Enriching Your Wiki with External Data — RESTful APIs and Skills

2026 DCI Summer Workshop, Day 3 Session 1

Kwok-leong Tang

2026-05-20

Table of contents

Today's session

In Session 08 you built your own LLM Wiki by ingesting a real research corpus — 29 sources on Tang–Song fiscal history. At the end of that session you met **Skills** — the layer that lets Codex reach *outside* your raw sources for data they never contained. Yesterday in Session 05 you already used **model APIs** (LM Studio, Google AI Studio, OpenRouter) and learned that the OpenAI-compatible request shape is portable across providers.

Today's session takes the same idea — *one workflow, many backends* — and applies it to the **other** kind of API you will meet constantly as a humanities researcher: the **RESTful data API**. Public humanities databases (CBDB, CHGIS, Harvard LibraryCloud, Wikidata) all expose their data this way. You will:

1. Distinguish a **RESTful API** from a model API, and understand why each one deserves a different trust level in your wiki's sources.
2. Watch one guided walkthrough against **Harvard LibraryCloud**, and turn that walkthrough into a **Skill**.
3. Pick **CBDB** or **CHGIS / TGAZ** and repeat the workflow on your own — docs → Codex → Skill → use it on your wiki.

> **Note**
>
> **Teaching goal**: leave with (a) the *workflow* — docs → LLM → Skill — that scales to any new API you meet for the rest of your research career; (b) one Skill you built yourself and used on your wiki; (c) a clear mental model for why a fact pulled from a REST API is worth more in a research artifact than a fact pulled from a model API.

Block 0 — Where we are

Eight sessions in. The map so far:

Sessions	What you got	Day
02	What an LLM actually is — tokens, probabilities, why hallucinations happen	Day 1
03	Codex Desktop as your AI agent — settings, files, approvals	Day 1
04	Deterministic vs non-deterministic — regex, scripts, when each wins	Day 1

Sessions	What you got	Day
05	A self-contained **HTML batch tool** calling LM Studio, Google AI Studio, and OpenRouter	Day 2
06	Memory, RAG, NotebookLM — how agents *remember*	Day 2
07	Reading someone else's LLM Wiki (`kltng/digital-china-wiki`)	Day 2
08	Building **your own** LLM Wiki from a 29-source fiscal-history corpus + Skills intro	Day 2

Two threads run through all of it:

- **Your tool, your data, your machine.** Every artifact you've built so far runs locally or stays under your control.
- **Codex is a collaborator, not an oracle.** You verify. The skill is reading what the agent produced and deciding whether it's good.

Today extends the *first* thread outward — from yesterday's model APIs to today's **RESTful data APIs** — while keeping the *second* discipline intact.

Before clean APIs — scrape a real page first

Before we walk through clean RESTful APIs in the next section, confront the more common situation: **the data you want lives inside a webpage, and there is no API at all.** Most humanities sources you encounter online are like this — a Chinese-language portal, an institutional bibliography, a journal index with a search box but no documented endpoint.

For these, you do not write a scraper. You ask Codex to.

We will work against one site: https://www.wenxianxue.cn/list.html — a Chinese 文獻學 (philology / textual studies) portal whose `list.html` page indexes article entries, each linking to its own detail page.

> **Note**
>
> **We are not teaching HTML, CSS, selectors, or BeautifulSoup today.** You do not need to know any of that. Codex does. Your job is to say what you want in plain

language and verify the result.

The prompt

In Codex Desktop, start a **new chat without a project** (no folder attached — this is a throwaway extraction, not something that belongs inside one of your wikis) and send:

```
Scrape the article entries listed on https://www.wenxianxue.cn/list.html.
For each entry on that page, follow the link to its detail page and pull
out the article's title, author, source or publication, date, and the
full body text. Save everything into one Excel file (wenxianxue.xlsx)
with one row per article and a column for each field. Be polite — add
a short delay between requests so we don't hammer the site.
```

That is the whole instruction. Approve the steps as Codex inspects the list page, plans the fields, writes the script, runs it, and produces the spreadsheet. Open `wenxianxue.xlsx` in Excel or Numbers when it finishes.

> **!** Verify, do not trust
>
> Open the spreadsheet and spot-check **three rows** against the live site. Did the title transfer cleanly? Is the body text complete, or truncated? Are author and date in the right columns? If a column is mostly empty, the page does not actually carry that field — tell Codex to drop it or rename it, then rerun.

