json({ reply: '⚠️ Invalid instruction. Reply 1 (approve) or 0 (abort).' });
  }
});

app.listen(8080, () => console.log('Mobile gateway listening on port 8080'));
```

8.4 Official Reference Implementation: Codex Feishu Sentinel

To eliminate the need for developers to manually build webhook gateways and polling logic, this book provides an out-of-the-box companion repository: **plugins-codex-feishu (Feishu Assistant)**.

It productizes this entire workflow into three primary capabilities:

1. **Daily Inspection & Sentinel Reports (Quick Start)**:
   - Automatically summarizes local Git commits and workspace status into rich notification cards sent directly to your personal Feishu chat.
   - Takes less than 3 minutes to verify with zero cognitive overhead.
2. **Mobile Alerts & Two-Way Approval (Offline Orchestration)**:
   - Vibrates your phone immediately whenever cloud/local Codex runs into testing failures or requests high-risk deployment authorization.
   - Reply directly on mobile to "Approve" or "Abort & Rollback".
3. **Knowledge Base Persistence (Team Collaboration)**:
   - Write project status into Feishu Docx documents and sync project metadata with Bitable multidimensional tables.

> 💡 ***3-Minute Quick Setup**: The project provides `feishu_app_manifest.json`, allowing you to create the complete Feishu app with pre-configured scopes and websocket events via a single JSON import in the Feishu Open Platform.*

🛡️ Troubleshooting & Pitfall Cheat Sheet

Common Pitfall	Root Cause	Rapid Diagnosis & Fix Guide
Phone does not receive Webhook messages	Bot security settings require keyword or IP whitelist	In bot settings under "Security", add matching keyword (e.g., `Codex`) or configure request signature
Replied "1" on phone but nothing happened locally	Relay gateway lost tunnel connectivity or signal file is unmonitored	Run `node scripts/codex-watchdog` to verify tunnel connectivity and inspect signal file status
Phone flooded with approval cards every minute	Minor edits routed directly to mobile without filtering	Enable `--approve-for-me` to delegate non-destructive routine actions to Guardian

---

8.5 Founder's Mantra: Reclaiming Your Freedom

Many tech practitioners using AI tools end up behaving like "manual testing monkeys" and "human git commit triggers." AI edits code, the human refreshes the tab; AI returns an error, the human copies the trace and pastes it back to the chat.

By delegating validation assertions to GitHub Actions, forwarding exceptions via mobile webhooks, and holding the deployment approval key on your mobile device, you can achieve the dream: **"enjoying your coffee while the product automatically evolves."**

Ch.09 Architecture Revitalization: Panoramic Analysis and Progressive Decoupling of Legacy

Systems

> 🎯 ***The Real Problem**: Inheriting tens of thousands of lines of undocumented legacy spaghetti code with zero tests, where editing one line breaks three unexpected modules.*
> 💡 ***Tangible Output & Takeaway**: GPT-5.6 Terra long-context panoramic topology prompts; interface behavioral snapshot baseline tests; and 3-step minimally invasive decoupling.*
> ⚡ ***Viral Screenshot Quote**: "Facing hundreds of thousands of lines of undocumented legacy code? Don't impulsively rewrite from scratch. Let AI map the topology, lock down regression tests, then perform microsurgery."*

When working on solo projects or inheriting legacy codebases, the biggest source of anxiety is taking over undocumented, test-free "spaghetti code" left behind by previous teams. Any minor code change can detonate hidden mines buried deep in the system.

When facing these systems, resist the urge to discard everything and rebuild from scratch. Powered by **GPT-5.6 Terra's ultra-long context reasoning** and sandboxed isolation testing in 2026, we can carry out textbook-grade "progressive minimally invasive surgery."

🎯 Intuitive Metaphor: Changing Tires on a Speeding Truck on the Highway

Refactoring an active legacy system is like this high-wire operation:

```
[Reckless Full Rewrite] ──> ❌ "This old truck is too beat-up; I'll build a brand-new truck on the shoulder!"
                            (The new build stalls for 6 months, while the business starves to death.)
[Progressive Decoupling] ──> ✅ The old truck continues hauling freight on the highway (zero business interruption):
                            1. Fly a drone under the chassis (AI scans full repository to generate a topology map);
                            2. Mount anti-rollover safety brackets (write behavioral snapshot tests for critical APIs);
                            3. Unscrew one rusty bolt at a time, test-drive it, and only proceed once verified green.
```

Codex acts as a millimeter-precision robotic surgical arm: as long as you wrap it in test guardrails, it excises dead code without nicking healthy tissue.

🚀 Beginner Quickstart (3 Easy Steps)

When taking over an unfamiliar, messy repository on Day 1, take these 3 safe steps:

1. **Step 1: Ask AI for a Full-Repo Topology Map (Read-Only Exploration)**
   Run a read-only sandboxed analysis to output a Mermaid dependency graph:

   ```
   codex exec --sandbox read-only "Analyze architecture and database models; output Mermaid dependency diagram in docs/topology.md"
   ```

2. **Step 2: Capture a "Behavioral Snapshot" for Target Endpoints (Baseline Recording)**
   Instruct Codex: "*Do not touch business code in `src/api/` ; only write 3 unit tests covering success and failure paths for the existing endpoints.*"
3. **Step 3: Single-Point Extraction Guarded by Test Assertions**
   Issue minimally invasive refactoring specs: "*Extract only one function at a time, running `npm test`*

automatically after each modification. All tests must stay 100% green."

9.1 Step 1: Panoramic Reverse Engineering - Generating Codebase Topology

The primary duty when taking over a chaotic project is **drawing the map**. Leverage GPT-5.6 Terra's long-context scanning to directly reverse-engineer Mermaid Entity-Relationship Diagrams (ERD) and route topologies:

```
# 🎯 Goal
Analyze the database configuration and routing layer of the current project, generating a
Mermaid ERD illustrating core table structures and foreign key relationships.

# 🛑 Constraints
- Only analyze prisma/schema.prisma or src/db/models/.
- Exclude temporary cache tables and third-party integration logs.

# 🧪 Validation Specs
- Output a syntactically valid Mermaid diagram in docs/database_topology.md.
```

Within seconds, Codex unpacks complex many-to-many relationships and foreign key cascades, far outstripping manual code audits.

9.2 Step 2: Safety Guardrails - Writing Behavioral Baseline Tests

The absolute golden rule of legacy code refactoring is: **Before performing surgery, hook up the vital-signs monitor (baseline tests).**

```
# 🎯 Goal
Write baseline unit tests for src/pages/api/checkout.ts, capturing real request inputs and
response snapshots.

