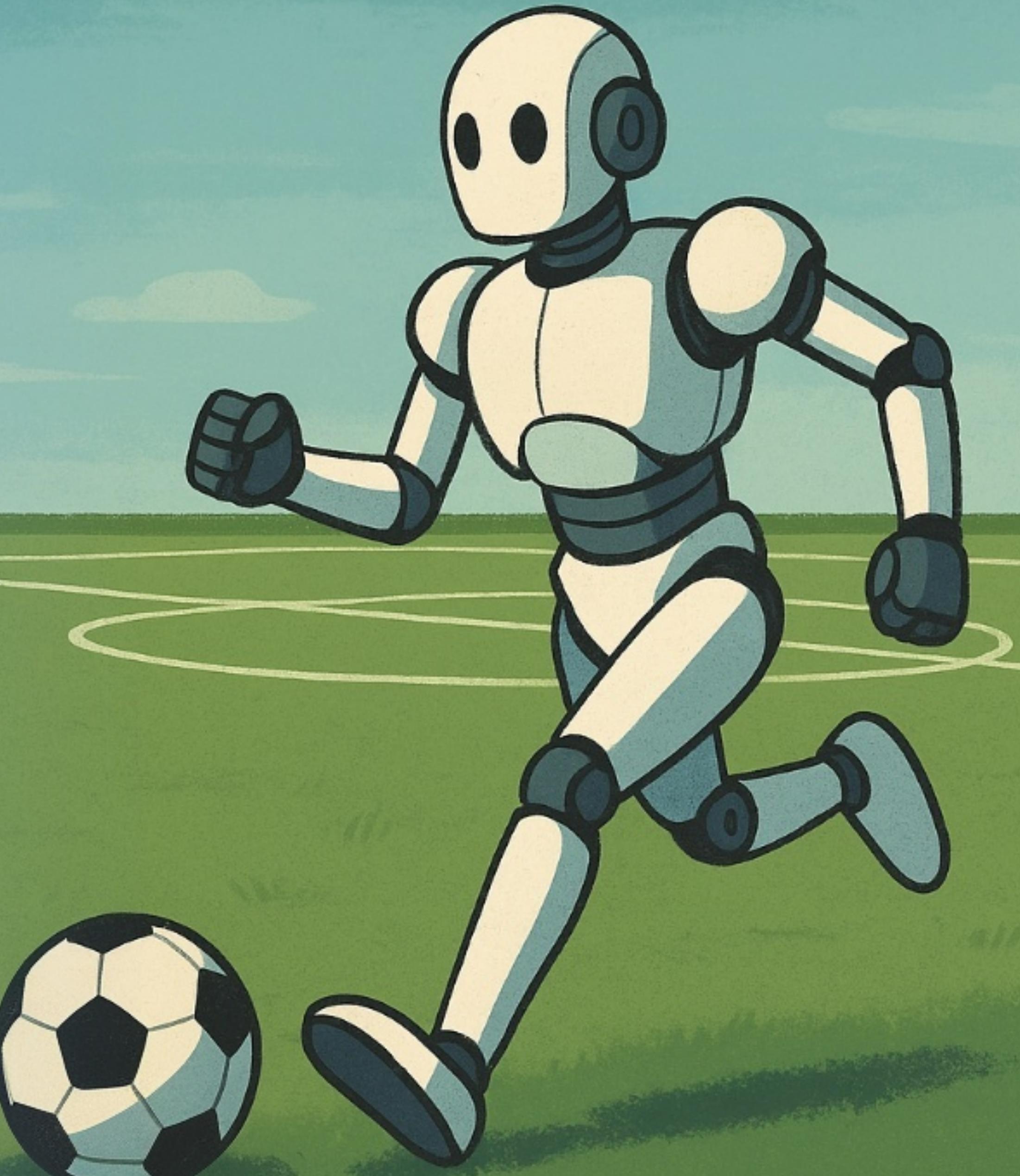


RELIABLE DATA-DRIVEN AI AGENTS

**Using Hugging Face's smolagents
framework to create a Fantasy
Premier League AI Agent**

Geir Freysson, November 4, 2025
geirfreysson.com



About me

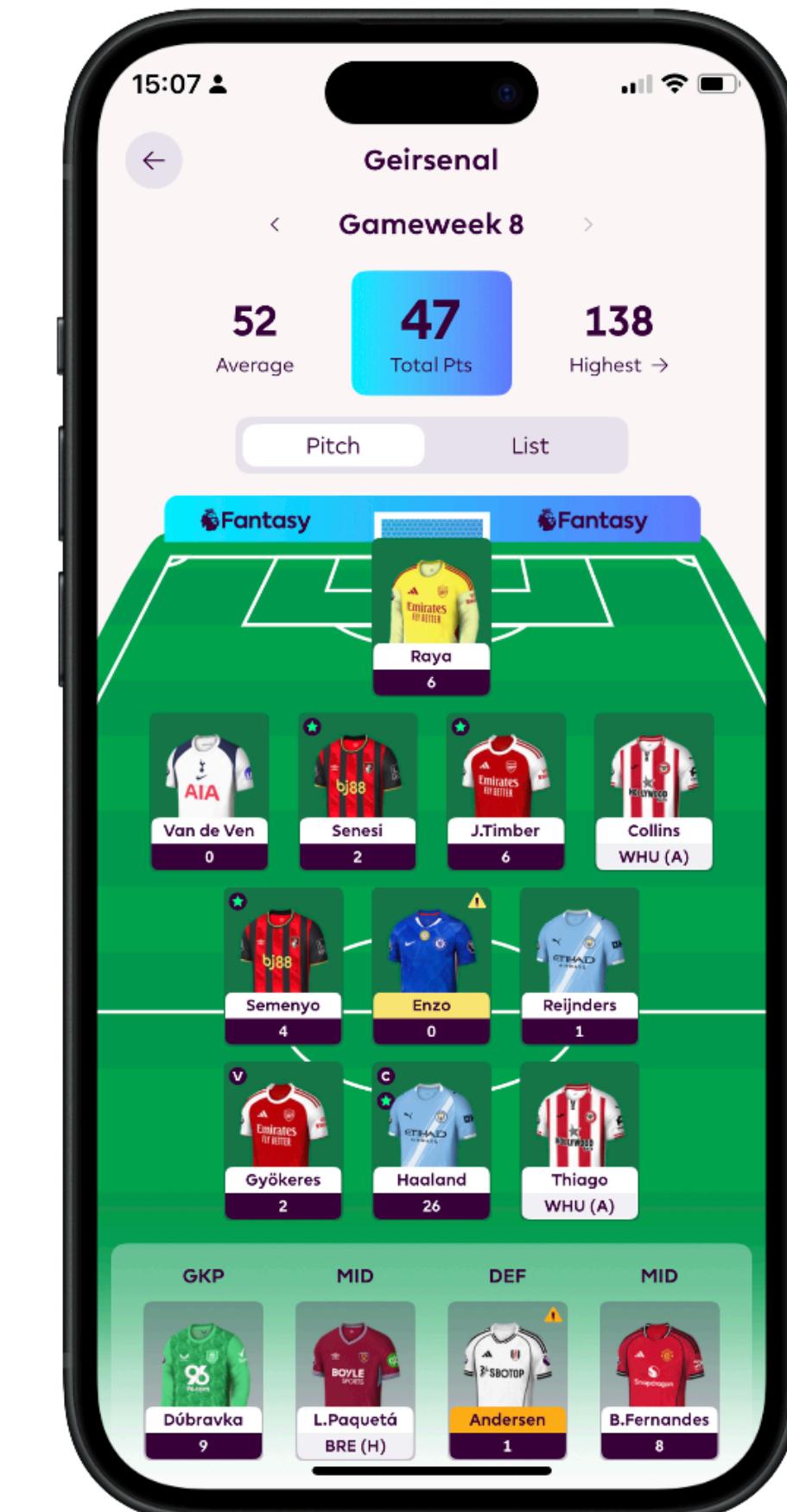
- Name: Geir Freysson
- From Iceland 
- Founder/CEO of Datasmoothie, bringing Python to market research - acquired by Ipsos
- Co-author of Quantipy, open source software for market research
- Addicted to FPL



What is the Fantasy Premier League?

Don't worry, this is not a talk about FPL

- In FPL you pick a team of football players and score points from their real matches.
- Use data to make decisions, like previous performance, easy or hard upcoming matches etc.
- Data is updated regularly, i.e. after each match



Problem, part I

access to data

- Dear ChatGPT: “What teams have the easiest fixtures in the next two games”
- Man City and S**rs are hardly both playing Everton next weekend
- Data fetched from ESPN’s website was interpreted incorrectly

ChatGPT 5 ▾

Share ...

what five premier league teams have the easiest fixtures in the next 2 games and what are these fixtures?

