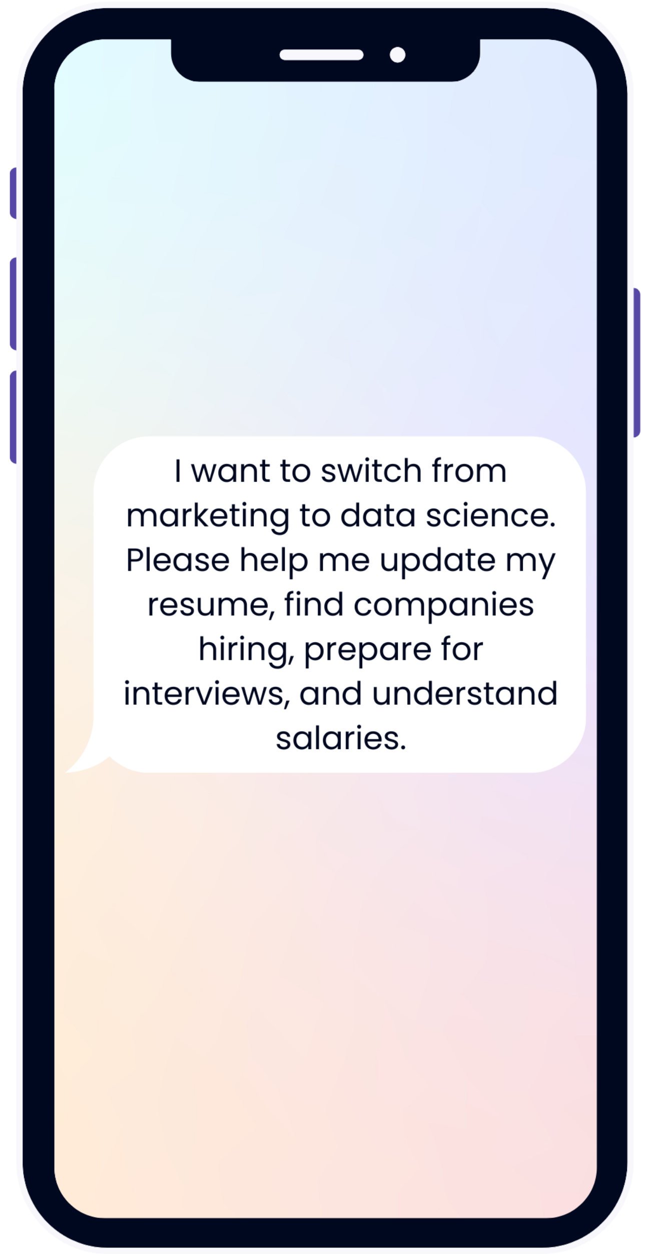
Introduction to Multi-Agent Systems

AI AGENTS WITH HUGGING FACE SMOLAGENTS

Adel Nehme
VP of AI Curriculum, DataCamp

Scenario: Career Advisor Agent


I want to switch from marketing to data science. Please help me update my resume, find companies hiring, prepare for interviews, and understand salaries.

