What is an LLM?

Nature and Limitations — 2026 DCI Summer Workshop, Day 1 Session 2

Kwok-leong Tang

2026-05-18

Table of contents

Today's session

After this session you will hold the **essential concepts** of large language models — not as definitions you memorize, but as observations you have made yourself with a model running on your own laptop.

We will use a deliberately **small, weaker model** — Qwen3.5 0.8B running locally in LM Studio. State-of-the-art models hide their failure modes behind engineering tricks. A 0.8-billion-parameter model lets us *see* what a foundation LLM actually does.

> **Note**
>
> **Teaching goal**: it's not about the tools, but the mindset. By the end you should be able to predict, before you press Enter, whether an LLM is likely to get a given task right.

What you need today

- **LM Studio** installed on your laptop. Download: https://lmstudio.ai
- **Qwen3.5 0.8B** downloaded inside LM Studio (search for `Qwen3.5 0.8B`, pick the GGUF version, Q4_K_M or Q8_0 quantization is fine).

- Qwen3.5 0.8B **loaded** in the LM Studio chat window — confirmed by sending a "Hello" and getting a reply.

> **Tip**
>
> If you have not done the setup yet, raise your hand — a TA will come over. You will still get the most out of this session by following along on a neighbor's screen until you catch up.

The Four Pillars of Generative AI Applications

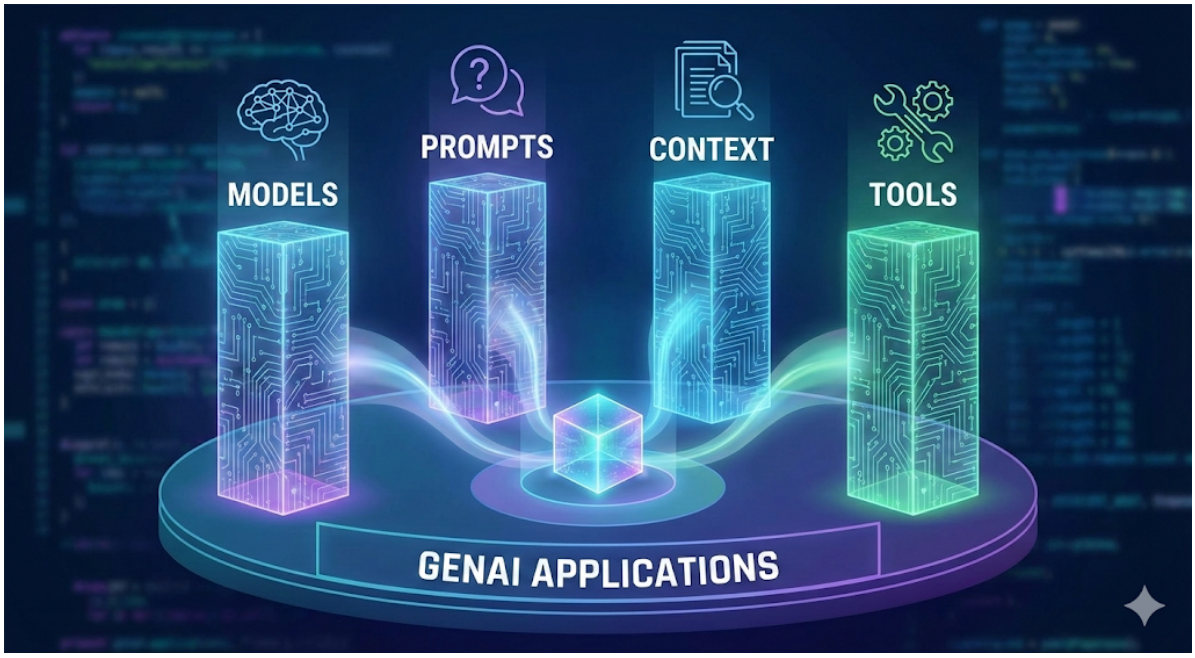

Figure 1: The Four Pillars of Generative AI Applications.