Here are **five Premier League clubs** that currently appear to have relatively easier upcoming fixtures (next two games) based on available schedule & difficulty indicators. Note: the fixture difficulty ranking (FDR) tool provides broad “easier vs harder” guidance. Premier Fantasy ... +2

Club	Next 2 Fixtures	Why these look easier
Manchester City	vs Everton (H) → vs AFC Bournemouth (H)	Everton are away to a top side (City) and struggling; City then host Bournemouth at home. A strong side at home plus weaker opponents suggest a favourable two-game window.
Chelsea	vs Sunderland (H) → vs Wolverhampton Wanderers (H)	Sunderland are newly promoted and likely vulnerable, Wolves haven't been in top-form; Chelsea's home advantage adds to the “easier fixture” appeal.
Arsenal	vs Crystal Palace (H) → vs Burnley (H)	Arsenal have two home games in a row, including one against Burnley who are newly promoted and among the weaker sides; home advantage matters.
Aston Villa	vs AFC Bournemouth (H) → vs Liverpool (A)	While the second game is tough (away at Liverpool), the first is a home game against Bournemouth (more manageable). For a two-game stretch this may still rank as relatively favourable compared to others with two very hard matches.
Tottenham Hotspur	vs Everton (A) → vs Chelsea (H)	The away game at Everton is less daunting than many top-six clashes; followed by a home game vs Chelsea (still tough, but home). So in relative terms a “better” two-game window for them.

Last weekend

Next weekend

Problem, part II

domain specific calculations

“ given a Python list of dicts with an FPL score and date, write a function that calculates the player’s form.

Problem, part II

Claude and ChatGPT suggest calculating form as average of 5

Claude

Given a python list of objects which record a fpl score on one hand and a date on the other, create a function that calculates the form of a player the list applies to

Architected flexible function to average recent player scores.

I'll create a function that calculates FPL player form based on recent gameweek scores:

```
python
def calculate_player_form(gameweeks, num_games=5):
    """
    Calculate a player's form based on recent gameweek scores.

    Args:
        gameweeks: List of dictionaries with 'score' and 'date' keys
            e.g., [{'score': 8, 'date': '2024-01-15'}, ...]
        num_games: Number of recent games to consider for form (default: 5)

    Returns:
        float: Average points over the specified number of recent games
            Returns 0 if no games available
    """
    if not gameweeks:
        return 0.0

    # Sort by date (most recent first)
    sorted_games = sorted(gameweeks, key=lambda x: x['date'], reverse=True)
```

ChatGPT

Given a python list of objects which record a fpl score on one hand and a date on the other, create a function that calculates the form of a player the list applies to

Here's a clean and flexible way to do that 👍

We'll assume you have a list of objects (or dicts) like this:

```
python
[{"date": "2025-09-14", "score": 6}, {"date": "2025-09-21", "score": 8}, {"date": "2025-09-28", "score": 2}, {"date": "2025-10-05", "score": 9}]
```

In FPL terms, "form" is usually defined as **the average score over the last 5 games** (or fewer if the player has played less).

