

What does ChatGPT know about agent skills?

“Agent skills” usually refers to the capabilities an **AI agent** has to perceive, decide, and act—often autonomously—across tools and environments. Here’s how ChatGPT understands and frames *agent skills*, from both a conceptual and practical standpoint.

1. Core Agent Skills (Conceptual Layer)

These are the foundational abilities that define an agent, regardless of implementation:

Reasoning & Planning

- Breaking down goals into steps (task decomposition)
- Choosing actions based on constraints and priorities
- Long-horizon planning (multi-step workflows)

Perception & Understanding

- Interpreting text, images, or structured data
- Extracting intent, entities, and context
- Tracking state over time (what’s been done, what remains)

Decision-Making

- Selecting the next best action
- Handling uncertainty and incomplete information
- Evaluating trade-offs (cost, time, risk)

Learning & Adaptation (limited, depending on system)

- Adjusting strategies based on feedback
- Using past interactions or memory (when enabled)
- Improving tool usage patterns

2. Tool-Use Skills (Practical Layer)

Modern agents are powerful because they can **use tools**, not just talk.



Tool Invocation

- Calling APIs (search, databases, internal services)
- Executing code (e.g., Python)
- Using browsers, calendars, email, CRMs, etc.



Orchestration

- Sequencing multiple tools to complete a task
- Handling tool errors and retries
- Passing outputs from one tool as inputs to another



Data Handling

- Reading/writing files
- Transforming data (CSV, JSON, SQL, docs)
- Generating reports, charts, or summaries

3. Autonomy & Control Skills

These determine *how independent* the agent is.



Goal Management

- Accepting high-level goals ("plan a trip")
- Asking for clarification only when necessary
- Knowing when a task is complete



State & Memory

- Maintaining working memory during execution
- Using long-term memory (if available)
- Avoiding repeated or redundant actions



Safety & Constraints

- Respecting rules, permissions, and guardrails
- Avoiding harmful or disallowed actions
- Deferring to humans when required

4. Communication Skills

An agent isn't useful if it can't collaborate.



Natural Language Interaction

- Explaining plans and decisions
- Asking precise follow-up questions
- Adjusting tone and detail to the user

Collaboration

- Working alongside humans-in-the-loop
- Handing off tasks or requesting approval
- Producing artifacts others can use (docs, code, tickets)

5. Common Agent Skill Frameworks ChatGPT Recognizes

You'll often see agent skills described using frameworks like:

- **ReAct**: Reason + Act + Observe loops
- **Plan-and-Execute**: Planner agent + executor agent
- **Toolformer-style agents**: Models trained to call tools
- **AutoGPT / BabyAGI patterns**: Goal-driven autonomous loops
- **Multi-agent systems**: Specialized agents (researcher, coder, reviewer)

ChatGPT is familiar with these patterns and can help **design, evaluate, or implement** them.

6. What *This* ChatGPT Can Do as an Agent

Depending on configuration, ChatGPT can:

- Act as a **task agent** (research, write, analyze)
- Act as a **tool-using agent** (code execution, data analysis, web search)
- Act as a **planning agent** (break down complex goals)
- Act as a **simulated agent** (design how an agent *should* behave)

What it *cannot* do without explicit tooling:

- Run indefinitely on its own
- Access systems without permission
- Learn persistently unless memory is enabled

7. If You're Asking This For a Reason...

If you want, I can:

- Help you **define agent skills** for a product or job role
- Design an **agent architecture** (skills + tools + control loop)
- Map agent skills to **LLM prompts or system instructions**
- Evaluate whether a system is truly an “agent” or just a chatbot

Just tell me the context you're working in.



How does what has been described compare to the open Agent Skills available at agentskills.io?

Here's how the *general concept* of agent skills that I described earlier compares with the **open Agent Skills** ecosystem and specification you find at agentskills.io and related repositories.

What “Agent Skills” Means at agentskills.io

Open Agent Skills (as defined at agentskills.io) is an **open standard and ecosystem** for packaging reusable capabilities that AI agents can load and execute. These skills are:

- **Modular packages** (folders) containing machine-readable instructions, resources, and optional code/scripts. The key file is `SKILL.md`, which includes metadata (`name`, `description`, etc.) and procedural guidance. Agent Skills +1
- Designed to be **discoverable and activated on demand** by compatible agents: agents load only metadata until a task matches a skill's description, then load full instructions. Agent Skills
- **Open and ecosystem-wide** — the format was released as a vendor-neutral open standard and is already supported by multiple agent products. Agent Skills +1

The open ecosystem has catalogs (e.g., *Open Agent Skills*) with hundreds to thousands of community and official skills across categories like documents, development, web

automation, etc. Open Agent Skills



How This Relates to the High-Level “Agent Skills” Description

General Concept (How I described it earlier)

Skills are *capabilities* or modules that extend an agent’s functionality

Skills help agents perform specific tasks reliably

Skills may include logic, reasoning steps, prompts, or tool usage patterns

Agents must discover and apply relevant skills

Agent skills are a way to modularize expertise

Open Agent Skills Standard (agentskills.io)

Skills are *actual structured packages* (folders with SKILL.md) describing that capability — an implementation of the concept Agent Skills

Open Agent Skills define *procedural, task-oriented guidance* agents follow exactly when relevant Agent Skills

In this standard, the logic *lives in SKILL.md plus optional scripts and assets*, so an agent can run code or follow detailed workflow specs Agent Skills

Skills are discovered via metadata at startup and dynamically activated when descriptions match a task context — progressive disclosure to manage context size Agent Skills

Agent Skills here are *literal modular expertise packages* that can be shared, versioned, and reused across tools and agents (§ open development) Agent Skills

Key Ideas the Open Standard Adds



Standard Structure & Metadata

The open standard defines a concrete format (YAML frontmatter + Markdown instructions), so tools and agents can interoperate. Agent Skills



Progressive Loading

Agents only load metadata initially (names & descriptions), then the full instructions when a task matches. This makes it context-efficient. Agent Skills



Ecosystem & Discoverability

There are open repositories/catalogs (e.g., Open Agent Skills) where anyone can browse, share, or contribute skills. Open Agent Skills

✓ Optional Code & Scripts

Skills can include executable scripts or assets (templates, documentation) that extend what an agent can do beyond text guidance. Agent Skills

🚀 Practical Outcome

Because of this open standard:

- **Agents across platforms** (Claude, Cursor, OpenAI's Codex/skills ecosystem, VS Code, etc.) can **use the same skill packages** without reinventing formats. Agent Skills +1
- **End users and teams** can share domain-specific workflows or procedures as reusable skills (e.g., document processing, automation tasks). Open Agent Skills
- Skills become a **modular layer outside of the model itself** — separating *capability descriptions* from *model training*, and enabling interoperability and versioning.

Agent Skills

📌 Summary

What I described conceptually — that agent skills are reusable capabilities that let an agent perform tasks better — maps directly to the **open Agent Skills standard**:

- The standard gives a **formal structure** and **ecosystem** to what agent skills *are*.
- It emphasizes **portability**, **modularity**, and **runtime activation**, not just abstract capabilities.
- It's adopted across tools and designed to let different agents share and reuse the same skill definitions. Agent Skills

If you'd like, I can walk through a concrete example of a specific Skill from Open Agent Skills (e.g., a PDF processing skill) and show how its structure enables an agent to work more reliably.

       Sources

Can the Open Agent Skills be used with ChatGPT?