# 🛑 Constraints
- Strictly forbid altering any logic inside src/pages/api/checkout.ts itself.
- Use local Jest / Vitest to execute the test suite.

# 🧪 Validation Specs
- Cover three core branches: successful checkout, out-of-stock error, and unauthenticated access
rejection.
- Ensure baseline tests pass 100% against the current legacy code.
```

9.3 Step 3: Minimally Invasive Surgery - Implementing Progressive Decoupling

1. Decoupling Fat Controllers

Gradually extract discount calculations, authentication, and inventory updates out of multi-hundred-line controller files into pure functions:

```
# 🎯 Goal
Extract "discount calculation logic" from src/pages/api/checkout.ts into a standalone service
src/services/discountService.ts.
```

```
# 🛑 Constraints
- External API Request / Response JSON schemas must maintain strict backward compatibility.
- Strictly avoid impacting other core logic branches inside checkout.ts.

# 🧪 Validation Specs
- Run npm run test to ensure the previously written checkout snapshot tests pass 100% green.
- Run npm run lint with zero type or syntax errors.
```

2. Local Validation and Safe Rollback

Protected by `--sandbox workspace-write`, Codex executes verification assertions immediately after code edits. The moment behavior breaks, revert instantly via Git:

```
# Instantly restore file on refactoring discrepancy
git checkout -- src/pages/api/checkout.ts
```

🛡️ Troubleshooting & Pitfall Cheat Sheet

Common Pitfall	Root Cause	Rapid Diagnosis & Fix Guide
Commanding AI to "refactor the entire project" causing total system collapse	Scope is too broad; AI suffers hallucinations and broken typings across large edits	Strictly constrain refactoring granularity; each prompt must touch only 1 specific service or function
Legacy functionality mysteriously lost or broken after refactoring	Missing pre-refactoring behavioral baseline snapshot tests	Enforce "test-first": require AI to write 100% passing baseline tests against old code before modifying logic
AI arbitrarily upgrades underlying framework packages in package.json	package.json was not locked down in constraints	Add hard rule to AGENTS.md & Constraints: "*Strictly forbid modifying any dependency versions in package.json*"

Ch.10 Monetization in Practice: Shipping a Commercial Next.js + Stripe SaaS MVP in 2 Hours

> 🎯 ***The Real Problem***: *Spending two weeks wrestling with authentication, database schemas, Stripe Webhook signature verification, and cloud hosting before shipping anything.*
> 💡 ***Tangible Output & Takeaway***: *Fully runnable production repo `examples/ch10-saas-mvp` (Next.js 15 + Supabase + Stripe subscriptions); Stripe CLI local payment test loops.*
> ⚡ ***Viral Screenshot Quote***: *"The ultimate milestone for indie developers isn't architectural perfection—it's receiving the first customer payment. Ship monetization in 2 hours."*

As an independent developer (Indie Hacker) or micro-startup founder, your core milestone is not building a "perfect architecture"—it is **"receiving your first payment."** Many developers waste weeks repeatedly configuring boilerplate setups, draining their momentum before ever launching.

In this chapter, in a fast-paced hacker style, we will teach you how to direct Codex to ship a SaaS MVP with a complete payment and subscription access control loop in under 2 hours using `Next.js 15 (App Router) + Supabase (PostgreSQL) + Stripe`.

> 📦 ***Companion Source Code***: *examples/ch10-saas-mvp — a fully runnable subscription-based AI translator (TransFlow) with its own CAP `AGENTS.md`. Verified with `npm install && npm run build`.*

🎯 Intuitive Metaphor: Launching a Cash-Generating Street Food Cart

Too many founders try to build a 5-star luxury hotel on Day 1:

```
[Daydreaming 5-Star Hotel] ──> Spending 6 months designing an opulent lobby, importing plush
carpets,
                               hiring dozens of waiters (over-engineering),
                               only to open doors and discover nobody wants to eat there.
Bankruptcy.
[Mobile Street Food Cart]  ──> ✅ You only need 3 essential tools:
                               1. The Griddle (Next.js core product page: delivers immediate
customer value);
                               2. The Storage Box (Supabase / PostgreSQL: stores user accounts
and orders);
                               3. The QR Payment Code (Stripe subscription: collects real money
into your bank).
```

Once a customer scans the code, pays $10, and receives a hot meal in their hands, your commercial loop is validated!

🚀 Beginner Quickstart (3 Easy Steps)

Set up and verify your local Stripe payment loop in 3 simple steps:

1. **Step 1: Obtain Stripe Test Keys and Declare in `.env.local`**
   Log into the Stripe Developer Dashboard, copy your test keys, and write them into `.env.local`:

   ```
   STRIPE_SECRET_KEY="sk_test_..."
   NEXT_PUBLIC_STRIPE_PUBLISHABLE_KEY="pk_test_..."
   ```

2. **Step 2: Start Stripe CLI Local Webhook Forwarding**
   Run the forwarding listener in your terminal to open a local tunnel:

   ```
   stripe listen --forward-to localhost:3000/api/webhooks/stripe
   # The terminal will print your signing secret: whsec_...
   ```

3. **Step 3: Direct Codex in Sandbox to Validate Webhook Signatures**
   Inject `whsec_...` into the environment and instruct Codex: "*Write automated test cases mocking `checkout.session.completed` ; it must return 200 OK and update user subscription status to ACTIVE.*"

10.1 Initialization and Database Schema Design

Our goal is to build a subscription-based AI translation service. First, initialize the project with Next.js 15 and configure core constraints. Next, instruct Codex to generate the database models.

1. Designing the Prisma Schema (Database Entity Modeling)

Dispatch the following goal-driven specs to Codex:

```
# 🎯 Goal
Write Prisma database models supporting User, Subscription, and TranslationRecord entities.
```

```
# 🛑 Constraints
- Use PostgreSQL (connected via Supabase) as the database provider.
- Subscription status must be an Enum type containing ACTIVE, CANCELED, and EXPIRED.