Why this comes before APIs

A scraping pass is what you do when **there is no API**. The output looks similar to what you will get from CBDB or LibraryCloud in a moment — a tidy structured dataset — but the path is different: you depend on the HTML layout staying the same, you respect the site's rate limits, and you keep the raw HTML somewhere in case you need to re-extract. The next section is what life looks like when the data provider has done that work *for* you and published a stable URL pattern. After today you have both moves in your toolkit, and you can tell, looking at a new source, which one applies.

RESTful APIs and the Skill-building workflow

In Session 05 you used two flavors of **model** API: **local model serving** (LM Studio) and **hosted model serving** (Google AI Studio, OpenRouter). Both happen to use the OpenAI

request shape, so they look similar. But neither is really a *RESTful* API in the textbook sense — they are streaming chat endpoints with a single "chat-completions" verb.

The humanities databases you want to enrich your wiki with — CBDB, CHGIS, Harvard LibraryCloud — are something different. They are **RESTful APIs over structured data**.

What "RESTful" means

A RESTful API exposes **resources** at **URLs** with **standard HTTP verbs** (mostly `GET` for reading). You build a URL that names what you want, send it, and get structured data back — almost always **JSON**, sometimes XML.

A handful of examples to anchor the shape:

```
GET https://cbdb.fas.harvard.edu/cbdbapi/person.php?name=朱熹
GET https://api.lib.harvard.edu/v2/items.json?q=su+shi&limit=10
GET https://chgis.hudci.org/tgaz/placename?n=Hangzhou&fmt=json
```

You can paste any of these into your browser. The response is text — usually JSON. There is no SDK. There is no chat. There is just *"build the right URL, get back data."*

What changes between two RESTful APIs is:

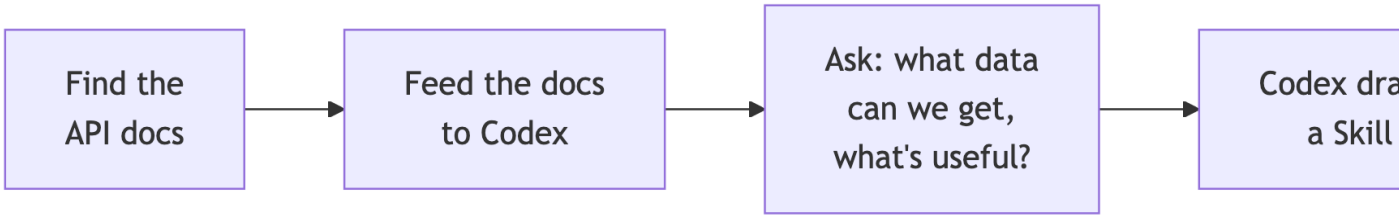
1. **The base URL** (different host).
2. **The path** (`/person.php`, `/items.json`, `/placename`).
3. **The parameters** (`name=`, `q=`, `limit=`).
4. **The response shape** (different JSON fields).

That is it. Once you know how to read one API's docs, you know how to read all of them.

> **i** Versus model APIs
>
> A model API is fundamentally **non-deterministic** — same prompt, different answer. A RESTful data API is **deterministic** — same URL, same data, today and a year from now (modulo the database itself changing). When you enrich a wiki page from a REST API, you are pulling a fact you can re-verify; when you enrich it from a model API, you are pulling a *generation*. Treat the two with different trust levels in your sources section.

The workflow we are actually teaching

You are not here to learn how to write API code. You are here to learn a workflow that scales to **any** API you meet for the rest of your career:



Every step is something you can do without writing code yourself. The output — a Skill — is portable: you can reuse it tomorrow, share it with a colleague, or hand it to a different agent.

What a Skill is, technically (briefly)

A Skill, in this curriculum, follows the **agentskills.io** specification — a cross-agent skill format that works in Codex, Claude Code, and other coding agents that support it. At the file level a Skill is a small folder:

```
my-skill/
  SKILL.md          # the agent-facing instructions
  README.md         # human-facing docs
  scripts/          # optional helper scripts the Skill calls
  references/       # optional fixed data the Skill ships with
```

The exact frontmatter and required fields are documented at https://agentskills.io. We will not memorize that schema today — **Codex will**, when we feed it the spec and ask it to write a SKILL.md.