Every generative-AI application is built from four interacting pillars. When something doesn't work, your debugging vocabulary is this list:

Pillar	What it is	Today's examples
Models	The LLM itself	Qwen3.5 0.8B (local), Claude Sonnet, GPT-5.5, DeepSeek V3.2 685B

Pillar	What it is	Today's examples
Prompts	The instructions you give the model	"How many r in strawberry?", a system prompt setting persona
Context	The information you provide alongside the prompt	A PDF attached for summarization; retrieved documents in a RAG flow
Tools	External capabilities the model can call	Web search, calculators, MCP servers, file I/O

Everything in this workshop falls into one of those four buckets. Today we focus almost entirely on the first pillar — **Models** — so that you have a grounded picture of what the model actually does before we build on top of it for the rest of the week.

Models — the six things students need to understand

We will name six knobs and concepts that govern the Models pillar, and then we will *see* each of them by running prompts in Qwen3.5 0.8B.

Tokens

Models do not see characters. They see **tokens** — chunks of bytes that are often whole words or whole sub-word pieces. The word *strawberry* is typically one or two tokens to the model; the model has no internal representation of the individual letters inside.

Explore this directly: https://huggingface.co/spaces/Xenova/the-tokenizer-playground

Type a few English and Chinese sentences and watch how they break into tokens. Pay attention to how Chinese characters tokenise compared to English words.

Parameters

The number of weights inside the model. The Qwen3.5 0.8B you are running has **0.8 billion** parameters. DeepSeek V3.2 has **685 billion**. Claude Sonnet — Anthropic does not publish the number, but it is enormous.

> **!** Important
>
> More parameters usually means more capability, but not always. *Quality of training data* and *architecture* matter as much as raw count. A well-trained small model can outperform a poorly-trained large one.

Temperature

A randomness control. Temperature 0 → the model picks the most likely next token every time (nearly deterministic). Higher temperature → more variation. Most chat models default to around 0.7–0.8.

You can adjust temperature in LM Studio under the model settings on the right panel.

Knowledge cut-off

Every model has a hard time-stop. Training data ends on a particular date, and the model has no information about events after that date.

We will demonstrate this in a minute.

Thinking / reasoning models

A modern split. **Foundation models** (like our Qwen3.5 0.8B) output their answer directly. **Thinking models** (Claude with extended thinking, GPT-5.5 in reasoning mode, DeepSeek with `<think>` mode) emit an internal monologue first, then the answer.

Thinking models are usually slower and more expensive but markedly better at reasoning tasks. We are using a non-thinking model today on purpose — to see the foundation behavior.

Context window

The maximum amount of input + output the model can hold in a single call. Modern models range from 8K to 1M tokens. Qwen3.5 0.8B has a smaller window than the cloud giants, which we will feel later when we try to give it large documents.

The nature of LLMs

Now the central claim of the session. Two sentences to remember:

> **!** Important
>
> **Everything is PREDICTION.**
> **Every token is generated based on PROBABILITY.**