# 🧪 Validation Specs
- Running `npx prisma validate` must return no syntax or definition errors.
```

Codex will automatically output a standard `prisma/schema.prisma` file containing foreign key relationships, cascade deletes, and database indexes:

```
datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}

generator client {
  provider = "prisma-client-js"
}

enum SubscriptionStatus {
  ACTIVE
  CANCELED
  EXPIRED
}

model User {
  id             String               @id @default(uuid())
  email          String               @unique
  createdAt      DateTime             @default(now())
  subscription   Subscription?
  records        TranslationRecord[]
}

model Subscription {
  id             String               @id @default(uuid())
  userId         String               @unique
  stripeSubId    String               @unique
  status         SubscriptionStatus
  priceId        String
  currentPeriodEnd DateTime
  user           User                 @relation(fields: [userId], references: [id], onDelete:
Cascade)
}

model TranslationRecord {
  id             String               @id @default(uuid())
  userId         String
  sourceText     String
  translatedText String
  createdAt      DateTime             @default(now())
  user           User                 @relation(fields: [userId], references: [id], onDelete:
Cascade)
}
```

10.2 Integrating Stripe Subscriptions and Webhook Handlers

The core of a payment system is **callback security**. When a user successfully checks out, Stripe's servers send a webhook request to your Next.js application.

Practice: Dispatching Webhook Route Specifications to Codex

```
# 🎯 Goal
Implement a Next.js 15 App Router style Stripe Webhook route handler `/api/webhooks/stripe`.

# 🛑 Constraints
- Verify the signature of incoming webhook requests using `stripe.webhooks.constructEvent` to prevent forgery attacks.
- Update user subscription status in the database on receiving `checkout.session.completed` or `invoice.payment_succeeded` events.
- Strictly avoid JSON parsing on the request body; Stripe verification requires the raw `req.text()` string.

# 🧪 Validation Specs
- Write unit tests verifying that invalid signatures return 400, while valid signatures return 200.
- The actual subscription billing period must be retrieved from the Stripe response, never hardcoded.
```

Codex generates the compliant handler:

```typescript
// File: src/app/api/webhooks/stripe/route.ts
import { NextResponse } from 'next/server';
import Stripe from 'stripe';
import { prisma } from '@/lib/prisma';

const stripe = new Stripe(process.env.STRIPE_SECRET_KEY!, {
  apiVersion: '2026-04-22.dahlia' as any,
});

export async function POST(req: Request) {
  const body = await req.text();
  const signature = req.headers.get('stripe-signature')!;

  let event: Stripe.Event;

  try {
    event = stripe.webhooks.constructEvent(
      body,
      signature,
      process.env.STRIPE_WEBHOOK_SECRET!
    );
  } catch (err: any) {
    return NextResponse.json({ error: `Webhook Error: ${err.message}` }, { status: 400 });
  }

  if (event.type === 'checkout.session.completed') {
    const session = event.data.object as Stripe.Checkout.Session;
    const stripeSubId = session.subscription as string;
    const customerEmail = session.customer_details?.email!;
```

```
    const subscription = await stripe.subscriptions.retrieve(stripeSubId);

    await prisma.subscription.upsert({
      where: { stripeSubId },
      update: {
        status: 'ACTIVE',
        currentPeriodEnd: new Date(subscription.current_period_end * 1000),
      },
      create: {
        stripeSubId,
        status: 'ACTIVE',
        priceId: subscription.items.data[0]?.price.id || 'default',
        currentPeriodEnd: new Date(subscription.current_period_end * 1000),
        user: { connect: { email: customerEmail } }
      }
    });
  }

  return NextResponse.json({ received: true });
}
```

10.3 Local Debugging: Using Stripe CLI for Payment Integration

1. Run Stripe forwarding locally:

```
stripe listen --forward-to localhost:3000/api/webhooks/stripe
```

2. Trigger a real checkout event:

```
stripe trigger checkout.session.completed
```

3. Observe terminal output for `200 OK`, and inspect the database via Prisma Studio:

```
npx prisma studio
```

The value of a commercial MVP lies in delivery speed. Enforce strict boundary specs on the AI to achieve the fastest time-to-revenue.

🛡️ Troubleshooting & Pitfall Cheat Sheet

Common Pitfall	Root Cause	Rapid Diagnosis & Fix Guide
`Webhook Error: No signatures found matching the expected signature`	Used `await req.json()` to parse body, corrupting raw payload bytes	Use `await req.text()` to get the raw string before passing to `constructEvent`
Credit card payment succeeds but database has zero new records	Webhook URL misconfigured or `stripe listen` forwarder not running locally	Run `stripe listen`, confirming listener terminal prints `200 OK [POST /api/webhooks/stripe]`
`DATABASE_URL` cannot connect to Supabase	Forgotten connection pooler (confusing Session vs.	Verify DB connection string; ensure port 6543 (Transaction mode) is used in

	Transaction ports)	Serverless environments

Ch.11 Mobile Extension: Expo Cross-Platform App Development and Cloud Packaging

> 🎯 ***The Real Problem***: *Web developers stuck in Xcode certificate signing, Android Gradle builds, and CocoaPods dependency hell when trying to ship mobile apps.*
> 💡 ***Tangible Output & Takeaway***: *Complete companion project* `examples/ch11-expo-mobile` *(Expo SDK 57 + Expo Router + NativeWind); zero-local-setup* `eas build` *cloud packaging pipelines.*
> ⚡ ***Viral Screenshot Quote***: "*Skip local Xcode and Android Studio configuration hell. Build and publish dual-platform native apps autonomously with cloud pipelines and AI error self-healing.*"

After shipping a web SaaS, many independent developers want to extend their reach to mobile. However, in traditional mobile development (React Native or Flutter), the most grueling friction is local environment setup: iOS certificates, Android Gradle build failures, and CocoaPods version conflicts.

I firmly believe that **"cloud compilation and packaging (EAS) is the only viable path for independent developers to build native apps."** Combined with Codex's automated diagnostic assistance, you can bypass local Xcode/Android Studio configuration entirely and ship production-ready native apps directly to app stores.

> 📦 ***Companion Source Code***: *examples/ch11-expo-mobile — a fully runnable Expo SDK 57 project (Expo Router* `src/app` *routing + NativeWind + three-tier EAS build profiles) with its own CAP* `AGENTS.md`*. Verified with* `npx expo lint` *(zero errors) and* `npx expo-doctor` *(20/20 checks passed).*

🎯 Intuitive Metaphor: Writing the Script and Letting the "Cloud Atelier" Tailor the Costumes

Cross-platform development shouldn't require turning your personal laptop into a heavy manufacturing plant:

```
[Traditional Native Dev] ──> Buying your own smelting furnace (installing 50GB+ Xcode and
Android Studio),
                             configuring loom machinery (fiddling with Gradle and CocoaPods
runtimes),
                             frequently tripping circuit breakers and stalling for days without
a working build.
[Expo + EAS Mode]        ──> ✅ You only focus on the script (writing React Native / TypeScript
code):
                             - Once the script is ready, send it to the "Cloud Custom Atelier"
(EAS Build);
                             - The cloud automatically tailors iOS IPAs and Android APKs;
                             - Scan the QR code on your phone to slip into the customized outfit
instantly!
```

Codex serves as your script editor in the atelier, automatically aligning versions whenever you miss a dependency or styling quirk.

🚀 Beginner Quickstart (3 Easy Steps)

See your cross-platform app running on a physical phone in 3 simple steps:

1. **Step 1: Install Expo Go on Your Mobile Device**
   Search for and install "Expo Go" from the App Store or Google Play.

2. **Step 2: Start the Local Development Server**
   Navigate to the project directory and run:

   ```
   npx expo start
   # The terminal will display a large QR code matrix
   ```

3. **Step 3: Scan the QR Code with Your Phone Camera**
   Scan the code to load the app immediately. Whenever you or Codex modify page code, the phone screen hot-reloads instantly!

11.1 Expo Project Rapid Initialization and Standards Setup

To use EAS for zero-configuration cloud packaging, initialize a standard Expo project:

1. Writing App Initialization Specs

Issue the following specs to Codex:

```
# 🎯 Goal
Initialize a React Native Expo project using TypeScript.

