Module 7: Building Financial AI Applications

MGMT 675: Generative AI for Finance

Kerry Back

From Questions to Goals

The Progression

Question

- "What is Apple's revenue growth?"
- One step, one answer
- *Chatbot*

Task

- "Build a DCF model for Apple"
- Multiple steps, you direct each one
- *Tool-using AI*

Goal

- "Evaluate Apple as an acquisition target"
- Agent plans and executes
- *Agent*

This module: how to build the third kind — **agents that pursue goals autonomously**.

What Makes an Agent?

Chatbot vs. Agent

Chatbot

- Receives prompts
- Generates text responses
- Cannot take actions

Agent

- Can generate text **or** request actions
- Has access to **tools**
- Takes actions to accomplish goals

An agent is a chatbot with tools and a loop.

What Are Tools?

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

Example Tools

- Execute SQL queries
- Run Python code
- Search the web
- Read and write files

How Tools Work

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

The Agent Loop

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

- **Think**: LLM decides next action
- **Act**: call a tool (SQL, Python, fetch data)
- **Observe**: get the result back
- **Repeat** until goal is achieved

This is exactly what Claude Code does

- Claude Code is a general-purpose agent
- LLM: Claude Opus/Sonnet
- Built-in tools: file I/O, bash, code execution, web search

Building Blocks: The API

The API: Talking to an LLM from Code

An **API** lets your code communicate with an LLM service over the internet.

What You Need

- API endpoint (URL)
- API key (authentication)
- Model name to use

OpenRouter

- Unified API for 100+ models
- Single API key for all models
- Some models are free!
- `openrouter.ai`

A Single API Call

Basic Structure

```
import requests

response = requests.post(
    "https://openrouter.ai/api/v1/chat/completions",
    headers={"Authorization": f"Bearer {API_KEY}"},
    json={
        "model": "mistralai/mistral-7b-instruct:free",
        "messages": [
            {"role": "user", "content": prompt}
        ]
    }
)
answer = response.json()["choices"][0]["message"]["content"]
```

Conversation History

Messages Are a List of Dictionaries

```
messages = [
    {"role": "system",
     "content": "You are a finance tutor..."},
    {"role": "user",
     "content": "What is a P/E ratio?"},
    {"role": "assistant",
     "content": "A P/E ratio is..."},
    {"role": "user",
     "content": "How do I interpret it?"}
]
```

- Each API call is independent — LLM has no memory
- You must send the **entire conversation history** each time
- The **system prompt** defines the agent's behavior and available tools

Building an Agent Step by Step

Agent Loop Pseudocode

The Agent's Decision Loop

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

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

The agent is part LLM intelligence, part traditional programming.

Example: Database Analytics Agent

User: "Analyze quarterly revenue trends for our top 5 customers."

What the LLM Sees at Report Time

#	Message
1	system: You are a data analyst with SQL and Python tools...
2	user: Analyze quarterly revenue for top 5 customers...
3	assistant: [tool call: SQL for top 5 customers]
4	tool: [results: ACME, Globex, Initech...]
5	assistant: [tool call: SQL for quarterly data]
6	tool: [results: Q1, Q2, Q3...]
7	assistant: [tool call: Python chart]
8	tool: [output: chart saved]
9	assistant: Here is my analysis... (final report)

Real Example: Rice Data Portal

Rice Data Portal: A Production Database Agent

- Production agent serving MBA students
- 59K companies, 17M+ stock prices
- Students ask questions in English
- Agent writes SQL, executes it, returns results

Architecture

Component	Implementation
LLM	OpenRouter API
Tool	SQL execution
Agent Loop	Python if/else
UI	Streamlit web app
Database	MotherDuck

The system prompt contains 53 rules encoding schema knowledge, business logic, and query patterns. **Same concept as a SKILL.md — just deployed as a web app.**

Finance Application: M&A Due Diligence

M&A Due Diligence with an Agent

Give an agent a goal: "Evaluate this acquisition target." The agent:

1. Reads the customer list with contracts (Excel)
2. Extracts detailed terms from the contract document (PDF)
3. Applies acquirer's evaluation criteria (Word)
4. Computes quality-adjusted revenue and concentration risk
5. Benchmarks against industry standards (CSV)
6. Produces a summary memo with flagged risks

This is a single prompt to Claude Code. The agent plans and executes all six steps autonomously, reading multiple file formats and combining the results.

User Interface and Deployment

Adding a User Interface

An agent doesn't need a web UI — it can run in a terminal. But a UI makes it accessible to non-technical users.

Streamlit

- Python library for web apps
- No HTML/CSS/JavaScript needed
- Built-in chat UI components
- Deploy to Streamlit Cloud (free)

Gradio

- Python library for ML demos
- Even simpler chat interface
- Built-in sharing via public links
- One-line deployment

Both let you build a chat UI in a few lines of Python.

From Idea to Deployed Agent

1. **Build the agent**: Define tools, system prompt, and agent loop
2. **Add a UI** (optional): Wrap with Streamlit or Gradio
3. **Version control**: Push to GitHub with Git
4. **Deploy**: Streamlit Cloud (free), Koyeb, Render, or cloud providers

Ask Claude Code to do each step — no commands to memorize.

Advanced: Orchestration

The Orchestration Layer

The agent's control logic can route different tasks to different models and prompts.

Different Prompts

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

Different Models

- Simple classification → fast, cheap model
- Complex reasoning → powerful model
- Cost and speed optimization

Sub-agents: dispatch specialized workers for parallel tasks. This is what Cowork does with its parallel VMs.

Exercises

Exercise 1: M&A Due Diligence

1. Download the due diligence data pack (Excel + PDF + Word + CSV)
2. Ask Claude Code to evaluate the acquisition target end-to-end
3. Submit: the summary report + screenshots of intermediate steps

Watch how Claude plans its approach, reads each file, and combines the results into a coherent analysis.

Exercise 2: Financial Chatbot

1. Get a free API key from OpenRouter
2. Ask Claude Code to build a Streamlit app that:
   - Connects to a financial dataset (CSV or database)
   - Lets users ask questions in English
   - Displays results with charts
3. Write a system prompt with schema knowledge
4. Test with 5 questions

Exercise 3: Deploy to Streamlit Cloud

1. Push one of your projects to GitHub
2. Connect your GitHub repo to Streamlit Cloud
3. Deploy and test the live URL
4. Submit: the live URL

Ask Claude Code to handle every step — from creating the Git repo to deploying the app.

Summary

Agents

- Chatbot + tools + loop
- LLM decides what to do
- Pursue goals autonomously

Building Blocks

- API calls to LLMs
- System prompts + tools
- Message history

Deployment

- Streamlit / Gradio UI
- Git + GitHub
- Streamlit Cloud (free)

Agents extend LLMs from conversation to action.