Chapter 1 - How AI Uses Tools: From Toolformer to MCP

Who is this course for

Developers who want to use MCP to build and develop applications.

Prerequisite

1. **Programming**

- **Python (Preferred)**

2. **AI Concepts**

- What an **LLM** is (basic structure, context window, limitations like hallucinations).

3. **Systems & Protocol Basics(If you don't know it's okay)**

- **Client–server model** (requests, responses, errors).
- **APIs**: What an API is and how it works within the client–server model.

🛠️ Step 1: What is a Tool?

- **Definition (AI perspective)**

  A *tool* is any external function, service, or resource that an AI model can call to extend its capabilities beyond text generation.

  Examples:

  - A calculator function for precise maths
  - A weather API to fetch live weather
  - A database connector for retrieving structured data

- **Why tools are needed**

  Imagine this: an AI like ChatGPT is kind of like a very smart person... but with one big limitation.

  It can only *read and write text*.

  That's powerful, but also limited. Let me explain.

1. **AI doesn't always know what's happening right now.**

   Once an AI finishes training, it only knows the information up to that point.

   So, if the AI was trained with data up to 2024, it has no clue what happened in 2025.

   It's like a wise person stuck on a digital island — it is very intelligent, but cut off from today's news, today's weather.

2. **Tools make AI smarter, just like they do for humans.**

   Think about doing maths homework without a calculator. You could still solve problems, but you'd be slower, make more mistakes, and get tired quickly.

   The same goes for AI — it can think, but it's not perfect at things like calculations. With the right tools, it becomes far more powerful.

3. **AI can't do everything by itself.**

   At its core, a language model is just a "text-in, text-out" machine.

   That means it can't directly analyse an image, listen to sound, or check the real world on its own.

   But when you give it tools — like an image or sound recognizer — it can suddenly go beyond those limits.

- **Example(ChatGPT)**
  - Now, explanations are fine... but it's always better to *see* it in action. So let's test this with ChatGPT.

    👉 I'll ask it a really big maths question:

    "What is 123,423,215 × 1,232,323,424?"

    Sounds simple, right? Just a multiplication problem.

    But here's the catch — without tools, solving this isn't easy for anyone.

    Even I would need a calculator, or at least a paper and pen, to handle numbers this big.

    So, let's see if our "intelligent AI" can do this giant multiplication *without any tools*.

    - Example(Use ChatGPT)

I'll give ChatGPT this instruction:

"Don't use any tools like a calculator. Don't think too long."

And then I'll ask:

"What is 123,423,215 × 1,232,323,424?"

Now, without tools, ChatGPT has to calculate everything by itself.

> 💡 <Instruction> You should not call the tools like **calculator.** You should not think long. </Instruction>
>
> <Question> What is 123,423,215 × 1,232,323,424? </Question>
>
> <Output> answer: int </Output>

Alright, GPT has given us a result. And honestly, it *looks* quite reasonable. But let's make sure whether the result is correct or not.

<Open the calculator and do the multiplication>

See the difference?

AI is smart, but just like humans, it needs the right tools to reach its full potential.

That's why tools matter.

📜 Step 2: Classical Tool Use – Toolformer (2023)

Alright, now we understand *why* AI needs tools.

But here's the next question: **How can an AI actually use a tool?** And, just as importantly, *when* should it use one?

Remember — an LLM is just a text-in, text-out machine. By itself, it has no real ability to use a calculator, run code, or call an external function.

So how do we fix that?

This is where **Toolformer** comes in.

The idea is surprisingly simple:

Step 1 — Train the AI model to learn *when* it should call a function and *how* to call it.

So instead of giving you a direct answer, it might generate something like:

> [→ API_CALL: calculator(25 * 12)]

Step 2 — When the AI produces that special “API call” text, we pause its response.

Step 3 — The system actually runs the function, for example by calling the calculator with those numbers.

Step 4 — The result of that function call is inserted back into the conversation.

Step 5 — The AI continues its response, now using the correct result.

And just like that, the AI is no longer limited to text alone — it can use external tools to get accurate answers, just like we humans use calculators, search engines.

- **Example(ChatGPT)**

  Let’s make it more concrete with an example.

  To train an AI model on *when* and *how* to call a function, I start by providing it with a prompt.

  The LLM can then understand the context and generate output in the correct format automatically.

  ```
  # source: https://github.com/lucidrains/toolformer-pytorch/blob/main/toolformer_pytorch/prompts.py

  system_prompt = """
  Your task is to add calls to a Calculator API to a piece of text.
  The calls should help you get information required to complete the text.
  You can call the API by writing "[Calculator(expression)]" where "expression" is the expression to be computed.
  Here are some examples of API calls:
  Input: The number in the next term is 18 + 12 × 3 = 54.
  Output: The number in the next term is 18 + 12 × 3 = [Calculator(18 + 12 * 3)] 54.

  ...(more examples)...

  Input: What is 123,423,215 × 1,232,323,424?
  ```

```
Output:
"""
```

If you test with this prompt, the LLM will produce something like this:

`What is 123,423,215 × 1,232,323,424 = [Calculator(123423215 * 1232323424)]?`

Using this output, AI researchers can train the LLM with the input-output pair:

```
Input: What is 123,423,215 × 1,232,323,424?
Output: What is 123,423,215 × 1,232,323,424 = [Calculator(123423215
* 1232323424)]?
```

This way, the LLM learns the general pattern of **when** and **how** to call the tool.