# 🛑 Constraints
- Use the latest stable Expo SDK 57, paired with Expo Router for file-system-based routing.
- Integrate NativeWind as the Tailwind-style CSS styling solution.
- Use the src/ prefix convention (i.e., src/app/ as the routing root).

# 🧪 Validation Specs
- Running `npx expo lint` must return zero errors.
- Run `npx expo-doctor` to verify dependency version alignment (must pass 20/20).
```

Directory structure layout:

```
src/
├── app/
│   ├── index.tsx         # App Home Page
│   └── _layout.tsx       # Global Routing Navigation Layout
├── components/
└── hooks/
app.json                  # Expo Core Configuration File
package.json
```

11.2 Resolving Mobile Obstacles: Native Module Conflicts

React Native development must avoid using standard `npm install` for third-party packages containing underlying native code (such as cameras, sensors, or gesture handlers).

Golden Rule: Use `npx expo install` for Version Alignment

If dependency versions drift, type this correction directly in the TUI:

```
Please use `npx expo install react-native-reanimated` to reinstall this dependency instead. It
will automatically adapt to the current Expo SDK version. In this project, using standard `npm
```

```
install` to install any native package is strictly prohibited. Please append this rule to
AGENTS.md.
```

> 💡 ***Founder's Mantra***: *`npx expo install` is the official certified version lock. Whenever native modules fail, force reinstalling with this command automatically wipes out 90% of dependency issues.*

11.3 EAS Build Cloud Packaging and Certificate Automation

In traditional app delivery, obtaining Apple developer certificates can stall newcomers for days. With EAS, this entire process is handled in the cloud.

1. Configuring `eas.json`

```
{
  "cli": {
    "version": ">= 9.0.0"
{{ ... }}
  }
}
```

2. Instructing Codex to Monitor Cloud Build Logs

```
# Start EAS iOS build task and stream logs to a local file
eas build --platform ios --profile production --non-interactive 2>&1 | tee eas-build.log
```

If the build hits an error, hand the logs directly to Codex for analysis:

```
codex exec --sandbox read-only "Analyze eas-build.log, locate failure cause, and provide
remediation commands"
```

Ultimately, EAS returns an installation QR code that you can scan to install the preview app directly on your phone.

🛡️ Troubleshooting & Pitfall Cheat Sheet

Common Pitfall	Root Cause	Rapid Diagnosis & Fix Guide
Phone scans QR code and shows "Could not connect to development server"	Phone and computer are not on the same Wi-Fi LAN, or router firewall blocks local traffic	Run `npx expo start --tunnel` to force public tunnel mode with zero network barriers
`Invariant Violation: "main" has not been registered`	Entry routing file missing or incorrect entry path in `app.json`	Verify `"main": "expo-router/entry"` in `package.json` was not accidentally modified
Terminal throws red screen error immediately after installing a new package	Accidental use of `npm install` introducing an incompatible SDK version	Run `npx expo install <package-name>`, or execute `npx expo-doctor` to diagnose and align

---

Ch.12 The Final Frontier: Building an Automated Growth Flywheel for a One-Person SaaS

> 🎯 ***The Real Problem***: *Spending 99% of effort writing code and 1% acquiring users, launching to crickets because a solo creator cannot balance engineering and growth.*
> 💡 ***Tangible Output & Takeaway***: *Automated marketing scripts (event-triggered telemetry digests, social distribution); core analytics dashboards; one-person SaaS growth pipelines.*
> ⚡ ***Viral Screenshot Quote***: *"Flawless code with zero users is useless. Let AI not only build your product, but also drive your automated distribution flywheel."*

On my WeChat public account "Real-World Product Talk" and pmer.cn, I have written numerous articles about "independent development and side hustles." The most common trap developers fall into is: **spending 99% of their energy polishing code syntax, but only 1% of their energy finding real users and actual pain points.**

No matter how elegant your code is or how perfect your architecture config is, as long as nobody uses it, it is just a pretty ornament. In the AI-native era, we must not only let Codex help us "manufacture products," but also let it help us "spin the commercial flywheel."

🎯 Intuitive Metaphor: Building a Day-and-Night Self-Spinning "Hydraulic Irrigation Waterwheel"

A commercial flywheel shouldn't depend on hauling water buckets by hand every day:

```
[Manual Bucket Hauling] ──> Writing code until midnight, then handing out flyers manually the
next morning.
                            If you get sick for a single day, product updates stall and traffic
drops to zero (fragile).
[Hydraulic Waterwheel]  ──> ✅ You assemble this self-spinning waterwheel by the river:
                            1. The Flume (SEO & automated sitemaps: continuously absorbing long-
tail search traffic);
                            2. The Milling Blades (Core SaaS features: solving real problems for
real users);
                            3. The Grain Funnel (Stripe checkout & subscriptions: continuously
generating cash flow);
                            4. The Water Level Gauge (Daily telemetry cards: monitoring
conversion rates and active users).
```

Once this pipeline is assembled, whether you are sleeping, eating, or traveling, the waterwheel spins day and night, generating compounding value.

🚀 Beginner Quickstart (3 Easy Steps)

Take your first step toward commercialization in 3 simple steps:

1. **Step 1: Deploy an Automated Sitemap Generation Script**
   Run `scripts/generate-sitemap.js` automatically in your deployment pipeline to ensure every new markdown post or route gets indexed by Google.
2. **Step 2: Hook Up an 8:00 AM Daily Revenue Telemetry Digest**
   Configure a scheduled script that pushes yesterday's new signups and active subscriptions to your phone every morning, sharpening your business sensitivity.

3. **Step 3: Post Your "One-Sentence Value Proposition" in Public Communities**
   Share your MVP link on X (Twitter), Reddit, or developer forums to gather real payment signals from your first seed cohort.

12.1 Automated Traffic Pipeline for a "One-Person SaaS"

For a healthy indie project, traffic acquisition (SEO, long-tail keywords, social media) should function as an automated conveyor belt just like its codebase.

Practice: Directing Codex to Autonomously Maintain SEO Blogs and Sitemaps

