Conversational Interfaces for Research

Transforming Data Interaction with LLMs and MCP

Cristian Dinu

1. Introduction

Hello. I'm Cristian, research software engineer at ARC I build software for a decarbonised future. My main project currently is *Energy Data Observatory and Laboratory* (aka *EDOL*) with the Bartlett's Energy Institute.

My research interests are appliance level energy insights, internet of things, and of course how Artificial Intelligence can improve the way we use technology.

2. Today's topic - Conversational Interfaces

Today, we are talking, just like everyone else apparently, about AI.

People have lots of opinions about it, about its effects, about our readiness as individuals or society for it. They all agree, however, that AI cannot be ignored and that it is an amazing tool, and that is here to stay.

However, we're approaching this from a different angle. It is not about how to use AI to produce code. So no "vibe coding", sorry. There is a "vibe" but not that that you throw spaghetti at a wall to see what sticks.

We are going to **delegate complexity** to the AI, we are going to **delegate user interface**.

3. Software generations

The same person that came with the name "vibe coding", Andrej Karpathy came up with a very interesting way to describe different eras of programming:

- Software 1.0 is written by human programmers. `If X, then Y, else Z`.
- Software 2.0 is created by machine learning engineer where the goal was to optimise neural network weights
- Software 3.0 is programmed by anyone, with human language.

Software 3.0 uses prompts as programs. And large language models (ChatGPT, Claude, Gemini) as interpreters.

Just like in human family, generations can coexist and each one contributes in their own way.

4. LLM

Five, let's now talk about the enablers of this generation. The *Large Language Model*, the LLM.

You've heard about them, haven't you? This competition drives innovation at a nauseating pace.

Some of the developments I'm talking about today weren't available until last week, and the main "protagonist" of our presentation was invented only eight months ago in November 2024.

5. "Humble" beginnings

The grand-grandfather of ChatGPT is the autocomplete system. It is an oversimplification to say that today's state-of-the-art models are just glorified autocompletes, but is true.

Some neural networks are trained on billions of sentences, so they "know" that when we write the word *"climate"*, the next words is probably *action* or *change* or *science*, and not *dog* or *sausage*.

Similarly, a large language model is trained on all human knowledge, good and bad, that exist online, and the result is a model that can predict with much higher precision what comes next.

While autocomplete looks back a few words, a large language model can look back at significantly more data.

6. Brain in a jar

But the model itself, despite being clever, it's quite helpless. Once trained, it stays the same. Updates are costly, it cannot "learn" new things. And by itself, it cannot do much.

This is why the metaphor I found to be appropriate is that of *a brain in a jar*.

An model by itself is just one very big file that contains numbers, probabilities of what comes next after a word or a concept.

8. Chat

What can we do with it, then?

A basic approach is to take the model, feed it some text inputs and see what output it gives us. This is what the original ChatGPT was.

I call it "basic", but in fact it is quite amazing. It took the world by storm, and not without a reason.

The model is fed an initial system prompt, plus the chat history, plus your new prompt, and outputs the most probable sequence of words according to the probability machine that was created as a result of its training and this context.

Most probable doesn't mean true or correct. The model does its best to be correct, but is sometimes wrong. Very wrong.

However, most of the times it is correct, especially for domains that are well-known and documented. It is highly unlikely for a model to hallucinate when asked to describe a symptom of flu.

```
[System Prompt: "You are a helpful weather assistant..."]
     ↓
[User: "Hi, what's the weather today?"] \____ Session
[Assistant: "It's sunny and 24°C."]     /
```

```
     ↓
[User: "What about tomorrow?"]  ← Current prompt
```

9. Tools

We have a “clever brain” that is isolated. cannot learn new things and cannot interact with the world. It cannot “see” outside, it doesn’t have hands to press buttons, physical or virtual. Having the ability to sense and action the world could increase the value quite a lot: instead of just talking it could do things. Not a “brain in the jar” but an “agent”.

One way we can do this is for us to be the AI’s proxy. We copy and paste text into the prompt and we allow the LLM to tell us what to do next.

This has the advantage of keeping the “alien intelligence” trapped in the jar and we humans stay in charge. But it is a slow process, non-scalable, expensive.