Yes — **Open Agent Skills can be used with ChatGPT**, but the level of support and how you use them today depends on which version of ChatGPT you're using and what you mean by *"use with"*:

✓ 1. Basic Use via Custom Instructions

You **can use Open Agent Skills with the ChatGPT web or app interface** by manually transferring the skill content into **Custom Instructions**.

- The *agentskills.io* docs explain that ChatGPT doesn't (yet) natively load skills like Claude or Codex, **but you can copy the SKILL.md content into Custom Instructions** so ChatGPT follows that guidance in your conversations. agentskillsindex.c...
- Because ChatGPT's instruction fields have character limits, you often need to **summarize or extract the key parts** of a skill you care about. agentskillsindex.c...

This is a **manual workflow**, but it lets you leverage *open agent skills* as structured guidance inside ChatGPT sessions.

📌 2. Via Custom GPTs in ChatGPT Builder

Another practical method is to **create a Custom GPT** (via ChatGPT's GPT Builder) and paste skill content into its instructions.

- Custom GPTs act like reusable skill-anchored assistants and can hold more detailed procedural guidance than Custom Instructions alone.
- This isn't *native skill loading*, but it effectively approximates the same behavior: structured procedural knowledge that ChatGPT uses across sessions. agentskillsindex.c...

⚠ 3. Native Skill Support in ChatGPT Platform





As of early 2026:

- **ChatGPT doesn't yet have full native support for the open Agent Skills format the**

way Claude Code or the OpenAI Codex CLI does. [agentskillsindex.c...](#) +1

- Some news and user reports suggest OpenAI *may soon add better native skills support (e.g., slash commands or built-in skill editors)* given its adoption of the same open standard, but this capability is not broadly available yet. [Gadgets 360](#)
- Users have noted that while OpenAI’s systems technically share the *agentskills.io* architecture (especially with Codex CLI and APIs), the full dynamic skill activation/management in ChatGPT itself is still limited. [GitHub](#)

Summary: What’s Possible Today

Platform / Feature	Native Agent Skills Support?	How to Use
ChatGPT (web/app)	 Not truly native yet	Manual import via Custom Instructions or Custom GPTs agentskillsindex.c...
ChatGPT Custom GPTs	 Partial support	Paste skill logic into instructions to approximate skills agentskillsindex.c...
OpenAI Codex CLI / IDE	 Yes	Full skills support, with automatic loading and invocation OpenAI Developers
Skills-aware wrappers / loaders	 Yes (with tooling)	Libraries like <i>openskills</i> can integrate SKILL.md into LLMs GitHub

Optional Ways to Integrate Skills at Scale

If you need *true automated skill loading* with ChatGPT-powered agents:

- Use **Wrapper Libraries (like openskills)** that make SKILL.md consumable by an LLM runtime. [GitHub](#)
- Use **Custom GPTs** to hard-bake the behaviors you want and distribute those as reusable mini-agents.

Final Thought

Yes — you *can* use Open Agent Skills with ChatGPT today, but it's primarily **manual or via Custom GPTs** rather than automatic native loading. Full native support (like you get in Codex and some Claude agents) is expected to roll out but isn't broadly available yet in ChatGPT itself. [agentskillsindex.c... +1](#)

If you want, I can show step-by-step how to import a specific Open Agent Skill for use inside ChatGPT!

         Sources

Can ChatGPT Apps be used from within OpenAI Codex CLI?

Here's how things stand regarding **using ChatGPT Apps from within the OpenAI Codex CLI**:

1. Codex CLI is a separate tool focused on coding workflows

The **OpenAI Codex CLI** is a **terminal-based coding agent** that helps you write, edit, and run code in your repository, automate workflows, and run tasks locally or via cloud sandboxes. It's designed for coding automation, not for running interactive chat-based apps the way ChatGPT does. [OpenAI Developers](#)

2. ChatGPT Apps currently live in the ChatGPT UI

"Apps" in ChatGPT are interactive experiences that appear inside the ChatGPT chat interface on the web/app, built with the **Apps SDK** and integrated into the ChatGPT conversation experience. They respond to natural language, show UI, and interact with users in ways tailored to that interface. [OpenAI](#)

There's **no official support** documented for launching or embedding *ChatGPT Apps* directly **from the Codex CLI** — the CLI doesn't support the ChatGPT Apps SDK or interactive app interfaces.

3. Codex can invoke models and workflows, but not full "Apps"

- You *can* authenticate the Codex CLI with your ChatGPT account and use it to run agentic coding tasks or leverage models locally/sandboxed. [OpenAI Developers](#)

- However, “Apps” in ChatGPT involve UI integration and user interaction that isn’t part of a CLI environment. There’s currently **no documented command or mechanism** in the Codex CLI that loads or runs ChatGPT Apps as they appear in the web/app experience.

🤔 4. Possible future integration paths (not yet available)

In theory there are two conceptual ways this *could* work in the future, but they are **not currently supported**:

- **Model Context Protocol (MCP) integrations** — industry standards like MCP aim to let models interact with external tools and UI layers, and some of the ChatGPT Apps SDK uses similar ideas. But this doesn’t yet mean that the Codex CLI can natively host ChatGPT Apps. Wikipedia
- **Local bridging between CLI and ChatGPT UI** — there’s an open discussion about connecting the ChatGPT app with local Codex CLI instances (e.g., letting the ChatGPT interface use the local CLI as a custom model), but this is still a requested/experimental feature and not a released product. GitHub

📌 Summary

❌ You **cannot currently run ChatGPT Apps directly from within the OpenAI Codex CLI** — the CLI is a coding agent environment, not an interactive app host.

✓ The CLI can use your ChatGPT credentials and models to do code-focused tasks.

OpenAI Developers

✓ ChatGPT Apps run in the ChatGPT web/app interface and rely on UI/SDK features not available in the CLI. OpenAI

If you need a workflow that *combines* App-like interactions with Codex-powered coding automation, the current pattern is to do that via the **ChatGPT UI (browser or desktop)** or by integrating the **Codex SDK** into your own tooling — not through the CLI itself.

📄 👍 💬 ↗️ ↺️ ⋮ 🔗 🌐 Sources

Tell me more about Model Context Protocol (MCP) integrations for the ChatGPT App SDK

Yes — the **Model Context Protocol (MCP)** plays a *central role* in how modern interactive AI integrations (including the **ChatGPT App SDK**) work under the hood, and understanding it helps clarify both current capabilities and how broader tool/agent integration standards are evolving. OpenAI Developers +1

What the Model Context Protocol (MCP) Is

MCP is an **open, standardized protocol** that defines how an AI **host** (like ChatGPT, Claude, a local agent, etc.) can:

1. **Discover tools**
2. **Invoke tools with structured arguments**
3. **Receive structured results and rich content**
4. **Render UI components returned as resources**

Effectively, MCP is like a *universal connector* that lets AI models interact with external capabilities — from APIs to UI widgets — in a consistent and extensible way.

OpenAI Developers +1

MCP's Role in the ChatGPT App SDK

The **ChatGPT Apps SDK** (the system used to build *ChatGPT Apps*) *builds on top of MCP*. This means:

1. Apps aren't just static prompts

When you build an app with the Apps SDK:

- Your backend exposes an **MCP server**.
- You register various **tools** that the model can call.
- You optionally include **UI resources** (like HTML/JS widgets) that the client can render.