Once trained, the LLM can generate the API or function call with the correct arguments.

You can then execute it and insert the result back into the prompt.

For example:

```
Input: What is 123,423,215 × 1,232,323,424?
Output: What is 123,423,215 × 1,232,323,424 = [Calculator(123423215
* 1232323424)→ 152097318909888160]?
```

Now, the LLM knows the API's result and can use it to answer the question accurately.

easy, right?

now, test again our big numbers multiplication with this API result.

```
<Instruction>
You should not call the tools like **calculator.**
You should not think long.
</Instruction>

<Question>
What is 123,423,215 × 1,232,323,424 = [Calculator(123423215 * 12323
23424) → 152097318909888160]?
```

```
</Question>

<Output> answer: int </Output>
```

Now you can see that GPT will produce the correct answer.

By giving the model access to a tool like this, we can **enhance the LLM's capabilities** and make it solve problems more accurately.

Before going to next sesion, I want to highlight this:

LLM can generate how to call and when to call the API even though an API is never shown in training time in the context.

Because by training like I mentioned, LLM can generalize the pattern of API calling mechanism.

At inference time, the LLM can:

1. Recognise that the question requires an external tool.
2. Generate a correctly formatted function/API call **following the pattern it learned**.
3. Use the result from the tool to produce the final answer.

So, if we give the prompt like this:

```
<System>
You have the ability to call the tool.
I will give you the tool description:
{
  "tool_name": "get_current_time",
  "Description": "Get the current time in a specific timezone.",
  "Arguments": "timezone (required): IANA timezone name (e.g., Americ
a/New_York, Europe/London). If not provided, defaults to America/New
_York."
}
</System>
```

LLM understand it's context and will generate how to call and when to call following by your question.

Reference: https://chatgpt.com/share/68b136fc-fbf8-8010-bbdc-b223e442e7f5

- **How can the tool help solve the problems we've discussed?**

  So we can solve all the problems we mentioned earlier.

  1. **AI can know what's happening right now using tool at proper time.**

     > How's the weather today?[Weather() → Rainny]

  2. **Tools make AI smarter**

     > (...After solving some very difficult mathmatical problem...),
     > the result is this: 37/20[calculator(37/20) → 1.85]

  3. **AI can't do everything by itself.**

     > Summarise this meeting recording[sound_extractor(file) → "Today, we have to discuss about ..."]

But unfortunately the problem sill remains...

- **Limitations of Toolformer-style approaches**

1. We keep making the same tools again and again

Here's the first issue: every time we want our AI model to use a tool, we have to build it from scratch.

For example, imagine our company wants to give our LLM access to a **calculator, a search engine, and a translator**. Great — we build those tools.

But then I start a personal project, and I want my LLM to use a **calculator, an image recogniser, and a translator**.

Guess what? I have to rebuild the calculator and translator tools all over again.

And it's not just me. Every developer ends up building the same tools again and again, each for slightly different projects.

Yes, we can share these tools as libraries, but here's the catch — everyone builds them in their own format, with their own style. So the tools aren't really reusable in a consistent way.

2. **Different AI models, different formats**

   The second issue is that every AI platform has its own way of registering and calling tools.

   Take OpenAI as an example — they define their own method for registering functions and calling APIs. Other platforms do it differently.

   So, even if you have the same tool, you often have to rewrite or reformat it to fit the model you're working with.

```
# 1. Define a list of callable tools for the model
tools = [
    {
        "type": "function",
        "name": "get_horoscope",
        "description": "Get today's horoscope for an astrological sign.",
        "parameters": {
            ....
            }
     }
]

# 2. Prompt the model with tools defined
response = client.responses.create(
    model="gpt-5",
    tools=tools,
    input=input_list,
)

# Save function call outputs for subsequent requests
input_list += response.output

for item in response.output:
    if item.type == "function_call":
        if item.name == "get_horoscope":
            # 3. Execute the function logic for get_horoscope
```

```
        horoscope = get_horoscope(json.loads(item.arguments))

        # 4. Provide function call results to the model
        input_list.append({
            "type": "function_call_output",
            "call_id": item.call_id,
            "output": json.dumps({
              "horoscope": horoscope
            })
        })

        ...
```