Here's a function that computes that:

```
python
from datetime import datetime

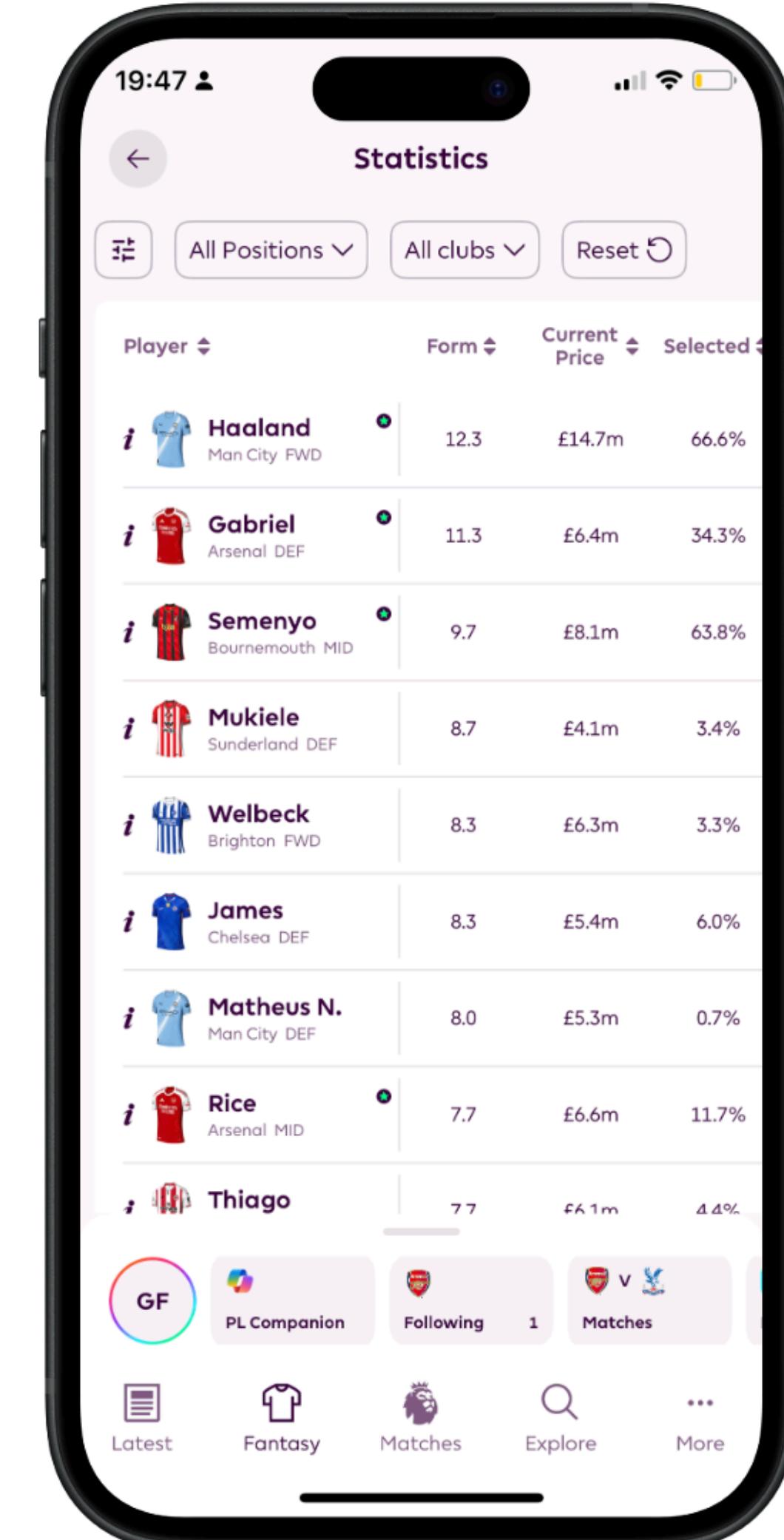
def calculate_form(records, n_games=5):
    """
```

