

Agentic AI Summary (M1–M5) — Interview-Ready Notes

1. Executive Summary (High-Level)

- Agentic AI = LLM systems that execute **multi-step workflows** using tools, planning, reflection, and multi-agent collaboration.
- Degrees of agenticness range from **single-shot LLM calls** to **highly autonomous, tool-using, multi-agent planners**.
- Core design patterns: **Reflection, Tool Use, Planning, Multi-Agent Collaboration**.
- Agentic workflows significantly outperform non-agentic (direct generation) workflows on complex tasks.
- **Task decomposition** is essential: break tasks into atomic steps that an LLM or tool can execute.
- **Parallelization** speeds up systems (e.g., multiple web searches or API calls in parallel).
- Tools extend LLMs: web search, PDF-to-text, database queries, code execution.
- Reflection loops provide substantial quality gains compared to zero-shot prompting.
- **Evals** (objective & subjective) guide development and prevent regressions.
- **Trace-based error analysis** identifies failing components and directs fixes.
- Component-level evals make it easier to optimize specific workflow steps.
- Planning workflows enable the LLM to generate structured execution plans (often JSON).
- Multi-agent systems mimic human team roles and improve reasoning and output quality.
- Practical challenges: latency, cost, sandboxed code execution, tuning search/RAG settings.
- Highly autonomous agents integrate planning, tool execution, reasoning loops, and delegation.

2. Core Concepts Breakdown

2.1 Agentic Workflows

Definition: LLM-driven systems executing multi-step tasks.

Why it matters: Reliable automation beyond simple text generation.

How it works:

1. Task decomposition →
2. Step execution →
3. Tool usage →
4. Reflection →

5. Refinement.

Pitfalls: Poor decomposition, missing tools, overlong steps.

2.2 Reflection

Definition: LLM reviews and improves its previous output.

Why: Fixes errors, improves reasoning and structure.

Loop: Generate → Critique → Rewrite.

Pitfalls: Vague critique → weak improvements.

2.3 Tool Use

Definition: LLM chooses and calls external functions.

Why: Gives LLM real-world abilities (data access, compute).

Pattern:

LLM → FUNCTION:{tool(args)} → System executes → LLM continues

Pitfalls: Wrong tool choice, malformed parameters.

2.4 Planning

Definition: LLM produces structured step-by-step plans before acting.

Why: Reduces hallucinations; improves reliability.

Pattern: JSON or textual plan → executed stepwise.

2.5 Multi-Agent Collaboration

Definition: Multiple specialized LLM agents collaborate.

Why: Mirrors human teams → better specificity & quality.

Patterns: Linear, all-to-all, hierarchical.

2.6 Evaluations (Evals)

Objective evals: Code checks, exact matches.

Subjective evals: LLM-as-judge, rubric scoring.

Why: Enables consistent measurement & iteration.

2.7 Error Analysis

Approach:

- Inspect workflow traces
- Identify which component failed

- Quantify error rates
- Why:** Focus efforts where they matter most.
-

3. Architecture & Workflow Summaries

3.1 Agentic Workflow Overview

User Input

↓

Plan (LLM)

↓

Act (Tools / LLM Steps)

↓

Reflect (LLM)

↓

Refine

↓

Final Output

3.2 Reasoning Loop

Plan → Act → Observe → Reflect → Revise

3.3 Retrieval-Augmented Agent Workflow

LLM → Generate search queries

Tool → Web Search

LLM → Extract & Summarize

LLM → Final Output

3.4 Tool Execution Pattern

LLM → FUNCTION:{tool(params)}

System → Tool execution

LLM → Continues reasoning

3.5 Multi-Agent Collaboration Pattern

Manager Agent

↓

Researcher → Designer → Writer

↑

↑

intermediate artifacts

3.6 JSON-Based Planning

```
{  
    "plan": [  
        {"step": 1, "tool": "get_item_descriptions"},  
        {"step": 2, "tool": "check_inventory"},  
        {"step": 3, "tool": "get_item_price"}  
    ]  
}
```

4. Practical Implementation Notes

Pseudocode

```
plan = llm("Create JSON plan")  
  
for step in plan["steps"]:  
  
    if step.get("tool"):  
  
        result = execute_tool(step["tool"], step["args"])  
  
    else:  
  
        result = llm(step["instruction"])  
  
    trace.append((step, result))
```

```
final = llm("Reflect and refine based on trace", trace)
```

Best Practices

- Decompose tasks into short, clear steps.
- Provide strict tool schemas.

