How to build AI agents for automating networking software development

Randy Zhang
Principal Architect

Shamin Aggarwal
Software Engineer

AIHUB-1013

Cisco Webex App

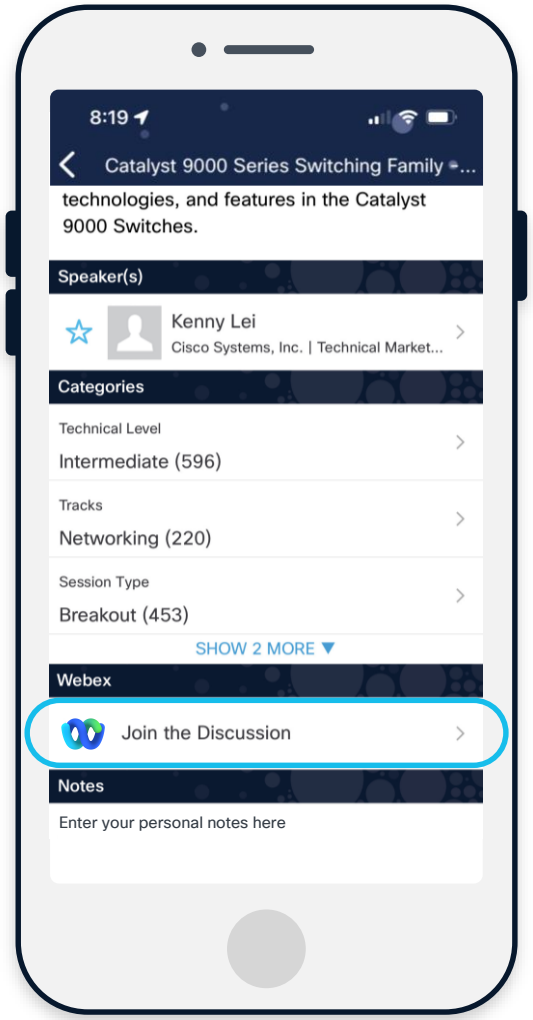
Questions?

Use Cisco Webex App to chat
with the speaker after the session

How

1. Find this session in the Cisco Live Mobile App
2. Click “Join the Discussion”
3. Install the Webex App or go directly to the Webex space
4. Enter messages/questions in the Webex space

Webex spaces will be moderated by the speaker until June 13, 2025.



https://ciscolive.ciscoevents.com/
ciscolivebot/**#AIHUB-1013**

Agenda

01 Introduction

02 Recommendations

03 Summary

Introduction

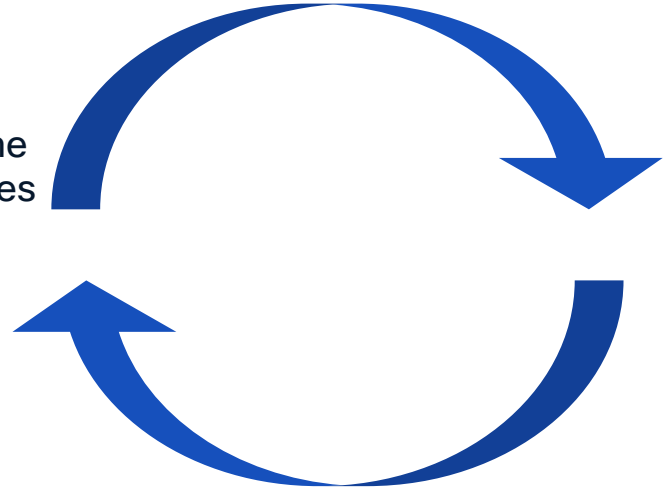
Software Development Lifecycle

1. Defining feature requirements

Clearly define the scope of the project with detailed objectives and requirements