```
// File: scripts/generate-sitemap.js
const fs = require('fs');
const path = require('path');

const BASE_URL = 'https://pmer.cn';
const blogDir = path.join(__dirname, '../content/blog');

function getBlogSlugs() {
  if (!fs.existsSync(blogDir)) return [];
  return fs.readdirSync(blogDir)
    .filter(file => file.endsWith('.md'))
    .map(file => `/blog/${file.replace('.md', '')}`);
}

function generate() {
  const staticPages = ['/', '/auth/login', '/features'];
  const blogPages = getBlogSlugs();
  const allUrls = [...staticPages, ...blogPages];

  const xml = `<?xml version="1.0" encoding="UTF-8"?>
<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
  ${allUrls.map(url => `
  <url>
    <loc>${BASE_URL}${url}</loc>
    <changefreq>daily</changefreq>
    <priority>${url === '/' ? '1.0' : '0.8'}</priority>
  </url>`).join('')}
</urlset>`;

  fs.writeFileSync(path.join(__dirname, '../public/sitemap.xml'), xml);
  console.log('✅ Sitemap.xml generated successfully!');
}

generate();
```

12.2 Connecting Business Data for Daily Telemetry

To keep yourself sensitive to cash flow dynamics, fetch Stripe earnings and user growth every morning and broadcast a report straight to your phone:

```
// File: scripts/daily-report.js
const { PrismaClient } = require('@prisma/client');
const prisma = new PrismaClient();
```

```
const axios = require('axios');

async function sendReport() {
  const yesterday = new Date(Date.now() - 24 * 60 * 60 * 1000);
  const userCount = await prisma.user.count({
    where: { createdAt: { gte: yesterday } }
  });

  const activeSubs = await prisma.subscription.count({
    where: { status: 'ACTIVE' }
  });

  const reportText = `📊 [Real-World Product Briefing]\nNew registered users yesterday:
${userCount}\nTotal active subscriptions: ${activeSubs}\n—— Keep going!`;

  await axios.post(process.env.MOBILE_WEBHOOK_URL, {
    msg_type: 'text',
    content: { text: reportText }
  });
}

sendReport();
```

12.3 The Final Moat: The Only Barrier in the AI-Native Era

When anyone can generate thousands of lines of code in two hours, build a native mobile app, and hook up automated payment pipelines, **pure code-writing becomes completely commoditized**. In this new era of "infinite code," the ultimate moat for independent developers and product managers lies in:

1. **Your deep empathy for user pain points (User Empathy)**.
2. **Your domain expertise accumulated over years in a specific industry (Domain Knowledge)**.
3. **Your execution speed in orchestrating AI agents to rapidly validate commercial loops.**

🛡️ Troubleshooting & Pitfall Cheat Sheet

Common Pitfall	Root Cause	Rapid Diagnosis & Fix Guide
Product launched with zero traffic for a month	Siloed development without early SEO or social seeding	Share your building journey on X/Twitter/Indie Hackers immediately; use scripts to output multilingual long-tail blog posts
Obsessing over refactoring and afraid to ship or tweet	Perfectionist trap and fear of public criticism	Remember: as long as core checkout works, ship minor flaws and iterate based on real feedback
Wasting half a day on minor layout tweaks	Failing to delegate visual micro-adjustments to Codex	Use Computer Use (from Ch.07) for automated visual verification, freeing your time to talk to paying customers

---

Ch.13 Frontier Watch: The 2026 Codex Ecosystem Overhaul

> 🎯 ***The Real Problem***: *Rapid tool churn, deprecated commands (Codex desktop merging into ChatGPT code mode), model evolutions (GPT-5.6), and unexpected CLI breaking changes.*
> 💡 ***Tangible Output & Takeaway***: *2026 migration command battle map (winget/brew setups, CLI 0.14x configurations, Plugins ecosystem integration), and compatibility test scripts.*
> ⚡ ***Viral Screenshot Quote***: *"Tools retool every quarter, but mental models remain your moat. Zero fluff—only an actionable battle map of what deprecated and what to adopt."*

The first twelve chapters built a version-independent orchestration methodology. But methodology must land on real tooling. In 2026, the Codex ecosystem went through four structural shifts: **the desktop merger, the model transition, the maturing plugin economy, and security becoming its own product line**. This chapter breaks down each shift with concrete migration commands and configurations.

🎯 Intuitive Metaphor: Avionics Retrofit of a Supersonic Airliner

Do not panic over frequent tool upgrades. View this transition through the eyes of a pilot:

```
[Pilot Mindset]   ──> Your flight manual remains identical: goal-driven execution (takeoff &
landing targets)
                      and strict boundary constraints (flight corridors & safe altitudes).
[Engines]         ──> Propulsion overhauled to next-generation powerplants:
                      - Frontier Flagship Core: "GPT-6 Astra" (beyond-visual-range radar &
autonomous computer operator);
                      - Daily Primary Thrust Engine: "GPT-5.6 Terra" (balanced deep reasoning);
                      - Auxiliary Cruise Engine: "GPT-5.6 Luna" (Guardian auto-approval & low-
latency checks).
[Cockpit Screen]  ──> The former standalone secondary monitor (separate Codex App) is officially
integrated
                      into the main instrument panel (ChatGPT Desktop Code Mode), adding Sites
radar & Annotations.
[Autopilot]       ──> Old simple cruise control (--full-auto) upgraded to an intelligent co-
pilot
                      with flight corridor self-checking (--sandbox workspace-write + Guardian).
```

With your mental model intact, upgraded avionics only help you fly faster and steadier.

🚀 Beginner Quickstart (3 Easy Steps)

Complete your 2026 toolchain health check and migration in 3 simple steps:

1. **Step 1: Scan for Deprecated Legacy Model Codenames**
   Run a grep scan in your project root:

   ```
   grep -rn "gpt-5.4" ~/.codex/config.toml .
   ```

2. **Step 2: Globally Upgrade to CLI 0.147.0+ Stable Release**
   Run the global npm update:

   ```
   npm install -g @openai/codex@latest
   codex --version
   ```

3. **Step 3: Eliminate the Deprecated `--full-auto` Flag**
   In all your shell aliases and CI scripts, replace `--full-auto` with:

