Integrating AI into your automation solutions

Calvin Remsburg

Today’s Workshop

Presentation:

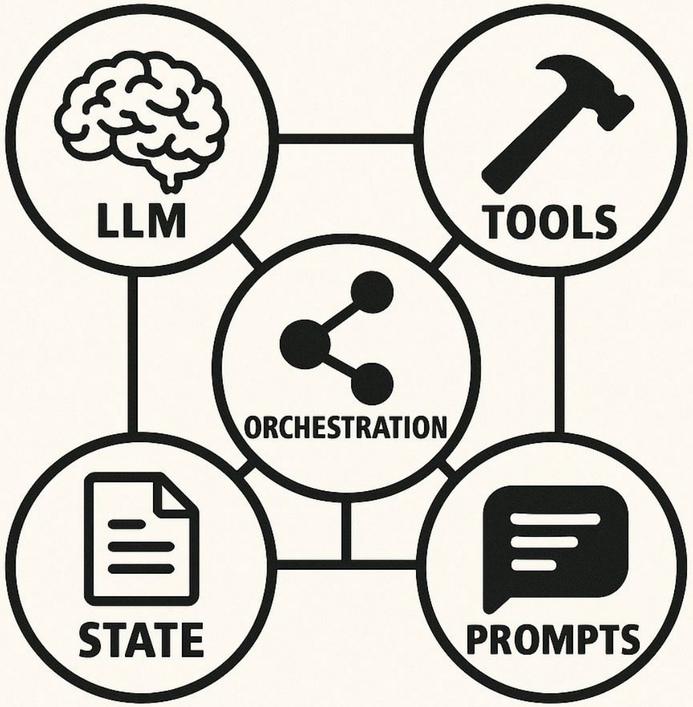
- Answers to “who, what, when, where, why, and how?”
- Detailed overview of AI Agents

Demos:

- Embedding a ReAct graph into Typer CLI project
- Executing a deterministic graph w/LangGraph Studio
- Fully autonomous agent over FastAPI / React web UI

Discussion time:

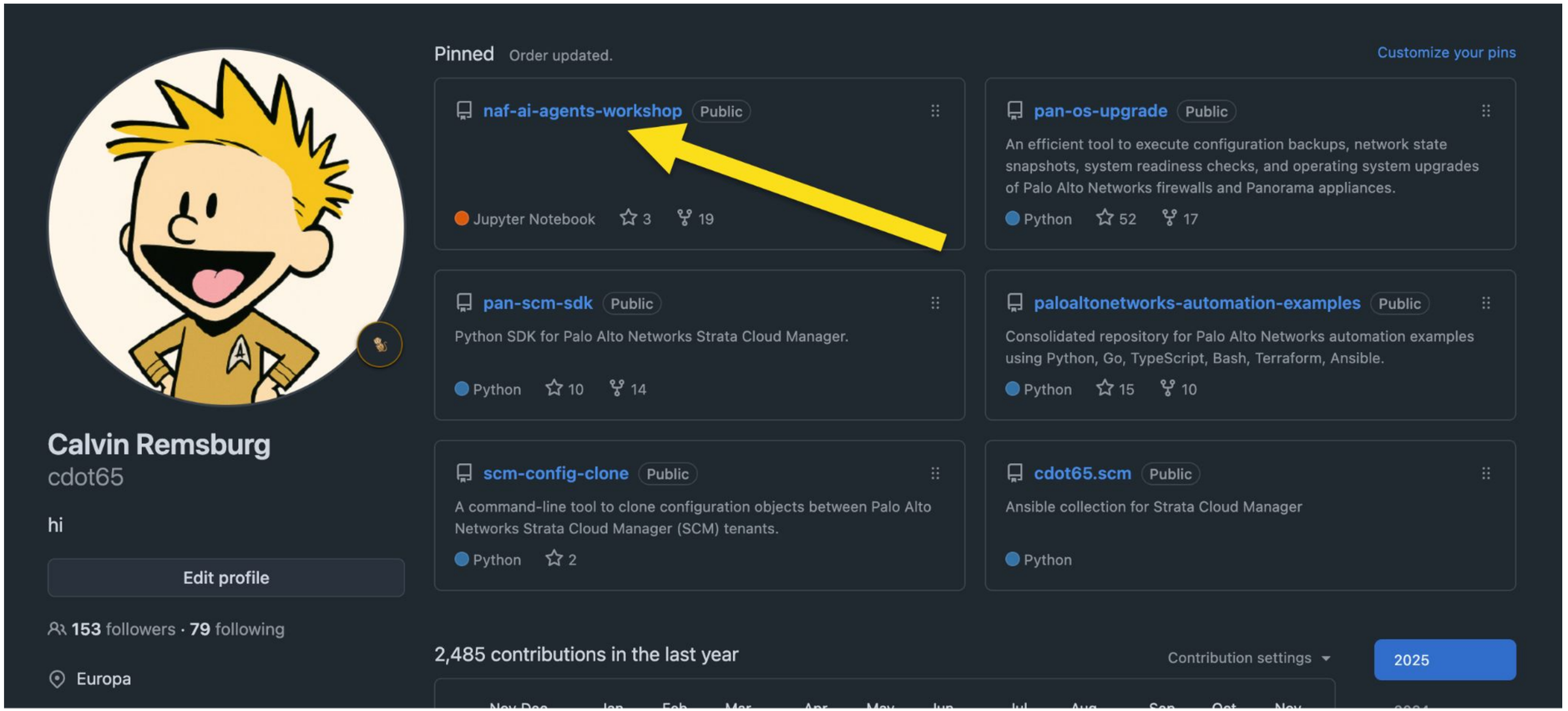
- How to build a secure AI Agent Infrastructure
- Job Risk and Security