2. Specifying functional components

Creating a plan for the overall software structure and architecture



3. Code development

Writing the code with a given language based on the specifications

4. Code testing and execution

Executing the code in a test environment

5. Bug fixes and enhancement

Fixing bugs and adding enhancements

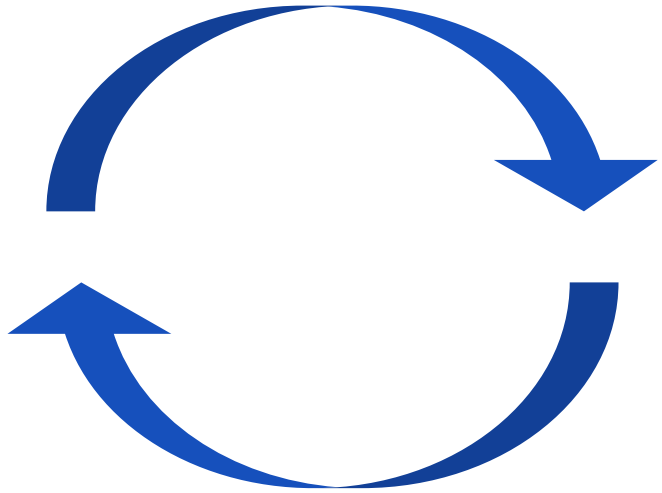
6. Code validation

Validating code in an end to end development environment

AI Can Help with an Agentic Workflow

2. Convert requirements into an action plan for coding

1. Understand my networking requirements



3. Write code based on the plan

4. Execute tests and review the code

5. Fix errors and improve the code

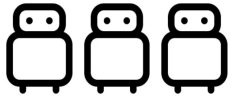
6. Validate the code

Benefits of Agentic AI

- Dynamic and adaptive workflow
- Proactive Planning
- Achieving complex goals
- Interacting with external environment
- Autonomous with no or little intervention

An AI system with the ability to learn from its environment, make decisions and perform tasks independently

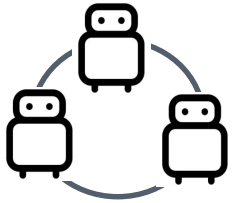
AI Agents and Agentic AI



AI Agents

A worker unit that leverages the reasoning power of a generative AI model to perform specific tasks.

- An individual entity that can perform one or more tasks
- It can make independent decisions using the power of an AI model
- Agents can collaborate to work together



Agentic AI

AI systems designed to autonomously pursue complex goals and workflows with limited direct human supervision.

- A system that leverages generative AI models
- Can handle complex goals, such as a software development project
- An autonomous system with workflows and limited human supervision

Agentic Functions

Planning

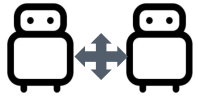
Breaking down complex tasks into manageable steps, determining sequence of actions, self prompting