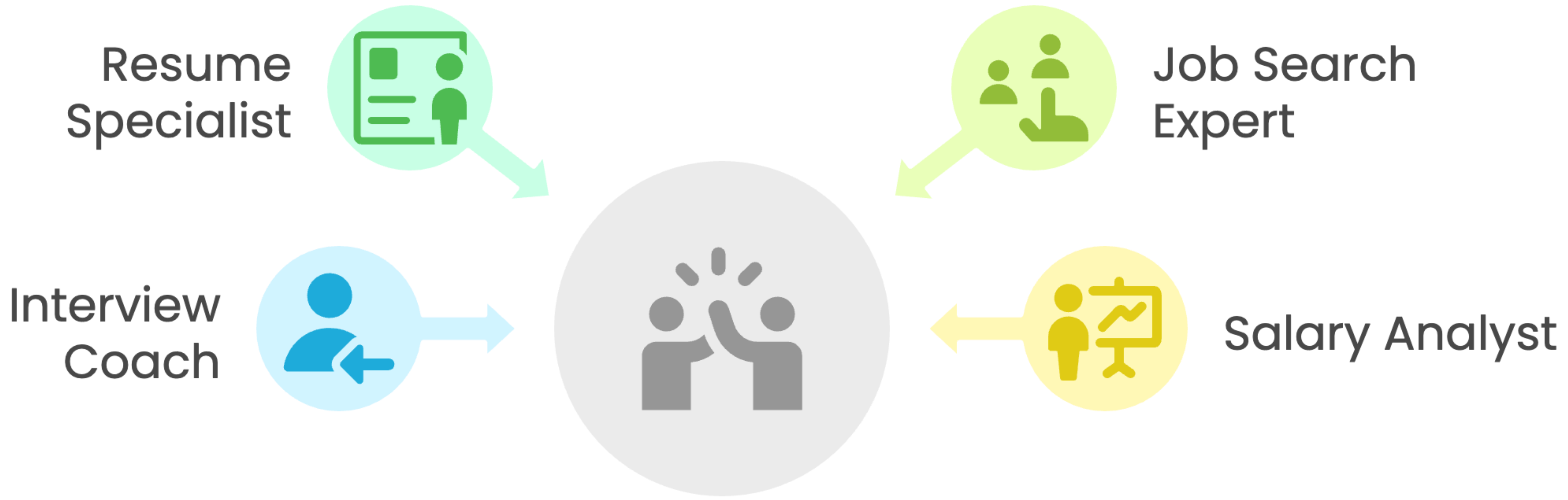
The request involves many tasks:

- Updating resumes
- Job searching
- Interview prep
- Salary research

Each task needs different tools and workflows!

Why Use Multi-Agent Systems?

- Use a team of specialized agents
- Each agent stays focused on one task or domain
- Prevents overload and confusion



A Specialized Resume Agent

```python
resume_agent = CodeAgent(
    tools=[WebSearchTool(), skill_translator, layout_generator],
    model=InferenceClientModel(),
    instructions="You are an expert in everything related to resumes.",
    name="resume_agent",
    description="Expert in resume writing and skill translation for career transitions"
)
```

A Company Research Specialist

```
company_agent = CodeAgent(