```
codex exec --sandbox workspace-write "<task>"
```

13.1 The Desktop Merger: Codex App Folds into the ChatGPT Client

In July 2026, the standalone Codex desktop app was retired. All of its capabilities moved into the **ChatGPT desktop client** as a dedicated "Codex Code Mode," available to Free, Plus, and Enterprise users.

Migration steps:

```
# macOS: download ChatGPT.dmg and drag it into Applications
# Windows: install via winget
winget install OpenAI.ChatGPT
```

After signing in, click the **Codex** tab in the left sidebar to enter Code Mode. Two caveats:

1. **Account vs. API Key**: Signing in with a ChatGPT account unlocks the full feature set (cloud tasks, cross-device sync, Sites hosting, Computer Use); an API key grants only basic local coding.
2. **Standalone components are unaffected**: the Codex CLI, the VS Code extension, and Codex Cloud continue to evolve independently. The multi-surface matrix from Ch.02 still holds — the "desktop app" cell is simply replaced by the ChatGPT client.

New capabilities at a glance:

- Upgraded built-in browser: search browsing history from the address bar; the Chrome extension can reference open tabs.
- **Multi-repo Review**: multi-folder projects show changed lines across all repositories in one review view — no more tab-hopping between diffs.
- **Sites**: create, deploy, and manage hosted web projects directly in the client, with Annotations for in-place "point and edit" workflows.

13.2 The Model Transition: GPT-6 Astra & the 2026 Landscape

In late 2026, OpenAI fundamentally restructured its model strategy away from single-model patches toward a persistent tiered architecture: introducing the frontier autonomous agent **GPT-6 Astra**, while solidifying the **GPT-5.6 family** into a clear three-tier capability hierarchy (Sol / Terra / Luna).

1. The Next-Gen Frontier Flagship: GPT-6 Astra (Launched September 3, 2026)

On September 3, 2026, OpenAI officially unveiled its most intelligent and aligned model to date: **GPT-6 Astra** (API model identifier: `gpt-6-astra`). Key specifications and capabilities include:

- **1.05 Million Token Context & 128k Output**: Natively supports a **1,050,000 token context window** with **128,000 maximum output tokens**, eliminating context truncations for massive monolithic repositories and multi-app codebases.
- **Native Computer Operator**: Purpose-built for **Computer Use**, Astra operates software like a human engineer—directly inspecting screen pixels, clicking GUI controls, navigating browsers, and executing end-to-end coding workflows across desktop applications.
- **Benchmark Dominance**: Set state-of-the-art records across software engineering (SWE-bench) and advanced mathematics (FrontierMath).
- **Safety Classification**: First OpenAI model rated at the "Critical" cybersecurity capability level under the Preparedness Framework, featuring advanced automated vulnerability discovery and reverse-engineering capabilities under gated access.

2. The 2026 Active Model Decision Matrix

Avoid the anti-pattern of blindly defaulting to the most expensive tier. Distribute tasks systematically:

Model ID	Public Tier Name	Core Positioning & Use Cases	Selection Guidance
`gpt-6-astra`	**GPT-6 Astra**	Frontier Agentic Flagship / Native Computer Operator	Multi-application desktop workflows, ultra-long context refactoring, high-fidelity visual audits
`gpt-5.6-sol` / `gpt-5.6`	**GPT-5.6 Sol**	High-Reasoning Architecture Flagship (Daybreak Blue)	Distributed system design, deep legacy refactoring, mission-critical security defenses
`gpt-5.6-terra`	**GPT-5.6 Terra**	Balanced Daily Coding Workhorse	Standard for everyday professional development; optimal balance of depth, latency, and cost
`gpt-5.6-luna`	**GPT-5.6 Luna**	Ultra-Fast, Low-Latency Guardian & Triaging	High-frequency micro-edits, test routing, automated Guardian (`--approve-for-me`) approvals

3. 2026 OpenAI Model Deprecation & Sunset Schedule

Throughout 2026, OpenAI executed a rigorous deprecation timeline. Audit your configurations to prevent production downtime:

Date	Action Type	Affected Models / Services	Impact & Official Replacement
March 26, 2026	Shutdown	`gpt-4-0314`, `gpt-4-1106-preview`, `gpt-4-0125-preview`	Legacy GPT-4 snapshots shut down; migrate to `gpt-5.6-terra`
May 12, 2026	API Shutdown	DALL-E 2 / DALL-E 3	Image generation APIs retired; migrate to `gpt-image-1` / `gpt-image-2`
June 2026	ChatGPT Retired	`o3`, `gpt-4.5`, early `gpt-5` thinking snapshots	Removed from ChatGPT UI; consolidated onto GPT-5.6 / 6 tiers
August 26, 2026	API Shutdown	Legacy Assistants API	Deprecated older architecture; superseded by Responses API & native agents
August 31, 2026	Codex Retired	`gpt-5.4`, `gpt-5.4-mini`	Retired from Codex ChatGPT login mode; replaced by `gpt-5.6-terra` / `luna`
September 3, 2026	General Launch	**GPT-6 Astra (`gpt-6-astra`)**	Frontier flagship launch with 1.05M context and native Computer Use
October 23, 2026	Scheduled Sunset	Remaining legacy Preview snapshot models	Batch shutdown of preview tags; developers must pin stable model IDs
December 1, 2026	Scheduled Sunset	`gpt-image-1-mini`, `gpt-image-1.5`, `chatgpt-image-latest`	Image models fully consolidated onto `gpt-image-2`

Recommended Configuration:

```
# ~/.codex/config.toml

# Default daily development engine (Terra: optimal balance of depth & speed)
model = "gpt-5.6-terra"
```

```toml
# Optional: Frontier autonomous agent & Computer Use (Astra)
# model = "gpt-6-astra"

# Fast automated approvals and guardian sentinels (Luna: low cost & sub-second latency)
[profiles.guardian]
model = "gpt-5.6-luna"

# High-stakes architectural refactoring & security defense (Sol: peak reasoning)
[profiles.architect]
model = "gpt-5.6-sol"
```

> ⚠️ ***Warning**: Audit all scheduled Automations, workspace default profiles, and CI scripts. Remove all hardcoded references to retired `gpt-5.4` or temporary `preview` models.*

13.3 CLI 0.14x: The Breaking Changes You Must Know

The Codex CLI is now in the `0.14x` era (latest stable: `0.147.0+`).