Reflection

Reviewing and critiquing their own outputs, iterate based on self feedbacks

Tool Use

Using specialized tools to finish specific tasks

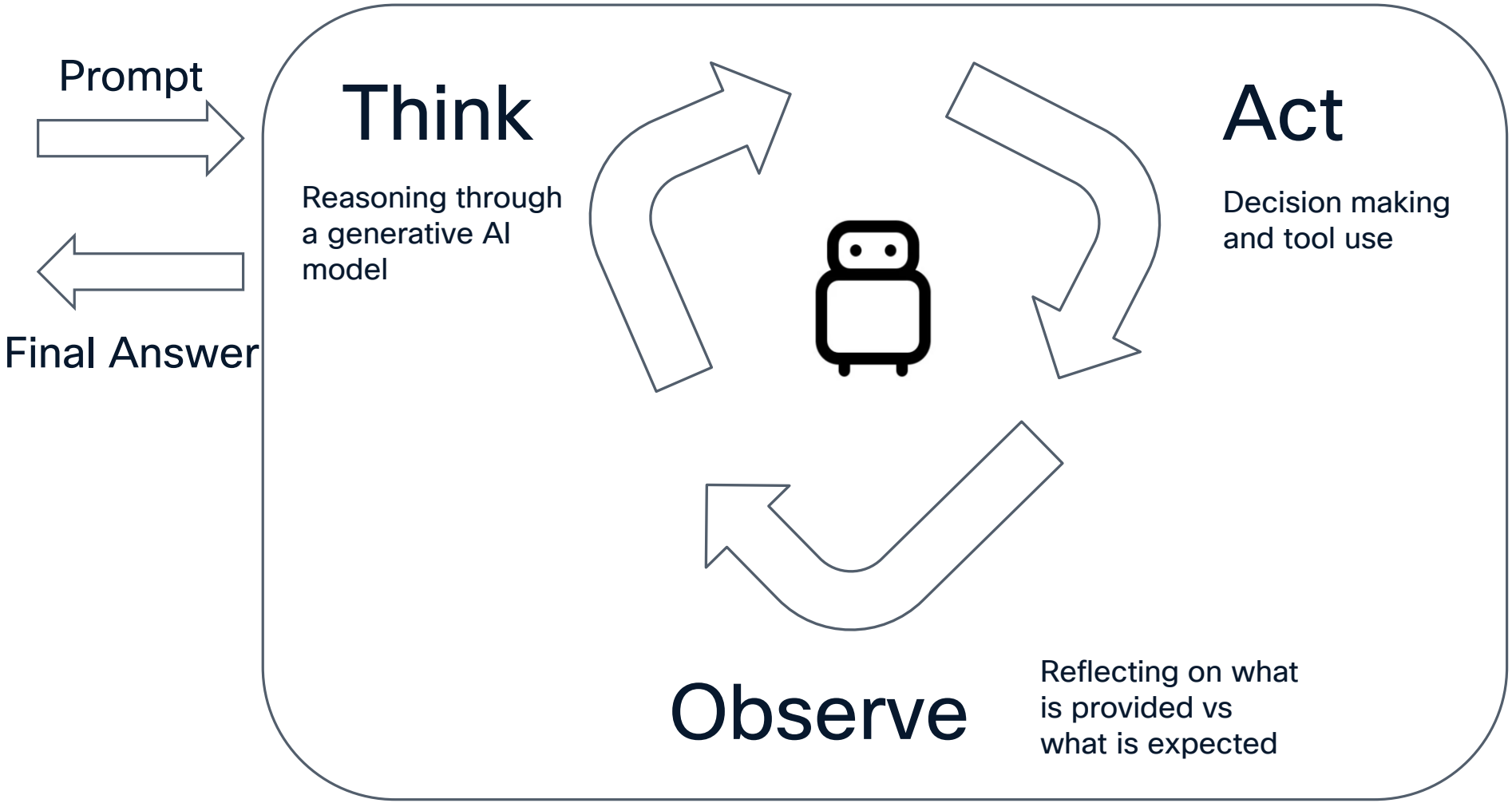


Collaboration

Delegating work to other agents

Agentic Loop

A continual loop until objective is achieved



Benefits of Multi-Agent Systems for Software Development

- Enables complex workflows by distributing tasks among agents with distinct roles
- Each agent is more focused on its core task, thus delivering better performance
- Separating agent memories reduces the input tokens at each step, thus reducing latency and cost

Instead of relying on a single agent, tasks are distributed among agents with distinct capabilities

A Multi-Agentic Example for Software Development

Agent	Functions
Planner	Kicks off the project with a detailed project plan
Coder	Writes and debugs code per requirements
Critic	Reviews code, points out issues and makes suggestions for enhancement
Unit Tester	Writes and runs unit tests
System Tester	Writes and runs system tests
Reporter	Summarizes the project and writes code documentation

Agentic Use Cases for Software Development

- New feature development
- Legacy code modernization (via codebase analysis)
- Bug fixes
- Unit testing
- Codebase standardization
- Validation & Documentation generation

These use-cases show where agentic AI can deliver rapid ROI, especially in organizations with complex, evolving codebases and compliance needs.

Recommendations

Special Topics to Deep-dive

1. Agentic Library Selection
Selection of an agentic framework for software development

2. Large Language Model Selection
Selection of an LLM that is optimized for code writing and debugging

3. Grounding Agents for Developing Quality Code
Providing AI agents with domain knowledge and environment to reduce hallucination

4. Handling Context between Agents
Sharing agent decisions and outputs for common knowledge to reduce noise and token usage

5. Optimizing Token Consumption
Getting the most amount of work done in least number of words for LLMs

6. Tool Calling
Enhancing agents' abilities to interact with workspace and/or external environment through custom functions

7. Dependency Analysis
Allowing Agents to lookup definitions of upstream code dependencies

8. Testing & Validation
Automated testing and review with iterative enhancement

Agentic Library Selection

The LLM engineering landscape is evolving rapidly. The optimal framework depends on your project's specific needs, such as required control granularity, available libraries tailored to your use case, and time constraints.

