

Creating AI Agents

MGMT 675: Generative AI for Finance

Kerry Back

Chatbot vs Agent

Chatbot

- Receives prompts
- Generates text responses
- That's it—just conversation
- Cannot take actions
- Cannot access external data

Agent

- Receives prompts
- Can generate text **or** request actions
- Has access to **tools**
- Can query databases, run code, search web
- Takes actions to accomplish goals

An agent is a chatbot with tools

What Are Tools?

Tools are functions the agent can call to interact with the outside world.

Example Tools

- Execute SQL queries
- Run Python code
- Read/write files
- Search the web
- Send emails
- Call APIs

How Tools Work

1. LLM decides to use a tool
2. Returns tool name + parameters
3. Agent executes the tool
4. Result sent back to LLM
5. LLM continues reasoning

Example: Database Analytics Agent

User request: “Analyze quarterly revenue trends for our top 5 customers and create a summary report.”

What the Agent Needs to Do

1. Write SQL to identify top 5 customers by revenue
2. Execute the SQL query against the database
3. Write SQL to get quarterly revenue for those customers
4. Execute that query
5. Write Python to analyze trends and create visualizations
6. Execute the Python code
7. Analyze the results
8. Generate a written report for the user

The Agent Loop

The agent runs in a loop: LLM thinks → tool executes → result returns → LLM thinks again.



- LLM decides next action
- Returns tool name + parameters
- Agent executes the tool

- Result added to message history
- LLM sees result, reasons again
- Loop continues until task complete

Step 1: User Prompt Arrives

Messages Sent to LLM

Role	Content
system	You are a data analyst with SQL and Python tools...
user	Analyze quarterly revenue trends for top 5 customers...

- System prompt defines capabilities
- Lists available tools
- Specifies database schema
- Sets response format

LLM Decides

"I need to first find the top 5 customers. I'll write a SQL query."

Step 2: LLM Requests SQL Tool

LLM Response (Not Text—A Tool Call)

```
{  
  "tool": "execute_sql",  
  "parameters": {  
    "query": "SELECT customer_id, SUM(amount) as total  
              FROM sales GROUP BY customer_id  
              ORDER BY total DESC LIMIT 5"  
  }  
}
```

- LLM doesn't return text to user yet
- Instead, requests a tool execution
- Agent code intercepts this and runs the SQL
- Database returns results

Step 3: Tool Result Returns to LLM

Messages Now Include Tool Result

```
[  
  {"role": "system", "content": "You are a data analyst..."},  
  {"role": "user", "content": "Analyze quarterly revenue..."},  
  {"role": "assistant", "tool_call": "execute_sql(...)"},  
  {"role": "tool", "content": "customer_id,total\n                               ACME,450000\n                               Globex,380000\n                               ..."}  
]
```

The LLM sees the full history including tool results

Step 4: LLM Continues Reasoning

With the top 5 customers identified, the LLM decides what to do next.

LLM Thinks

"I have the top 5 customers: ACME, Globex, Initech, Umbrella, Wayne. Now I need quarterly data for each."

Next Tool Call

`execute_sql`: Get quarterly revenue for these 5 customers...

The loop continues: tool call → result → reasoning → next action

Step 5: Python for Analysis

LLM Requests Python Execution

```
{  
  "tool": "execute_python",  
  "parameters": {  
    "code": "import pandas as pd  
             import matplotlib.pyplot as plt  
             df = pd.DataFrame(quarterly_data)  
             # Calculate growth rates  
             # Create trend visualization  
             plt.savefig('trends.png')"  
  }  
}
```

- LLM writes Python code as a string
- Agent executes it in a sandbox
- Output (prints, files, errors) returned to LLM

Interim Communication with User

Sometimes the agent needs clarification or wants to provide progress updates.

Agent Might Ask

- “Should I include returns/refunds in revenue?”
- “The data goes back 3 years. How many quarters?”
- “Globex has two divisions. Combine or separate?”

Progress Updates

- “Found top 5 customers...”
- “Retrieved quarterly data...”
- “Generating visualizations...”

The agent can pause the loop to interact with the user

Step 6: Final Report

After all tool calls complete, the LLM generates the final response.