Hands On Workshop Setup

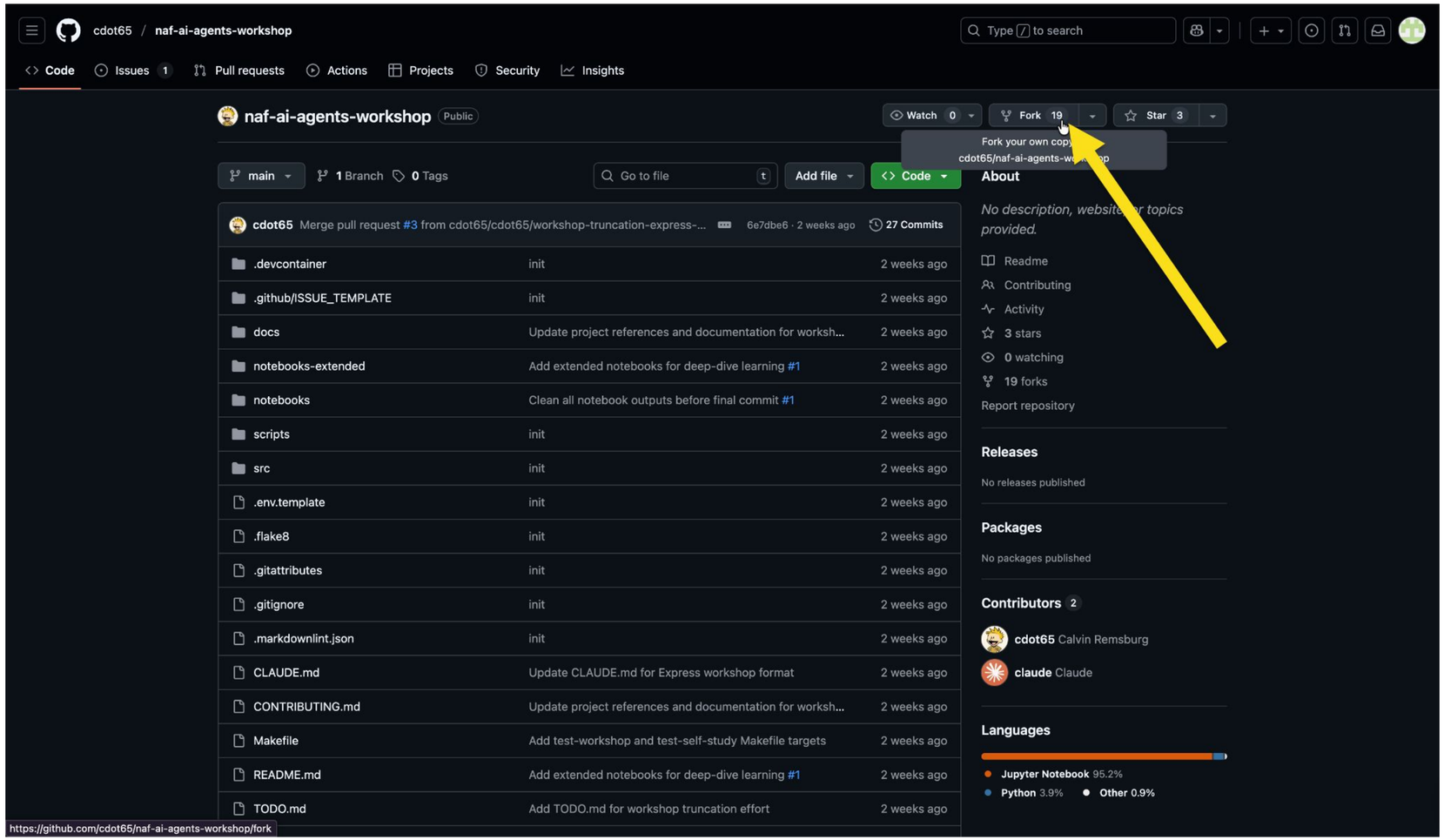
Hands-on Workshop

github.com/cdot65



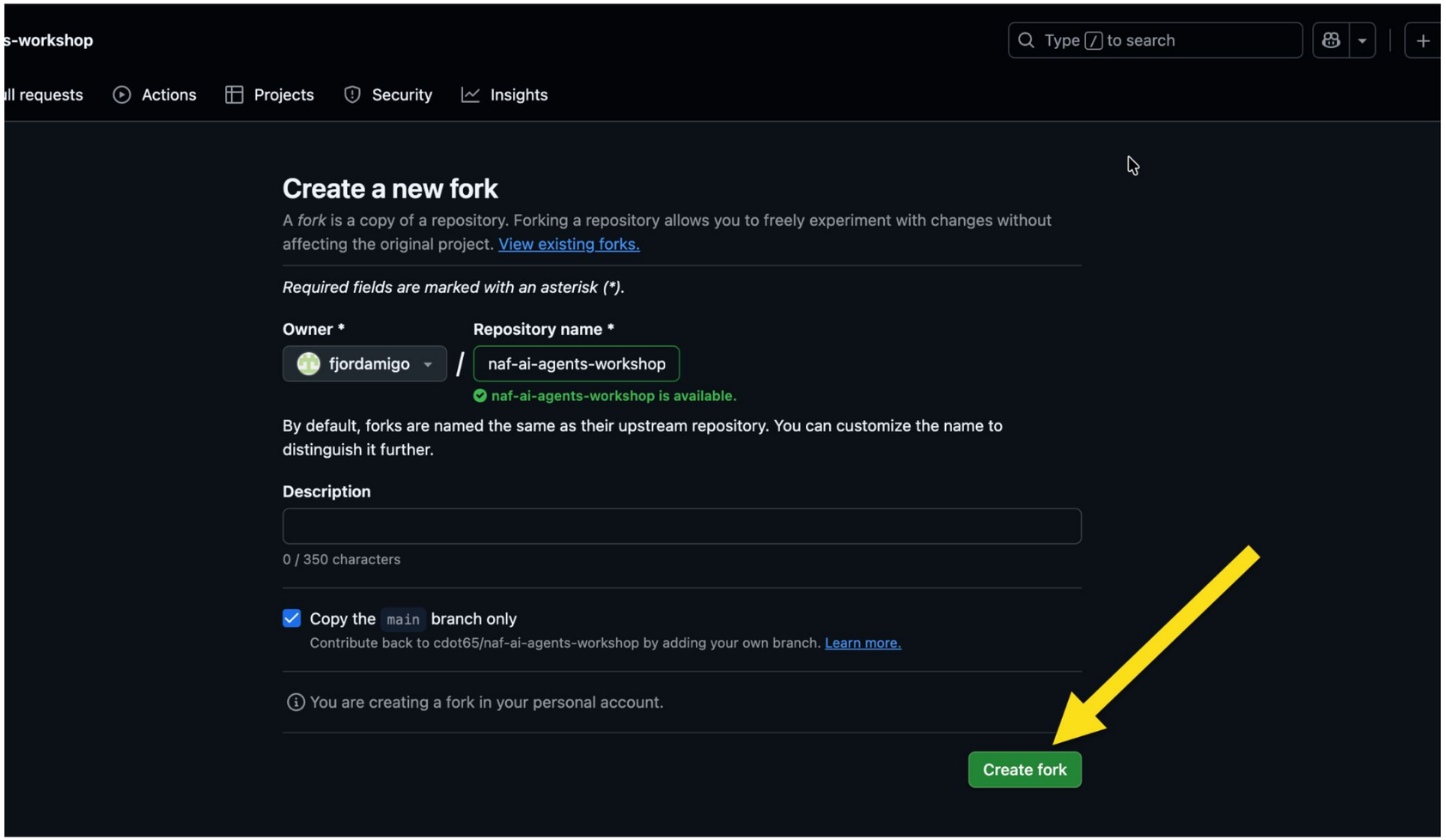
Hands-on Workshop

Fork The Repo



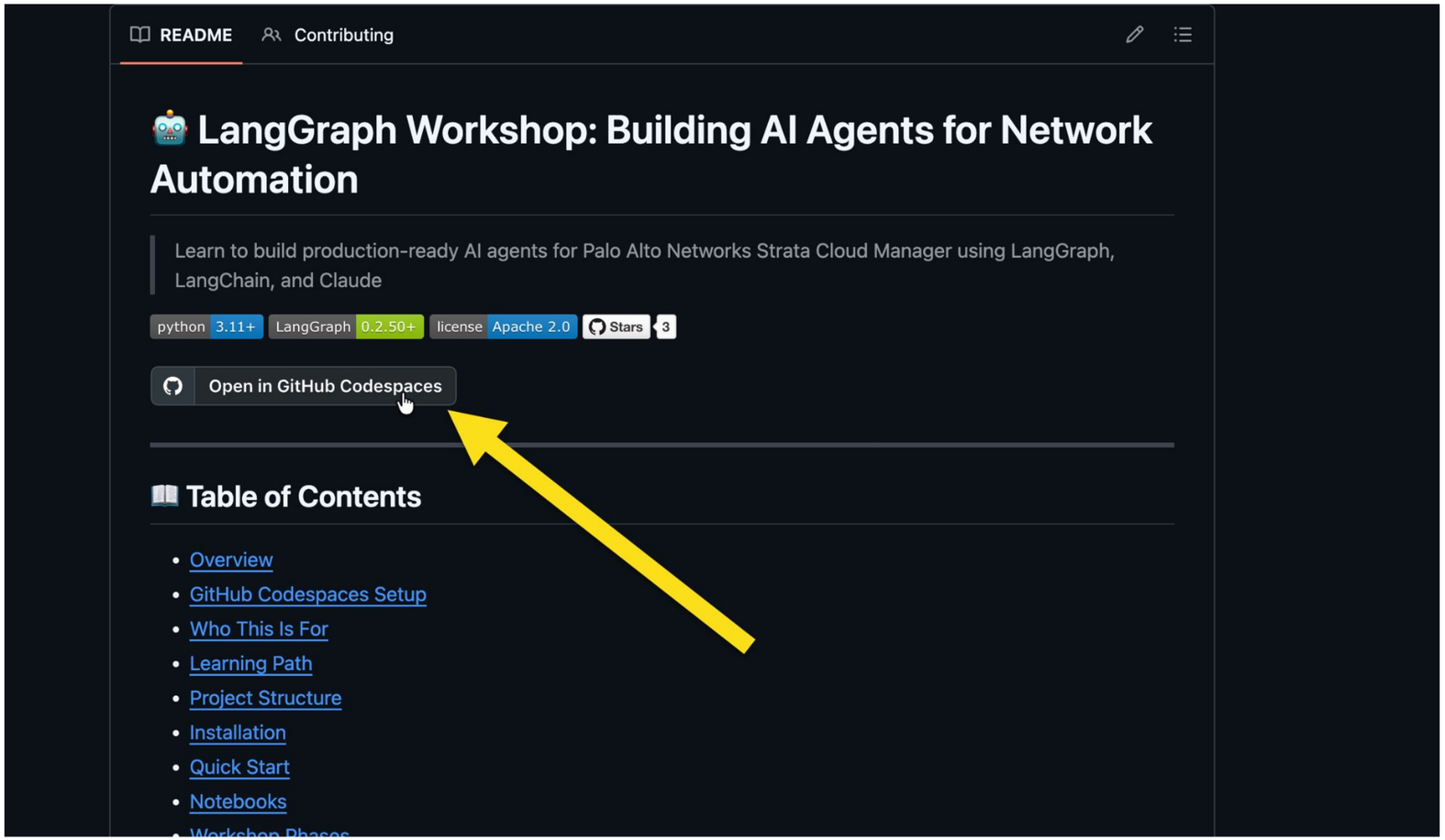
Hands-on Workshop

Create The Fork



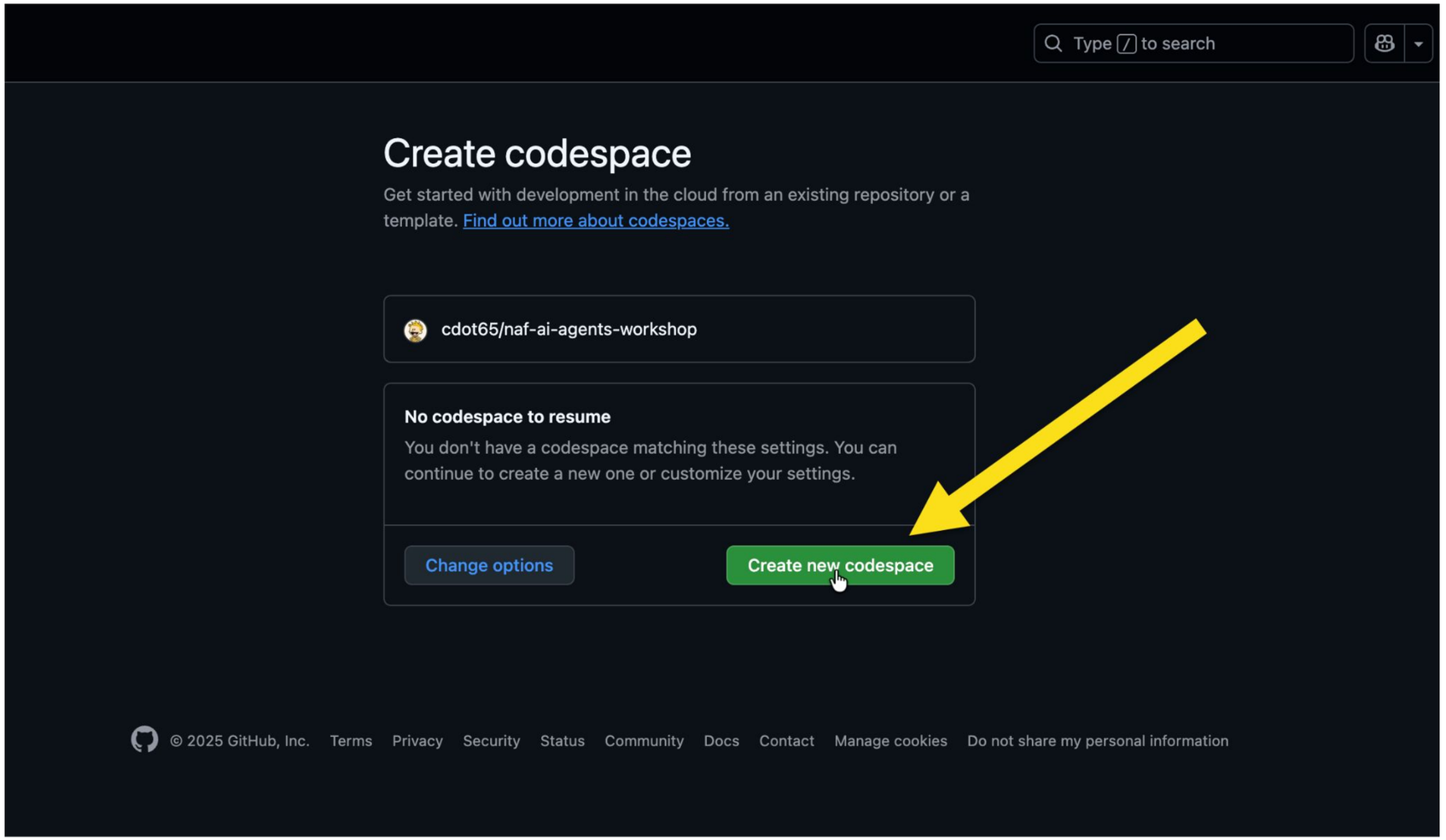
Hands-on Workshop

Open In GitHub Codespaces



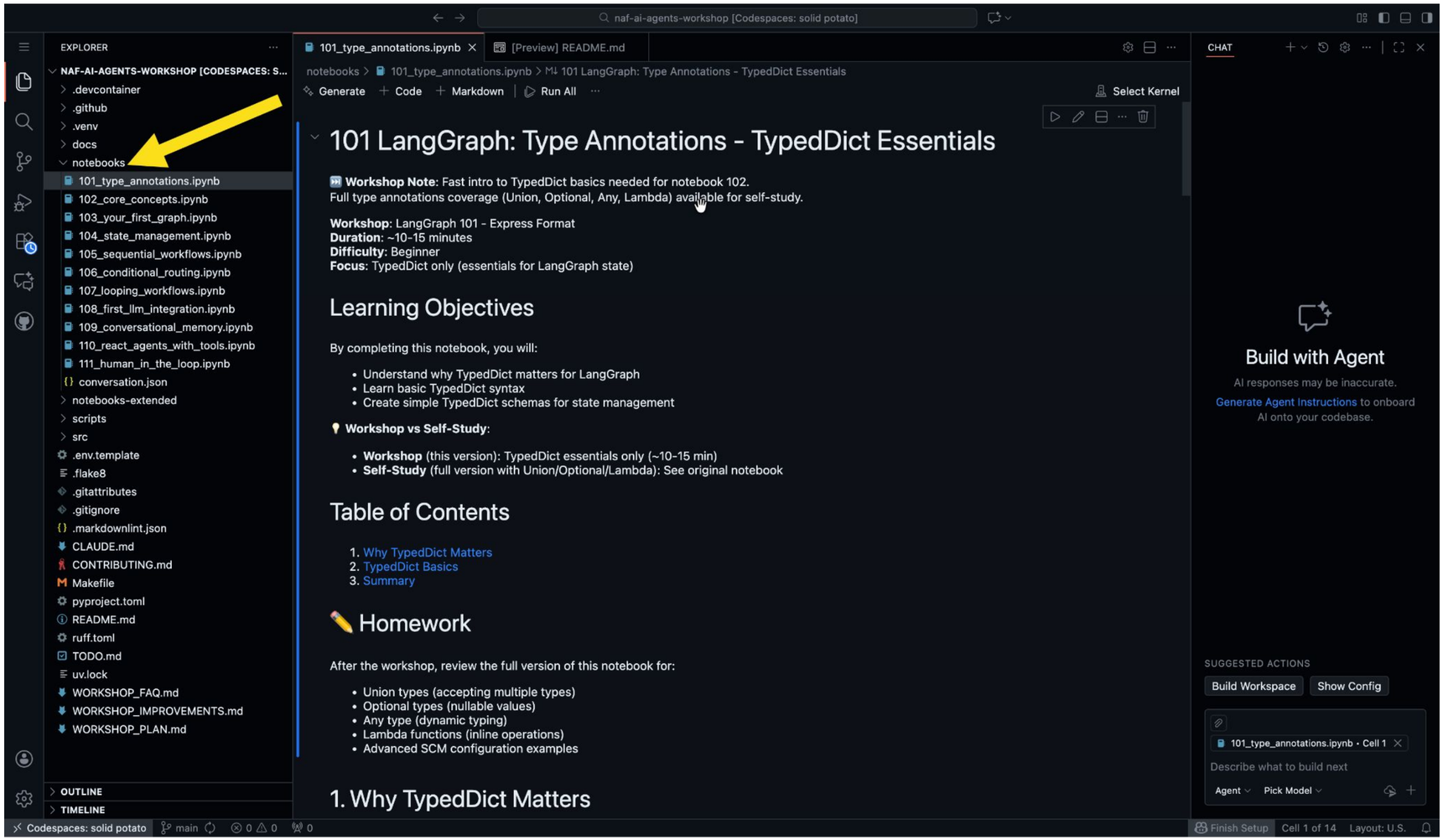
Hands-on Workshop

Create New Codespace



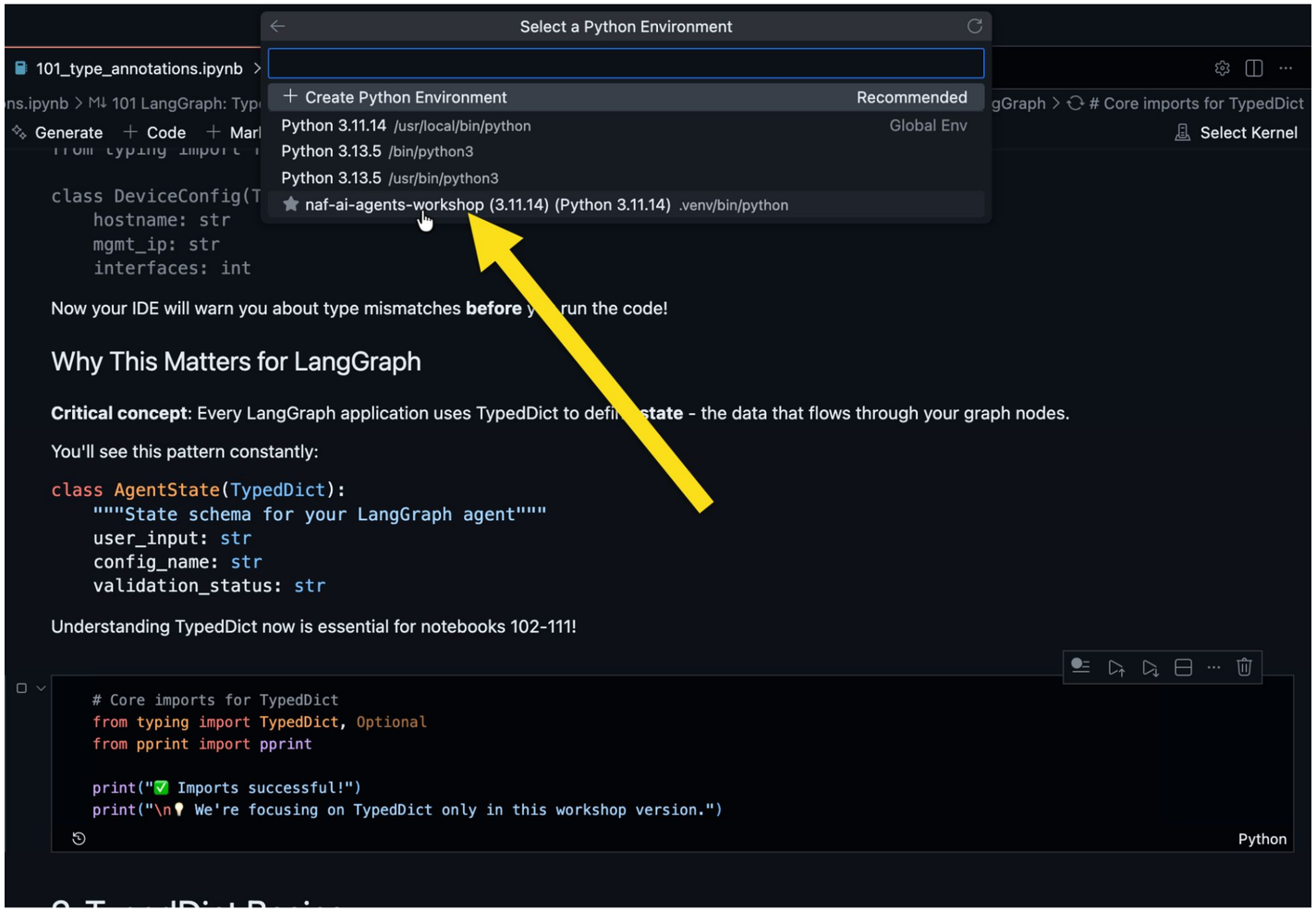
Hands-on Workshop

Open The Notebooks, Do Them



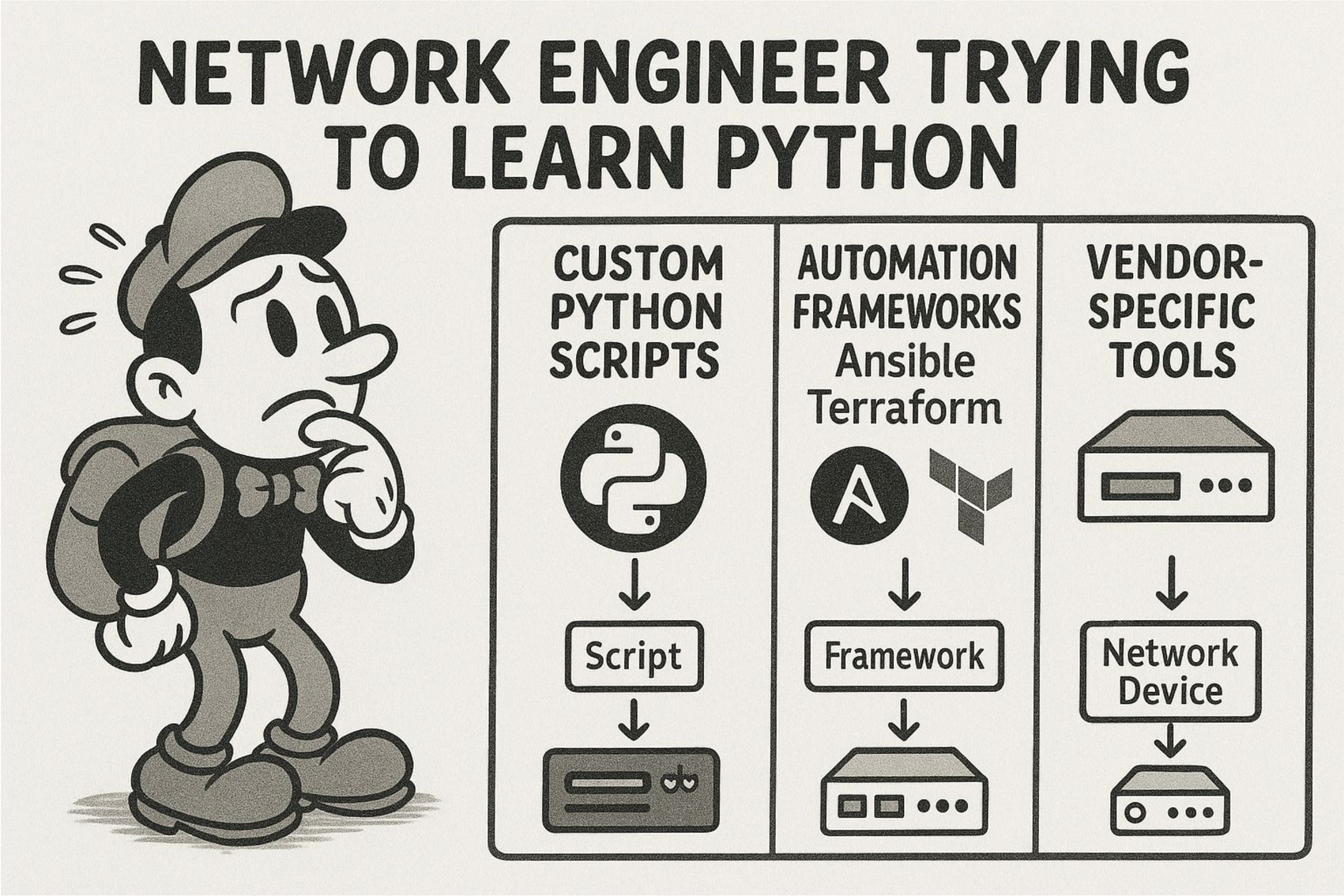
Hands-on Workshop

When Prompted, Select VirtualEnv



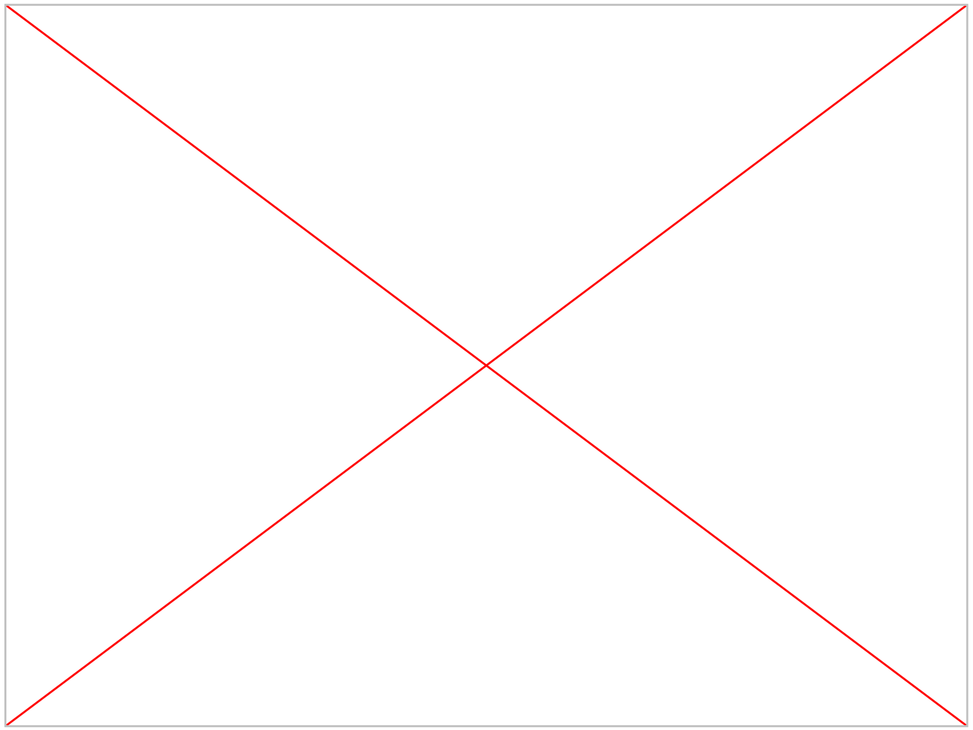
The Problems We Face Today

Where We Are: The Automation Landscape



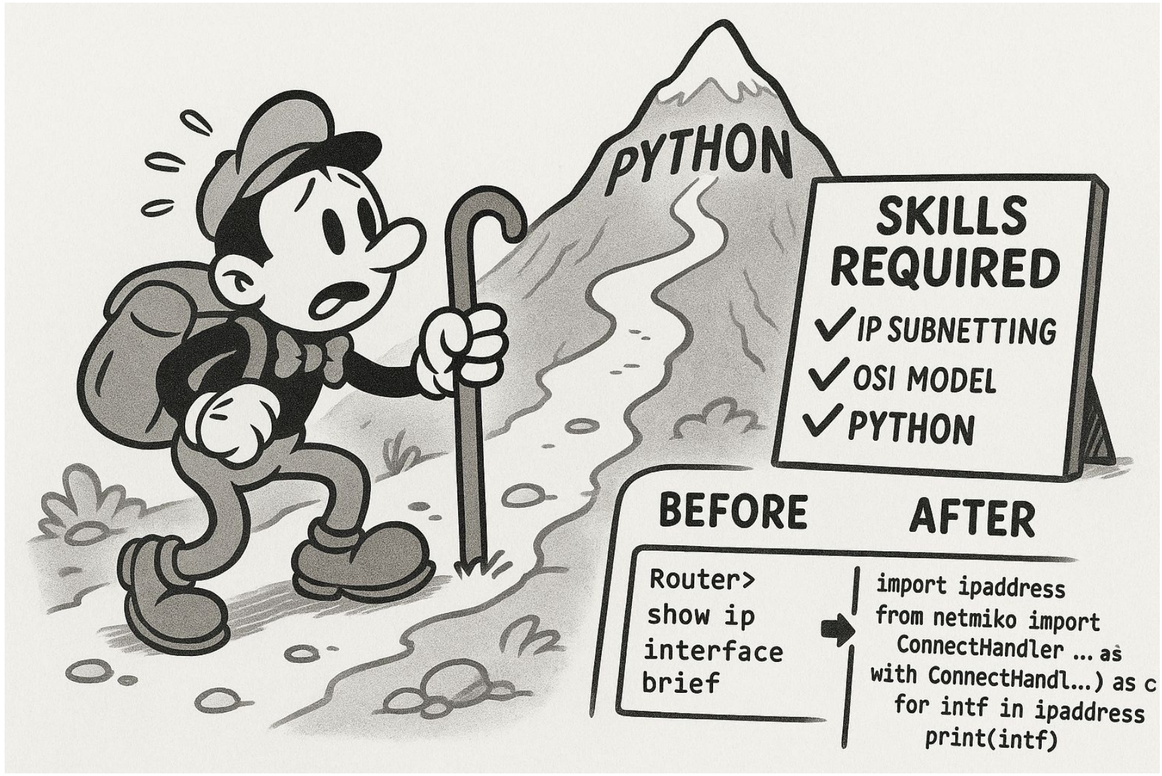
Why Automation Remains Expert-Only

“I can configure a firewall in my sleep, but I feel like it would be easier to learn Klingon than Python”



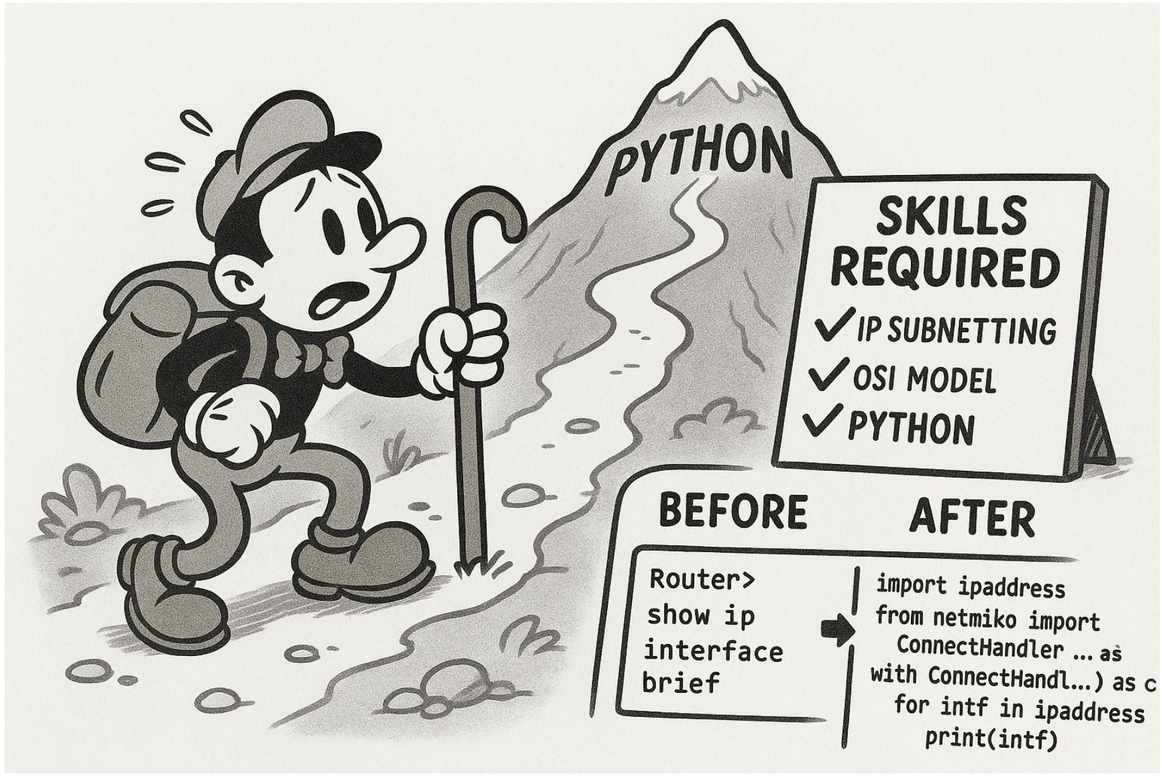
Why Automation Remains Expert-Only

- Most network/infrastructure engineers didn't start their career planning to become developers