> Skills vs MCP servers
>
> You will sometimes hear "Skill" and "MCP server" used interchangeably. They are not the same. An **MCP server** is a long-running process that exposes tools over a wire protocol — heavier setup, multi-agent reuse, language-independent. A **Skill** is a folder of markdown the agent reads on session start — lighter, single-agent, edits in your editor. For most humanities workflows, Skills are the right grain. Later today we will return to when an MCP server is worth the extra work.

Guided walkthrough — Harvard LibraryCloud

We will go through the full workflow once, together. Pick a wiki page you have that mentions a book, an author, or a Chinese title — anything bibliographic. (If you do not have one yet, write a one-line stub page for *Su Shi* in `concepts/` and use that.)

Step A — Find the docs

Open a browser tab to https://library.harvard.edu/services-tools/harvard-library-apis-datasets and find the LibraryCloud Item API page. Note the base URL, the example query, and the response format. **Do not read the docs in detail yet** — that is what Codex is for.

Copy the URL of the docs page.

Step B — Feed the docs to Codex

In Codex Desktop, in your `my-wiki` project from Session 06 (or your `liangsui-wiki` from Session 08 — any wiki with a `concepts/` folder works), send:

```
I want to build a Skill that lets you query the Harvard LibraryCloud Item API
 ↪  to enrich wiki pages with bibliographic data.

Step 1: Fetch the documentation at <paste URL>. Read it carefully.
Step 2: Summarize for me, in 5 short bullets:
  - What is the base URL?
  - What are the most useful query parameters for a humanities researcher
 ↪  (especially East Asian)?
  - What does a typical response look like — which fields are reliable,
 ↪  which are sparse?
  - Are there rate limits or auth requirements I need to know about?
  - What is one example query that would return data about Su Shi's published
 ↪  works in the Harvard collection?

Do not write any code yet. Just answer those five questions.
```

Read what Codex says. Verify it against the actual docs page if anything looks off — the docs are the source of truth, Codex's summary is a *draft*. This is the same "verify, don't trust" stance you applied to wiki pages in Session 08.

Step C — Have Codex draft the Skill

Now the construction step:

```
Good. Now build me a Skill that wraps that API, following the agentskills.io
 ↪  specification (see https://agentskills.io for the SKILL.md format).

Create the Skill in skills/harvard-librarycloud/ inside this project.
```

```
Requirements:
- A SKILL.md following the agentskills.io frontmatter and structure. The name
 ↪  should be "harvard-librarycloud". The description should make clear when
 ↪  to invoke it — "use this skill when the user asks for bibliographic data
 ↪  about a person, work, or title that might be in the Harvard library."
- The skill should expose one capability: search the LibraryCloud Item API by
 ↪  free-text query, with optional limit. Return the top results as a short
 ↪  structured summary (title, author, date, call number, identifier) — not
 ↪  the raw JSON dump.
- Include 2-3 example invocations in SKILL.md so future-me can see what good
 ↪  queries look like.
- A README.md in the same folder explaining the skill in human terms.

Do not invent fields. If you are unsure whether a response field exists, mark
 ↪  it with a TODO comment. We will verify by running the skill in a moment.
```

Approve the diff. The new folder `skills/harvard-librarycloud/` should now exist with `SKILL.md` and `README.md`.

Step D — Use the Skill, live

Restart Codex (or whatever your agent requires to pick up new Skills — check the Skill's README). Then ask:

```
Using the harvard-librarycloud skill, look up "Su Shi" in the Harvard library
 ↪  catalog. Return the top 5 results.

Then open concepts/su-shi.md and add a "Holdings in Harvard library"
 ↪  subsection citing the skill's results. Put the skill query in a comment
 ↪  so I can re-run it.
```

Approve. Read the result. **Did the Skill actually call the API and return real data?** Check by clicking through one of the returned identifiers in your browser. If Codex hallucinated results, the SKILL.md needs sharpening — go back to the draft and fix it.

This is the moment students often discover their SKILL.md was too vague. That is **the right thing to discover now**, not in a week when a colleague is depending on the skill.

Your turn — pick CBDB or CHGIS / TGAZ

Now you do the same workflow against one of:

- **CBDB API** — China Biographical Database. Person records, official titles, kinship, dates. Docs: https://projects.iq.harvard.edu/cbdb/cbdb-api (find the latest API spec page).
- **CHGIS / TGAZ** — China Historical GIS Temporal Gazetteer. Historical place names with coordinates and time spans. Docs: https://chgis.hudci.org/tgaz/.