Looking at the problem, some clever people thought “Let’s teach the language model to use some tools”. The first tool that I know of is a web search.

LLM is trained to use a tool when it needs data it doesn’t have. For example if I ask about some revent events, instead of hallucinating, it calls a function that returns web pages and their context and injects these results in its current context. We can do this manually, of course, but it’s tedious and if we can automate it it’s better.

Then we can have another tool that looks at source files in your code editor and checks for bugs or writes functions like GitHub copilot does.

The idea is great, but writing custom modules and training the model on how to use them is costly, and it becomes too complex too soon, especially if we want to use these tools together.

10. Model Context Protocol (MCP)

This is why someone did something that now feels obvious,common sense: Create a standard, a protocol to manage the context for the model.

The Model Context Protocol.

It is a a very simple protocol, which makes it especially effective, and everyone loves it. Microsoft announced it will be available at the operating system level for every Windows computer. All vendors have either added support or added it on top of their roadmap.

So now we have a USB-C for AI models to plug shiny new extensions.

11. Concepts

MCP needs a “client”, which is a part of a larger piece of software (e.g. Claude Desktop, VS Code, your Python script)

The client knows how to request a list of available tools and resources from a MCP server. It knows how to look at human language and translate it to a computer function call and format the result.

So the client calls a server – for security reasons until a few weeks ago all servers had to run on your computer. There are already thousands of pre-written MCP servers, we're going to play with some of them, and write one of our own.

The server is usually a "Software 1.0" that advertises its capabilities, implements these capabilities and executes them on demand.

There are three core primitivesthat a server exposes:

- **Tools** – expose executable functionality to clients (but in practice used to retrieve information, too)
- Resources – expose data and content
- Context – expose prompt templates to clients

Let's play with some ready-made MCPs

- Fetch MCP
  - `Go to the Bank of England website and compare interest rate with inflation.`
  - `Go to the https://www.bankofengland.co.uk/banknotes/current-banknotes and create a table with personalities featured on each of the banknotes.`
  - `Visit CNN website and show me happy news in a markdown table`
- Save them to a CSV
  - `Go to FT.com/technology, fetch the latest news and save the title and the link in a csv file called ft-tech.csv§`

Let's create our own MCP

Prerequisites:

- `nodejs` - if you want to run the inspector
- `uv` - brilliant Python project manager

Both are useful for running ready-made

Conclusions

Large mode models and model context protocol are not here to replace all the software. They create new opportunities some of them to improve the efficiency by order of magnitude, some other cases to deliver a level of customisation and features that would be absolutely impossible without them not all the details are clear yet security, deployment, et cetera, but these are details that are going to be sold soon enough

Final Thoughts

I heard Stephen Fry saying that for us, as individuals, the focus should be not on efficiency. Efficiency is not going to be so important in the future, as the machines will catch up and get ahead of us, anyway. The focus should be on how we can be better humans, how to make others life better, how we can enlighten a room when we enter it.

Sure, it's much easier to do this when you're Stephen Fry; things will certainly change.

It's Chilling and Thrilling.

```
uv init intensity_mcp
cd intensity_mcp

uv add "mcp[cli]" requests
```

```
from mcp.server.fastmcp import FastMCP
import datetime
import requests

mcp = FastMCP("CarbonIntensityMCP")

@mcp.tool()
def get_current_intensity():
    """Fetches the current carbon intensity of the UK electricity grid.
    Returns a JSON with the current carbon intensity in gCO2eq/kWh."""
    url = "https://api.carbonintensity.org.uk/intensity"
    try:
        response = requests.get(url)
        response.raise_for_status()
        data = response.json()
        return data['data']

    except Exception as e:
        return f"Failed to retrieve current carbon intensity: {str(e)}"

@mcp.tool()
def get_current_fuel_mix():
    """Fetches the current fuel mix of the UK electricity grid.
    Returns a JSON with the current fuel mix percentages."""
    url = "https://api.carbonintensity.org.uk/generation"
    try:
        response = requests.get(url)
        response.raise_for_status()
        data = response.json()
        return data['data']['generationmix']

    except Exception as e:
        return f"Failed to retrieve current fuel mix: {str(e)}"
```