Considerations

- **Multi-Agent Conversation Patterns:**
  - Supports diverse chat models (e.g., two-agent, group chat, hierarchical).
  - Allowed us to model a dev team for collaborative task completion.
  - **Tip:** Clearly define agent roles and communication protocols.
- **Tool Use Integration (File Ops, Code Execution):**
  - Allows agents to execute code (sandboxed) and manage files.
  - Agents can write, save, execute, and analyze code files.
- **Human-in-the-Loop (HITL) Workflows:**
  - Enables human participation for input, decisions, approvals, or corrections.
  - Crucial for validating critical code or guiding agents in ambiguous situations.
  - **Tip:** Implement HITL at key checkpoints (e.g., code merges, complex bug fixes).
- **Suited for Fast Prototyping & Complex Task Orchestration:**
  - High-level abstractions facilitate quick setup of multi-agent systems.
  - Allowed us to rapidly test automation ideas for various software development lifecycle stages.
  - **Consideration:** For production, evaluate operational aspects (monitoring, logging, error handling).

AutoGen Example: Setting up Agents

Configure an LLM

```
openai_config_list = [
  {
    "model": "gpt-4o",
    "api_key": OPENAI_API_KEY,
    "base_url": azure_endpoint,
  },
  {
    "model": "gpt-4-turbo",
    "api_key": OPENAI_API_KEY,
    "base_url": azure_endpoint,
  }
]
```

```
llm_config_openai = {
        "config_list": openai_config_list,
    }

llm_config = llm_config_openai
```

Set up a Planner agent

```
Planner = ConversableAgent(
    name="Planner",
    system_message="""You are a software architect. You generate clear and detail
coding instructions per requirements for Coder.
     You can generate low level logic to help Coder but you do not write the code""",
    llm_config= llm_config,
)
```

Set up the User Proxy agent for user interactions

```
user_proxy = UserProxyAgent(
    name = "user_proxy",
    human_input_mode = "ALWAYS",
    code_execution_config={
        # the executor to run the generated code
        "executor":LocalCommandLineCodeExecutor(work_dir="tmp_dir"),
    },
```

AutoGen Example: Launching the Workflow

Set up the group chat for agent messaging

```
group_chat = GroupChat(
    agents = [user_proxy, Planner, Coder, Critic,
Reporter],
    messages = [],
    send_introductions= True,
    speaker_selection_method = “auto”,)

group_chat_manager = GroupChatManager(
    groupchat = group_chat,
    llm_config = llm_config,)
```

Initiate the agentic workflow

```
chat_result = user_proxy.initiate_chat(
    group_chat_manager,
    message=""" You lead a team to collaborate on a networking
feature development project using Python.
    The Python code should perform the following 3 tasks:
    <snip>    """,
    summary_method="reflection_with_llm",
    max_turns=5,
)
```

Large Language Model Selection

The choice of LLM significantly impacts your AI agent's performance, cost, and capabilities in software development automation. A diverse range of models exists, varying in size, specialization, access (open-source vs. proprietary), and cost.

For most software development tasks, a larger context window is highly beneficial, if not essential.

LLM Selection Considerations

1. Model Size & Capabilities vs. Use-Case Performance: Match model capability to task complexity. Don't overpay for a powerful model on a simple task.
2. Context Window vs. API Cost & Latency: A larger context window allows more information to be processed, leading to better understanding but typically incurs higher costs and potentially higher latency.
3. Essential Reasoning & Coding Abilities:

- **Tool/Function Calling:** Critical for agents to interact with external tools (compilers, linters, APIs, file systems). *Non-negotiable for most software agents.*
- **Chain-of-Thought (CoT) / Step-by-Step Reasoning:** Enables models to break down complex problems, show their work, and arrive at more accurate solutions. Look for models strong in this.
- **Code Understanding & Generation Quality:** Evaluate proficiency in target programming languages, ability to understand existing code, generate new code, debug, and explain code.
- **Tip:** Test models on representative coding tasks specific to your domain.