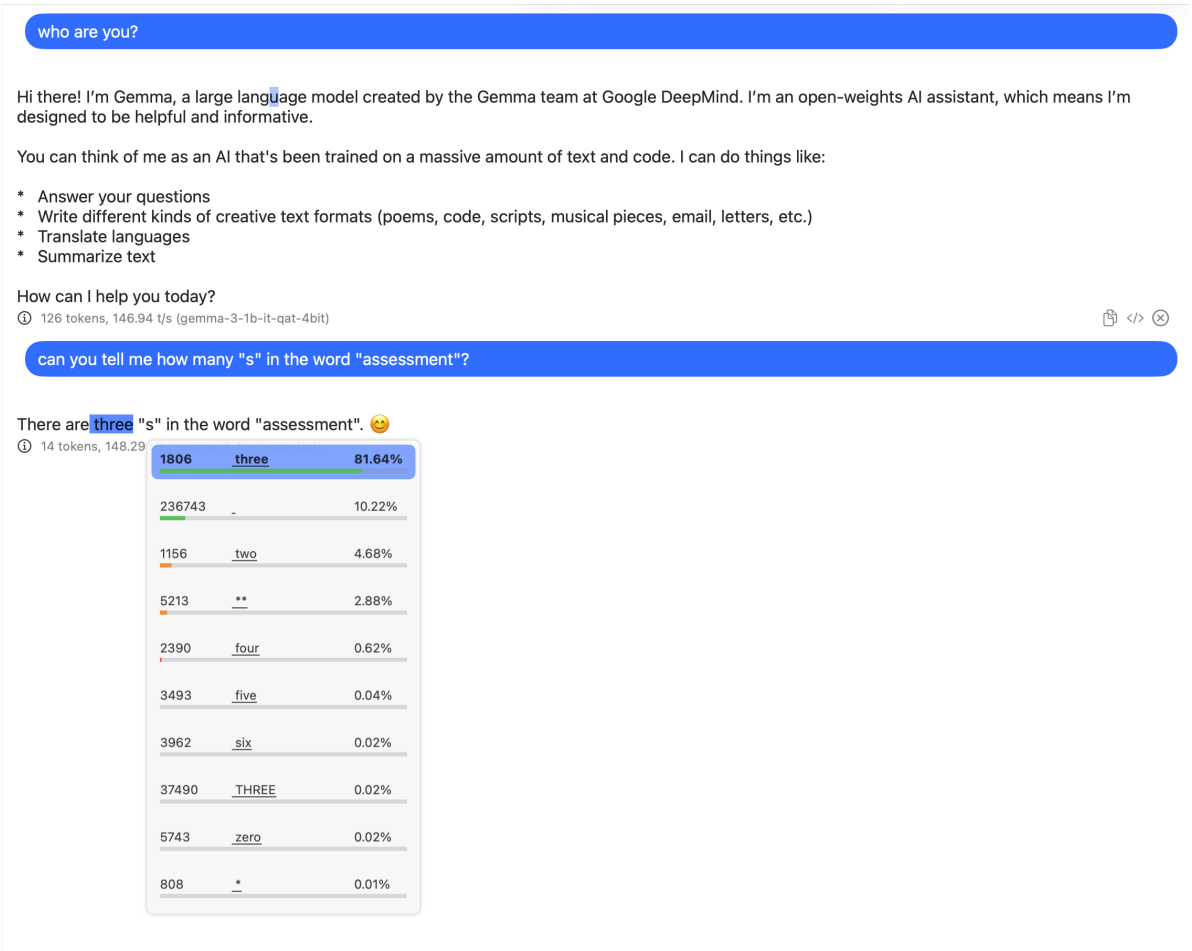

Figure 2: Probability over a vocabulary — image source: Inferencer.

The model is not "looking up" answers from a database. It is rolling weighted dice over its vocabulary, one token at a time, each die shaped by all the tokens before it. The "intelligence" is in *how the dice are shaped*.

The image below is from Chip Huyen's *AI Engineering: Building Applications with Foundation Models* (O'Reilly, 2025), Chapter 1. It shows an autoregressive LM emitting one token at a time, each token conditioned on every token before it.

Figure 1-2 shows these two types of language models.

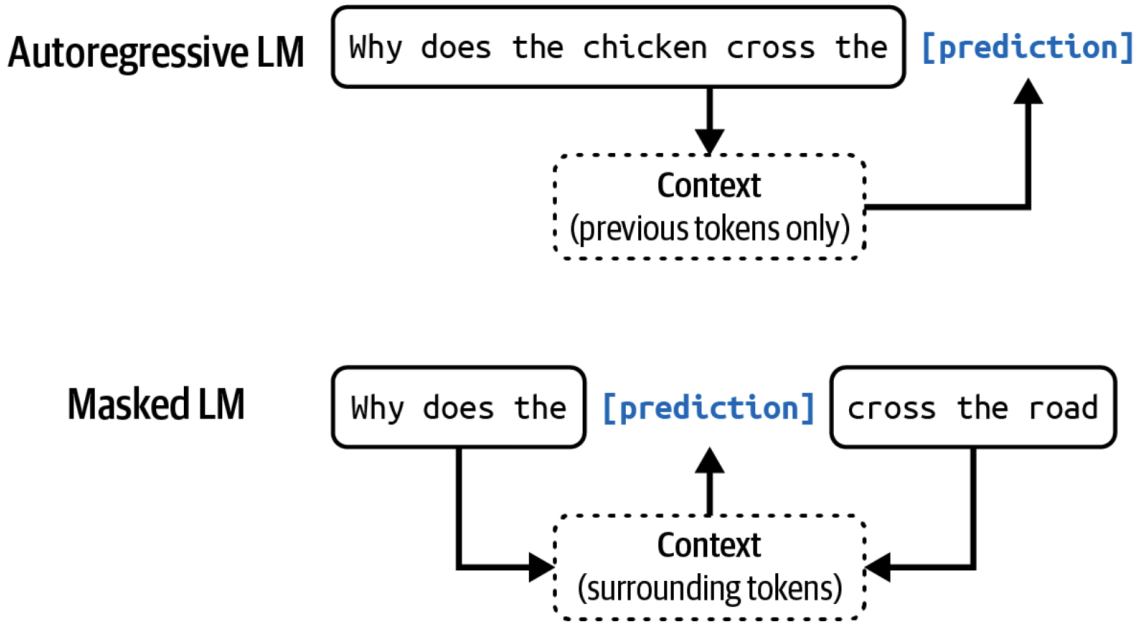

Figure 1-2. Autoregressive language model and masked language model.