OpenAI Developers

2. MCP connects the model, server, and UI

MCP provides the *communication protocol* that allows the AI model to do things like:

- List available tools (and their input/output schemas)
- Invoke a tool with structured arguments
- Receive structured results
- Return UI components to the ChatGPT client for rendering

In other words, MCP is the plumbing that lets an AI app run *beyond plain text* — including interactive dashboards, forms, visualizations, or even multi-step workflows — by seamlessly syncing tools, results, and UI state. OpenAI Developers

Key MCP Components in the Apps SDK

Here's how MCP fits into the standard ChatGPT App experience:

1. Tool Discovery & Catalog

Your MCP server tells the model which tools you offer and how they work. The model can use that metadata to decide what to call. OpenAI Developers

2. Structured Tool Invocation

The model doesn't just *imagine* what to do — it calls `call_tool` with real parameters, and your server executes them and returns structured data. OpenAI Developers

3. UI Resources

Beyond raw data, tools can return *resources* that render in ChatGPT clients as UI widgets (HTML/JS). This is how rich app experiences appear inside the conversation. OpenAI Developers

4. Transport Flexibility

MCP isn't tied to a specific transport — you can host a server over **streamable HTTP** or **STDIO**. The Apps SDK handles the connection. OpenAI Developers

Why MCP Matters for Apps and Agents

Consistency Across Platforms

MCP lets you build tools that:

- Work in ChatGPT

- Work in other MCP-aware environments (e.g., Claude or custom agents)
- Don't require reimplementing connection logic per platform

In fact, MCP is increasingly seen as a **cross-platform interoperability standard** for AI integrations. The Verge

Rich Interactive Experiences

Because MCP supports returning UI resources as *components*, apps can go far beyond text output — they can provide interactive interfaces that live inside the ChatGPT conversation.

OpenAI Developers

Standardized Tooling

Instead of building ad-hoc connectors for each model, MCP gives you:

- JSON schema descriptions for tools
 - Standardized authentication flows (like OAuth)
 - Structured results the model can reason about
- ... all without reinventing the wheel. OpenAI Developers

MCP vs. "Traditional" App SDK Development

A few things to be aware of:

Traditional Function Calls / Plugins	MCP + Apps SDK
Text-based tool invocation only	Structured tool calls with schemas <small>OpenAI Developers</small>
Limited UI integration	Full client-rendered UI components from tools <small>OpenAI Developers</small>
Platform-specific	Protocol-agnostic and potentially cross-platform <small>Model Context Pr...</small>
Custom integrations per model	Standardized interactions via MCP <small>OpenAI Developers</small>

The Broader Landscape

MCP is **gaining adoption beyond just ChatGPT Apps**. Major AI platforms, ecosystems, and third-party connectors are building around it — including extensions called **MCP Apps** that enable interactive UI elements in any MCP-aware client. Model Context Pr...

There's also growing community and industry support aimed at making MCP an open standard for *agentic inter-AI interoperability*, with OpenAI, Anthropic, Google, and others contributing to foundations and tooling around it. WIRED

In Summary

- **MCP is a core enabler** of rich Apps in ChatGPT — it standardizes how models discover and interact with tools and context. OpenAI Developers
- The Apps SDK *uses* MCP to provide schema-driven tool invocation, structured outputs, and widget-rendered UI. OpenAI Developers
- MCP's open standard approach means your integrations could be portable beyond ChatGPT and into other AI ecosystems. Model Context Pr...

If you want, I can walk you through a **simple MCP server example** for use with the ChatGPT Apps SDK — including how to define a tool with a schema and render a UI widget in the client.

     ...  Sources

Tell me more about how the Apps SDK uses MCP to provide widget rendered UI and its relationship to the Resources part of the MCP specification.

Below is a **deeper, architectural explanation** of how the **ChatGPT Apps SDK** uses **MCP** to deliver **widget-rendered UI**, and how that maps specifically to the **Resources** section of the MCP specification. I'll keep it concrete and layered, because this is one of the most subtle (and powerful) parts of the system.

1. Big Picture: MCP Separates *Reasoning* from *Rendering*

At a high level, MCP enforces a clean separation:

- **The model reasons** → decides *what* should happen
- **The MCP server executes** → performs actions and returns results
- **The client (ChatGPT app) renders** → displays UI widgets

The **Apps SDK** is essentially a **specialized MCP client + runtime** that knows how to render **MCP Resources** as interactive UI inside the ChatGPT conversation.

The model never “draws UI.”

It requests or returns *resources*, and the client renders them.

2. Where Widgets Come From in MCP Terms

In MCP, **widgets are not a special concept**.

They are implemented using **Resources**.

MCP Resource (Core Concept)

A **Resource** is:

- A structured object returned by a tool
- Identified by a `type`
- Accompanied by metadata and optional lifecycle hooks

In the Apps SDK, **UI widgets are just Resources whose type the ChatGPT client knows how to render**.

3. The Apps SDK Rendering Pipeline (Step by Step)

Let’s walk through what actually happens.

Step 1: Tool Invocation (Model → MCP)

The model decides to call a tool:

```
json
```



```
{
  "tool": "get_dashboard",
  "arguments": {
    "user_id": "123"
  }
}
```

This is **pure MCP**: schema-driven, structured, no UI yet.

Step 2: MCP Server Returns Data + Resources

Your MCP server responds with something like:

json

 Copy code

```
{
  "content": [
    { "type": "text", "text": "Here's your dashboard:" }
  ],
  "resources": [
    {
      "id": "sales_chart",
      "type": "ui.chart",
      "data": {
        "title": "Monthly Revenue",
        "series": [120, 140, 180, 160]
      }
    }
  ]
}
```

Key points:

- The **tool output** is still model-readable
- The **resource is client-renderable**
- The model doesn't need to understand *how* the chart is drawn

Step 3: ChatGPT Client Recognizes the Resource

The ChatGPT App runtime:

- Sees `type: "ui.chart"`
- Looks up a **registered renderer** for that resource type
- Mounts the widget inline in the conversation

This is where the **Apps SDK adds value**:

- It ships with **built-in renderers**
- And supports **custom renderers** for app-defined resource types

4. Relationship to the MCP “Resources” Specification

The MCP spec defines **Resources** as a general-purpose mechanism:

MCP Spec Concept	Apps SDK Interpretation
Resource	UI widget, file, or visual artifact
Resource type	Determines renderer
Resource data	Props / state
Resource id	Component identity
Resource lifecycle	UI updates, replacement, removal

The Apps SDK **does not change MCP** — it **constrains and extends** it with conventions.

Important: MCP Itself Is UI-Agnostic

MCP does *not* say:

- “This is a button”
- “This is a chart”

It only says:

“Here is a resource of type X with data Y.”

The **Apps SDK defines the UI vocabulary** on top of that.

5. Stateful UI via Resource Identity

One of the most powerful (and underappreciated) parts of this design is **resource identity**.

If your MCP server returns:

json

 Copy code

```
{
  "id": "sales_chart",
  "type": "ui.chart",
  "data": { "series": [200, 220, 210] }
}
```

The ChatGPT client:

- Updates the existing widget
- Instead of creating a new one

This enables:

- Live updates
- Streaming UI
- Step-by-step workflows
- Agent-driven dashboards

All **without the model managing UI state explicitly**.

6. Interactivity: User → Resource → Tool → Model

Widgets are not just output — they can be **interactive**.