Problem, part II

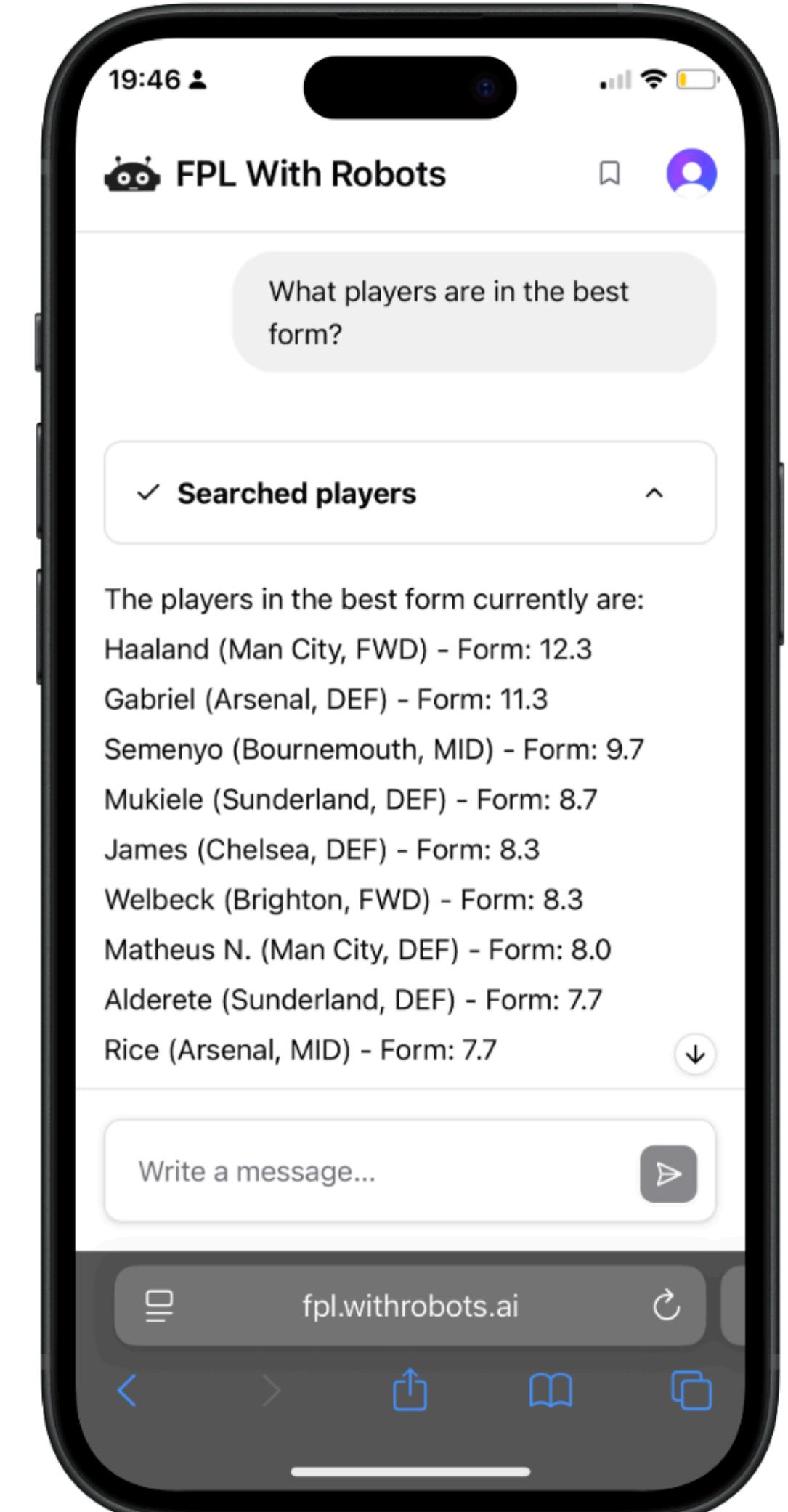
domain specific calculations

- FPL defines form as average result over 30 days, not 5 matches
- Our agent should calculate form with the industry standard approach
- This builds trust and provides consistency