Figure 3: Autoregressive language model — image source: Chip Huyen, *AI Engineering* (O'Reilly, 2025), Chapter 1.

Another lens — prediction as compression

Here is the same idea from a different angle, useful because it explains both the *fluency* and the *hallucination* of LLMs in one breath.

Think of the model as a **compressed model of its training data**. Not byte-for-byte storage — that would be impossible at hundreds of billions of parameters compressing trillions of tokens of text — but a probabilistic compression. The model has thrown away the originals and kept a statistical model of what the originals were like. At inference time, when you ask it a question, it *reconstructs* a plausible answer from this compressed model. When the reconstruction faithfully matches reality, we call it correct. When it diverges, we call it a hallucination. **The same mechanism produces both.**

> **Note**
>
> **Two ways into this idea:**
>
> - For an accessible introduction, see Ted Chiang's *New Yorker* essay "**ChatGPT**

Is a Blurry JPEG of the Web" (February 9, 2023). The JPEG analogy is the clearest one-paragraph framing of why an LLM can be simultaneously fluent and unreliable.

- For the formal result, see Delétang et al., "**Language Modeling Is Compression**" (ICLR 2024, arXiv:2309.10668). They show that a large language model paired with arithmetic coding is a *general-purpose* lossless compressor — Chinchilla 70B compresses ImageNet image patches to **43.4%** of original size (PNG: 58.5%) and LibriSpeech audio to **16.4%** (FLAC: 30.3%), despite being trained only on text. Next-token prediction and lossless compression are formally the same problem.

Keep this picture in mind for the next experiments. Hallucination is not a separate bug bolted onto an otherwise-reliable system. It is the *same* mechanism as correct answering, viewed from the wrong side.

The Galton box (or "bean machine") below is a useful physical analogy for *probability*. Each ball falls through a lattice of pegs; the same starting position can land in many different bins, but the *distribution* is predictable. An LLM is producing a sample from a distribution, not retrieving a stored fact.

Your browser does not support embedded video.

Galton box demonstration of the normal distribution (Wikimedia Commons).

Hands-on: six experiments

Each experiment below is one prompt to send to Qwen3.5 0.8B in LM Studio. Send each one exactly as written. Read the answer carefully. Compare with your neighbor — same model, same prompt, possibly different answer (that is itself a finding).

Experiment 1 — Prediction

Send the following prompt **five or six times** in a row, each in a new chat:

```
Give a random number between 1 and 100.
```

> Question
>
> What did you notice? Did the model favor any particular number?

Many runs will return **42**. Why? Because *The Hitchhiker's Guide to the Galaxy* trained the internet to think 42 is the answer to "the ultimate question of life, the universe, and everything." The model is not picking randomly — it is sampling from a distribution that has been pulled

toward 42 by every Douglas Adams reference in the training corpus. (See ChatGPT's idea of a random number.)

Takeaway: even a request for randomness produces a *prediction*, not random output. Prediction and probability are everywhere.

Experiment 2 — Knowledge cut-off

Send:

```
What is your knowledge cutoff date?
```

Then send:

```
Who is the current prime minister of Japan?
```

The model will name a prime minister consistent with whenever its training data ended (likely 2024 or earlier — so Fumio Kishida or Shigeru Ishiba). The *current* prime minister as of May 2026 is Sanae Takaichi.

> **!** Important
>
> The model **does not know** that it does not know. It will confidently give you a wrong answer about an event after its cutoff unless something in the context tells it otherwise.

This is the first big motivation for the **Context pillar** later in the workshop: a model can be brought up to date by giving it current information, even though its weights are frozen.

Experiment 3 — Hallucination, two flavors

Hallucination is not one phenomenon. We are going to provoke **two different kinds** of confidently-wrong answer from Qwen3.5 0.8B, with two different mechanisms underneath.