Interaction Loop

1. User clicks a button / submits a form
2. Client emits an MCP event
3. Event invokes a tool or updates context
4. Model reacts

5. New resources are returned

Conceptually:

sql

 Copy code

User Action

↓

UI Resource Event

↓

MCP Tool Call

↓

Model Reasoning

↓

Updated Resources

This turns ChatGPT Apps into **event-driven systems**, not just chatbots.

7. Why This Architecture Matters

◆ Models Stay UI-Blind

The model:

- Doesn't know React, HTML, CSS
- Only reasons about *intent and structure*

◆ Clients Stay Model-Agnostic

The ChatGPT client:

- Doesn't care which model is running
- Just renders resources

◆ Servers Stay Portable

Your MCP server:

- Can be used by ChatGPT
- Or any MCP-aware client
- With zero changes

This is why MCP is often described as **“the HTTP of agent UI.”**

8. How This Differs from Old Plugins / Function Calling

Old Plugins	MCP + Apps SDK
Text-only responses	Structured + UI
UI hardcoded in client	UI described via resources
Model-specific	Cross-client capable
Stateless	Stateful resource identity

9. Mental Model to Keep

If you want a concise mental model:

Tools do work.

Resources describe artifacts.

The Apps SDK renders artifacts as widgets.

The model orchestrates — but never paints pixels.

10. Where This Is Going

This design strongly suggests:

- Cross-platform app portability
- Shared widget standards
- Agent-driven UIs outside ChatGPT
- IDEs, CLIs, and browsers all rendering the same MCP resources differently

In other words:

Apps are not “ChatGPT features.”

They’re MCP applications with a ChatGPT client.

If you want, next we can:

- Walk through a **minimal MCP server that renders a widget**
- Compare **MCP Resources vs Open Agent Skills**
- Or explore **how Codex CLI could consume resources differently (e.g., TUI instead of UI)**



What built-in renderers does the Apps SDK ship with?

As of the latest **OpenAI Apps SDK** documentation and ecosystem information, there *aren't* “built-in” renderers in the sense of a closed list of components that automatically get rendered by ChatGPT without developer UI assets. Instead, the Apps SDK supports interactive **widget UI rendering by design**, but the actual visuals come from the **UI resources your app provides** rather than from a pre-packaged set of native UI primitives built into ChatGPT itself. OpenAI Developers +1

Here's how rendering support actually works:

1. Apps SDK Does *Not* Include Fixed Browser UI Controls by Default

Unlike some UI frameworks that ship with built-in components (like cards, tables, charts), the Apps SDK **doesn't come with a fixed library of renderers that automatically render structured data** in ChatGPT.

Instead, the system expects your app to:

- Return **HTML/JS widget bundles** from your **MCP server** in the tool response metadata
- Let the ChatGPT client **load those UI bundles in a sandboxed iframe** and use the runtime APIs (`window.openai`) to connect the widget to the conversation context

OpenAI Developers

So from ChatGPT's **rendering perspective**, the core built-in ability is:

- ✓ **Displaying arbitrary HTML/JS iframes** as widgets
 - ✓ **Injecting data into them via the** `window.openai` **API bridge**
 - ✓ **Rendering moderator UI container + sandbox environment**
- 📌 But *the content and interactivity come from your app's widget.* OpenAI Developers

✂ 2. Optional UI Toolkit for Developers

Although ChatGPT *doesn't automatically render a fixed component set*, OpenAI provides an **optional UI kit** (often referenced as `apps-sdk-ui`) that developers can use to build consistent UI elements that fit well inside the ChatGPT interface. This kit gives you:

- ✓ Buttons
- ✓ Cards
- ✓ Input controls
- ✓ Layout primitives

These aren't ChatGPT's built-in renderers — they're reusable pieces you include in your widget bundle to make consistent, chat-styled UIs more easily. OpenAI Developers

📦 3. Widget Rendering Workflow

Here's what actually happens under the hood when a widget is shown in ChatGPT:

1. **Your MCP server returns structured content** with text/data and metadata pointing to a UI resource (HTML/JS).
2. ChatGPT's client loads the HTML (via a sandboxed iframe).
3. Inside the iframe your bundle runs — usually a framework like React, Svelte, or vanilla JS.
4. Your script connects to ChatGPT's **widget runtime** (`window.openai`), which lets it:
 - Read the tool output (`toolOutput`)
 - Call tools (`callTool`)
 - Persist UI state (`widgetState`)
 - Request layout changes (fullscreen, PiP) OpenAI Developers +1

The ChatGPT host renders the container and iframe around your widget; your widget *renders its own UI elements*.


4. Examples of What Developers Actually Bundle

Although there aren't "built-in" renderers like Angular Material or iOS UIKit, developers *effectively create their own UI* using:


- **HTML/JS** widgets compiled into single bundles
- UI frameworks like **React or Svelte**
- Style/design matching ChatGPT's UI
- Helper libraries and optional packages (`apps-sdk-ui`) for common patterns


OpenAI Developers


Example apps in the official *openai-apps-sdk-examples* GitHub repo showcase variations like:

 List views

 Carousel-like components

 Map integrations (via Mapbox)

 Cards and image galleries

 Media players

... and more, all served as widget HTML/JS bundles that ChatGPT loads and displays.

DeepWiki

5. ChatGPT's Role vs. Your Widget's Role

Responsibility	Who Provides It?
Rendering container & sandbox	ChatGPT client
Loading widget HTML/JS	ChatGPT client
Actual UI implementation (buttons, charts, lists, maps)	Your bundled widget code

Responsibility

Layout control (inline, fullscreen, PiP)

Interaction & calls back to server

Who Provides It?

ChatGPT widget runtime via
`requestDisplayMode`

Widget → `window.openai.callTool()`

Summary

- ◆ There is **no built-in library of UI renderers in the Apps SDK** that automatically show tables, cards, or charts without you providing the widget UI code.
- ◆ The **Apps SDK enables UI widgets** by allowing your MCP response to return HTML/JS components that render inside ChatGPT's iframe environment. OpenAI Developers
- ◆ Developers are encouraged to use **optional UI kits** (`apps-sdk-ui`) and standard web technologies (React, Svelte, etc.) to implement the visuals that ChatGPT will render for the user. OpenAI Developers

If you'd like, I can outline some of the **common UI component patterns developers build** with the Apps SDK (e.g., lists with actions, interactive charts, forms) and how they integrate with the MCP tool outputs!

     ...  Sources

What can you tell me about how this compares to the recent MCP Apps extension described at <https://modelcontextprotocol.io/docs/extensions/apps?> Is the MCP Apps extension supported by the ChatGPT Apps SDK?

Here's a clear comparison between the **MCP Apps extension** (as defined in the official Model Context Protocol extensions spec) and the **ChatGPT Apps SDK**, plus whether and

how they relate to each other.

What the *MCP Apps* Extension Is

The **MCP Apps extension** is an **official extension to the Model Context Protocol (MCP)** designed to standardize how **interactive user interfaces** — e.g., dashboards, forms, live widgets, etc. — are delivered from an MCP server to any *MCP-aware host* (e.g., conversational agents, IDEs, other clients). It specifies:

Key Concepts of MCP Apps (SEP-1865)

- **UI resources (`ui://` URIs):** Tools declare interactive UIs as part of their metadata so hosts know a widget should be rendered. modelcontextprot...
- **Sandboxed rendering:** HTML/JS interfaces are rendered securely in iframes by the host. modelcontextprot...
- **Bidirectional communication:** The UI can request tool calls and push interactions back to the host using JSON-RPC and `postMessage`. modelcontextprot...
- **Standardized protocols:** Instead of per-host bespoke integrations, MCP Apps defines a shared JSON-RPC based extension to MCP to enable UI rendering and interaction.