- The mental model shift is enormous
- This barrier keeps automation in the hands of a few experts
- Results in bottlenecks and slow adoption



Problem #1 - The Learning Curve Wall

- Setting up development environment
  - venv
  - IDE
- Learning programming fundamentals
  - data structures
  - control flow
  - OOP
- Error and exception handling
- Testing and debugging code
- Version control and collaboration
- The cognitive shift from imperative to programmatic thinking



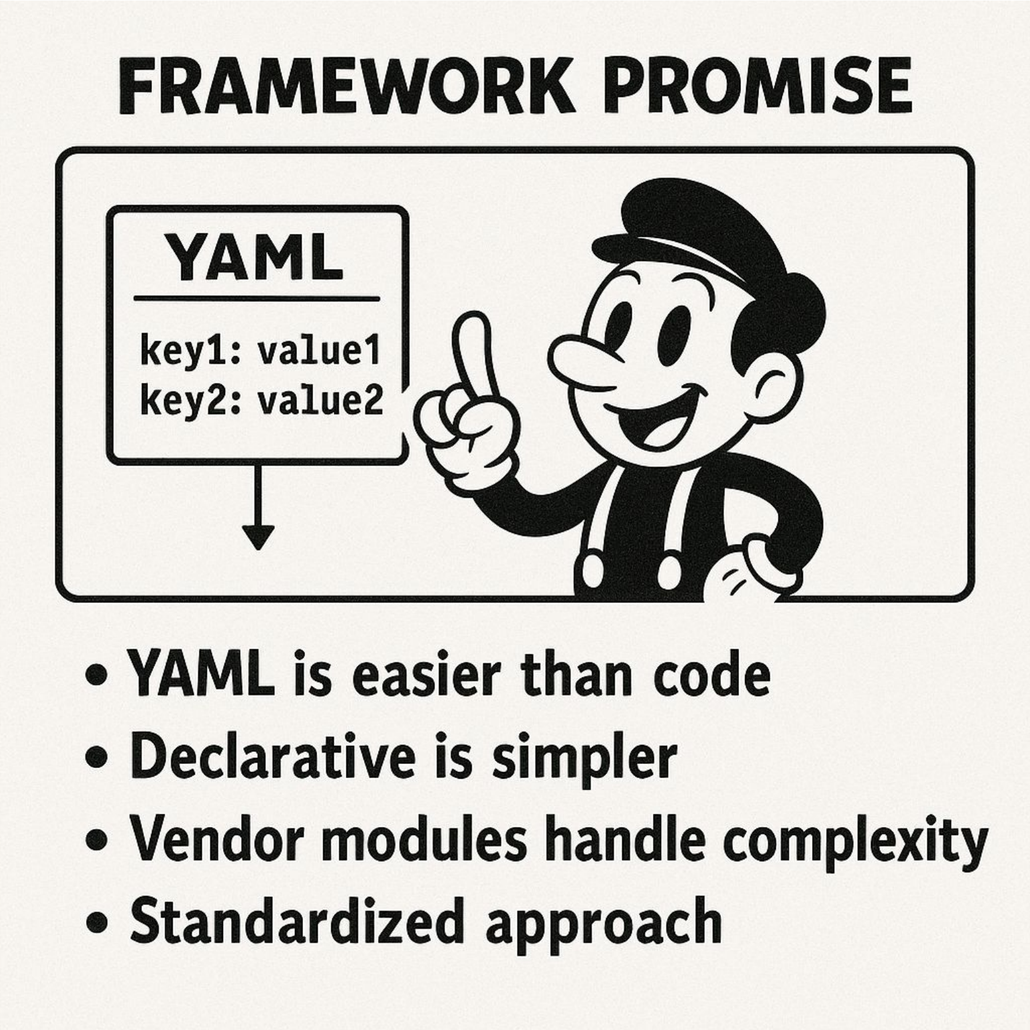
Problem #1 - The Learning Curve Wall

“Permit traffic of type TCP from 10.0.0.0/8 to any address”

```python
# Infrastructure folks think: "permit tcp 10.0.0.0/8 any"
# They must write:
def create_rule(source_net, dest, protocol, action):
    try:
        rule = {
            'source': source_net,
            'destination': dest,
            'protocol': protocol,
            'action': action
        }
        validate_rule(rule)
        return api_call('security/rules', rule)
    except ValidationError as e:
        logger.error(f"Rule validation failed: {e}")
        raise
    except APIException as e:
        handle_api_error(e)
```

Problem #2 - Framework Trade-Off: Simplicity vs Capability

Promise
YAML is easier than code
Declarative is simpler
Vendor modules handle complexity
Standardized approach



Problem #2 - Framework Trade-Off: Simplicity vs Capability

Reality
Abstractions requires learning new syntax
Procedural logic is awkward or impossible
Limited to what module supports
Rigid patterns don't fit all use cases

Problem #2 - Framework Trade-Off: Simplicity vs Capability

“I want to configure a new firewall rule, perform validations and conditionally rollback”

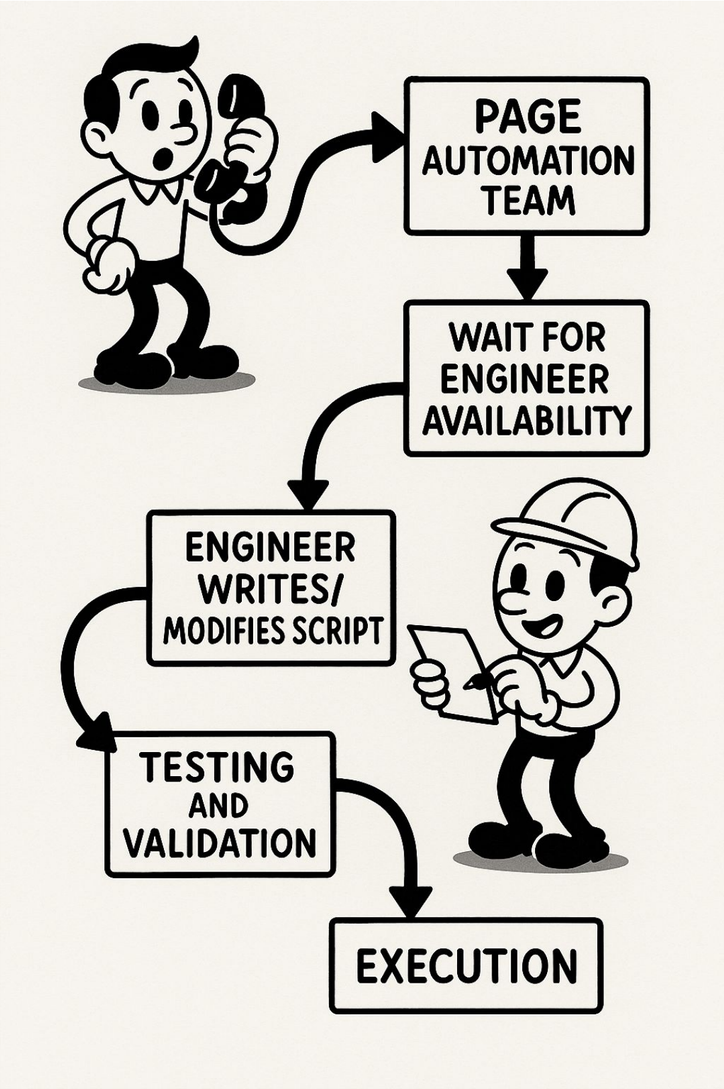
```yaml
# Simple task works great
- name: Configure firewall rule
  panos_security_rule:
    rule_name: "allow_web"
    source_zone: "trust"
    destination_zone: "untrust"

# But complex conditional logic becomes painful
- name: Validate and conditionally rollback
  # Have to use multiple tasks, set_fact, when clauses
  # No real state awareness
  # Can't easily "think" about the problem
```

Problem #3 - The Accessibility Gap

“We built tools that only we can use!”