```bash
npm install -g @openai/codex@latest
```

1. `--full-auto` Is Officially Removed

```bash
# ❌ Old syntax (now errors out)
codex exec --full-auto "fix all lint errors"

# ✅ New syntax: express autonomy via sandbox mode
codex exec --sandbox workspace-write "fix all lint errors"
```

2. The Hooks Engine Graduates to Stable

Configure `SessionStart` / `Stop` hooks directly in `config.toml`, observing MCP tool calls, `apply_patch`, and long-running Bash sessions:

```toml
# ~/.codex/config.toml
[hooks]
session_start = "bash scripts/bootstrap-env.sh"
stop = "npm run lint --silent"
```

3. Agent Plugins and Marketplaces

Support for local, personal, workspace, and remote catalogs:

```bash
# Unified plugin management entry point
codex plugin list
codex plugin marketplace add https://plugins.example.com/catalog.json
codex plugin install security-workbench
```

Mention a plugin in chat with `@plugin` to auto-inject its context.

4. Subagents and Multi-Agent Orchestration

The CLI natively spawns subagents with independent context windows:

```
> Spawn two subagents: one adds integration tests for src/api,
  another refactors state management in src/hooks in parallel.
  Aggregate the diffs for my review at the end.
```

5. Guardian Auto-Approval and `--approve-for-me`

Human confirmation is no longer the only path for high-risk operations. The new `--approve-for-me` flag routes approval requests to a Guardian subagent (powered by GPT-5.6 Luna):

```
codex --approve-for-me "upgrade dependencies and make tests pass"
```

13.4 The Skills Economy: Compounding Repeated Workflows into Assets

If AGENTS.md is a project's "constitutional law," **Skills are the standard operating procedures for specific task types**:

```
skills/
└── pr-review/
    └── SKILL.md      # A procedural manual with YAML frontmatter
```

13.5 Codex Security and Daybreak: Security Becomes a Product Line

For enterprise defense and mission-critical infrastructure, OpenAI established the **Daybreak** defense architecture and official security plugins:

- **Daybreak Blue**: general-purpose engineering defense and automated patch generation powered by the high-reasoning **GPT-5.6 Sol**.
- **Daybreak Red / Frontier Eval**: red-teaming offensive evaluation and penetration testing powered by the frontier flagship **GPT-6 Astra** (gated under the Critical cybersecurity capability framework), delivering end-to-end zero-day discovery and automated binary reverse-engineering.

```
codex plugin install codex-security
codex-security scan --deep --report sarif > results.sarif
```

13.6 Spatial Intelligence & 3D Cross-Domain Frontiers: GPT-6 Astra with Blender and Tripo3D

As frontier models evolved from pure text and 2D web code toward spatial physical reasoning, the second half of 2026 unleashed a **cross-disciplinary revolution** across 3D modeling, spatial computing, and game engineering.

1. Industry Breakthrough: From 2D Code to Spatial Geometry

In September 2026, OpenAI demonstrated GPT-6 Astra's spatial geometry capabilities (tweet #2100679992720142459) and unveiled **BenchCAD**—where Astra achieved a record-breaking **95.9% mean voxel IoU** in multi-view CAD code reconstruction. This shattered the long-standing limitation that large language models lack three-dimensional spatial intuition.

Global 3D generation leader **Tripo3D** swiftly announced deep integration, launching a dedicated **GPT-6 Astra 3D Prompt Engineering Center**. Indie developers can now leverage an ultra-lean pipeline to accomplish 3D modeling, lighting setup, and material baking in minutes instead of weeks.

2. The "Dual-Engine" Architectural Mental Model

Solo developers must discard a common misconception: expecting an LLM to generate raw 3D mesh files containing tens of thousands of polygons from pure text is inefficient. The state-of-the-art production workflow relies on **complementary synergy between GPT-6 Astra, native 3D generation (Tripo3D), and professional DCC software (Blender)**:

```
[ Creative Concept & Spatial Requirements ]
                    │
                    ▼
[ GPT-6 Astra (3D Art Director & Technical Artist) ] ──> Structured Prompt Expansion (Topology / PBR / Scaling)
                    │
                    ▼
[ Tripo3D (Native 3D Geometry & Material Engine) ]   ──> Generates Continuous Mesh (Quad Topology, UVs, PBR GLB)
                    │
                    ▼
[ Blender + bpy / Blender MCP (Astra Auto-Control) ] ──> Origin Centering, 3-Point Lighting, Viewport Regression
                    │
                    ▼
[ Frontend Web / Game Engine (Three.js / React Three Fiber) ] ──> Zero-Cost Interactive Distribution
```

- **Tripo3D (Native Geometry & Material Synthesis)**: Trained on hundreds of millions of 3D meshes, it generates clean, quad-dominant watertight meshes with PBR maps (Albedo / Roughness / Metallic / Normal) from single images or text prompts.
- **GPT-6 Astra (Technical Director & Automation Pipeline Hub)**: Acts as the Lead Technical Artist (TA). It structures prompts, manages project specs, and takes over Blender via Python (`bpy`) or **Blender MCP** to automate asset centering, studio lighting, camera rigs, and autonomous visual inspection.

3. Step-by-Step Production Walkthrough

Step 1: Structure 3D Prompts via GPT-6 Astra

Instruct Astra in your terminal or desktop prompt:

```
Act as a Lead 3D Technical Artist. I want to build a "Cyberpunk Holographic Vending Machine" for a Web 3D product showcase.
Generate an engine-optimized, structured 3D prompt for Tripo3D.

Requirements:
1. Emphasize silhouette and geometric symmetry;
2. Enforce mesh topology tags (Clean quad topology, game-ready, low-poly);
3. Define PBR material specifications (Matte painted metal, glowing neon emissive panels).
```

Astra outputs a production-ready prompt:

```
Cyberpunk holographic vending machine, freestanding rectangular kiosk with chamfered edges.
Center features transparent acrylic dispenser bay with internal glowing LED strip.
Materials: Brushed matte dark gray titanium alloy (roughness: 0.35, metallic: 0.85),
glowing cyan neon emissive trim lines (emission strength: 5.0), scratched hazard decals on base.
Topology tags: clean quad-dominant topology, watertight mesh, manifold geometry, PBR 4K
textures, game-ready asset.
```

Step 2: Generate & Export Standardized GLB via Tripo3D

Paste the refined prompt into Tripo3D. Within 30 seconds, Tripo3D synthesizes the mesh, UV layout, and textures. Download the asset as **`vending_machine.glb`** and save it into `public/models/`.

Step 3: Automate Blender via GPT-6 Astra (`pipeline.py`)

Instead of wrestling with manual GUI dials, have Astra write and execute a headless Blender Python automation script:

```
import bpy
import math

# 1. Clean factory environment
bpy.ops.wm.read_factory_settings(use_empty=True)

# 2. Import Tripo3D GLB asset
asset_path = "./public/models/vending_machine.glb"
bpy.ops.import_scene.gltf(filepath=asset_path)