Model Context Pr...

MCP Apps is an **open, tool-agnostic standard** that multiple clients and server ecosystems can adopt (e.g., Claude, VS Code, Goose, Postman). modelcontextprot...

How the ChatGPT Apps SDK Works Today

The **ChatGPT Apps SDK** is OpenAI's current *host-specific* tooling for building interactive experiences inside ChatGPT using MCP underneath. It allows developers to:

- **Build an MCP server** that exposes tools and UI resources. OpenAI Developers
- **Return UI resources** bundled as HTML/JS that ChatGPT renders in a sandboxed iframe. OpenAI Developers
- **Communicate between UI widget and server** using a host-specific runtime API.

OpenAI Developers

The **Apps SDK** is already heavily influenced by MCP — it uses MCP's JSON-RPC to handle tool calls and resource delivery — but it originally had **proprietary runtime expectations**

and metadata structures that were unique to ChatGPT’s client (before MCP Apps was formalized). [Reddit](#)

Comparison: MCP Apps vs ChatGPT Apps SDK

Aspect	MCP Apps (Official MCP Extension)	ChatGPT Apps SDK
Standard	Formal extension to MCP (SEP-1865) — open standard for interactive UI. modelcontextprot...	A host-specific SDK built on MCP for ChatGPT interactive experiences. OpenAI Developers
UI Delivery	Declares interactive HTML resources using standard ui:// scheme. modelcontextprot...	Declares UI via resource metadata; historically used proprietary metadata fields. Reddit
Bidirectional UI Communication	Standardized JSON-RPC over postMessage . modelcontextprot...	Bidirectional, but originally via ChatGPT-specific widget runtime. Reddit
Host Support	Designed to work in any MCP host that implements the extension (e.g., Claude, VS Code). modelcontextprot...	Designed to work in ChatGPT clients. Support elsewhere depends on adapters. Reddit
Portability	High — standardized so servers don’t need host-specific code. Model Context Pr...	Moderate — works well in ChatGPT but not standardized across clients. Reddit
Security & Sandbox	Standardized sandboxing and CSP through the MCP Apps spec. modelcontextprot...	Similar sandboxing when hosted in ChatGPT, but tied to ChatGPT’s client implementation. Reddit

Is MCP Apps Supported by the *ChatGPT Apps SDK*?

Short Answer: Yes — with caveats.

- ChatGPT's Apps SDK already implements UI rendering for MCP-based applications, and many of the ideas in *MCP Apps* (like sandboxed iframes and bidirectional communication) originate from the work behind the Apps SDK. Model Context Pr...
- As of early 2026, **ChatGPT apps support the official MCP Apps extension**, meaning they can understand and render `ui://` UI resources according to the MCP Apps spec. Clients like Claude and ChatGPT now support this standardized extension.

Model Context Pr...

- This means **UI built to the MCP Apps standard should work in ChatGPT** without requiring proprietary metadata formats, assuming your MCP server implements the extension correctly. modelcontextprot...

However:

- The *ChatGPT Apps SDK* still includes **ChatGPT-specific runtime APIs and helpers** that go beyond the core MCP Apps spec (e.g., widget state APIs, display modes, etc.). These are SDK conveniences that might not exist in other MCP clients. Reddit
- Some ChatGPT SDK tooling may still emit metadata outside the strict MCP Apps schema for additional ChatGPT-specific features. Reddit

So while **MCP Apps is now a shared open standard that ChatGPT supports**, developers should expect some **ChatGPT-specific enhancements** layered on top in the Apps SDK.



Why This Matters

Building to the MCP Apps Extension:

- Gives you *portable UI apps* that work in **any MCP-aware host** (e.g., Claude, ChatGPT, IDEs). modelcontextprot...
- Lets the server ship UI once and more clients render it without new adapters. modelcontextprot...
- Promotes a **unified ecosystem** where interactive agent UIs are standardized rather than siloed. Model Context Pr...

ChatGPT Apps SDK still adds value:

- Provides tooling, helpers, and integrations specific to ChatGPT clients. OpenAI Developers
- Eases development with built-in lifecycle and widget APIs beyond the core spec.

Reddit

Summary

✓ **MCP Apps** is now an *official MCP extension* standardizing how interactive UI experiences are declared and rendered within any MCP host. modelcontextprot...

✓ **ChatGPT Apps SDK** uses MCP and has informed the MCP Apps spec, and modern ChatGPT clients support the MCP Apps extension. Model Context Pr...

⚠ The Apps SDK includes *additional ChatGPT-specific features* that extend beyond the core MCP Apps spec for convenience and richer integration. Reddit

If you want, I can give you an example MCP Apps server tool definition and how that same definition works in both ChatGPT and other MCP-aware hosts!

       Sources

Give me a strictly standard MCP Apps server tool definition and describe how it would work in ChatGPT.

Below is a **strictly standard MCP Apps tool definition**—using **only what's in the MCP core + MCP Apps extension**, with **no ChatGPT-specific SDK helpers**—followed by a precise explanation of **how ChatGPT would execute and render it**.

I'll keep this aligned with the MCP Apps spec concepts: **tools, resources, and ui:// apps**.

1. A Strict MCP Apps Tool Definition

This is what an MCP server would expose during **tool discovery**.

Tool metadata (JSON-RPC `tools/list` result)

json

 Copy code


```
{
  "name": "show_sales_dashboard",
  "description": "Display an interactive sales dashboard for a given account",
  "inputSchema": {
    "type": "object",
    "properties": {
      "account_id": {
        "type": "string",
        "description": "The account identifier"
      }
    },
    "required": ["account_id"]
  }
}
```

Nothing here is UI-specific yet. This is **pure MCP core**.

2. Tool Invocation Result (With MCP Apps UI)

When the model calls the tool, the MCP server responds like this:

json

 Copy code

```
{
  "content": [
    {
      "type": "text",
      "text": "Here is the sales dashboard."
    }
  ],
  "resources": [
    {
      "uri": "ui://sales-dashboard",
      "mimeType": "text/html",
      "name": "Sales Dashboard",
      "description": "Interactive sales metrics and charts",
      "annotations": {
        "mcp.app": {
```



```
        "entrypoint": "/index.html",
        "sandbox": true
    }
}
}
]
```

Why this is strictly standard

- ✓ Uses resources (MCP core)
- ✓ Uses ui:// URI scheme (MCP Apps extension)
- ✓ Uses text/html for UI payload
- ✓ Uses extension-scoped annotations (mcp.app)
- ✓ No ChatGPT runtime APIs
- ✓ No proprietary fields

The actual HTML/JS bundle is served by the MCP server at /index.html .

3. What the UI Bundle Looks Like (Conceptually)

Your /index.html might do something like:

html

 Copy code

```
<!DOCTYPE html>
<html>
  <body>
    <h1>Sales Dashboard</h1>
    <div id="chart"></div>

    <script>
      // MCP Apps standard postMessage channel
      window.parent.postMessage(
        {
          jsonrpc: "2.0",
          method: "mcp.app.ready"
        },

```



```
        """
    );
</script>
</body>
</html>
```

Interaction (e.g. clicking a button) would send messages like:

json

 Copy code

```
{
  "jsonrpc": "2.0",
  "method": "mcp.app.callTool",
  "params": {
    "tool": "get_sales_data",
    "arguments": { "range": "30d" }
  }
}
```

This is **exactly what the MCP Apps extension standardizes**.

