Memory, Context, and the LLM Wiki

Why models forget, what we put back — 2026 DCI Summer Workshop, Day 2 Session 2

Kwok-leong Tang

2026-05-19

Table of contents

Today's session

This morning you built a tool and pushed it to the public web. You changed what *your laptop* can do.

In this block we step back from the tools and ask a deeper question: **how does the model itself know anything**, and when it doesn't know something, what is our recourse? The answer reframes everything you will do for the rest of the workshop — and is, more than any single tool, the curriculum's central idea.

> **Note**
>
> **Teaching goal**: students should leave with a clear mental model of (a) what an LLM cannot do at inference time, (b) the two main strategies for giving it the context it lacks — RAG and the LLM Wiki — and (c) the trade-off between them. We will use this distinction for the rest of Day 2 and Day 3.

> **Session materials**
>
> **Download `session_06_materials.zip`** — or grab the individual file: `README.md`.

The stubborn genius

Across the last five sessions you have run the same model — Qwen3.5 0.8B — in many ways: in chat, in scripts, in browser tools, through agents. **It is the same fixed object every time.** Now consider what that object can and cannot do.

At inference time, the model …	Can it?
Use everything it learned during training to write fluent prose	☐
Read what's in the current prompt and respond to it	☐
Change its own parameters based on something you said	☐
Remember what you said in a previous chat	☐
Look up a fact that was published after its training cutoff	☐
Learn from anything new about you, your data, your research	☐

A useful image: the model is **a stubborn genius**. Enormously well-read, very fluent, frozen since its training cutoff, unable to update its mind, with the conversation-memory of a goldfish between calls. You cannot teach it anything new at runtime by *changing* it.

But you have one knob: **what you put in its prompt**. The prompt — system message + user message + any attachments or tool outputs — is the only fresh information the model sees every call. Everything else is hard-coded into the weights.

> **!** Important
>
> **The whole rest of the workshop is about how to use that one knob well.** Better prompts (session 02), better tool architecture (session 03), better message structure (session 04), better deployable interfaces (session 05) — all variations on the same theme: *we cannot change the model, so we change what it sees.*

Two paths to giving the model context

If the model only sees what is in the prompt, and the prompt has a budget (the context window), then the practical question becomes: **how do we choose what to put in the prompt for any given query?**

The field has two dominant answers.

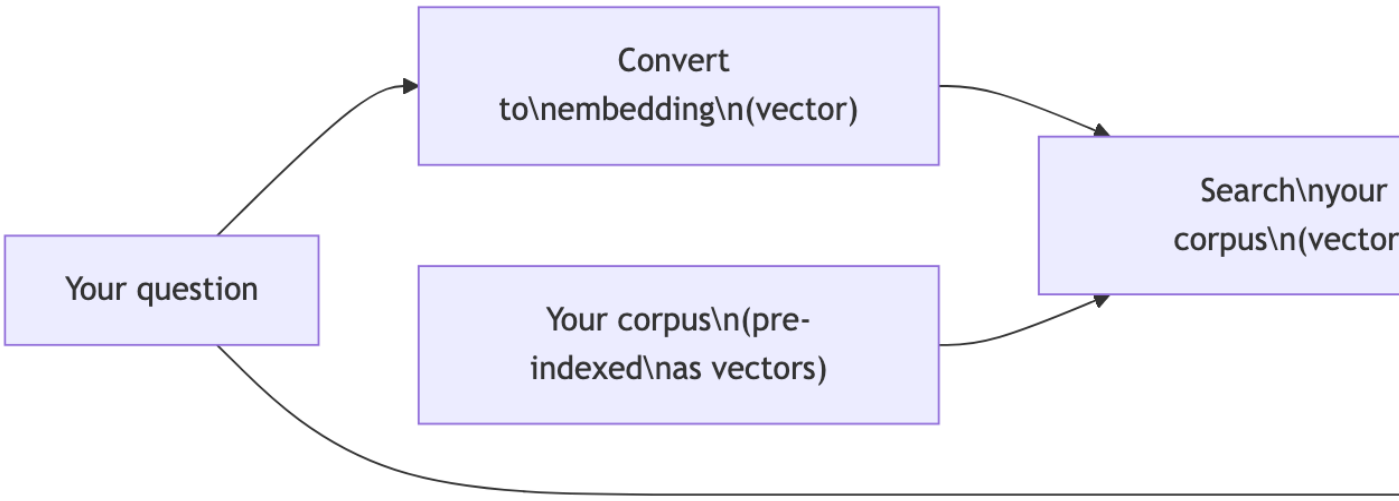
Approach	The idea	Slogan
RAG (Retrieval-Augmented Generation)	**Find** the relevant chunks of your corpus at the moment of the query and inject them into the prompt	*Retrieve, then generate*
LLM Wiki	**Compile** your corpus into durable, well-written wiki pages ahead of time; load whole pages as context	*Compile, then load*

The rest of today is about both. We will look at RAG first (because it is the better-known industry term), demonstrate it with NotebookLM, then introduce the LLM Wiki as the alternative philosophy. Sessions 07 and 08 then build out the LLM Wiki side in depth.