# 3. Compute bounding box, center origin, and ground asset (Z=0)
imported_objs = [obj for obj in bpy.context.selected_objects if obj.type == 'MESH']
if imported_objs:
    bpy.ops.object.select_all(action='DESELECT')
    for obj in imported_objs:
        obj.select_set(True)
    bpy.context.view_layer.objects.active = imported_objs[0]
    bpy.ops.object.origin_set(type='ORIGIN_GEOMETRY', center='BOUNDS')

    min_z = min([v[2] for obj in imported_objs for v in [obj.matrix_world @ v.co for v in
obj.data.vertices]])
    for obj in imported_objs:
        obj.location.z -= min_z

# 4. Automated Studio Three-Point Lighting
# Key Light
key_light_data = bpy.data.lights.new(name="Key_Light", type='AREA')
key_light_data.energy = 500
key_light_data.size = 2.0
key_light = bpy.data.objects.new(name="Key_Light", object_data=key_light_data)
key_light.location = (3.0, -3.0, 4.0)
bpy.context.collection.objects.link(key_light)

# Fill Light
fill_light_data = bpy.data.lights.new(name="Fill_Light", type='AREA')
fill_light_data.energy = 200
fill_light_data.size = 3.0
fill_light = bpy.data.objects.new(name="Fill_Light", object_data=fill_light_data)
fill_light.location = (-3.0, -2.0, 2.0)
bpy.context.collection.objects.link(fill_light)

# Rim Light
rim_light_data = bpy.data.lights.new(name="Rim_Light", type='SUN')
rim_light_data.energy = 3.0
rim_light = bpy.data.objects.new(name="Rim_Light", object_data=rim_light_data)
rim_light.location = (0.0, 4.0, 3.0)
rim_light.rotation_euler = (math.radians(-45), 0, math.radians(180))
```

```
bpy.context.collection.objects.link(rim_light)

# 5. Product Showcase Camera
cam_data = bpy.data.cameras.new("Product_Camera")
cam_obj = bpy.data.objects.new("Product_Camera", cam_data)
cam_obj.location = (0, -4.5, 2.0)
cam_obj.rotation_euler = (math.radians(72), 0, 0)
bpy.context.collection.objects.link(cam_obj)
bpy.context.scene.camera = cam_obj

# 6. Eevee Next Headless Inspection Render
bpy.context.scene.render.engine = 'BLENDER_EEVEE_NEXT'
bpy.context.scene.render.resolution_x = 1280
bpy.context.scene.render.resolution_y = 720
bpy.context.scene.render.filepath = "./public/renders/validation_preview.png"

bpy.ops.render.render(write_still=True)
print("✅ Headless Blender validation frame generated: ./public/renders/validation_preview.png")
```

Execute in background mode:

```
blender --background --python pipeline.py
```

Astra immediately inspects `validation_preview.png` using multimodal vision, identifying any inverted normals or lighting washouts and self-correcting script parameters automatically.

Step 4: Instant Web Embed (React Three Fiber / Three.js)

Distribute the optimized GLB asset directly inside your Next.js 15 or Vite web application:

```
// components/VendingMachineViewer.tsx
import { Canvas } from '@react-three/fiber';
import { useGLTF, OrbitControls, Environment } from '@react-three/drei';

export function VendingMachineViewer() {
  const { scene } = useGLTF('/models/vending_machine.glb');
  return (
    <div className="w-full h-[500px] bg-slate-900 rounded-xl overflow-hidden shadow-2xl">
      <Canvas camera={{ position: [0, 2, 4], fov: 45 }}>
        <ambientLight intensity={0.7} />
        <directionalLight position={[5, 10, 5]} intensity={1.2} />
        <primitive object={scene} position={[0, -1, 0]} />
        <Environment preset="city" />
        <OrbitControls autoRotate autoRotateSpeed={2.0} enableZoom={true} />
      </Canvas>
    </div>
  );
}
```

13.7 Cost Optimization & Elastic Routing: Codex Switch in Practice

In the 2026 multi-model ecosystem, seasoned indie developers avoid vendor lock-in with a resilient routing strategy:

1. **Tiered Model Routing**: Standard everyday development runs on `gpt-5.6-terra`, lightweight approvals run on `gpt-5.6-luna`, reserving `gpt-6-astra` for complex multi-app visual audits, 3D spatial computing,

and full-system architecture.

2. **Multi-Provider Failover & Seamless Continuation**: When OpenAI account limits (rate limits or 3-hour quotas) are hit, use the open-source companion tool **Codex Switch** to toggle instantly to DeepSeek or Kimi Code. Its **cross-provider session continuation engine** preserves existing conversation context and SQLite thread history without losing state.

13.8 Migration Checklist (TL;DR)

```
# ① Desktop: install the ChatGPT client
winget install OpenAI.ChatGPT        # Windows

# ② Upgrade the CLI to 0.147.0+
npm install -g @openai/codex@latest

# ③ Upgrade models to the GPT-6 Astra / GPT-5.6 hierarchy
grep -rn "gpt-5.4" ~/.codex/ .       # full audit to replace retired models

# ④ Replace the removed --full-auto flag
codex exec --sandbox workspace-write "<task>"

# ⑤ Enable Guardian auto-approval
codex --approve-for-me "<task>"

# ⑥ Seamless multi-provider failover with Codex Switch
git clone https://github.com/aipmer/codex-switch.git && cd codex-switch && ./install.sh
codex-switch --to deepseek

# ⑦ Install plugin marketplaces and the security plugin
codex plugin marketplace add <catalog-url>
codex plugin install codex-security
```

🛡️ Troubleshooting & Pitfall Cheat Sheet

Common Pitfall	Root Cause	Rapid Diagnosis & Fix Guide
`Unknown argument: --full-auto`	0.14x completely removed this flag, aborting scripts	Replace with `--sandbox workspace-write`
`API Error: Model 'gpt-5.4' is deprecated`	Model retired after August 31	Edit `~/.codex/config.toml` and change model to `gpt-5.6-terra`
`Failed to install plugin: catalog not found`	Marketplace catalog URL is unreachable or blocked	Check network connectivity, or install via local path: `codex plugin install ./my-plugin`

13.7 The Constants That Survive Every Version

The tooling lesson is singular: **every version number, CLI flag, and model codename hard-coded into scripts is technical debt**. Centralize them in `config.toml` and `AGENTS.md`, so migration cost collapses to "edit one line."

And the three principles that outlive every release cycle are exactly what this book keeps hammering: goal-driven rather than step-driven, boundaries first rather than post-mortem blame, and human-in-the-loop orchestrating the AI rather than being led by it. Tools change blood; your mindset compounds.