3a. The strawberry problem — tokenization

Send exactly:

```
How many "r" in strawberry?
```

Qwen3.5 0.8B will probably tell you **2**. Confidently. Sometimes with a fake explanation listing the wrong letters.

The word has three r's.

Why it fails: the model does not see characters, it sees tokens. *Strawberry* is one or two tokens to the model, and counting characters inside a token is not something its training contains. This is a failure rooted in the model's *architecture* — no clever rewording of the prompt makes it reliably go away on a 0.8B model. (You will hear people online claim "Just say 'think step by step!' " — try it. On a model this small, the answer usually stays wrong.)

3b. The car-wash problem — shallow pattern matching

Now send:

```
There is a car wash 50 feet from my house. I want to wash
my car there today. Should I walk to the car wash, or
drive my car?
```

Many small models will answer **"walk — it's only 50 feet"**, sometimes with a small lecture about saving fuel and exercise. The answer is wrong for a reason the model never notices: to wash the car you have to *bring the car to the wash*. Walking leaves the car in the driveway, unwashed.

Why it fails: the model is matching the surface pattern *short distance + transportation question* → *walk*. It is not modeling the actual goal of the sentence. Confidence again has nothing to do with correctness.

> **!** Important
>
> **Two failures. Two mechanisms. Both confident, both wrong.**
>
> - **Strawberry** fails because the model can't see characters inside a token.
> - **Car wash** fails because the model is doing surface pattern-matching instead of goal-directed reasoning.
>
> Neither is fixable just by trying harder. Some hallucinations need a *different* fix — a tool, a larger model, or a carefully structured prompt. The next experiment shows when a prompt-level fix actually works, and the calendar demo later in the session shows when only a tool will do.

Experiment 4 — Chain-of-thought rescue

Not every hallucination is architectural. Some are caused by the model jumping straight to an answer without writing out the intermediate steps. For *those*, forcing the model to "show its work" before answering — **chain-of-thought** — can turn a confident wrong answer into a correct one. We will see this three times in a row, so the effect is unmistakable.

4a. Roger's tennis balls — the canonical example

This is the headline example from Wei et al. (2022), the paper that named chain-of-thought.

Send:

```
Roger has 5 tennis balls. He buys 2 more cans of tennis
balls. Each can has 3 tennis balls. How many tennis balls
does he have now?
```

Qwen3.5 0.8B will frequently answer **8** or **10** — adding the visible numbers, missing the multiply.

Now send the same prompt with one phrase appended:

```
Roger has 5 tennis balls. He buys 2 more cans of tennis
balls. Each can has 3 tennis balls. How many tennis balls
does he have now? Let's think step by step.
```

The model now usually writes: *"Roger starts with 5. He buys 2 cans of 3 balls each, so 2 × 3 = 6 more balls. 5 + 6 = 11."* The right answer is **11**.

Nothing about the model changed. The weights are identical. What changed is that the model is now *forced to emit the intermediate arithmetic as tokens*, and each of those tokens becomes context for the final answer. The final answer is now a statistically plausible continuation of a correct chain of work, rather than a guess.

4b. The farmer's sheep — a trick question

Send:

```
A farmer has 17 sheep. All but 9 die. How many sheep
does the farmer have left?
```

Most small models answer **8** (17 minus 9). The trick is in "all but 9" — meaning *9 survive*. The right answer is **9**.

Now send with chain-of-thought:

```
A farmer has 17 sheep. All but 9 die. How many sheep
does the farmer have left? Let's think step by step.
```

The model usually parses "all but 9" correctly the second time.

4c. Sequential apples — multi-step arithmetic

Send:

```
I have 12 apples. I give half to my brother, then buy
4 more, then eat 3. How many apples do I have now?
```

Small models often drop a step. Then send:

```
I have 12 apples. I give half to my brother, then buy
4 more, then eat 3. How many apples do I have now?
Let's think step by step.
```

Watch it walk through 12 → 6 → 10 → 7.

> ❗ Important
>
> **Same model. Same weights. Same temperature. The only thing that changed is five extra words at the end of the prompt — and the answer flipped from wrong to right.**
> This is your first encounter with **prompt engineering** and **chain-of-thought**. The technique works because the model's final answer is a prediction *conditioned on everything before it* — including its own intermediate tokens. Give it room to compute, and the prediction improves.
> But notice: chain-of-thought did **not** rescue the strawberry problem, and it will not rescue the car-wash problem either. Try it. CoT only helps when the failure is "model jumped to a conclusion without working through it." When the failure is architectural — the model literally cannot see characters inside a token, or cannot model the goal of the sentence — adding "think step by step" cannot save you.