4. How ChatGPT Executes This Step-by-Step

Here's the lifecycle inside **ChatGPT**, mapped explicitly to MCP concepts.

Step 1: Tool discovery

ChatGPT (as an MCP host):

- Calls `tools/list`
- Sees `show_sales_dashboard`
- Reads its description and schema

No UI yet.

Step 2: Model decides to call the tool

The model emits a structured call:

json

 Copy code

```
{  
  "tool": "show_sales_dashboard",  
  "arguments": {  
    "account_id": "acct_123"  
  }  
}
```

Step 3: MCP server responds with a UI resource

ChatGPT receives:

- Text content
- A `ui://sales-dashboard` resource

ChatGPT recognizes:

“This is an MCP Apps UI resource.”

Step 4: ChatGPT renders the UI

Because ChatGPT **supports the MCP Apps extension**:

- It creates a sandboxed iframe
- Loads `/index.html` from the MCP server
- Applies CSP + sandbox rules per spec
- Renders the UI inline in the conversation

At this point, the user sees a **live dashboard widget**.

Step 5: User interacts with the UI

When the user clicks or submits something:





1. The UI sends a JSON-RPC message via `postMessage`

2. ChatGPT validates it against MCP Apps rules
3. ChatGPT forwards it as a tool call
4. The model reasons about the result
5. The MCP server responds (possibly with updated resources)

If the server returns the **same** `ui://sales-dashboard` **URI again**, ChatGPT **updates the existing widget** instead of creating a new one.

5. What ChatGPT Does *Not* Add Here

Importantly, in this strictly standard flow:

-  No `window.openai`
-  No ChatGPT widget runtime APIs
-  No ChatGPT-specific metadata
-  No SDK-specific lifecycle helpers

ChatGPT is acting purely as:

An MCP Apps-compliant host

6. How This Differs From Using the ChatGPT Apps SDK

The **Apps SDK** would optionally add:

- Convenience JS APIs
- Widget state helpers
- Display mode controls (fullscreen, PiP)
- Dev tooling and templates

But those are **layered on top**, not required.

Your MCP Apps server above:

- Works in **ChatGPT**
- Works in **Claude**
- Works in **VS Code**
- Works in **any MCP Apps-aware host**

7. Mental Model (Very Important)

Think of it this way:

MCP Apps defines *what* a UI is and how it communicates.
ChatGPT decides *how* to render it in its client.

ChatGPT is not special here—it's just a very capable MCP host.

8. One-Sentence Summary

A **strict MCP Apps server tool** returns a `ui://` resource describing an HTML app;
ChatGPT natively recognizes that resource, renders it in a sandboxed iframe, and routes user interactions back through MCP JSON-RPC exactly as the MCP Apps specification defines—without any ChatGPT-specific code required.

If you want, next we can:

- Compare this with **Open Agent Skills**
- Show how the **same app degrades gracefully in a CLI**
- Or diagram the **full JSON-RPC message flow end-to-end**



How would the same app degrade gracefully in a CLI?

Great question. This is *exactly* one of the design goals behind **MCP Apps**: the same server-side “app” should work across rich UIs *and* text-only environments without special-casing the server.

Below is a **precise, standard-aligned explanation** of how a UI-centric MCP App **degrades gracefully in a CLI**, and what responsibilities fall on the **host, model, and server**.

1. Key Principle: Graceful Degradation Is a *Host Capability Issue*

In MCP, the server never checks “am I in a CLI?”

It just returns:

- content → model-readable
- resources → host-renderable

The **host decides** what it can render.

So degradation happens because:

- The CLI host **does not support the MCP Apps extension**
- Or supports it only partially

2. The Same Tool Response (Unchanged)

Recall the server response:

json

 Copy code

```
{
  "content": [
    {
      "type": "text",
      "text": "Here is the sales dashboard."
    }
  ],
  "resources": [
    {
      "uri": "ui://sales-dashboard",
      "mimeType": "text/html",
      "name": "Sales Dashboard"
    }
  ]
}
```

This response is **identical** whether the host is:

- ChatGPT (rich UI)
- A terminal CLI

- An IDE
- A headless agent

3. What a CLI Host Does Differently

A CLI host typically:

- ❌ Does not support `ui://` rendering
- ❌ Cannot sandbox or display HTML
- ✅ Can render text
- ✅ Can invoke tools

So the CLI applies **MCP fallback rules**.

4. CLI Rendering Behavior (Step-by-Step)

Step 1: Ignore unsupported resources

The CLI host sees:

```
json
```

 Copy code

```
{
  "uri": "ui://sales-dashboard",
  "mimeType": "text/html"
}
```

It says:

"I don't support MCP Apps UI resources."

So it:


- Skips rendering the resource
- Keeps it available in context for the model

This is **expected and compliant behavior**.

Step 2: Render content normally

The CLI prints:

```
csharp
```

 Copy code

```
Here is the sales dashboard.
```

At minimum, the user sees something meaningful.

Step 3: Model adapts its response

Because the UI wasn't rendered, the **model compensates**.

Typical behaviors:

- Summarize key metrics in text
- Offer a table instead of a chart
- Ask follow-up questions instead of relying on UI

Example model output:

```
css
```

 Copy code

```
I can't display the interactive dashboard here, but here's a summary:
```

- Revenue (**30d**): **\$180,000** (**+12%**)
- Top region: EMEA
- Best product: Pro Plan

```
Would you like a CSV export or a detailed breakdown?
```


This adaptation happens **at the reasoning layer**, not in the server.

5. Optional: Server-Provided Text Fallbacks (Best Practice)

Although not required, **well-designed MCP Apps servers include text fallbacks** in content .

Example improved response:

json

 Copy code

```
{
  "content": [
    {
      "type": "text",
      "text": "Sales (30d): $180k (+12%). Top region: EMEA. Best product: I"
    }
  ],
  "resources": [
    {
      "uri": "ui://sales-dashboard",
      "mimeType": "text/html",
      "name": "Sales Dashboard"
    }
  ]
}
```

Now:

- ChatGPT shows the widget
- CLI users get a usable summary
- No branching logic needed

This is **the recommended MCP pattern**.

6. Interactions in a CLI (No Widgets)

In ChatGPT:

- User clicks buttons
- UI sends `mcp.app.callTool`

In a CLI:

- The **model prompts the user explicitly**

Example:

vbnet

 Copy code

Would you like:

- 1) Revenue by day
- 2) Revenue by product
- 3) Export CSV

Enter a number:

When the user answers, the model:

- Calls the same underlying tools
- Produces text output instead of UI

Same tools. Same server. Different interaction modality.

7. Resource Identity Still Matters (Even in CLI)

Even though the CLI doesn't render UI:

- The resource URI (`ui://sales-dashboard`) is still tracked
- If it reappears, the model knows:

"This is the same conceptual artifact"

This helps:

- Prevent repetition
- Maintain conversational continuity
- Allow later re-rendering in a UI host

8. Why This Works So Well Architecturally

Layer	Responsibility
MCP Server	Describe artifacts
MCP Host	Render what it can
Model	Adapt explanation & interaction

Layer	Responsibility
User	Chooses next step

No layer needs to know about the others' limitations.