RAG — Retrieval-Augmented Generation

The logic

The classical RAG flow has four steps:



Two pre-conditions and four runtime steps:

- **Pre**: every document in the corpus has been chunked into passages and each passage has been turned into a vector. Vectors are stored in a database that supports nearest-neighbour search.
- **Runtime step 1**: the user's question is turned into a vector with the same procedure as the corpus.
- **Step 2**: the search finds the passages whose vectors are *closest* to the question's vector.
- **Step 3**: the top matches are injected into the model's prompt as context, alongside the question.
- **Step 4**: the model writes an answer grounded in the injected context.

Two terms in this story need unpacking: **embedding** and **vector search**. We will explain both in plain English.

What is an embedding?

An **embedding** is a list of numbers — typically a few hundred to a few thousand of them — that represents the *meaning* of a piece of text. Two pieces of text whose meanings are similar end up with similar lists of numbers. Two pieces whose meanings are different end up with different lists.

You do not compute embeddings by hand. A specialized model (an *embedding model* — often a small transformer) takes text in and outputs the vector. Common embedding models include `text-embedding-3-small` (OpenAI), `bge-large` (open-source), `Qwen3-Embedding`, and many others.

The classic illustration uses a low-dimensional simplification. Imagine reducing each text's embedding to **just two numbers** so we can plot it on a 2D graph. Then:

Two things to see in this picture:

1. **Semantically similar words cluster.** `boat` and `ship` are near each other in vector space; both are far from `king` and `queen`. The geometry encodes meaning.
2. **Relationships have direction.** The arrow from `man` to `woman` points in roughly the same direction as the arrow from `king` to `queen`. The famous *"king − man + woman ≈ queen"* equation is just vector arithmetic on these representations.

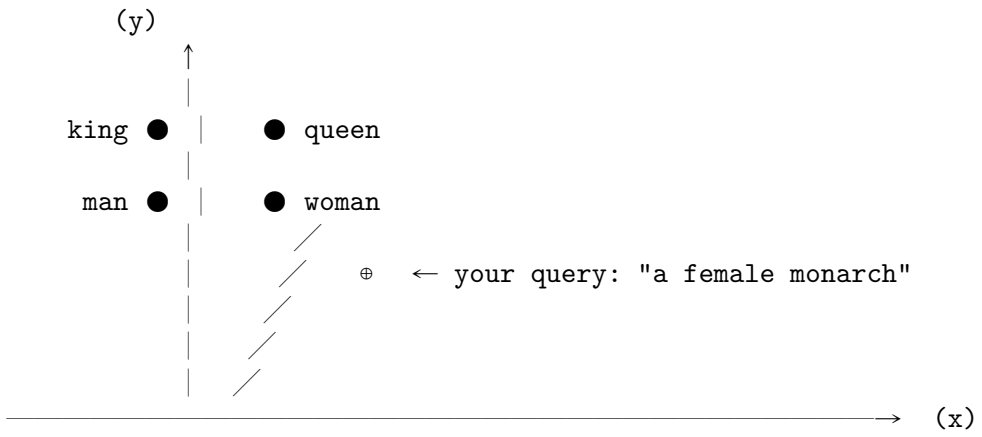
Real embeddings live in 384, 768, 1024, or more dimensions — far more than humans can visualize — but the same intuition applies. *Closeness in vector space approximates similarity in meaning*, and the embedding model has been trained on enormous amounts of text to make that true.

> Interactive demo
>
> Google's free **TensorFlow Embedding Projector** lets you explore real word embeddings in 3D: https://projector.tensorflow.org. Pick the "Word2Vec 10K" dataset, type a word into the search box, and watch the nearest words appear. This is the same principle every RAG system uses, scaled up to sentences and paragraphs.

What is vector search?

Vector search is **finding the nearest vectors to a given vector**. If your corpus has 100,000 passages, each represented by a 768-number vector, vector search is the algorithm that — given a *query* vector — returns the (say) 5 closest corpus vectors and the passages they came from.