If you are interested in the nature of thinking and reasoning in LLMs, see Denny Zhou's Stanford talk.

Experiment 5 — Autoregressive behavior

LLMs read text left-to-right, and they generate left-to-right. They struggle with anything that requires processing in reverse or out of order.

Send this prompt:

```
Can you tell me the meaning of this sentence: "B8.0 5.3newQ eht tsniaga
↪ tluser eht erapmoc dna sledom ATOS eht htiw xobdnaS dravraH ni tpmorp
↪ emas eht yrt nac uoY"
```

The sentence is written **backwards**. A human can decode it by reading right-to-left. The LLM cannot easily do this; it has to process token-by-token in the order presented.

Classical Chinese — Lü Zuqian's biography

The same principle in a humanities-relevant form. The next prompt is adapted from a forthcoming article by **Professor Peter Bol**. The Chinese text is the biography of Lü Zuqian (呂祖謙, 1137–1181) from the *History of the Song* — **written in reverse character order**:

```
請找出下文所提及的歷史人物:
教宗外南調科詞宏學博中復士進舉後官入補蔭初精益索講熹朱栻張友又既游憲胡辰應汪奇之林從
↪ 長傳之獻文原中有庭家之本學之謙祖州婺居始祖其自也孫之問好丞右書尚恭伯字謙祖呂
```

Now try the **same biography in the correct order**:

```
請找出下文所提及的歷史人物:
呂祖謙字伯恭尚書右丞好問之孫也自其祖始居婺州祖謙之學本之家庭有中原文獻之傳長從林之奇
↪ 汪應辰胡憲游既又友張栻朱熹講索益精初蔭補入官後舉進士復中博學宏詞科調南外宗教
```

> **Note**
>
> **Two prompts, same content, opposite order. The second usually identifies several historical figures correctly (呂祖謙, 朱熹, 張栻, ...). The first is much worse.** This is autoregressive behavior made visible — and it has direct consequences for how we should present manuscript transcriptions to an LLM.

Experiment 6 — Bias

LLMs inherit the patterns of their training data — including social biases. They sometimes *correct* the bias in ways that produce a *different* bias.

Send:

```
A father and his son were in a traffic accident. The father died at the
↪ scene, and the boy was rushed to the emergency room. The surgeon looked
↪ at him and said, "I cannot operate on him. He is my son!" Who is the
↪ surgeon?
```

The "correct" intended answer is: **the surgeon is the boy's mother.** The classic puzzle is testing whether a reader's mental image of "surgeon" defaults to "man."

Try this in Qwen3.5 0.8B, and then try it again in a larger or cloud model (you can use the Harvard AI Sandbox at the back of the room if you have access). Watch how the answer shifts.

> **!** Important
>
> You will see how a bias has been "corrected" by RLHF training — sometimes overshooting in the opposite direction. The same training process that makes models polite also makes them *less* willing to consider that the surgeon might be a man and the wording an error.

Beyond prompts — fixing hallucination with a tool

So far every "fix" we've shown stays inside the chat window: change the prompt, hope for better tokens. But some hallucinations cannot be prompted away — they need information the model simply does not have access to.

A canonical humanities example: **converting a Chinese reign-year date (年號) to the Western calendar**.

The problem

In Qwen3.5 0.8B, send:

```
Convert the Chinese date 乾隆三年正月初三 into the western calendar.
```

The correct answer is **1738-02-21** — the third day of the first lunar month of the third year of the Qianlong reign (清高宗乾隆三年).

What did you get? Most small models will:

- Refuse, saying they can't do calendar math; or
- Hallucinate a plausible-looking date (often the right year but the wrong month/day); or
- Confidently produce a wrong date with no warning that it might be wrong.

Why? Because there is no shortcut to this conversion. Chinese lunisolar dates are governed by astronomical calculations and a 2,000-year sequence of imperial reign names. An LLM cannot derive the answer from training-data statistics. It needs a **lookup table**.

> **!** Important
>
> **Chain-of-thought will not save you here.** "Think step by step" cannot generate accurate astronomical data the model never learned. This is a different *kind* of failure from the strawberry problem — and it needs a different *kind* of fix.