9. Comparison to Older Systems

Old Approach	MCP Apps
UI hard-coded	UI optional
CLI breaks	CLI degrades
Separate APIs	Same tools
Host-specific logic	Host capability detection

10. One-Sentence Summary

In a CLI, the same MCP App degrades gracefully by ignoring `ui://` resources, rendering only text content, and letting the model adapt the interaction flow—while the server remains completely unchanged and standards-compliant.

If you want, next we can:

- Show a **real Codex CLI example**
- Diagram **host capability negotiation**
- Or compare this to **how Open Agent Skills degrade**






What are the required steps to implement an strictly MCP App extension compliant App for ChatGPT? Do I still use

the Apps SDK?

Below is a **clear, standards-first answer** that separates **what is strictly required by the MCP Apps extension** from **what the ChatGPT Apps SDK adds**, and tells you exactly **when you do and do not need the SDK**.

Short Answer (TL;DR)

-  You do *not* need to use the ChatGPT Apps SDK to build a **strictly MCP Apps–compliant app** that works in ChatGPT.
-  You **do need** to implement:
 1. A compliant **MCP server**
 2. The **MCP Apps extension** (`ui://` resources + JSON-RPC UI bridge)
 3. A sandboxable **HTML/JS UI**
-  You may *optionally* use the ChatGPT Apps SDK **as a convenience layer**, but doing so means your app is no longer *strictly* MCP Apps–only (it becomes MCP Apps + ChatGPT extensions).

1. What “Strictly MCP Apps–Compliant” Means

A strictly compliant MCP Apps app:

- Uses **only**:
 - MCP core
 - MCP Apps extension
- Avoids:
 - `window.openai`
 - ChatGPT widget runtime APIs
 - ChatGPT-specific metadata or helpers
- Works unchanged in:
 - ChatGPT
 - Claude
 - IDEs

- CLI hosts (with graceful degradation)

ChatGPT acts purely as an **MCP Apps host**, not a special platform.

2. Required Steps (Standards-Only)

Step 1: Implement an MCP Server (Core)

Your server must support MCP JSON-RPC, at minimum:

- `initialize`
- `tools/list`
- `tools/call`

This is mandatory regardless of host.

✓ No OpenAI SDK required

✓ No Apps SDK required

Step 2: Define Tools Normally (No UI Yet)

Example:

json

 Copy code

```
{
  "name": "show_sales_dashboard",
  "description": "Display sales metrics for an account",
  "inputSchema": {
    "type": "object",
    "properties": {
      "account_id": { "type": "string" }
    },
    "required": ["account_id"]
  }
}
```

This is **pure MCP core**.

Step 3: Return a `ui://` Resource (MCP Apps Extension)

When the tool is called, your response must include:

- `content` (text fallback)
- `resources` with a `ui://` URI

Example:

json

 Copy code

```
{
  "content": [
    {
      "type": "text",
      "text": "Sales dashboard loaded."
    }
  ],
  "resources": [
    {
      "uri": "ui://sales-dashboard",
      "mimeType": "text/html",
      "name": "Sales Dashboard",
      "annotations": {
        "mcp.app": {
          "entrypoint": "/index.html",
          "sandbox": true
        }
      }
    }
  ]
}
```

This is the **core of MCP Apps compliance**.

Step 4: Serve a Sandboxed HTML/JS UI

Your MCP server must serve the UI bundle (e.g. `/index.html`).

Your UI must:

- Run in an iframe
- Communicate **only** via MCP Apps JSON-RPC
- Use `postMessage`, not ChatGPT APIs

Example (simplified):

js

 Copy code

```
window.parent.postMessage({  
  jsonrpc: "2.0",  
  method: "mcp.app.ready"  
}, "*");
```

And for interaction:

json

 Copy code

```
{  
  "jsonrpc": "2.0",  
  "method": "mcp.app.callTool",  
  "params": {  
    "tool": "get_sales_data",  
    "arguments": { "range": "30d" }  
  }  
}
```

This is **required** for strict compliance.

Step 5: Handle Tool Calls from the UI

Your MCP server must accept:

- Tool calls initiated by:
 - The model
 - The UI (via MCP Apps bridge)

No distinction is required server-side.

Step 6: Include Text Fallbacks (Strongly Recommended)

To ensure graceful degradation:

- Always include useful `content` `text`
- Never rely solely on UI for meaning

This is not strictly required by the spec, but **expected behavior** for MCP Apps.

3. What ChatGPT Does Automatically

If you follow the steps above, **ChatGPT will**:

- Detect `ui://` resources
- Create a sandboxed iframe
- Load your HTML
- Route JSON-RPC messages correctly
- Ignore the UI in CLI / non-UI contexts

You do **not** need to register anything special with ChatGPT beyond making your MCP server available.

4. Do You Still Use the ChatGPT Apps SDK?

Not Required

For **strict MCP Apps compliance**:

- You should **not** use the Apps SDK
- You should **not** rely on ChatGPT-specific runtime APIs

Optional (Convenience Only)

You *may* use the Apps SDK if:







- You want faster local dev
- You want ChatGPT-only enhancements
- You're okay with reduced portability

But then your app becomes:

MCP Apps—compatible, but ChatGPT-extended

This is fine for many products — just not “strict”.

5. Comparison Summary

Goal	Use Apps SDK?
Strict MCP Apps compliance	 No
Maximum portability	 No
Works in ChatGPT	 Yes
Works in Claude / IDEs	 Yes
Faster dev / helpers	 Optional
ChatGPT-only features	 Yes

6. Recommended Architecture (Best Practice)

Server

- MCP core + MCP Apps
- No host-specific logic

UI

- Plain HTML/JS
- MCP Apps JSON-RPC only

Host

- ChatGPT (or others) render what they support

This is the cleanest, future-proof design.

7. One-Sentence Final Answer

To implement a **strictly MCP Apps-compliant app for ChatGPT**, you build a standard MCP server that returns `ui://` resources and uses the MCP Apps JSON-RPC UI bridge; **you do not need the ChatGPT Apps SDK**, which is optional and adds ChatGPT-specific conveniences on top of the standard.

If you want next, I can:

- Provide a **minimal reference implementation**
- Show how to **validate strict compliance**
- Or map **MCP Apps vs Open Agent Skills** end-to-end



Using an MCP App strict approach is it still possible to create an integrated App for ChatGPT that becomes available in the Apps section of ChatGPT?

Short answer: **No — not by itself.**

A strict MCP Apps-only implementation is sufficient for ChatGPT to *run* your app, but **not sufficient for it to be *listed* in the ChatGPT Apps section.**

Below is the precise breakdown, without hand-waving.

1. Two Different Things That Often Get Conflated

It helps to separate these clearly:

A. Runtime compatibility

Can ChatGPT execute and render this app?

B. Product integration & distribution

Does this app appear in the ChatGPT “Apps” gallery/section and get installable by users?

These are governed by **different mechanisms**.

2. What a Strict MCP Apps Implementation Gives You

If you build a **strict MCP Apps-compliant app** (as described earlier):

✓ ChatGPT can:

- Discover your tools via MCP
- Render `ui://` resources
- Host your UI in a sandboxed iframe
- Route UI → tool → model interactions
- Gracefully degrade in non-UI contexts

In other words:

ChatGPT can act as a fully compliant MCP Apps host.

This is a **technical capability**.