LLM Has Seen

- Original user request
- SQL query results (raw data)
- Python execution output
- Any user clarifications
- Generated charts/files

LLM Generates

- Written analysis
- Key findings
- Trends identified
- Recommendations
- References charts

The Complete Message History

What the LLM Sees at Report Time

#	Message
---	---------

- | | |
|---|--|
| 1 | system: You are a data analyst... |
| 2 | user: Analyze quarterly revenue... |
| 3 | assistant: [tool call: SQL for top 5] |
| 4 | tool: [results: ACME, Globex...] |
| 5 | assistant: [tool call: SQL for quarterly data] |
| 6 | tool: [results: Q1, Q2, Q3...] |
| 7 | assistant: [tool call: Python analysis] |
| 8 | tool: [output: stats, saved trends.png] |
| 9 | assistant: Here is my analysis... (final report) |

Agent Architecture Summary

Components

- **LLM**: Reasoning engine
- **System Prompt**: Defines tools & behavior
- **Tools**: SQL, Python, file I/O, etc.
- **Agent Loop**: Orchestrates everything
- **Message History**: Context for LLM

Flow

1. User prompt arrives
2. LLM reasons, picks tool
3. Agent executes tool
4. Result added to history
5. LLM reasons again
6. Repeat until done
7. Final response to user

The Orchestration Layer

The agent's control logic coordinates everything: which LLM to call, which system prompt to use, and what to do with each response.

Different Tasks, Different Prompts

- SQL generation → database schema prompt
- Python analysis → data science prompt
- Report writing → communication prompt
- Each task gets specialized instructions

Different Tasks, Different LLMs

- Simple classification → fast, cheap model
- Complex reasoning → powerful model
- Code generation → code-specialized model
- Cost and speed optimization

How Coordination Works

The agent combines **LLM responses** with **pre-programmed logic** to decide the next action.

LLM Decides

- Which tool to call next
- What parameters to pass
- When the task is complete
- What to ask the user

Code Decides

- If tool_call → execute tool
- If error → retry or report
- If question → prompt user
- If done → return response

The agent is part LLM intelligence, part traditional programming

Example: Multi-Model Routing

Our Database Agent with Model Selection

Task	Model	System Prompt
Parse user request	GPT-4	Extract intent and entities
Generate SQL	Claude	Database schema + SQL examples
Validate SQL	GPT-3.5 (fast)	Check syntax only
Analyze results	Claude	Data analysis instructions
Write report	GPT-4	Business writing style

- Each step uses the best model for that task
- Faster models for simple tasks save time and money
- Specialized prompts improve quality at each step

The Agent's Decision Loop

```
while not done:
    response = call_llm(messages, current_system_prompt)

    if response.has_tool_call:
        result = execute_tool(response.tool_call)
        messages.append(tool_result)
    elif response.needs_user_input:
        answer = ask_user(response.question)
        messages.append(user_answer)
    elif response.is_final:
        done = True
        return response.content
```

Why This Matters

Agents can accomplish complex, multi-step tasks that chatbots cannot.

Chatbot Limitations

- Can only suggest SQL to run
- Cannot see actual data
- Cannot execute analysis
- User must do all the work

Agent Capabilities

- Writes and executes SQL
- Sees and reasons about data
- Performs analysis end-to-end
- Delivers finished product

Claude Code is an agent—it has tools for files, code, web, and more

Building Your Own Agent

The key pieces you need to build an agent:

1. **Define tools:** What actions can the agent take?
2. **Write system prompt:** Describe tools, schema, rules
3. **Implement agent loop:** Send messages, parse tool calls, execute, repeat
4. **Handle tool execution:** Safely run SQL, Python, etc.
5. **Manage context:** Keep message history, handle token limits

Or use an agent framework: **LangChain, CrewAI, Claude Code SDK**

Summary

Key Concepts

- Agent = Chatbot + Tools
- LLM decides which tools to use
- Tool results feed back to LLM
- Agent loop orchestrates flow
- Full history provides context

Our Example

- SQL tool for database queries
- Python tool for analysis
- Multiple tool calls in sequence
- Interim user communication
- Final report from LLM

Agents extend LLMs from conversation to action