    tools=[WebSearchTool(), background_compatibility_checker],
    model=InferenceClientModel(),
    instructions="You are an expert in everything related to company research",
    name="company_agent",
    description="Expert in researching companies, culture, and hiring practices for job seekers"
)
```

- Create additional specialized agents using the same pattern.

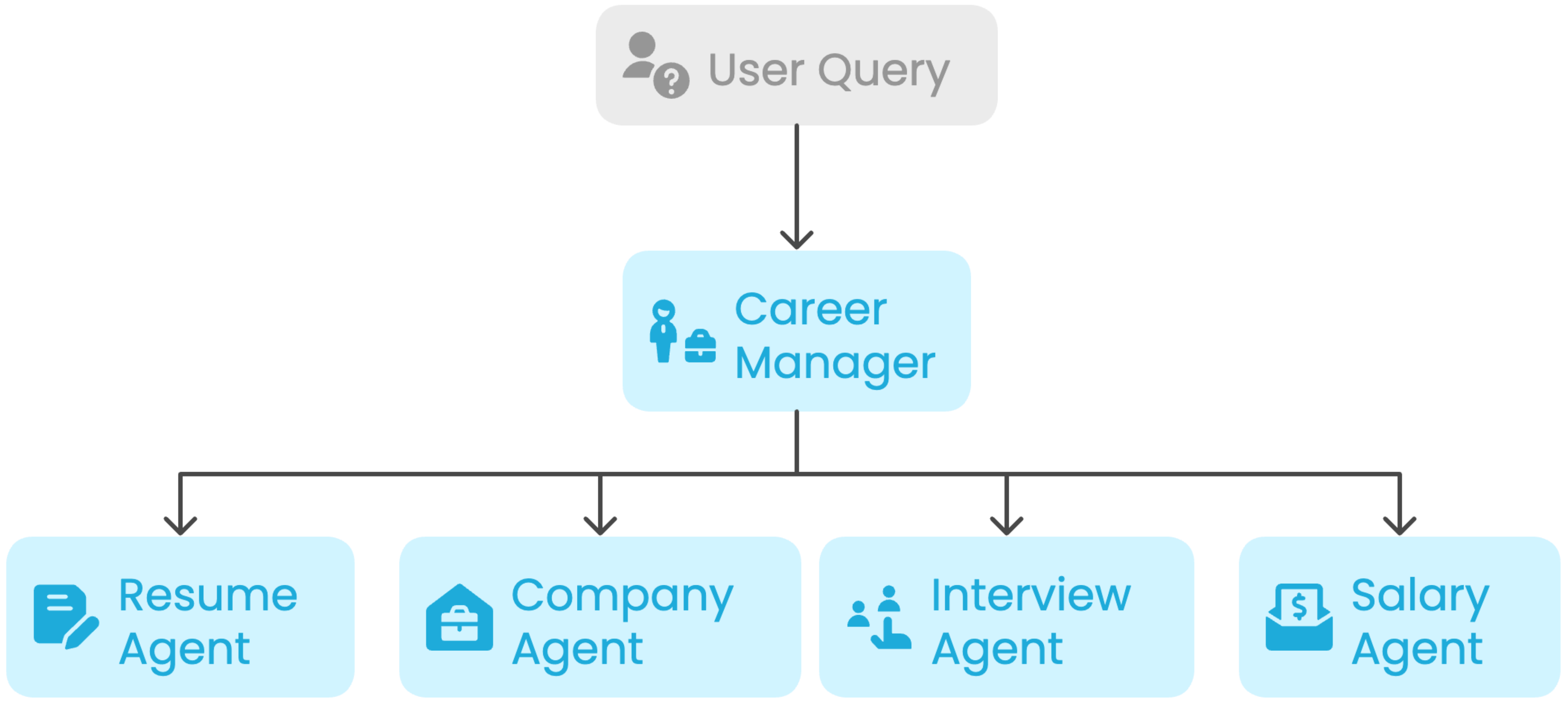
The Manager Agent

```
career_manager = CodeAgent(
    tools=[],
    model=InferenceClientModel(model_id="deepseek-ai/DeepSeek-R1"), # Reasoning model
    instructions="You are an advisory agent to help professionals build stellar careers",
    managed_agents=[resume_agent, company_agent, interview_agent, salary_agent]
)
```