Operationalizing automation requires

- Understanding the code/playbook structure
- Access to development environment
- Knowledge of execution parameters
- Ability to interpret logs and errors
- Understanding of edge cases and failure modes

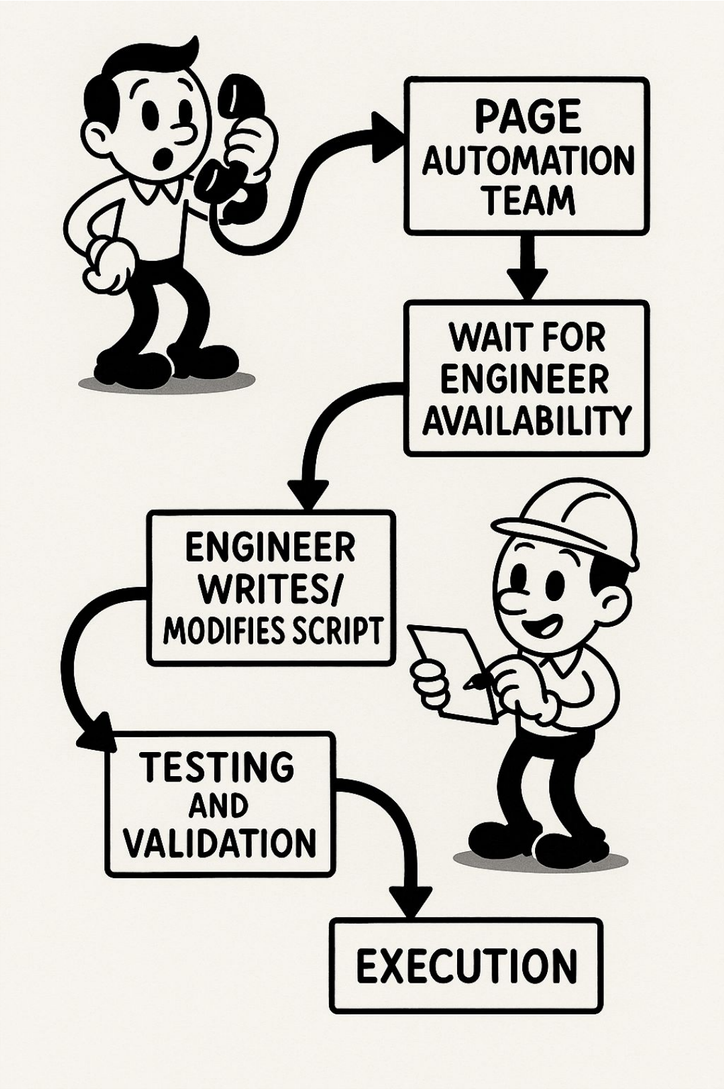


Problem #3 - The Accessibility Gap

“I need to block IP 192.0.2.100 urgently”

Workflow:

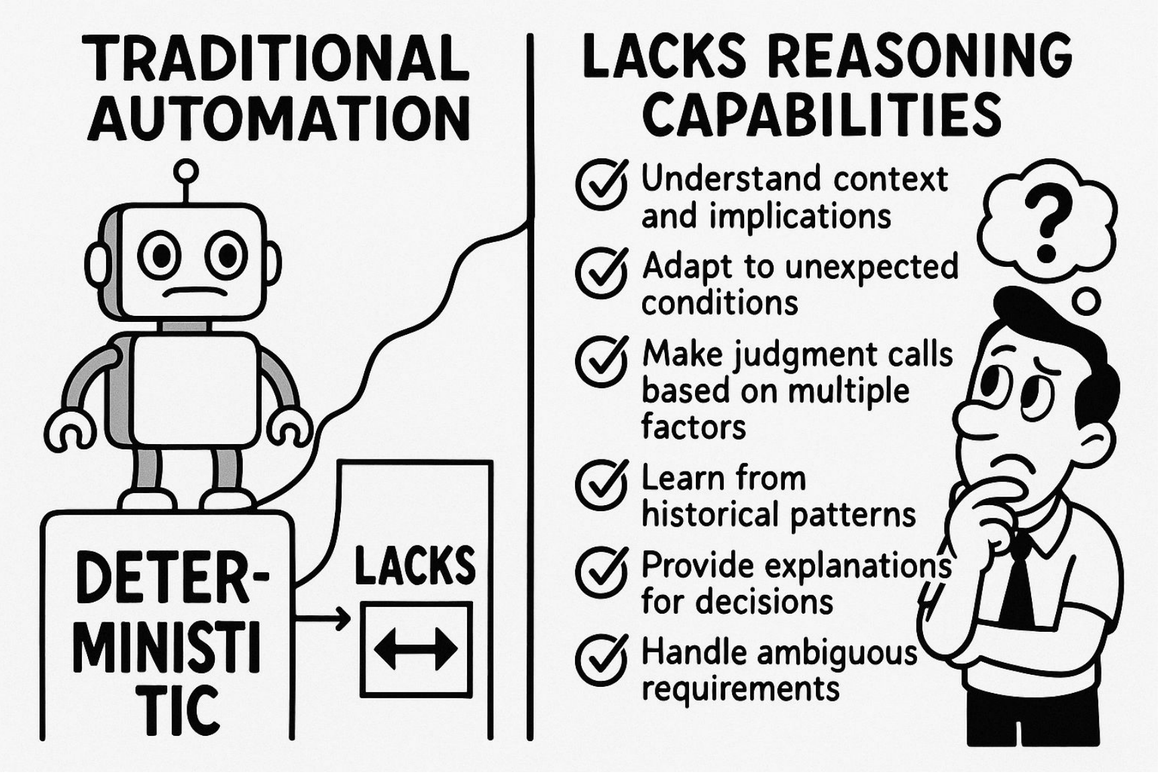
1. Page automation team
2. Wait for engineer availability
3. Engineer writes/modifies script
4. Testing and validation
5. Execution



Problem #4 - The Reasoning Gap

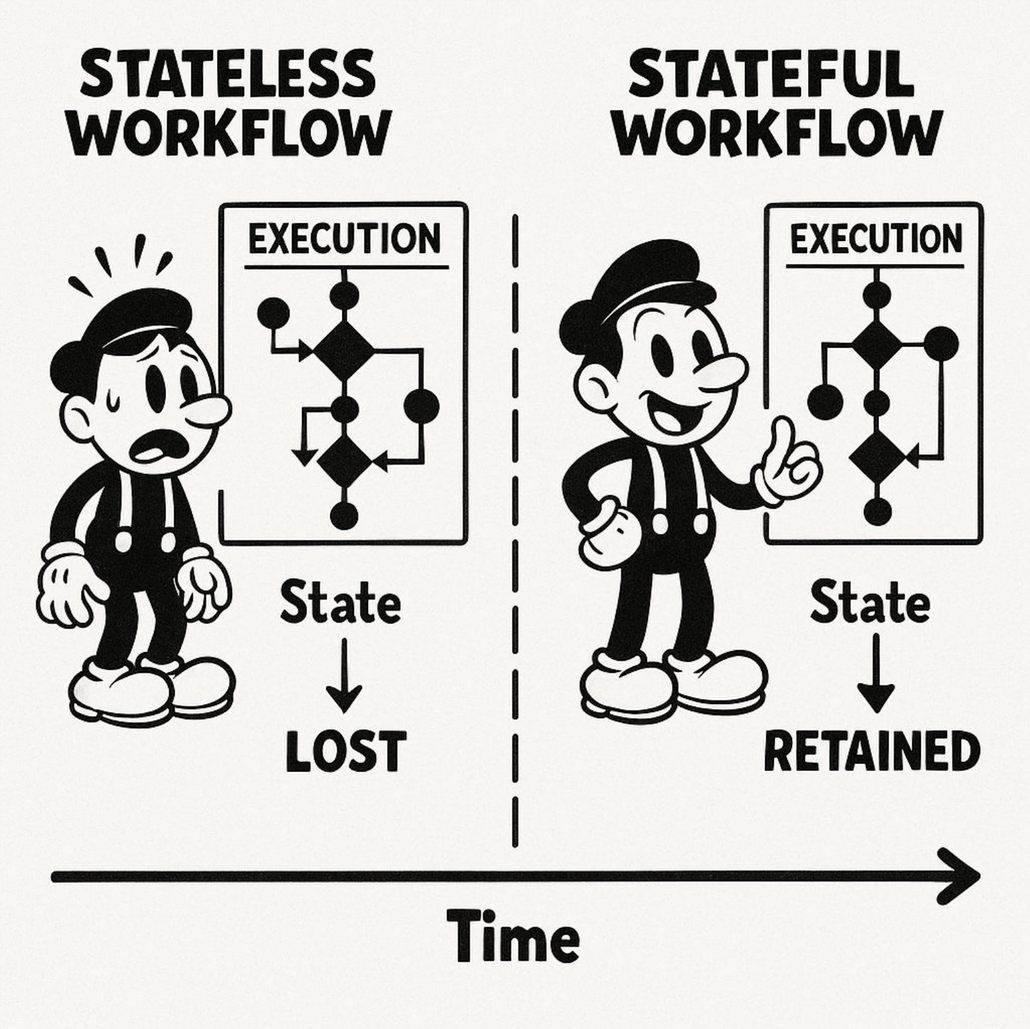
Traditional Automation:

- Apply predefined firewall template
- No context about current state
- No understanding of "secure" in context
- Can't evaluate if it's actually more secure
- Just executes predefined steps



Problem #5 - State Blindness

- Ansible is stateless by design
- Scripts don't persist context between runs
- No memory of previous operations
- Can't maintain workflow context
- Difficult to implement rollback/recovery
- Multi-step processes lose coordination



The Cost of These Problems

Quantifiable Costs:

- **Time**: 30-60 minutes average for simple changes
- **Bottlenecks**: Small automation teams become blockers
- **Errors**: Human intervention points = error opportunities
- **Coverage**: <30% of operational tasks actually automated
- **Utilization**: Skilled engineers spending time on routine tasks
- **Risk**: Delays in security response due to accessibility issues

Introducing AI Agents

Enter AI Agents: Intelligent Automation

“What if your automation could reason, learn, and adapt?”