In our 2D picture:



The query *"a female monarch"* becomes a vector that lands somewhere between `queen` and `woman`. The vector search returns its top neighbours: `queen` first, `woman` and `king` not far behind. The system then takes the passages those vectors came from and injects them into the prompt.

The "nearest" calculation is usually **cosine similarity** (how aligned two vectors point) rather than literal Euclidean distance, but the visual intuition is the same.

> **i** Why this matters for humanities research
>
> Vector search lets you ask **conceptual** questions of a corpus, not just keyword questions. A keyword search for "ancestor worship" misses every passage that uses different vocabulary for the same idea. A vector search retrieves passages whose meaning is close, regardless of which words were used. This is the single biggest reason humanities researchers care about RAG.

What lives where in a RAG system

Putting it all together:

Component	What it does	Examples
Embedding model	Turns text into vectors	OpenAI `text-embedding-3-small`, Qwen3-Embedding, BGE
Vector database	Stores vectors and runs nearest-neighbour search	Qdrant, Chroma, FAISS, pgvector
Chunking strategy	Decides how to split source documents	By paragraph, by token count, by semantic boundary

Component	What it does	Examples
Retriever	The pipeline that takes a question, embeds it, queries the DB, and returns matches	Custom code, or `LangChain` / `LlamaIndex`
Generator	The LLM that writes the final answer	Qwen, Claude, GPT, anything that takes a prompt

You do not need to build all of this by hand for today. **NotebookLM** does it all for you, and that is what we will use to feel RAG in action.

Hands-on — NotebookLM as a working RAG system

Google NotebookLM is a free product that takes a small corpus of sources you upload and lets you ask grounded questions of them. Under the hood, NotebookLM chunks each source, embeds the chunks, indexes them in a vector store, and assembles answers by retrieving relevant chunks and generating with Gemini. You will not see the moving parts — that is the point — but everything we just discussed is happening.

Step 1 — Open NotebookLM and create a notebook

Visit https://notebooklm.google.com and sign in with your Google account (any Google account works — Harvard Google Workspace or personal).

Click **+ Create new notebook**.

Step 2 — Load the three papers

The `materials/` folder for this session points to **three open-access Chinese-language papers** on digital humanities. They share a theme — *how do we organise, extract, and represent knowledge in digital humanities?* — and form a theory → method → application progression.

#	论文 (Paper)	链接
1	《数字人文的研究范式与平台建设》*(Research paradigms and platform construction)*	http://dik.whu.edu.cn/jwk3/tsqbzs/CN/article/downloadArticleFile.do?attachType=PDF&id=5728
2	《知识图谱在数字人文中的应用研究》*(Knowledge graphs in DH)*	https://cbdb.hsites.harvard.edu/file_url/618

#	论文 (Paper)	链接
3	《数字人文中的文本挖掘研究》 *(Text mining in DH)*	https://ccj.pku.edu.cn/Article/DownLoad?id=292861688&type=ArticleFile

⚠ Verify the URLs before relying on them

PDFs hosted on Chinese university servers occasionally move, or are reachable only from inside China. **Check that each URL opens in your browser first**; if one fails, use one of the alternative papers listed in `materials/README.md`.

In NotebookLM, click + **Add source** for each. You can either paste the URL directly (NotebookLM will fetch and index) or download the file first and upload it. Once all three are loaded, NotebookLM tells you how many tokens it indexed.

Step 3 — Ask grounded questions

Try these, one at a time, and watch the citations NotebookLM attaches to each answer (you can ask in Chinese or English — Gemini handles both):

1. **☒三篇☒文☒"数字人文"的核心☒☒各自提出了什么☒的☒述**? / *"What core question of digital humanities does each paper address?"*
2. **三篇☒文在方法☒上有哪些共同点和不同点**? / *"What methodological commonalities and differences exist across the three papers?"*
3. **每篇☒文☒出的具体案例** *(case studies)* **分别☒明了什么**? / *"What does each paper's case study illustrate?"*
4. **如果要☒用其中一个框架到自己的研究☒域, 哪个最合适?☒什么**? / *"Which framework would you most likely apply to your own research, and why?"*

Each answer should come back with **citation markers** — small numbered links that take you to the exact passage in the source PDF the answer was drawn from. **Click them.** The citations are the visible evidence that RAG is doing what it claims.

Question

Ask one question the sources collectively *cannot* answer — e.g., *"***☒三篇☒文如何☒价大型☒言模型在数字人文中的☒用***?"* (most papers predate the LLM era, so they probably do not discuss LLMs at all). Watch what NotebookLM does. A well-behaved RAG system will say *"the sources don't say"* or note that the topic is outside the corpus. A poorly-behaved one will hallucinate. NotebookLM is in the well-behaved camp; this is one reason it is a useful teaching tool.