- Delegates tasks to the appropriate specialists based on their descriptions.
- Coordinates four specialist agents using the `managed_agents` parameter.
- Benefit from models with strong reasoning and coordination capabilities.

Multi-Agent Orchestration

```
result = career_manager.run("I want to switch from marketing to data science.
Help me update my resume, find companies hiring, prepare for interviews, and understand salaries.")
```



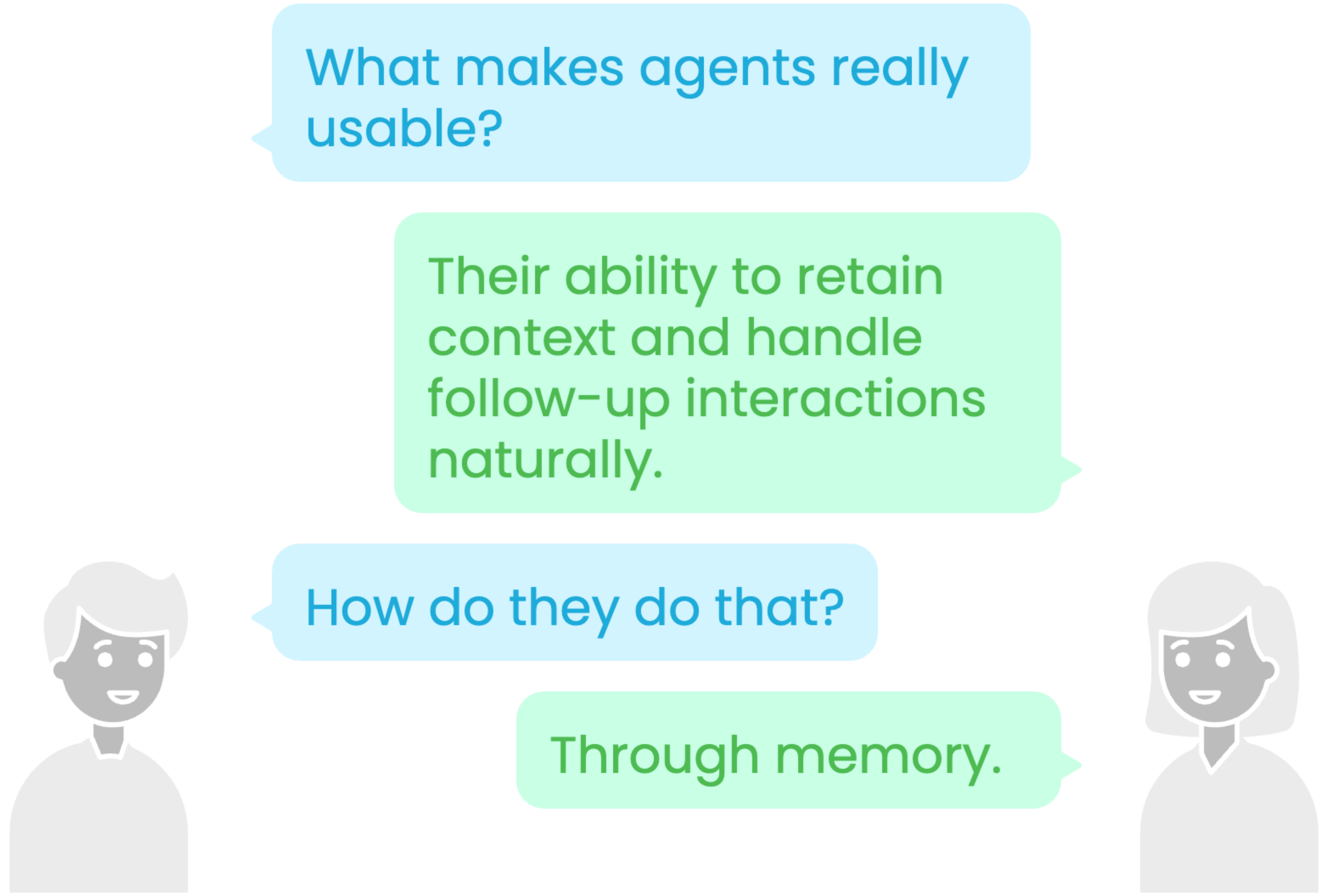
Let's practice!

AI AGENTS WITH HUGGING FACE SMOLAGENTS

Managing Agent Memory

AI AGENTS WITH HUGGING FACE SMOLAGENTS

Adel Nehme
VP of AI Curriculum, DataCamp



What makes agents really usable?

Their ability to retain context and handle follow-up interactions naturally.

How do they do that?

Through memory.

Stateless by Default

Each `.run()` call is a fresh start.

- Example:

```
career_advisor.run("What career skills should I highlight?")
```

```
You should highlight Python, SQL, data visualization,
machine learning fundamentals, and communication skills tailored to business outcomes.
```

```
career_advisor.run("Can you format those skills as bullet points?")
```

```
Sorry, I'm not sure which skills you're referring to. Could you clarify?
```

Retaining Memory Between Interactions

```
career_advisor.run("What career skills should I highlight?")
```

```
You should highlight Python, SQL, data visualization,
machine learning fundamentals, and communication skills tailored to business outcomes.
```

- Pass `reset=False` :

```
career_advisor.run("Can you format those skills as bullet points?", reset=False)
```

```
Sure! Here are the skills as bullet points:
- Python
- SQL
- Data visualization
...
```

Memory Helps You Debug, Too

```
User: What's the expected salary?
```

```
Agent: It's $80,000
```

```
User: Wait, that seems wrong...
```

```
Agent: Sorry, I'm not sure what you mean
```

Inspect what happened in the agent's run:

- Reviewing all code the agent generated
- Tracing its reasoning, actions, and tool usage
- Debugging incorrect answers or broken logic

What Code Did the Agent Run?

The `.return_full_code()` method lets you see all executed code.

```
executed_code = career_advisor.memory.return_full_code()
print(executed_code)
```

```
# ...other steps omitted for brevity

salary = 80000  # <- hardcoded?

# script continues...
```

What Was the Agent Thinking?

```python
conversation_steps = career_advisor.memory.get_succinct_steps()
print(conversation_steps[5])
```

```
{
  "step_number": 5,
  "tool_calls": [
    {"function": {"name": "python_interpreter"}},
    {"function": {"name": "web_search"}}
  ],
  "code_action": "import requests\nskills = requests.get('api.jobsearch.com').json()",
  "observations": "resume_agent found 15 relevant skills for transition",
  "token_usage": {"total_tokens": 334},
  ...
}
```