AI Agents are software that can:

- Understand natural language intent
- Reason about problems and context
- Use tools and APIs to take actions
- Maintain state and memory
- Adapt to changing conditions
- Explain its decisions
- Learn from feedback

Value Proposition #1 - Natural Language Interface

Natural language as the new CLI

Before

```
scm-agent ❯ $ python deploy_rule.py \
--source 10.0.0.0/8 \
--dest any \
--action deny \
--priority 100 \
--commit
```

After

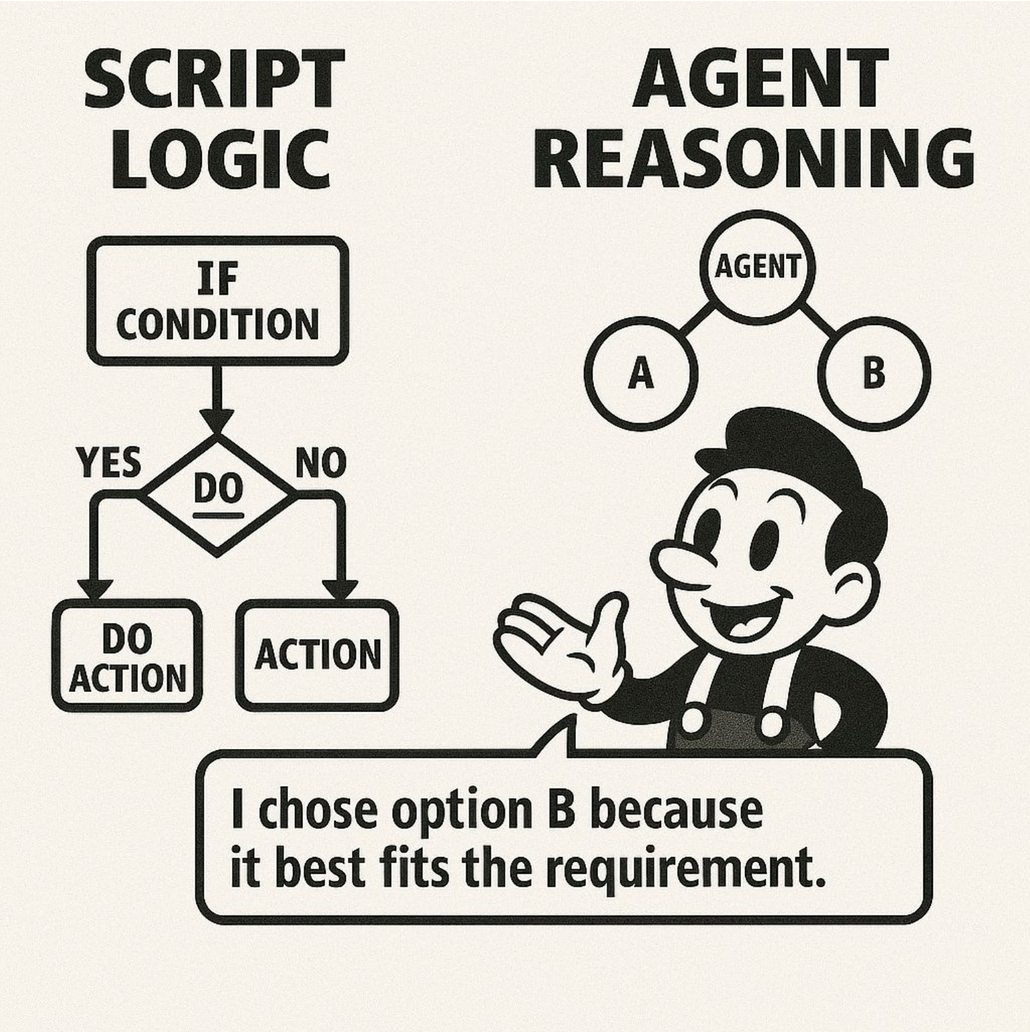
```
You: Block all traffic from the 10.0.0.0/8 network
```

Value Proposition #2 - Intelligent Reasoning

Automation that understands and reasons, not just executes

Reasoning Capabilities

- Analyzes current state before acting
- Considers security, performance, compliance
- Adjusts approach based on conditions
- Evaluates potential impacts
- Provides rationale for decisions

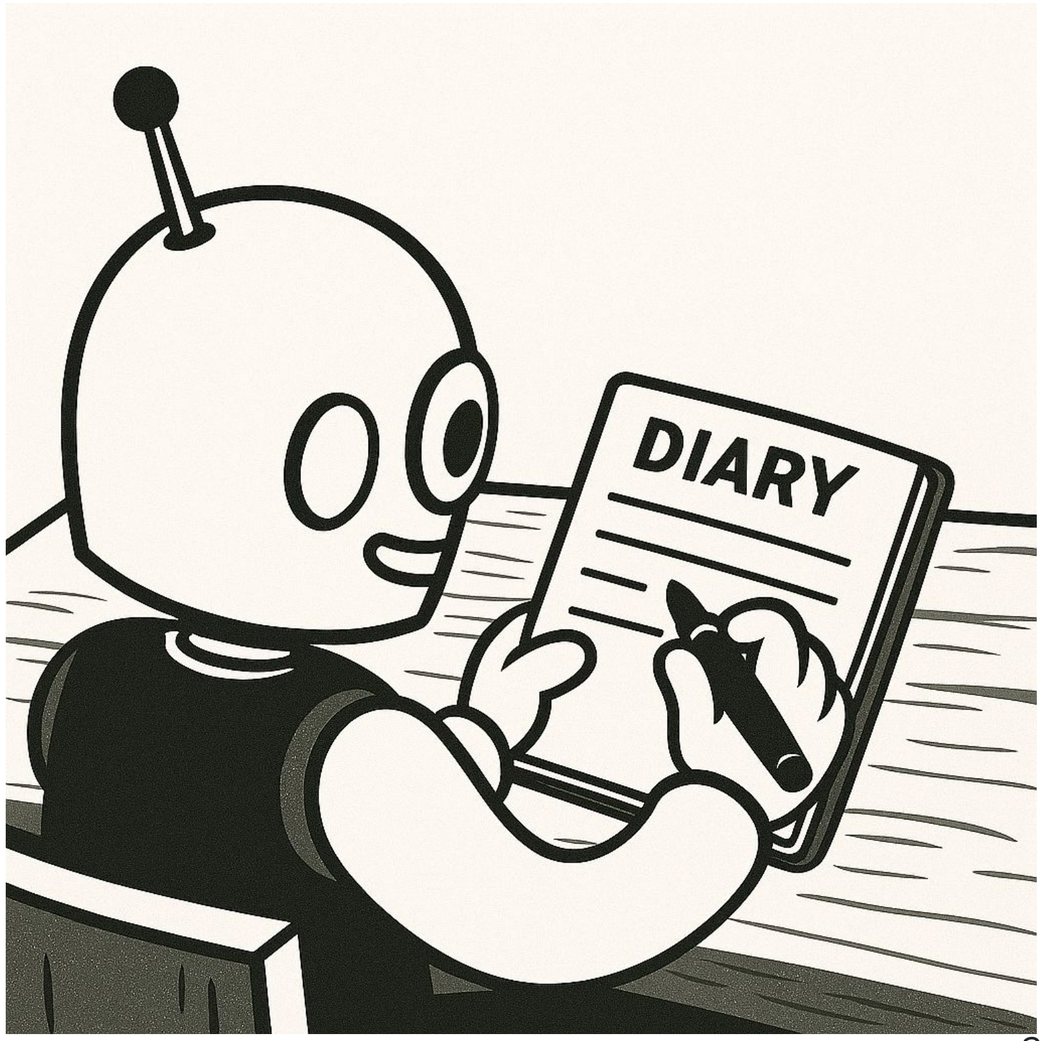


Value Proposition #3 - Built-In State Management

Workflows that remember and adapt

State Management Benefits

- Maintains state across entire workflow
- Multi-step coordination of complex processes
- Can rollback based on validation results
- Complete history of decisions and actions
- Pick up where you left off after interruption
- Support for processes that span hours/days



Value Proposition #4 - Flexible Integration

Your existing automation become an agent's superpowers

Tool Calling Explained

- Agents can invoke any function or API as a "tool"
- Your Python functions become agent capabilities
- Existing playbooks can be wrapped as tools
- APIs, scripts, databases - all accessible
- Dynamic tool selection based on need

Value Proposition #4 - Flexible Integration

```python
# Your existing automation becomes a tool
@tool
def create_firewall_rule(source: str, destination: str, action: str):
    """Create a security rule on the firewall."""
    # Your existing code here
    return deploy_to_firewall(source, destination, action)

# Agent can now use it
agent = create_agent(tools=[create_firewall_rule, get_rules, validate_config])

# Agent decides when and how to use tools based on conversation
result = agent.invoke("Block traffic from 10.0.0.0/8")
```

Deterministic + Autonomous: Best of Both Worlds

Deterministic Workflows:

- Predefined, explicit flow
- Validation gates and approval checkpoints
- Consistent, repeatable execution
- Full audit and compliance tracking
- Best for: Changes, deployments, compliance

Autonomous Workflows (ReAct Agents):

- Agent decides the path
- Dynamic tool selection
- Adaptive to conditions
- Self-correcting loops
- Best for: Troubleshooting, investigation, incident response

Deterministic + Autonomous: Best of Both Worlds

Use Case	Workflow Type	Why
Deploy configuration	Deterministic	Compliance and consistency required
Investigate security alert	Autonomous	Unknown path, requires reasoning
Compliance audit	Deterministic	Standard checks, predictable
Performance troubleshooting	Autonomous	Exploratory, adaptive analysis
Rollback procedure	Deterministic	Critical path, no improvisation

The Business Case Summary

ROI of AI Agents in Automation

Key Metrics:

- Time Reduction: 62% faster incident response
- Cost Savings: 20% reduction in operational costs
- Accuracy: 47% fewer configuration errors
- Coverage: Expand automation use cases by 3x
- Accessibility: 10x more teammates can use automation
- Availability: 24/7 intelligent response

Who are the AI Agent players?

Popular Open Source AI Agents Frameworks

Google ADK

Modular, model-agnostic agent framework

Crew AI

Collaborative, workflow driven agent orchestration

n8n

Visual, extensible workflow automation

LangGraph

Structured, stateful agent orchestration

Popular SaaS Solutions With AI Agent Frameworks

Zoom

Provide personalized help to enhance team's productivity

ServiceNow

Provides intelligent automation and workflow optimization

ChatGPT Enterprise

Secure, no-code enterprise agent creation

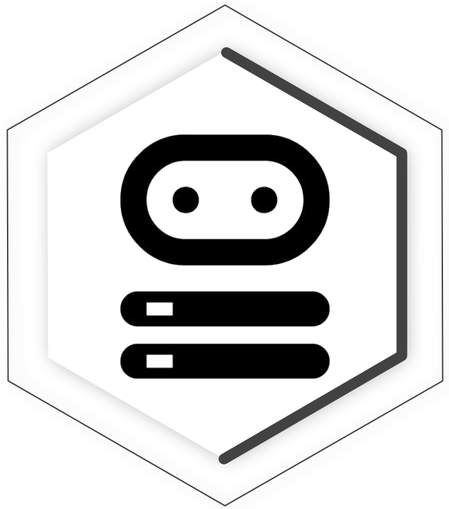
Copilot Studio

Integrated, low-code Microsoft agent platform

Popular Cloud Service Provider Agent Orchestration Solutions

Amazon Bedrock

Unified, scalable foundation model platform



Google Agent Engine

Integrated, generative agent runtime
A2A, LangGraph, CrewAI

Azure AI Foundry

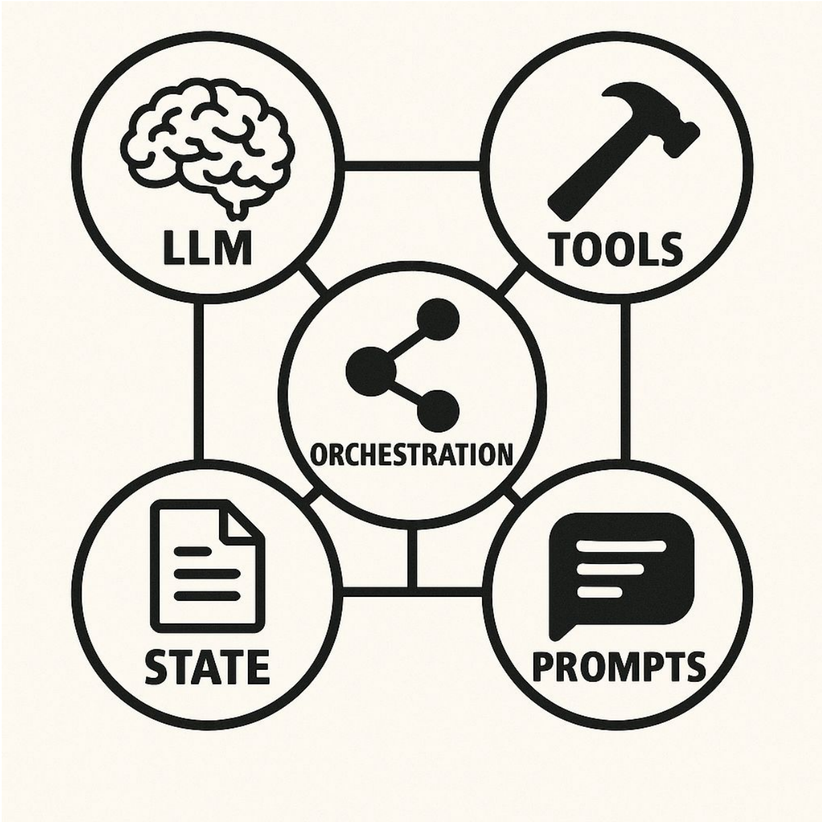
Comprehensive, enterprise AI development hub