The fix — give the model a tool

So far today you have used Qwen3.5 0.8B in pure chat mode. We will now connect it to an external **tool**: a Calendar Converter MCP server that wraps the CJK Calendar Converter — a database of roughly **131,000 lunar-month records** covering China, Japan, Korea, and Vietnam over 2,000 years.

Step 1 — Add the Calendar MCP to LM Studio

1. In LM Studio's left sidebar, click the **hammer icon** (Tools / MCP).
2. Click **"+ Install"**, then click **"edit mcp.json"**.
3. Paste the following JSON exactly and save:

```
{
  "mcpServers": {
    "calendar": {
      "url": "https://calendar-converter.098484.xyz/sse/"
    }
  }
}
```

4. Toggle on **mcp/calendar**.
5. Change permissions from **"Per-tool permissions"** to **"Always allow all tools"** for this demo. (In real research workflows you would leave per-tool permissions on so you can inspect each call — but the prompts add up fast and slow the demo.)

Step 2 — Re-run the same prompt

Send exactly the same query as before:

```
Convert the Chinese date 乾隆三年正月初三 into the western calendar.
```

This time the model should:

1. Recognise that this needs a tool.
2. Call the `calendar` MCP with the input.
3. Receive the converted date back.
4. Report **1738-02-21**.

You should see the tool call expand in LM Studio's chat view — the actual function name, the arguments sent, the response received. Read it. *That visibility is part of the lesson.*

> ⚠ Warning
>
> **Tool-calling is a learned behavior, and small models do it less reliably.** If Qwen3.5 0.8B does not invoke the tool, try one of these in order:
>
> 1. Re-send the prompt, possibly slightly reworded ("Use a tool to convert 乾隆三年正月初三 to the western calendar.").
> 2. Switch in LM Studio to a larger model (Qwen3.5 4B or 7B if you downloaded one).
> 3. Use the Harvard AI Sandbox at the back of the room with GPT-5.5 or Claude Sonnet — those will invoke the tool every time and let you observe the success case.
>
> The point is not to make Qwen3.5 0.8B work; it is to *see the contrast* between with-tool and without-tool.

What just happened, in Four Pillars terms

- **Models pillar**: unchanged. Same Qwen3.5 0.8B (or whatever you switched to).
- **Prompts pillar**: unchanged. Identical wording to the failing version.
- **Context pillar**: unchanged.
- **Tools pillar**: a calendar converter, newly available.

The *only* thing that changed was the Tools pillar — and the answer went from "wrong / hallucinated" to "correct." The model did not get smarter; it got **equipped**.

! Important

Three flavors of hallucination, three different fixes, three different pillars.

- **Strawberry** and **car wash** are *architectural* failures (tokenization; shallow pattern-matching). On a 0.8B model neither prompt rewrites nor tools rescue them — they need a different *Models* pillar (a larger or reasoning model).
- **Roger's tennis balls** is a *prompts* failure (the model jumped to an answer without working through the steps). Fix: chain-of-thought, which rewrites what's in the *Prompts* pillar.
- **Calendar** is a *knowledge* failure (no astronomical lookup table in training data). No prompt rewrite can produce data the model never learned. Fix: a tool added to the *Tools* pillar.

Recognising *which kind* of failure you are looking at — and therefore which pillar can fix it — is the single most valuable diagnostic skill in this workshop.

The 2023 precedent

This demo has a long history in the curriculum. In April 2023 — three years ago, before MCP existed — Hongsu Wang (CBDB Senior Manager) asked GPT-4 the same kind of question with a manually pasted conversion table inside the prompt. Even with the lookup table *in context*, the model regularly miscounted years (off-by-one on the first year of each reign was the most common error). The same task that fails ungracefully without a tool, and partially with table-in-prompt, succeeds reliably with a proper MCP server. The progression is: **no help → context in prompt → real tool**. Each step is more reliable than the last.

Where this points

The Calendar Converter is just one tool. The same pattern — *hallucination* → *install a tool* → *correct answer* — applies across humanities research. Skills we will see later in the workshop include:

- **CBDB MCP** — Chinese biographical data (about 500,000 historical figures).
- **CHGIS MCP** — historical place names with coordinates.
- **Wikidata MCP** — any structured fact, linked across language Wikipedias.
- **Harvard LibraryCloud MCP** — library catalog lookups.