Save Agent Sessions for Analysis

```python
import json

def save_agent_memory(agent):
    with open("agent_memory.json", "w") as f:
        json.dump(agent.memory.get_succinct_steps(), f, indent=2, default=str)

# Save memory to a file
save_agent_memory(career_advisor)
```

Logs can help with:

- Post-hoc analysis
- Regression testing
- Improving agent behavior over time

Fixing Agent Failures: What to Adjust

- **Memory issues:** Use `reset=False` or reset intentionally
- **Reasoning problems:** Try a stronger model
- **Inconsistent behavior:** Improve system prompt
- **Tool confusion:** Clarify tool docstrings

Let's practice!

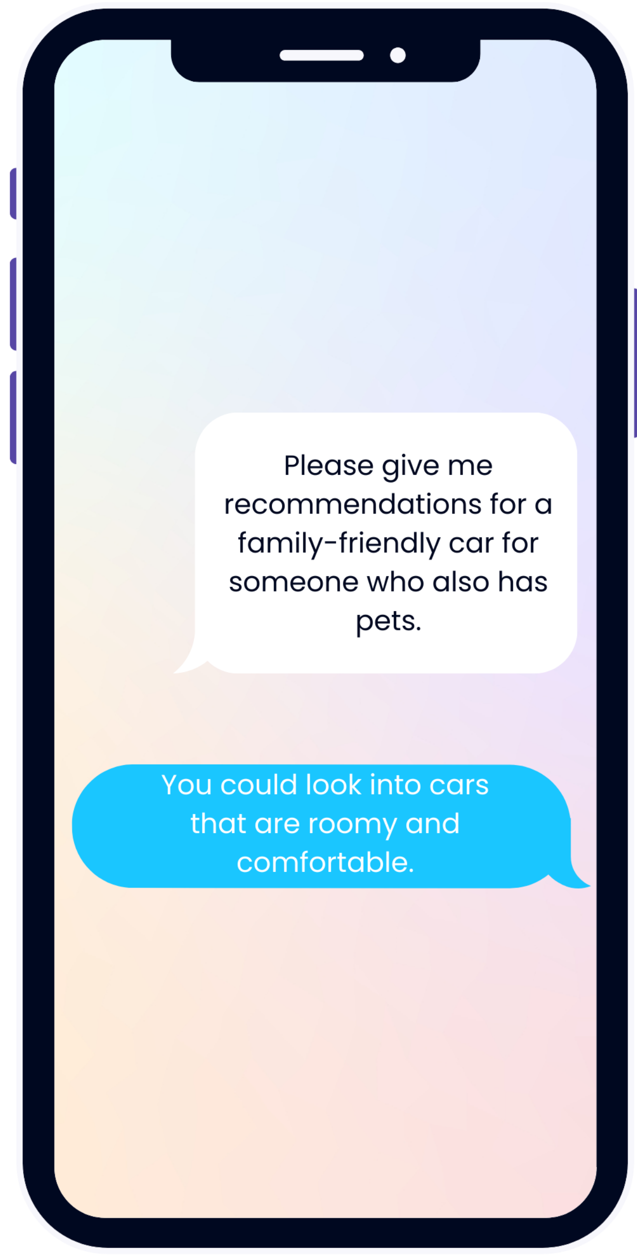
AI AGENTS WITH HUGGING FACE SMOLAGENTS

Agent Final Answer Validation

AI AGENTS WITH HUGGING FACE SMOLAGENTS

Adel Nehme
VP of AI Curriculum, DataCamp

Why Validation Matters


Please give me recommendations for a family-friendly car for someone who also has pets.
You could look into cars that are roomy and comfortable.

- Agent's answer was not helpful
- Customer experience was lost

To avoid this, smolagents lets you validate final answers!