3. What It Does *Not* Give You

A strict MCP Apps server **does not automatically**:

- Register your app with ChatGPT
- Make it discoverable in the Apps section
- Give it an icon, listing, permissions page, or install flow
- Enable user-level installation / removal
- Go through OpenAI's app review & policy checks

Those are **product-level concerns**, not protocol concerns.

4. How the ChatGPT “Apps” Section Actually Works

The **Apps section** in ChatGPT is:

- A **curated, registered catalog**
- Backed by **OpenAI-managed metadata**

- With:
 - App name, icon, description
 - Permission scopes
 - Review & approval
 - User install state
 - Versioning and rollout controls

Today, **that catalog is driven by the ChatGPT Apps platform**, not by generic MCP discovery.

In practical terms:

Being MCP Apps-compliant \neq being an App listing

5. So How *Do* Apps Get Into the Apps Section?

To appear in the Apps section, you currently must:

1. **Register the app with OpenAI** as a ChatGPT App
2. Provide required **app metadata** (name, icon, description, policies)
3. Pass **review & safety checks**
4. Use the **ChatGPT Apps integration path** (which today is surfaced via the Apps SDK / App registration flow)

Even if your runtime logic is **100% MCP Apps-standard**, the *listing* still goes through ChatGPT's app system.

6. The Important Subtlety (This Is the Key Insight)

You can combine both approaches:

Strict MCP Apps at runtime

- No `window.openai`
- No ChatGPT-only APIs
- Fully portable app

✓ ChatGPT App registration at distribution time

- App appears in the Apps section
- Users can install it
- ChatGPT launches it as an MCP App host

In that case:

ChatGPT is just one MCP Apps host — but it's the one providing discovery and distribution.

This does **not** violate strict MCP Apps principles, as long as:

- Your *server + UI* remain MCP-only
- ChatGPT-specific behavior is limited to *listing and launch*

7. Why This Distinction Exists (Architectural Reason)

MCP (and MCP Apps) intentionally **does not define**:

- App stores
- Trust & safety review
- Monetization
- User install flows

Those are:

- Host-specific
- Product-layer responsibilities

So it's expected—and correct—that:

- **MCP Apps standardizes execution**
- **ChatGPT product controls distribution**

8. Final Answer (Very Explicit)

Using a strict MCP Apps approach, you can build an app that runs perfectly inside ChatGPT — but that alone will not make it appear in the ChatGPT Apps section.

To be listed there, you still need to go through ChatGPT's App registration and review process. The good news is that your app's *runtime implementation* can remain fully

MCP Apps—compliant and portable.

If you want, next we can:

- Map **exactly which parts are “runtime MCP” vs “ChatGPT product”**
- Discuss **future convergence** (where MCP Apps *may* become the native app listing format)
- Or compare this with **how Open Agent Skills are discovered vs listed**

This is a genuinely important design boundary, and you’re asking exactly the right questions.



Is it possible to use the Codex CLI to help generate a ChatGPT App that is MCP App extension compliant?

Yes — **the Codex CLI can absolutely be used to help generate a ChatGPT App that is MCP Apps—extension compliant**, with a few important clarifications about *what it helps with* and *what it cannot do for you*.

Below is a precise, non-marketing explanation.

Short Answer

Yes, the Codex CLI is well-suited to:

- Generate a **strict MCP Apps—compliant MCP server**
- Generate a **standards-compliant MCP Apps UI (HTML/JS)**
- Validate **graceful CLI degradation**
- Keep you honest about **not using ChatGPT-specific APIs**

No, it cannot:

- Register the app in the ChatGPT Apps catalog

- Handle App review, listing metadata, or permissions
- Replace the ChatGPT Apps submission process

Think of Codex CLI as a **code-generation and validation tool**, not a **distribution channel**.

What Codex CLI Is Good At (for MCP Apps)

1. Generating a Strict MCP Server

Codex CLI excels at producing:

- JSON-RPC MCP servers
- Tool schemas
- MCP Apps-compatible tool responses
- `ui://` resources
- Text fallback logic

Example prompts you can give Codex CLI:

“Generate an MCP server in Node.js that implements one tool and returns a `ui://` resource strictly according to the MCP Apps extension. Do not use any ChatGPT-specific APIs.”

Because Codex runs in a **code-first, CLI environment**, it naturally:

- Avoids ChatGPT runtime globals
- Encourages spec-driven correctness
- Produces portable code

This is a *feature*, not a limitation.

2. Generating the MCP Apps UI (HTML/JS)

Codex CLI is also very effective at generating:

- `index.html`
- Vanilla JS UI
- `postMessage` -based MCP Apps JSON-RPC bridge
- No reliance on `window.openai`, React SDKs, or host globals

You can explicitly instruct:

“The UI must communicate with the host only via MCP Apps JSON-RPC messages and must run in a sandboxed iframe.”

That makes the result **strictly MCP Apps-compliant** by construction.

3. Enforcing Graceful Degradation

Because Codex CLI itself is **non-graphical**, it’s ideal for testing:

- Text-only output paths
- CLI-safe tool responses
- Model-driven interaction instead of UI-only logic

You can ask it to:

- Simulate a CLI host
- Verify that `content` is sufficient without UI
- Ensure no UI-only assumptions leak into the server

This is much harder to do if you start inside a rich UI.

4. Avoiding Accidental ChatGPT Lock-In

Using Codex CLI helps prevent common mistakes such as:

- Using ChatGPT-only widget APIs
- Assuming iframe globals
- Hardcoding host behavior
- Mixing distribution metadata into runtime logic

In practice, **Codex CLI nudges you toward portable MCP-first design.**

What Codex CLI Cannot Do

1. It Cannot Make Your App Appear in ChatGPT’s Apps Section

Codex CLI:

- Does not register apps
- Does not publish listings
- Does not manage permissions or user installs

Those steps are **product-level** and must be done via OpenAI's ChatGPT App registration flow.

You can generate the *code*, but not complete the *listing*.

2. It Cannot Replace the ChatGPT Apps SDK (If You Choose to Use It)

If you decide to:

- Use ChatGPT-only UI helpers
- Use SDK-specific components
- Optimize specifically for ChatGPT

Then Codex CLI can still generate code — but **that code will no longer be strictly MCP Apps-only**.

This is a choice, not a limitation.

Recommended Workflow (Best Practice)

Here's the cleanest, future-proof approach:

Step 1: Use Codex CLI to Generate a Strict MCP Apps App

- MCP server
- Tools
- `ui://` resources
- HTML/JS UI
- Text fallbacks

Step 2: Test in a CLI Host

- Validate graceful degradation
- Ensure meaningful text output
- Confirm no UI assumptions

Step 3: Run in ChatGPT (as an MCP Host)

- Confirm UI renders
- Confirm tool calls work
- Confirm no ChatGPT-specific dependencies

Step 4: (Optional) Register as a ChatGPT App

- Keep runtime code unchanged
- Add listing metadata only
- Let ChatGPT act as *one MCP Apps host among many*

This preserves **strict MCP compliance** while enabling **ChatGPT distribution**.

One-Sentence Bottom Line

Yes — the Codex CLI is an excellent tool for generating and validating a strictly MCP Apps–compliant ChatGPT App, but it helps with code and standards compliance, not with app listing or distribution in the ChatGPT Apps section.

If you want, I can next:

- Provide a **Codex CLI prompt pack** for MCP Apps
- Show a **minimal reference repo layout**
- Or map **Codex CLI → MCP Apps → ChatGPT App registration** end-to-end