Technical Implementation Details

Under the Hood: AI Agent Architecture

The components that make it work

1. **Large Language Model (LLM)** - The reasoning engine
2. **Tools** - Functions and APIs the agent can call
3. **State Management** - Persistent context and memory
4. **Orchestration Layer** - Workflow coordination
5. **Prompt Engineering** - Instructions and guardrails

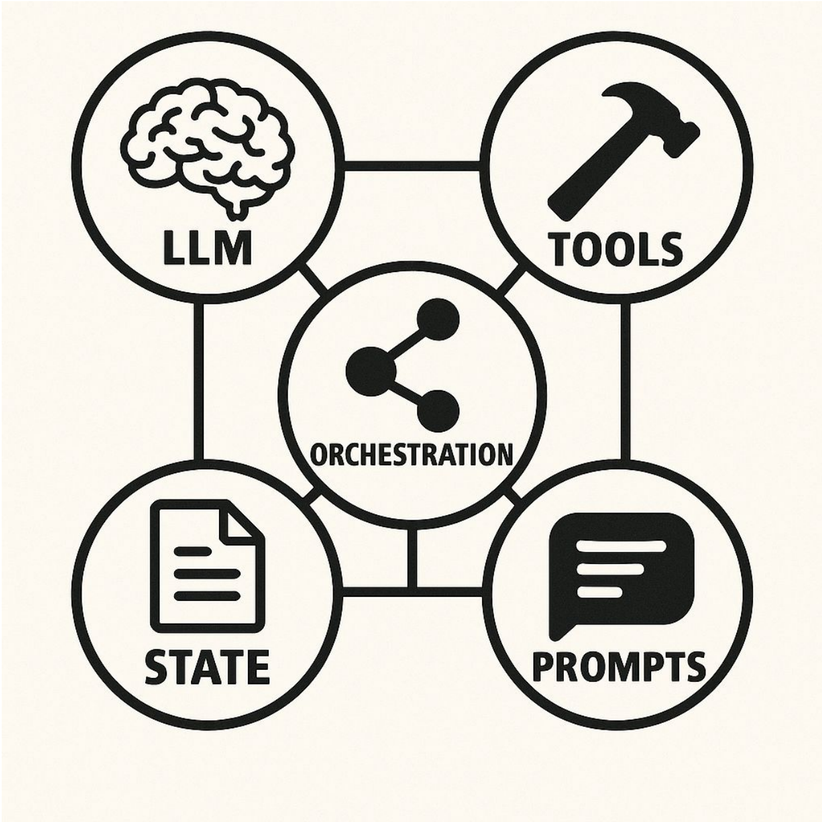


The LLM as Reasoning Engine

The Brain: How LLMs Enable Reasoning

What LLMs Bring:

- Natural language understanding
- Context analysis and interpretation
- Pattern recognition from training
- Structured output generation
- Chain-of-thought reasoning
- Decision making based on criteria



The LLM as Reasoning Engine

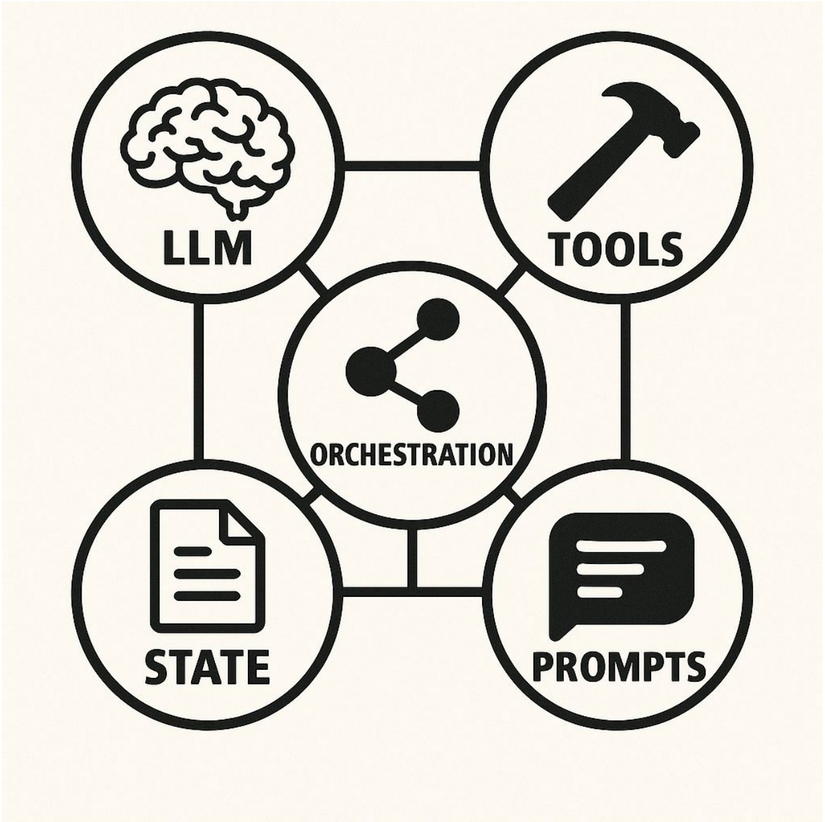
```python
# LLM receives context and tools
prompt = """
You are a network security agent. A user has requested:
"Block all traffic from 10.0.0.0/8"

Current firewall state: [retrieved context]
Available tools: [create_rule, get_rules, commit_config]

Analyze this request and determine:
1. Is this safe to execute?
2. Are there existing rules that conflict?
3. What specific steps are needed?
4. What should be validated?

Provide your reasoning and proposed action plan.
"""

# LLM responds with structured reasoning
response = llm.invoke(prompt)
```

Tool Calling Mechanism

Connecting reasoning to execution

Tool Calling Process:

1. Define tools with clear descriptions and schemas
2. LLM analyzes available tools
3. LLM selects appropriate tool based on task
4. LLM generates parameters from context
5. Framework executes the tool call
6. Result returned to LLM for analysis



Tool Calling Mechanism

```python
from langchain_core.tools import tool

@tool
def get_security_rules(firewall_name: str, zone: str = None) -> list:
    """
    Retrieve security rules from a PAN-OS firewall.

    Args:
        firewall_name: Name of the firewall device
        zone: Optional zone to filter rules

    Returns:
        List of security rule objects
    """
    # Your existing API integration code
    client = get_firewall_client(firewall_name)
    rules = client.security_rules.list(zone=zone)
    return [rule.dict() for rule in rules]

# Register tools with agent
tools = [get_security_rules]
```

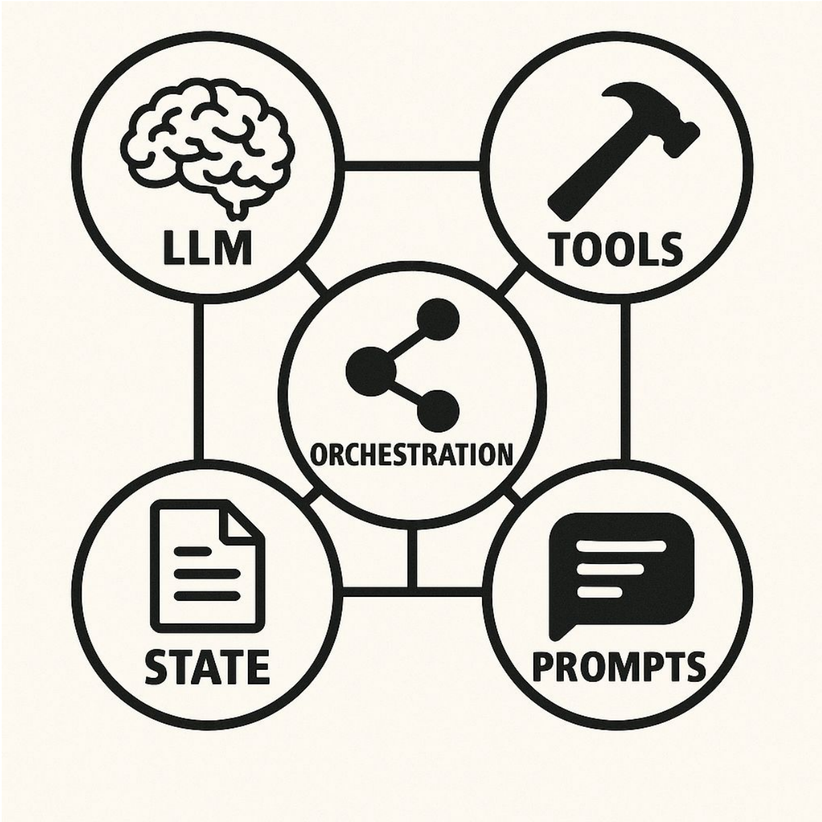
```text
User: "Show me rules blocking 10.0.0.0/8"

LLM reasoning:
1. User wants to see rules
2. Specific to a network range
3. Tool: get_security_rules is appropriate
4. Need firewall_name - should ask or use default
5. Can filter in results for the network

LLM action:
{
  "tool": "get_security_rules",
  "args": {
    "firewall_name": "primary-fw",
    "zone": null
  }
}

Result processed and presented to user
```

State Management in Practice

Maintaining context across workflow steps

State Management Concepts:

- **State Schema**: Structured definition of what to track
- **State Updates**: How nodes modify state
- **Persistence**: Saving state between sessions
- **State Inspection**: Debugging and observability
- **Checkpointing**: Save points for rollback



State Management in Practice

```python
from typing import TypedDict, Annotated, Sequence
from langgraph.graph import StateGraph, add_messages

# Define what we track throughout workflow
class AutomationState(TypedDict):
    """State for infrastructure automation workflow."""

    # Original request
    user_request: str
    intent: dict  # Parsed intent

    # Workflow tracking
    current_step: str
    steps_completed: Sequence[str]

    # Context data
    current_config: dict
    proposed_changes: dict
    validation_results: dict

    # Communication
    messages: Annotated[list, add_messages]

    # Status
    approved: bool
    deployed: bool
    errors: Sequence[str]
```

```python
# Nodes update state
def validate_change(state: AutomationState) -> AutomationState:
    """Validate proposed changes against policy."""
    validation_result = run_validation(state["proposed_changes"])

    return {
        "validation_results": validation_result,
        "current_step": "validation_complete",
        "steps_completed": state["steps_completed"] + ["validation"],
        "messages": [f"Validation complete: {validation_result['status']}"]
    }

# State is preserved across all workflow steps
workflow = StateGraph(AutomationState)
workflow.add_node("validate", validate_change)
```