FPL app



Agent



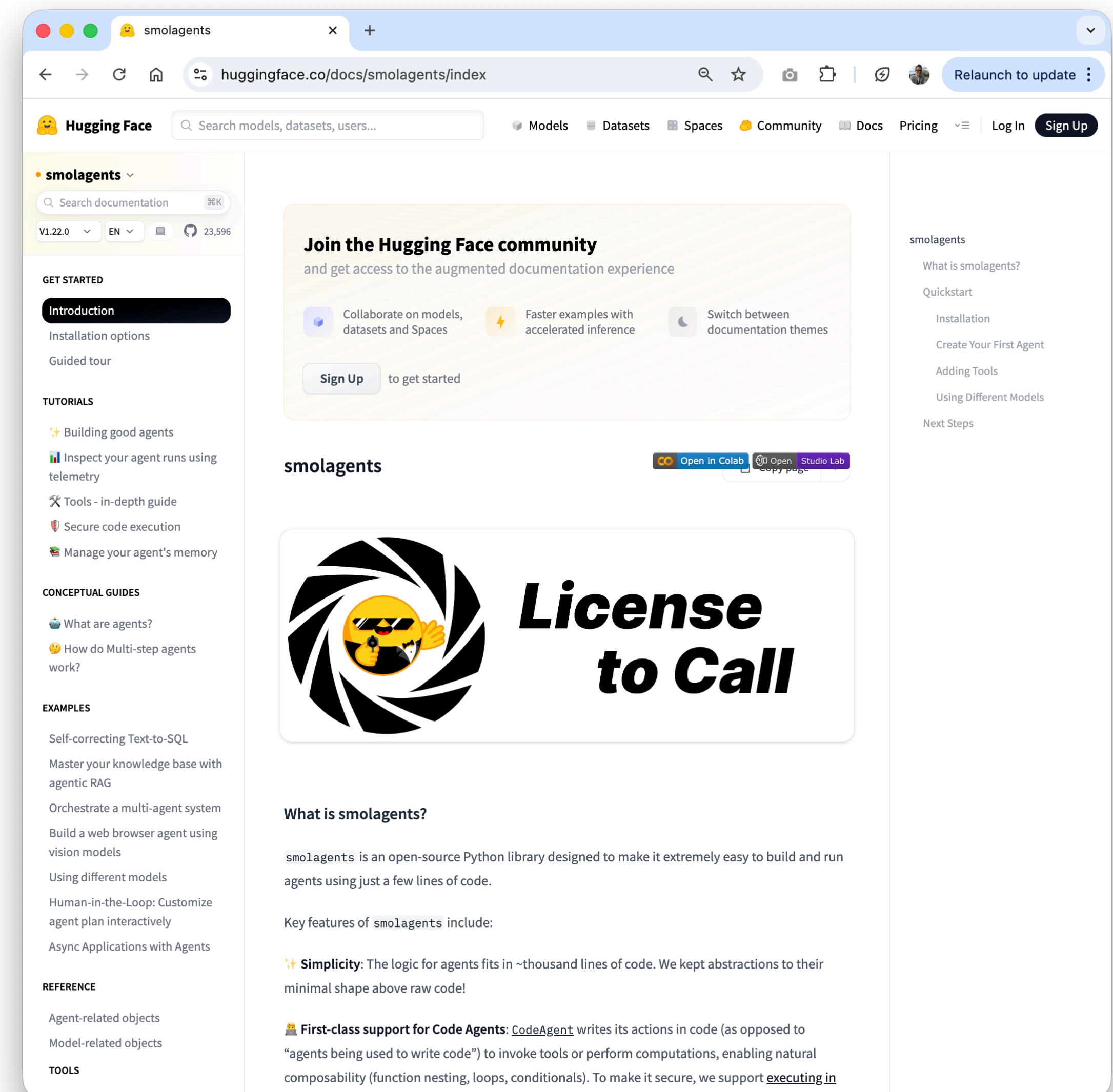
Let's build an agent

We're calling it fpl.withrobots.ai

smolagents

from Hugging Face

- Open source library for building agents.
- Emphasis is on simplicity: Logic for agents fits in ~thousand lines of code.
- Makes it easy to use the force, and read the source. 



smolagents size makes it a good fit for AI

```
$ git clone https://github.com/huggingface/smolagents && cd smolagents
$ claude
> looking at @agents.py, @tools.py, how would you implement streaming?
  Write your plan in streaming.md
$ cp streaming.md ..../fpl-agent/

$ cd ..../fpl-agent
$ claude
> use the @streaming.md guide and make sure the agent returns
  a streaming response to the api
```

Code for a basic agent entering the all-caps era of docstrings

```
from smolagents import ToolCallingAgent, LiteLLMModel, tool

@tool
def sum_numbers(numbers: list[float]) -> float:
    """
    PLEASE Calculate the sum of a list of numbers.

    Args:
        numbers: A list of numbers to sum
    """
    return sum(numbers)

@tool
def divide_sum(total: float, length: int) -> float:
    """
    Divide a sum by a length to get the average.

    Args:
        total: The sum/total to divide
        length: The number to divide by (typically the count of numbers)
    """
    return total / length

model = LiteLLMModel(model_id="gpt-4o-mini")
agent = ToolCallingAgent(tools=[sum_numbers, divide_sum], model=model)
agent.run("What is the average of 10, 20, 30, 40, and 50?")
```

smolagents system prompt

You are an expert assistant who can solve any task using tool calls. You will be given a task to solve as best you can. To do so, you have been given access to some tools.

... bla bla bla

Above example were using notional tools that might not exist for you. You only have access to these tools:

- sum_numbers: PLEASE Calculate the sum of a list of numbers.

Takes inputs: {'numbers': {'type': 'array', 'items': {'type': 'number'}, 'description': 'A list of numbers to sum'}}
Returns an output of type: number

- divide_sum: Divide a sum by a length to get the average.