- Use sandboxed code execution.
 - Add reflection loops for improvement.
 - Implement evals early.
 - Use parallelization for search-heavy tasks.
-

Common Mistakes

- Overly complex plans.
 - Poor tool descriptions.
 - No evals or weak eval metrics.
 - Not reviewing traces.
 - Over-trusting LLM-as-judge without rubrics.
-

Integrating into ML Pipelines

- Use RAG for domain grounding.
 - Tools for database & API access.
 - Component-level evals for regression safety.
 - Track cost & latency across iterations.
-

5. Interview Q&A Bank

20 Conceptual Questions

1. What is agentic AI?
2. Difference between agentic and non-agentic systems?
3. Why is task decomposition essential?
4. Four design patterns of agentic AI?
5. Benefits of agentic workflows?
6. What is reflection?
7. Why is reflection more effective than direct generation?
8. What is tool use?
9. Why do LLMs need external tools?
10. What is planning?
11. Why JSON-based plans?

12. What are multi-agent systems?
13. Why use multiple agents?
14. What are evals?
15. What is LLM-as-judge?
16. What are component-level evals?
17. Why inspect traces?
18. Common causes of tool-use errors?
19. Common planning failures?
20. Why sandbox code execution?

(Each answer is included in the conceptual section above.)

10 Scenario-Based Questions

1. Customer email workflow produces wrong outputs — debug?
 2. Invoice processor misreads dates — next steps?
 3. LLM picks wrong tool — fix?
 4. Web search returns poor results — fix?
 5. LLM violates output length — what to do?
 6. Multi-agent workflow inconsistent — resolve?
 7. Plan includes invalid steps — fix?
 8. LLM-as-judge inconsistent — how to evaluate?
 9. Workflow too slow — optimize?
 10. Workflow too costly — optimize?
-

10 Hiring-Manager-Level Questions

1. Why agentic AI?
2. How planning improves accuracy?
3. Why tools matter?
4. Why reflection loops?
5. Why multi-agent collaboration?
6. Why evals are necessary?
7. Why error analysis is crucial?

8. Why parallelization matters?
 9. Why JSON plans?
 10. Why sandboxed code execution?
-

6. Real-World Applications (FinTech-Aligned)

Fraud Detection

- Multi-step investigations
- Database queries as tools
- Reflection to refine reasoning
- Planning to structure workflows

Risk Scoring

- Tool-based retrieval of financial attributes
- Reflection to validate reasoning
- Multi-step structured evaluation

Customer Analytics

- Code execution tools for plots
- Reflection to improve chart quality
- Multi-step data workflows

Automation

- KYC, invoice extraction
 - Customer service workflows
 - Tool-driven database + retrieval pipelines
-

7. Final One-Page Revision Sheet

Agentic Concepts

- Multi-step workflows → planning + tools + reflection.
- Patterns: Reflection, Tool Use, Planning, Multi-Agent.

Reflection

- V1 → critique → V2 (large gains).

Tool Use

- Function calling extends LLM abilities.

Planning

- JSON or text plans → reliable execution.

Multi-Agent Systems

- Specialized roles → better reasoning.

Evals

- Objective (code-based)
- Subjective (LLM-as-judge; rubrics better).

Error Analysis

- Inspect traces
- Count errors per component

Optimization

- Parallel search
- Reduce LLM calls
- Tune thresholds

Architecture

Plan → Act → Reflect → Refine → Output