Validating Agent Responses

```python
def check_answer_length(final_answer, agent_memory):
    # Check if the answer is substantial enough
    if len(final_answer) < 200:
        raise Exception("Car recommendation is too brief")
    return True
```

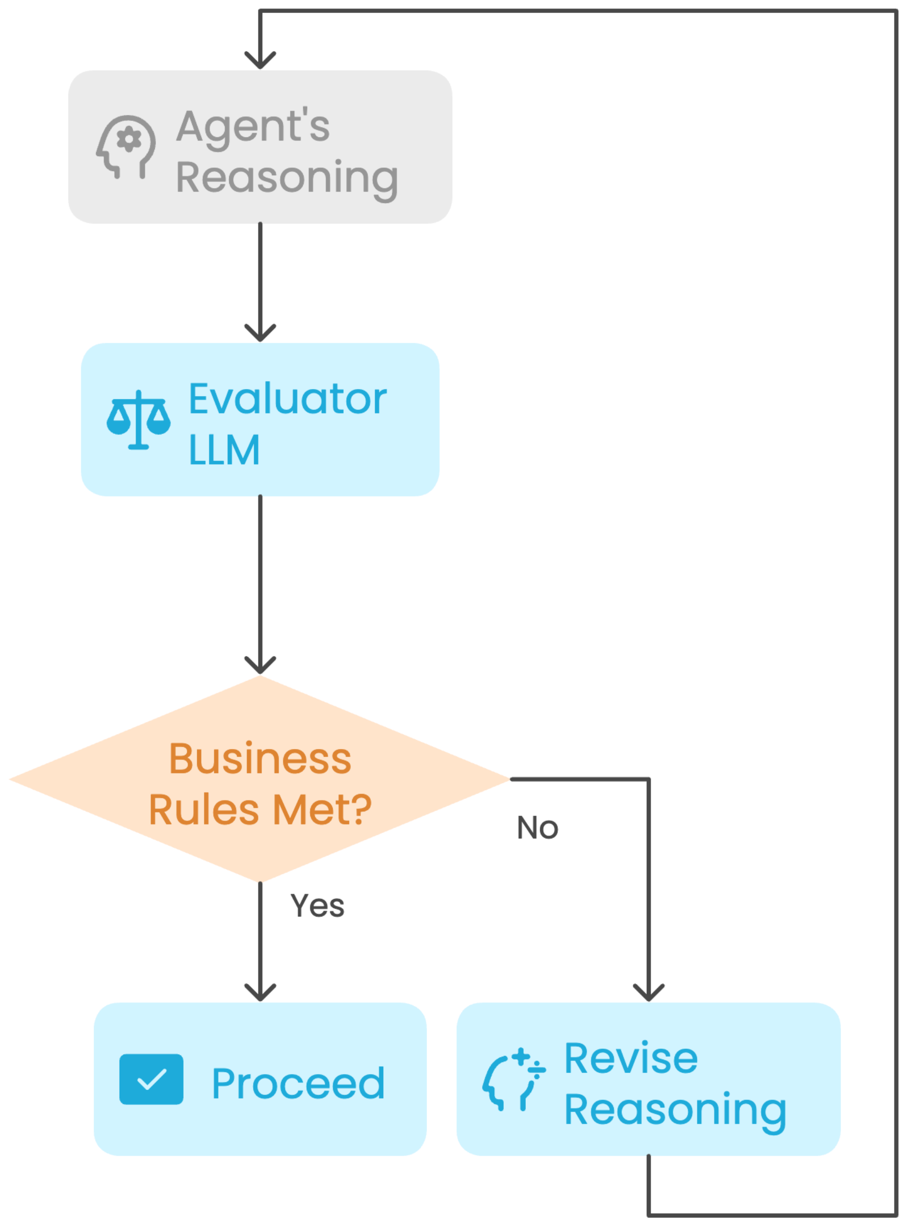
- If `final_answer` fails rule, raise an exception. Otherwise, return `True` .

Using Output Validation in Your Agent

```
car_advisor = CodeAgent(
    tools=[WebSearchTool()],
    model=InferenceClientModel(),
    final_answer_checks=[check_answer_length],
    verbosity_level=0
)
```

- Run `check_answer_length` validation before responding.
- Retry automatically based on exception message defined in the function.