Takes inputs: {'total': {'type': 'number', 'description': 'The sum/total to divide'},
'length': {'type': 'integer', 'description': 'The number to divide by (typically the count of numbers)'}}
Returns an output of type: number

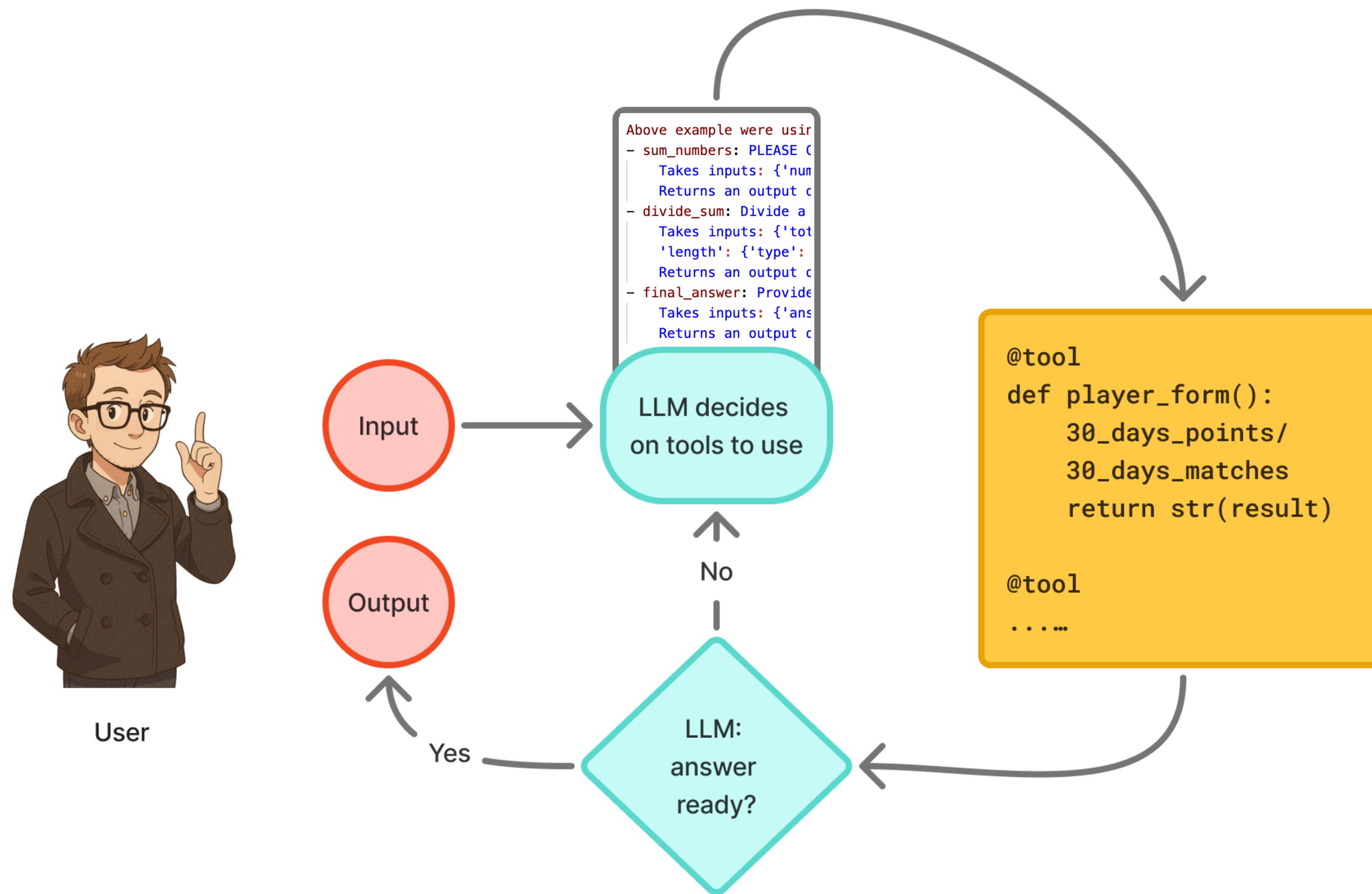
- final_answer: Provides a final answer to the given problem.

Takes inputs: {'answer': {'type': 'any', 'description': 'The final answer to the problem'}}
Returns an output of type: any

Here are the rules you should always follow to solve your task:

1. ALWAYS provide a tool call, else you will fail.
2. Always use the right arguments for the tools. Never use variable names as the action arguments, use the value instead.
3. Call a tool only when needed: do not call the search agent if you do not need information,
try to solve the task yourself. If no tool call is needed, use final_answer tool to return your answer.
4. Never re-do a tool call that you previously did with the exact same parameters.