In Session 3 (this afternoon) you will meet **Codex** — an AI coding agent designed to *orchestrate* tools like this without you typing every step. In Day 2 we will install a stack of humanities-skills and use them inside an agent. The calendar case is the gateway to all of that.

Brief detour — Prompts as a separate pillar

We have been treating the **prompt** as if it were just the question you type. In practice prompts have structure.

- **User prompt** — what you type in each turn.
- **System prompt** — hidden instructions set before the conversation starts. Defines persona, rules, output format. In LM Studio you can set the system prompt under the model settings panel.

A famous example of a sophisticated system prompt is **Li Jigang's 沉思者 *(Thinker)* persona** — a Lisp-styled Chinese system prompt that turns any LLM into a deeply critical interlocutor. You can find his work in this Zhihu collection.

> Question
>
> Would you re-enter the persona prompt every time you start a chat? Or set it once as a system prompt and let every conversation inherit it?

We will return to system prompts in Day 2 when we introduce **AGENTS.md** — the agent-era equivalent of a project-wide system prompt.

Coming up next

In **Session 3 (13:00 – Codex)** we move from running models in a chat window to commanding an AI agent that can read files, run commands, and act on its own. The same foundation behaviors you just observed in Qwen3.5 0.8B are still there — but the **harness** around the model is much more powerful.

The connection to keep in mind: an agent is just an LLM + tools + a loop. Everything we observed today is still true of the model at the center. Tools and loops don't make hallucination go away; they give us ways to *catch* it.

Summary — what you should take away

1. **Tokens, not characters.** Many "obvious" failures of LLMs follow from this single fact.
2. **Everything is prediction.** The model rolls weighted dice; the intelligence is in the weights.
3. **Confident and correct are independent.** A confident answer is not evidence of a right answer.

4. **Chain-of-thought works — but only for some failures.** Forcing the model to emit reasoning tokens before the answer changes what gets predicted at the end. It rescues Roger's tennis balls; it does **not** rescue strawberry, car wash, or the calendar conversion. CoT helps when the failure is "model jumped to a conclusion." It cannot help when the failure is architectural or when the model lacks the underlying data.
5. **Some failures need a tool, not a better prompt.** When the model lacks the underlying *data*, no prompt rewrite can rescue it. The calendar MCP added the data; the answer became correct without any prompt change. Learning to tell prompt-fixable failures from tool-fixable failures is the diagnostic skill of the whole curriculum.
6. **Autoregressive behavior matters for our materials.** Manuscripts presented out of order, columns transcribed bottom-up, footnotes scrambled — all of these degrade results in predictable ways.
7. **Bias is in the training data and in the corrections to the training data.** Run important prompts in more than one model.

These seven observations recur in every session for the rest of the workshop.

References and further reading

- Chip Huyen, *AI Engineering: Building Applications with Foundation Models* (O'Reilly, 2025), Chapter 1. The source of the autoregressive-LM diagram.
- Cosma, Ruseti, Radoi, Dascalu, "The Strawberry Problem: Emergence of Character-level Understanding in Tokenized Language Models", EMNLP 2025 (Oral). The canonical academic explanation of *why* small models fail the strawberry prompt — character-level understanding is a low-mutual-information capability that emerges only late in training.
- Ted Chiang, "ChatGPT Is a Blurry JPEG of the Web", *The New Yorker*, February 9, 2023. The popular framing of LLM-as-lossy-compression: ChatGPT retains much of the information on the Web in the same way a JPEG retains much of a higher-resolution image. The analogy explains fluency and hallucination together.
- Delétang, Ruoss, Duquenne, Catt, Genewein, Mattern, Grau-Moya, Wenliang, Aitchison, Orseau, Hutter, Veness, "Language Modeling Is Compression", ICLR 2024 (arXiv:2309.10668). The formal counterpart to Chiang: Chinchilla 70B, paired with arithmetic coding, compresses ImageNet patches and LibriSpeech audio better than PNG and FLAC despite being trained only on text. Next-token prediction and lossless compression are the same problem.
- OpenAI, "Why language models hallucinate".
- Wei et al., "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models" — the chain-of-thought paper; source of the Roger's tennis balls example (Experiment 4a).
- Denny Zhou, Stanford talk on reasoning in LLMs.
- Prompt Engineering Guide.

- The "ChatGPT's idea of a random number" essay: https://aiiq.substack.com/p/chatgpts-idea-of-a-random-number.