Both are well-suited to the kinds of pages you have in your wiki. Pick the one whose data you would actually use this week.

> Neutral note on docs quality
>
> Read the docs page first before committing. Some humanities APIs have richer, more current documentation than others. If the docs feel thin, that is not a reason to give up — it is part of what you are learning to handle. Codex can often piece together a working endpoint from sparse docs plus the examples it finds via fetch, but **the thinner the docs, the more carefully you verify**. If you finish early, do the other one.

The workflow is the same as the Harvard walkthrough — **A** (find docs), **B** (Codex summarizes), **C** (Codex drafts the Skill into `skills/<name>/`), **D** (use the Skill on a wiki page, verify the data is real).

Constraints to keep yourself honest:

1. **Do not** write SKILL.md by hand. The whole point is that Codex authors it from the docs.
2. **Do** run the Skill on a real wiki page before declaring victory. Authored ≠ working.
3. **Do** check at least one returned data point against the source database directly.
4. Commit your Skill (`git add skills/ && git commit -m "Add <name> skill"`) before the wrap-up.

Reveal — `kltng/humanities-skills`

Once everyone has a working Skill, open https://github.com/kltng/humanities-skills.

This is a repository of Skills — built using exactly the workflow you just walked through — covering CBDB, CHGIS, Wikidata for East Asian entities, calendar conversion, Harvard library, and a few more. **It exists because the workflow scales**: every new database I needed for my research became one more Skill, drafted with Codex against the database's own docs, committed in an evening.

Compare what you wrote against the equivalent skill in this repo:

- Where is theirs more detailed than yours? (Usually the examples and the failure handling.)
- Where is yours actually clearer? (Often the case — fresh eyes write better top matter.)
- Are there fields or parameters in theirs you missed? Why?

Borrow what you want. The repo is permissively licensed; cloning it into your own wiki's `skills/` folder is encouraged, not cheating.

> **!** Why we held this back
>
> You needed to build one yourself before seeing a polished version. If we had opened with the repo, the question would have been *"which skill do I clone?"* and you would have stopped reading docs forever. Now the question is *"which database needs a skill next, and how do I write it?"* — and that question is yours to keep.

Wrap-up

By the end of this session you have:

1. Distinguished **model APIs** (yesterday — chat-completions endpoints) from **RESTful data APIs** (today — URL-pattern queries over structured data), and understood why each one deserves a different trust level in your wiki's sources.
2. Walked the **docs → Codex → Skill → use** workflow end-to-end against Harvard LibraryCloud, then **repeated it independently** on CBDB or CHGIS.
3. Seen a real-world skills repository and confirmed your workflow produces something that fits in it.

Looking ahead — rest of Day 3

The next sessions go further into the same arc:

- **Classical DH methods, driven by agents** (Session 10, next) — TEI, NLP, historical GIS, network analysis, with the agent as the front door.
- **More raw sources** — your own research material, not a corpus we hand you. OCR pipelines and file-system organization for the raw layer.
- **Writing your own Skill from scratch** — going beyond wrapping someone else's API, into building Skills that compose multiple sources or run deterministic computations.
- **Skill vs MCP server** — when the lightweight Skill folder is not enough, and how the upgrade path looks.

Summary — what you should take away

1. **A RESTful API is a URL pattern.** Once you can read one API's docs, you can read all of them. Codex shortcuts the reading; you verify the result.

2. **Model APIs and data APIs deserve different trust.** A fact pulled from CBDB or LibraryCloud is reproducible; a "fact" generated by an LLM is not. Capture the source — and the URL — in your wiki page.
3. **The workflow is the lesson.** Docs → Codex → Skill → use → commit. Repeat for every new database you meet. Skills accumulate; the wiki grows wider.
4. **Authored ≠ working.** Always run a new Skill on a real wiki page and verify at least one returned data point against the source database before declaring it done.

References

- agentskills.io — the cross-agent Skill specification.
- Harvard LibraryCloud — bibliographic data API.
- CBDB API — China Biographical Database.
- CHGIS / TGAZ — China Historical GIS Temporal Gazetteer.
- `kltng/humanities-skills` — the worked-example skills repository revealed at the end of this session.