Meta-Evaluation: Using AI to Validate AI



```
validation_prompt = """
Reasoning process: {}

Agent's final answer: {}

Does the final answer logically follow
from the reasoning and solve the user's
question?

Respond only TRUE or FALSE.
No other text.
"""
```

Validating Reasoning with a Meta-Evaluator

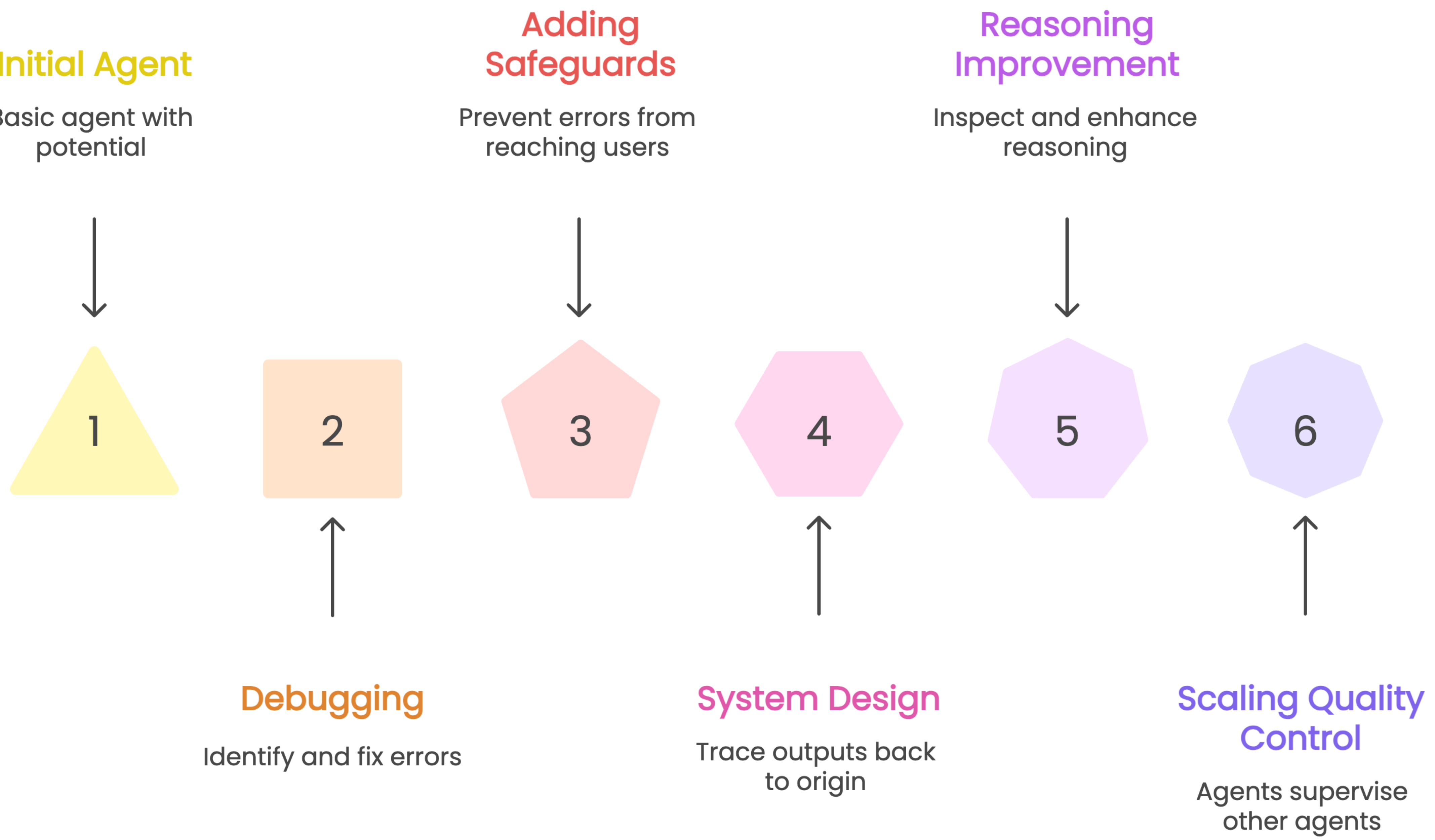
```python
def check_reasoning_accuracy(final_answer, agent_memory):
    evaluator_model = InferenceClientModel()
    reasoning_steps = agent_memory.get_succinct_steps()
    final_prompt = validation_prompt.format(reasoning_steps, final_answer)

    message = ChatMessage(role='user', content=final_prompt)
    evaluation = evaluator_model([message])

    if evaluation.content == "FALSE":
        raise Exception("The agent's reasoning process contains logical errors")
    else:
        return True
```

Combining Multiple Validations

```
car_advisor = CodeAgent(
    tools=[WebSearchTool()],
    model=InferenceClientModel(),
    final_answer_checks=[check_answer_length, check_reasoning_accuracy],
    verbosity_level=0
)
```

More likely to catch and correct errors before the user ever sees them!