Step 4 — Generate and download your takeaway artifact

In NotebookLM's right pane, find the **Studio** section. Two options worth knowing:

- **Briefing doc** — a written summary stitched from the sources. Click *Generate*, wait ~15 seconds, then download (top-right menu → Save as PDF or copy markdown).
- **Audio overview** — generates an ~8-minute podcast where two synthetic voices discuss the sources. Can be downloaded as `.mp3`.

Save one of these before moving on. It is your session 06 takeaway: a NotebookLM-built summary of the three papers, generated entirely from a vector-indexed corpus you assembled in five minutes.

What NotebookLM does well, and where it has limits

Strength	Limit
Five-minute setup; no plumbing	Locked to Google's ecosystem; you can't swap out the embedding model or the generator
Citations are real and clickable	Citations point only inside the sources you loaded, not the wider literature
Handles PDFs, web pages, audio, video transcripts	No durable editable corpus — what you load is *only* loaded
Answers stay grounded in the sources	If you add another paper later, old answers don't update; you re-ask

The last point is worth dwelling on. **NotebookLM's index regenerates as you add sources**, but the *answers* you generated yesterday are not automatically refreshed when today's source is added. There is no durable, edit-as-you-go knowledge artifact that accumulates between sessions. The wiki you build inside NotebookLM is ephemeral.

This is the gap the **LLM Wiki** approach is designed to close.

The LLM Wiki — context by compilation, not retrieval

So far we have used the model **plus retrieval**: the model itself stays a stubborn genius; at query time we ferret out the right passages and inject them. That works for many tasks. It also has a specific failure mode:

- **Every query starts from raw passages.** The model has to re-read and re-synthesise the same source material every time you ask a question.

- **The retrieval can miss.** If the chunking split a key sentence across two chunks, or if the question uses vocabulary that doesn't match the corpus, the nearest-neighbour search returns nothing useful.
- **The output is lossy.** RAG answers are one-shot synthesis — they exist only as long as the conversation does.

In April 2026 Andrej Karpathy proposed a different paradigm. Have the LLM read each source carefully **once**, write a **durable wiki page** about that source's contents, and then — at query time — load *whole wiki pages* into context instead of retrieved chunks. The wiki is a **compounding, persistent artifact**. It is built once and reused many times.

He calls this the **LLM Wiki** pattern. The slogan: **compile over retrieve**.

RAG vs LLM Wiki — different philosophies

	RAG	LLM Wiki
When is work done?	At query time	Ahead of time (ingest phase)
What is the durable artifact?	A vector database of chunks	A folder of human-readable markdown pages
What does the user read?	The model's answer + citations	The pages directly, or an answer with page references
Editable by a human?	Hard — chunks are abstract	Easy — markdown files
Compounds across sessions?	No	Yes
Easy to share with a colleague?	Send them the vector DB + retrieval code	Send them the folder
Best for	Fast lookup over a large corpus	Long-running scholarship on a focused topic

The two are **not mutually exclusive**. You can build a wiki *and* RAG it. But the wiki is what you *keep*; the retrieval is just an option for navigating it.

A short tour of an LLM Wiki

An LLM Wiki has roughly this canonical shape — and this afternoon, in Session 07, we will open `kltng/digital-china-wiki` (a public, ~1000-page LLM Wiki on digital resources for China studies) and walk through one in detail. For now, the *structure*:

```
my-wiki/
├── README.md
├── index.md            ← master catalog
├── log.md              ← change history (every wiki edit logged here)
```

```
├── AGENTS.md           ← rules for agents writing into the wiki
├── concepts/           ← short pages, one per concept
├── patterns/           ← larger architectural ideas
├── tools/              ← one page per tool
├── models/             ← one page per LLM family
├── people/             ← one page per relevant author
├── examples/           ← worked teaching demos
├── readings/           ← annotated source papers
├── datasets/, skills/, ...
```

Every page is plain Markdown with YAML frontmatter (title, type, status, tags, dates). Cross-references between pages use `[[wikilinks]]`. The whole wiki is a flat folder of text files — no database, no server, version-controlled with Git. It compiles to nothing; the *files themselves* are the artifact.

A typical page has: a one-paragraph definition, a "why it matters for humanities research" framing, common misconceptions, in-teaching cross-links, related concepts, and sources.