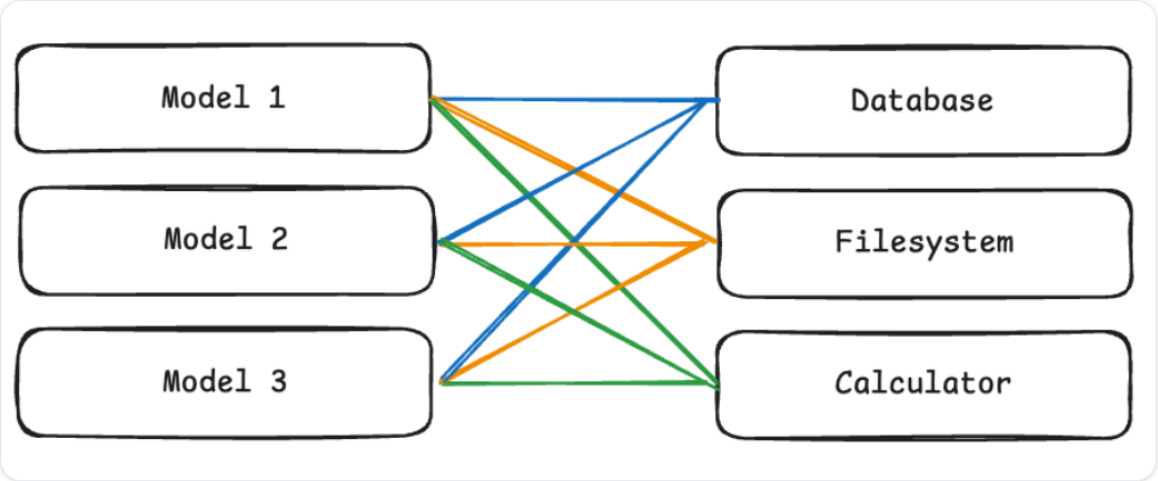
👉 And this is what we call the **M × N problem**: many models, many tools, but no standard way to connect them all together.

References:

1. https://platform.openai.com/docs/guides/function-calling#function-tool-example
2. https://docs.anthropic.com/en/docs/agents-and-tools/tool-use/overview

---

⚡ Step 3: Transition to MCP



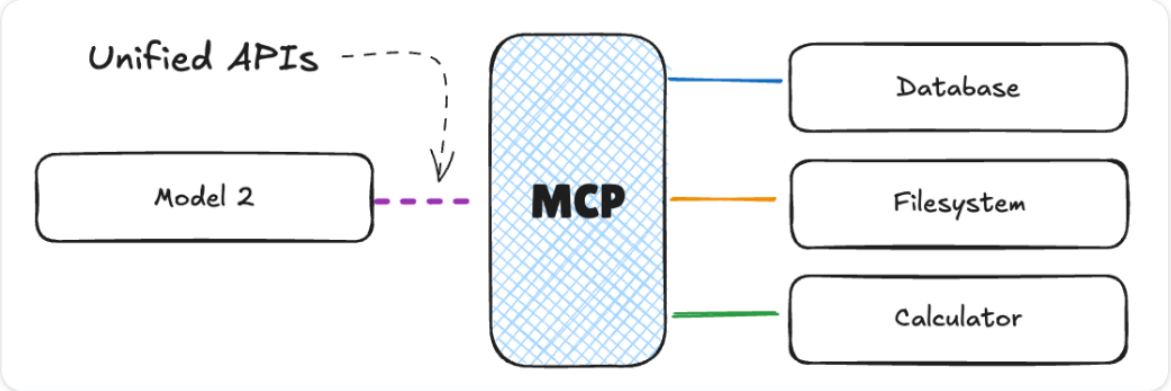
So, how do we solve this problem?

What we really need is a **standard** — a universal way to connect AI models and tools.

Instead of forcing every developer to build custom connections between each AI model and each tool, why not create an **intermediate layer** that handles the communication for us?

And this idea isn't new. We've seen the same thing happen in other fields

for instance Operating systems, Compilers and USB-C port



In the AI world, the equivalent solution is **Model Context Protocol**, or **MCP**.

That's why the creators of MCP describe it as the **USB-C port of AI** — a common standard that lets models and tools connect seamlessly, without endless rewrites.

Image reference:

https://huggingface.co/learn/mcp-course/unit1/key-concepts

Recap

- **Tools**: A *tool* is any external function, service, or resource that an AI model can call to extend its capabilities beyond text generation.
- **Why tools matter**:
  - LLMs can't access real-time info.
  - Tools make AI smarter, just like they do for humans.
  - AI can't do everything by itself.
- **Toolformer (2023)**:

    - Trains LLMs to learn *when* and *how* to call tools.
        - It can solve the problems we mentioned before.
    - Limitation: different platforms require different tool formats → **M × N problem**.
- **MCP (Model Context Protocol)**:
    - A **standard** for tool use.
    - Works like USB-C → one universal connector for AI and tools.