Designing Intelligent Systems


Initial Agent
Basic agent with potential
Adding Safeguards
Prevent errors from reaching users
Reasoning Improvement
Inspect and enhance reasoning
1
2
3
4
5
6
Debugging
Identify and fix errors
System Design
Trace outputs back to origin
Scaling Quality Control
Agents supervise other agents

Let's practice!

AI AGENTS WITH HUGGING FACE SMOLAGENTS

Congratulations!

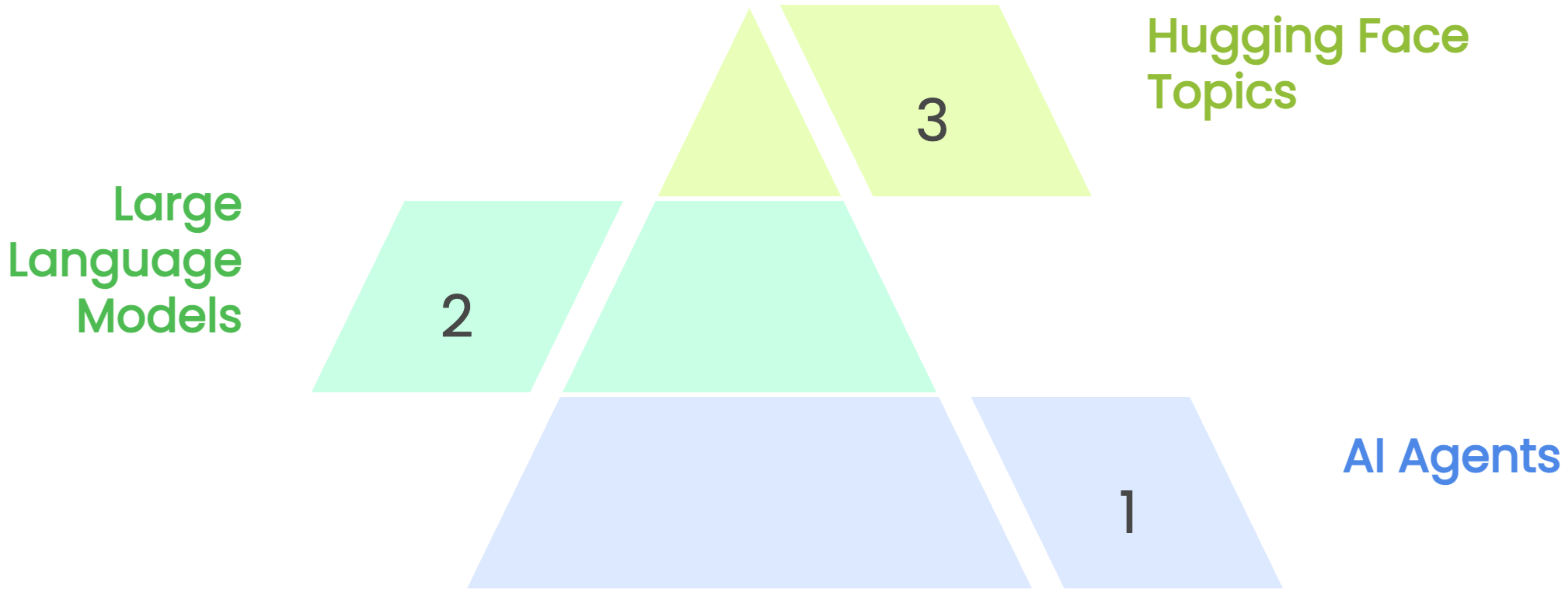
AI AGENTS WITH HUGGING FACE SMOLAGENTS

Adel Nehme
VP of AI Curriculum, DataCamp

What You Learned

- How code agents work, and why they're powerful.
- Built agents that can solve meaningful tasks.
- Extended their capabilities with custom tools.
- Experimented with multi-agent workflows.
- Patterns for debugging and managing agents.

This Is Only The Beginning



See You Soon!

AI AGENTS WITH HUGGING FACE SMOLAGENTS