4. And Many More: Open-Source vs. Proprietary Models, Fine-Tuning & Customization, Cost, Latency, and Rate Limits

Grounding Agents for Developing Quality Code

To ensure AI agents generate accurate, relevant, and high-quality code, they must be "grounded." This involves providing them with the necessary context, constraints, and feedback mechanisms related to the specific software development task.

Key Strategies

1. Provide Comprehensive Domain Knowledge & Define Scope:

- **Benefit:** Narrows the solution space, ensures adherence to project conventions, and reduces irrelevant outputs.
- **Tip:** Curate and structure your knowledge base for efficient retrieval by the agent.

2. Give Clear, Unambiguous Directions & Objectives:

- **Benefit:** Minimizes misinterpretations and guides the agent towards the desired solution.
- **Tip:** Break down complex tasks into smaller, well-defined sub-tasks with clear objectives for each agent or step.

3 . Implement Iterative Multi-Agent Review, Testing & Refinement:

- **Benefit:** Simulates a peer review process, catching errors, improving code quality, and ensuring adherence to requirements iteratively. Integrated testing also provides continuous feedback on code correctness and functionality, enabling agents to iteratively improve their output. Catches regressions early.
- **Tip:** Define clear review and testing criteria.

Handling Context between Agents

For AI agents to collaborate effectively on software development tasks, mechanisms for managing shared understanding (context) is essential. Poor context handling leads to errors, rework, and inefficient collaboration.

Data & Conversational Context Management Techniques

- **Explicit Context Passing & Summarization:**
  - **Action:** At the end of an agent's task or phase, explicitly summarize key findings, decisions, outputs, and unresolved issues.
  - **Mechanism:** This summary (structured or natural language) is then passed as focused input to the next agent(s) in the workflow.
  - **Benefit:** Reduces noise, manages token limits for LLMs, and ensures the next agent receives relevant information.

- **Threaded Conversations & Message History:**
  - **Action:** Maintain a clear history of interactions, decisions, and rationales within a specific task or sub-task.
  - **Mechanism:** Most agent frameworks (like AutoGen) manage this inherently within agent conversations.
  - **Benefit:** Allows agents (and humans) to trace back decisions and understand the evolution of a solution.
  - **Tip:** Ensure that the full relevant history is accessible or adequately summarized when an agent needs to make a context-dependent decision.

Optimizing Token Consumption

- **Cost Efficiency:** LLM usage costs are typically proportional to the number of tokens processed (both input and output). Optimizing "work done per token" is vital.
- **Performance & Scalability:** Effective token management is essential for handling complex tasks, scaling agentic workflows, and preventing errors due to exceeding context window limits (e.g., context loss, truncation).

Challenge	Mitigation
Finite LLM Context Windows	**Context Summarization**: Use prompts that ask for concise summaries or key takeaways necessary for next Agent
Long tasks or large codebases can exceed limits	**Focused Processing**: Use tools like Abstract Syntax Tree parsers and dependency analyzers, to allow Agents to request 'focused' code blocks
Verbose/Redundant Prompts & LLM Outputs	· **Precise Prompt Engineering** · **Instruction Re-use**: Utilize system prompts for standing instructions or roles, rather than repeating them in every user message · **Requesting Structured/Concise Output**
Inefficient Agent Communication Loops	· **Purposeful Agent Interactions:** Restrict Agent interactions with an FSM-like graph · **Caching LLM Responses**

Tool Calling

Well-designed tool calling (also known as function calling) transforms AI agents from mere text generators into active participants that can interact with real-world systems, access information, and perform concrete actions, enabling true automation.

General Considerations for Designing & Integrating Tools

1. Robustness & Reliability:
Tools must handle a variety of inputs gracefully and recover from or report errors clearly.

2. Flexibility & Adaptability:
Tools should be configurable and adaptable to different scenarios or parameters provided by the agent.

3. Security & Safety:
- **Sandboxing:** Execute code or commands in isolated environments.
- **Permissions & Access Control:** Limit what tools can do based on agent roles or task requirements.
- **Human-in-the-Loop (HITL):** Require human approval for high-risk tool operations.

4. Clear Interface & Discoverability (for the Agent):
Provide clear, concise, and accurate descriptions (function signatures, docstrings) of tools to the LLM. This enables the LLM to correctly decide when and how to use a tool.