Hands-on — write your first wiki page with Codex

Pick **one concept you have learned this morning** that you want to remember six months from now. Candidates from sessions 04 and 05:

- *CORS* (cross-origin resource sharing) and the localhost exception
- *Mixed content* (HTTPS pages and HTTP resources)
- *FileReader* (reading a local file in the browser)
- *The OpenAI chat-completions message shape*
- *git revert vs git reset*
- *The deterministic vs non-deterministic split*
- *Why HTML is a good deliverable for humanities tools*

…or any other.

Create a folder for your wiki and open it as a Codex project:

```
mkdir -p ~/genai-workshop/my-wiki
```

(On Windows: PowerShell users run `mkdir "$HOME\genai-workshop\my-wiki"`; `cmd` users run `mkdir %USERPROFILE%\genai-workshop\my-wiki`.)

In Codex Desktop, **Projects → + → Use an existing folder** → pick `~/genai-workshop/my-wiki`.

Send this prompt (replace the topic with yours):

```
Create a markdown file in this folder for a wiki page about [YOUR TOPIC]. The
 ↪  file should follow this structure:

---
title: "[YOUR TOPIC]"
type: concept
status: draft
created: 2026-05-19
---

## Definition

(1–3 sentences in plain English.)

## Why it matters for humanities research

(Concrete framing — why would a humanities researcher need to know this?)

## Common misconceptions

(Optional — what people typically get wrong.)

## In teaching

- Where it came up for me (e.g., "2026 DCI Summer Workshop, Session 05")

## Related

- [Related concept](./other-page.md)
- [Related tool](./other-page.md)

## Sources

- (Anything you read that informed this page)

Fill in the sections from what you know about [YOUR TOPIC]. Be concrete, use
 ↪  plain English, no jargon you wouldn't use with a colleague. About 30–50
 ↪  lines is fine.
```

Approve. Open the resulting file and **read it carefully**. Does it match your understanding? If not, ask Codex to revise specific parts. If yes, this is your first page.

What you have just done

You used an LLM (via Codex) to write a *durable, plain-text, version-controllable* artifact that captures something *you just learned*. This is the atomic unit of the LLM Wiki pattern: a human picks the topic, the agent drafts, the human verifies. Repeat 50 times and you have a personal wiki on your research area. Repeat 500 times and you have something a colleague can cite.
In Session 07 you will explore a real ~1000-page LLM Wiki in Obsidian. In Session 08 you will build your own wiki from a 29-source research corpus and install **Skills** that let Codex query external databases (CBDB, Wikidata, etc.).

Looking ahead

- **Session 07 (13:30 — Obsidian and the LLM Wiki You Already Have)**: open `kltng/digital-china-wiki` (~1000 pages) in Obsidian — wikilinks, backlinks, graph view, and the four canonical Codex-on-wiki operations: read · improve · query · lint.
- **Session 08 (15:00 — Build Your Own LLM Wiki)**: scaffold and ingest a real 29-source corpus on Tang–Song fiscal history — the three-layer architecture (raw / wiki / schema) in practice, plus your first Skill.

Summary — what you should take away

1. **The model is a stubborn genius.** You cannot change it at inference time. You can only change the context — what goes into the prompt.
2. **The whole rest of this workshop is about choosing context well.** RAG and the LLM Wiki are the two dominant strategies.
3. **RAG** finds chunks at query time using embeddings + vector search, then injects them into the prompt. NotebookLM is a clean off-the-shelf example.
4. **Embeddings** are a way of representing meaning as numbers, so that similar meanings end up close in vector space. Vector search finds the nearest passages to a query.
5. **The LLM Wiki** is the alternative: have the agent read sources carefully *ahead of time* and write durable, human-readable wiki pages. Compile-once, reuse-many. Editable, shareable, version-controllable.
6. **They are not mutually exclusive.** You can RAG a wiki. But the wiki is what survives across sessions.

References

- NotebookLM — Google's RAG-as-a-product.

- TensorFlow Embedding Projector — interactive 3D embeddings.
- Andrej Karpathy, "LLM Wiki" gist (April 2026) — the canonical statement of the LLM Wiki pattern.
- Vannevar Bush, "As We May Think", *The Atlantic*, July 1945 — the 1945 ancestor.
- Andy Matuschak, "Evergreen notes" and the note-taking writings — the strongest contemporary statement of compounding personal knowledge bases.
- Patrick Lewis et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks" (2020) — the paper that named RAG.
- Wiki: `concepts/retrieval-augmented-generation.md`, `patterns/llm-wiki-pattern.md`, `patterns/compilation-over-retrieval.md` (or `concepts/compilation-over-retrieval.md`).