The “reasoning and acting” agent (ReAct)



Memory refresh: Basic agent code

```
from smolagents import ToolCallingAgent, LiteLLMModel, tool

@tool
def sum_numbers(numbers: list[float]) -> float:
    """
    PLEASE Calculate the sum of a list of numbers.

    Args:
        numbers: A list of numbers to sum
    """
    return sum(numbers)

@tool
def divide_sum(total: float, length: int) -> float:
    """
    Divide a sum by a length to get the average.

    Args:
        total: The sum/total to divide
        length: The number to divide by (typically the count of numbers)
    """
    return total / length

model = LiteLLMModel(model_id="gpt-4o-mini")
agent = ToolCallingAgent(tools=[sum_numbers, divide_sum], model=model)
agent.run("What is the average of 10, 20, 30, 40, and 50?")
```



```
GBPL-M12W606526:pydata-london-2025 geir.freysson$ uv run two_tool_agent.py  
Reading inline script metadata from `two_tool_agent.py`
```

New run

What is the average of 10, 20, 30, 40, and 50?

LiteLLMModel - gpt-4o-mini

Step 1

```
Calling tool: 'sum_numbers' with arguments: {'numbers': [10, 20, 30, 40, 50]}
```

Observations: 150

```
[Step 1: Duration 1.18 seconds | Input tokens: 1,068 | Output tokens: 22]
```

Step 2

```
Calling tool: 'divide_sum' with arguments: {'total': 150, 'length': 5}
```

Observations: 30.0

```
[Step 2: Duration 0.77 seconds | Input tokens: 2,214 | Output tokens: 40]
```

Step 3

```
Calling tool: 'final_answer' with arguments: {'answer': '30.0'}
```

Observations: 30.0

Final answer: 30.0

```
[Step 3: Duration 0.71 seconds | Input tokens: 3,432 | Output tokens: 56]
```

```
GBPL-M12W606526:pydata-london-2025 geir.freysson$ █
```

Agent code in FPL with robots

```
model = LiteLLMModel(  
    #model_id="gpt-4o",  
    model_id="gpt-4.1-mini",  
    api_key=os.getenv("OPENAI_API_KEY")  
)  
  
instructions = """"  
All lists should be displayed as markdown tables for an extra prize.  
  
Markdown tables should have a short, descriptive headline, like  
## Differentials with easy fixtures  
  
(remember to add ## to the table titles for markdown)  
  
Fantasy Premier League terms:  
- GW = Gameweek  
- H = Home  
- A = Away  
- Differential: Player with low ownership  
- xG: Expected goals  
- xA: Expected assists  
"""  
  
agent = ToolCallingAgent(  
    tools=[help,  
          search_players,  
          get_player_fixtures,  
          get_player_details,  
          find_player_replacements,  
          suggest_captain,  
          get_easiest_fixtures,  
          ],  
    model=model,  
    stream_outputs=True, # Enable streaming to get ToolCall/ToolOutput events  
    instructions=instructions  
)
```

CallingAgent(
 help,
 search_players,
 get_player_fixtures,
 get_player_details,
 find_player_replacements,
 suggest_captain,
 get_easiest_fixtures,
,
 model,
 stream_outputs=True

FPL With Robots Your Fantasy Premier League AI Assistant[Releases](#)

How can I help you today?

Who should I
captain this week?

Replacement for
Bowen

High xA def for
less than 5M

Top 5 undervalued
goalies?

Write a message...



Tool calling in other contexts

Travel AI agent tools:

- Search hotel rooms
- Search flights
- Book hotel room
- Book flight
- Compose result email

Coding AI agent tools:

- **find**: ls, glob, grep
- **edit**: read, write, edit/multi-edit
- **execute**: bash
- **research**: websearch, webfetch
- **Q/A**: todoRead, todoWrite

Frameworks to build agents

Backend

- smolagents*
- LangChain/LangGraph
- Google ADK
- LlamalIndex

I've never tried:

- CrewAI
- MS ADK
- Pydantic AI

Frontend (for web)

- assistant-ui*
- Gradio from Hugging Face
- Streamlit
- Chainlit
- ... bring me your suggestions!

* used by FPL with robots

Final thoughts

- smolagents is great for learning to code agents, and also in production
- Deciding between general purpose tools (search_players) vs task specific tools (suggest_captain) is a balancing act
- Deterministic is easier than AI, so only add a tool if absolutely necessary, because:
 - A. More tools means a bigger context for the agent's system prompt, increasing errors
 - B. Deterministic code is easier to test, debug, reproduce errors
- I haven't found a nice out-of-the-box front-end for agents

Thank you!