5. Observability & Debuggability:
Log tool invocations, inputs, outputs, and any errors. It's essential for understanding agent behavior, debugging issues, and monitoring tool usage.

6. Relevant Tools for Software Development AI Agents:
- File System Operations
- Code Execution & Analysis
- Testing & Validation
- Dependency & Version Control
- Information Retrieval & External APIs

Dependency Analysis

Challenge: Ensuring that code generated or modified by one agent correctly integrates with code from other parts of the project or external libraries.

Tool-Assisted Dependency Verification

- **Action:** Equip agents with tools to inspect and verify code dependencies.
- **Examples:**
  - **Static Analysis Tools:** To parse import statements, check for undefined functions/classes, and analyze call graphs.
  - **Build System Integration:** To verify that dependencies are correctly declared and resolvable.
  - **IDE-like Functionality (via tools):** To perform "go to definition" or look up function signatures from imported modules.
- **Benefit:** Proactively identifies potential integration issues before full compilation or runtime.
- **Tip:** Provide agents with access to the project's dependency manifest files (e.g., requirements.txt, pom.xml, package.json).

Code Validation

Rigorous code validation is essential to ensure that AI-generated or modified code is correct, reliable, adheres to standards, and meets all specified requirements before integration. This involves a collaborative effort, often mimicked by specialized AI agents.

Recommended Agents for an Automated Code Validation Process

1. **Tester Agent:**
   - **Role:** Focuses on functional correctness and robustness by executing automated tests.
   - **Actions:**
     - Runs unit tests against new/modified code.
     - Executes integration tests to check interactions with other modules.
   - **Output:** Detailed test results (pass/fail status, error messages, code coverage reports).

2. **Critic Agent (or Reviewer Agent):**
   - **Role:** Evaluates the code against a broader set of criteria, including requirements, quality standards, and best practices.
   - **Actions:**
     - Reviews code (and Tester agent outputs) against functional requirements (does it solve the problem?).
     - Assesses non-functional requirements (e.g., performance, security vulnerabilities, maintainability, readability, adherence to coding standards).
     - Flags discrepancies, inconsistencies, or areas for improvement with specific, actionable feedback.
   - **Output:** A structured review report, list of issues, suggestions for refinement, or approval.

Code Validation

The Iterative Improvement Loop:

- **Feedback Flow:** A structured cycle, typically: **Coder Agent** --> **Tester Agent** --> **Critic Agent --> Coder Agent**
- **Repetition:** This loop repeats until the code passes all tests and meets the Critic agent's approval criteria.
- **Human Oversight:** Human-in-the-loop (HITL) can be integrated at any stage, especially for complex issues or final approval.

Benefits of Multi-Agent Validation:

- **Ensured Quality:** Systematically improves code quality by enforcing functional correctness and adherence to non-functional standards.
- **Increased Confidence:** Builds trust that the AI-generated code meets project objectives and quality bars before it's integrated into the main codebase.
- **Efficient Refinement:** Provides targeted feedback, allowing the Coder agent to make more effective revisions.
- **Enhanced Specialization & Performance:** Enables the use of different LLMs optimized or fine-tuned for each specialized agent's role.

Summary

Summary and Additional Information

Agentic AI presents a new low code alternative to networking software development

Additional information:

Building AI Agents to Automate Enterprise-Level Software Development: A Practical Perspective

Automated networking script development with AI agents

Project Neo: an AI-powered automated event-routing and response system

Demystifying AI Networking Protocols

Complete Your Session Evaluations

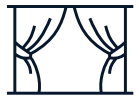
Complete a minimum of 4 session surveys and the Overall Event Survey to be entered in a drawing to win 1 of 5 full conference passes to Cisco Live 2026.

Earn 100 points per survey completed and compete on the Cisco Live Challenge leaderboard.

Level up and earn exclusive prizes!

Complete your surveys in the Cisco Live mobile app.

Continue your education



Visit the Cisco Showcase for related demos

Book your one-on-one Meet the Engineer meeting

Attend the interactive education with DevNet, Capture the Flag, and Walk-in Labs

Visit the On-Demand Library for more sessions at www.CiscoLive.com/on-demand

Contact us at: the Webex team space for this session

Thank you

cisco Live !