State Inspection:

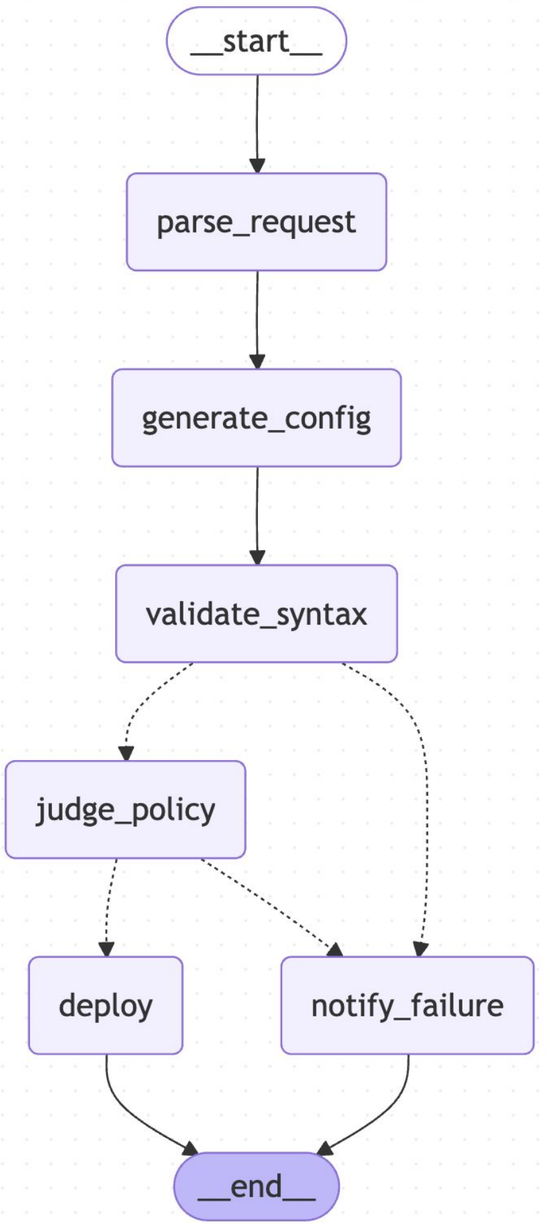
```python
# At any point, inspect current state
current_state = workflow.get_state()
print(f"Current step: {current_state['current_step']}")
print(f"Validation status: {current_state['validation_results']}")

# Can replay or resume from checkpoints
if current_state["errors"]:
    workflow.rollback_to_checkpoint("pre_deployment")
```

Deterministic Workflows

Predefined paths with intelligent validation

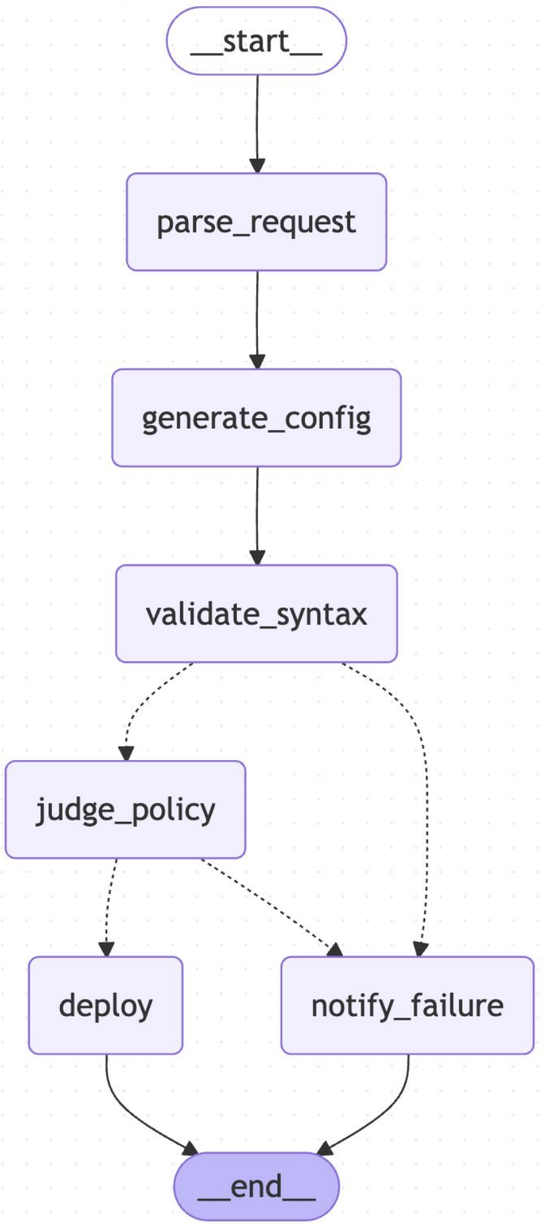
When to Use Deterministic:

- Standard operating procedures
- Compliance-required workflows
- Multi-step deployments
- Approval-based processes
- Any workflow with a known optimal path



Deterministic Workflows

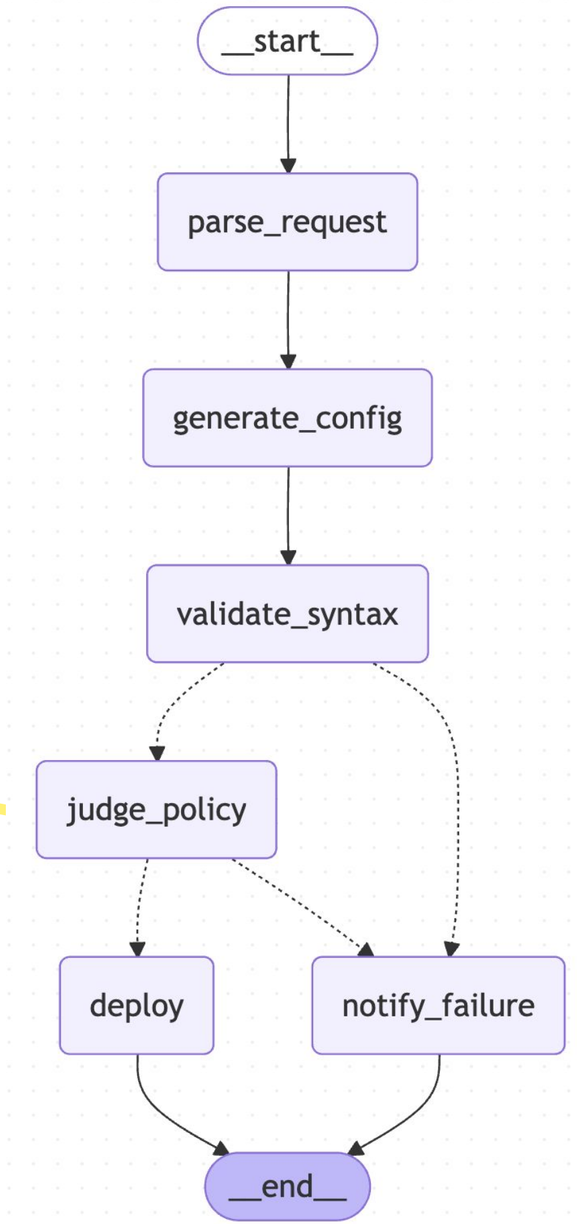
```python
from langgraph.graph import StateGraph, START, END

# Define workflow graph
workflow = StateGraph(AutomationState)

# Add nodes (steps in workflow)
workflow.add_node("parse_request", parse_user_intent)
workflow.add_node("generate_config", create_configuration)
workflow.add_node("validate_syntax", check_syntax)
workflow.add_node("judge_policy", llm_policy_validation)  # LLM validation
workflow.add_node("deploy", deploy_to_infrastructure)
workflow.add_node("notify_failure", send_error_notification)

# Define explicit flow
workflow.add_edge(START, "parse_request")
workflow.add_edge("parse_request", "generate_config")
workflow.add_edge("generate_config", "validate_syntax")

# Conditional routing based on validation
workflow.add_conditional_edges(
    "validate_syntax",
    lambda state: "judge_policy" if state["validation_results"]["syntax_ok"]
                  else "notify_failure"
)

workflow.add_conditional_edges(
    "judge_policy",
    lambda state: "deploy" if state["validation_results"]["policy_compliant"]
                  else "notify_failure"
)

workflow.add_edge("deploy", END)
workflow.add_edge("notify_failure", END)

# Compile and run
app = workflow.compile()
result = app.invoke({"user_request": "Block traffic from 10.0.0.0/8"})
```



Deterministic Workflows

Judge Node Example:

```python
def llm_policy_validation(state: AutomationState) -> AutomationState:
    """Use LLM to validate against security policies."""

    prompt = f"""
    Analyze this proposed firewall rule:
    {state['proposed_changes']}

    Security Policy Requirements:
    - No 'any-any' rules
    - All rules must specify application
    - Source must be specific network or object
    - Must have business justification

    Determine: Does this rule comply with policy?
    Return: {{"compliant": bool, "violations": [list], "explanation": str}}
    """

    result = llm.invoke(prompt)

    return {
        "validation_results": {
            **state["validation_results"],
            "policy_compliant": result["compliant"],
            "policy_check": result
        }
    }
```

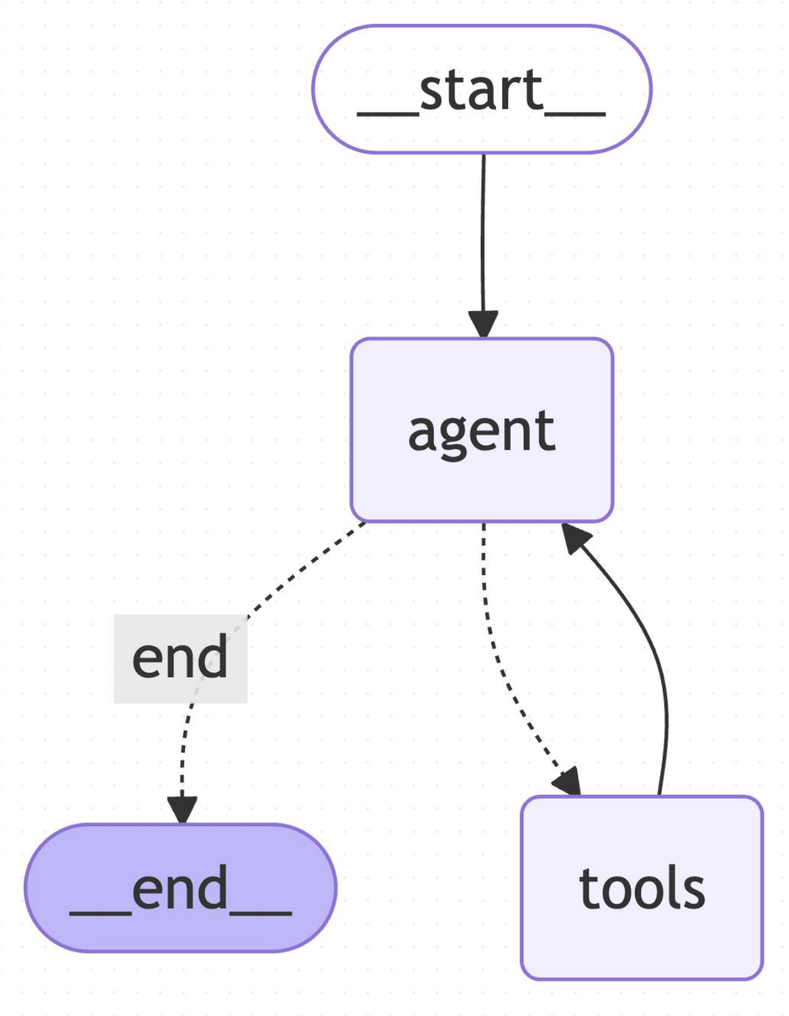


Autonomous Workflows with ReAct Agents

When the agent decides the path

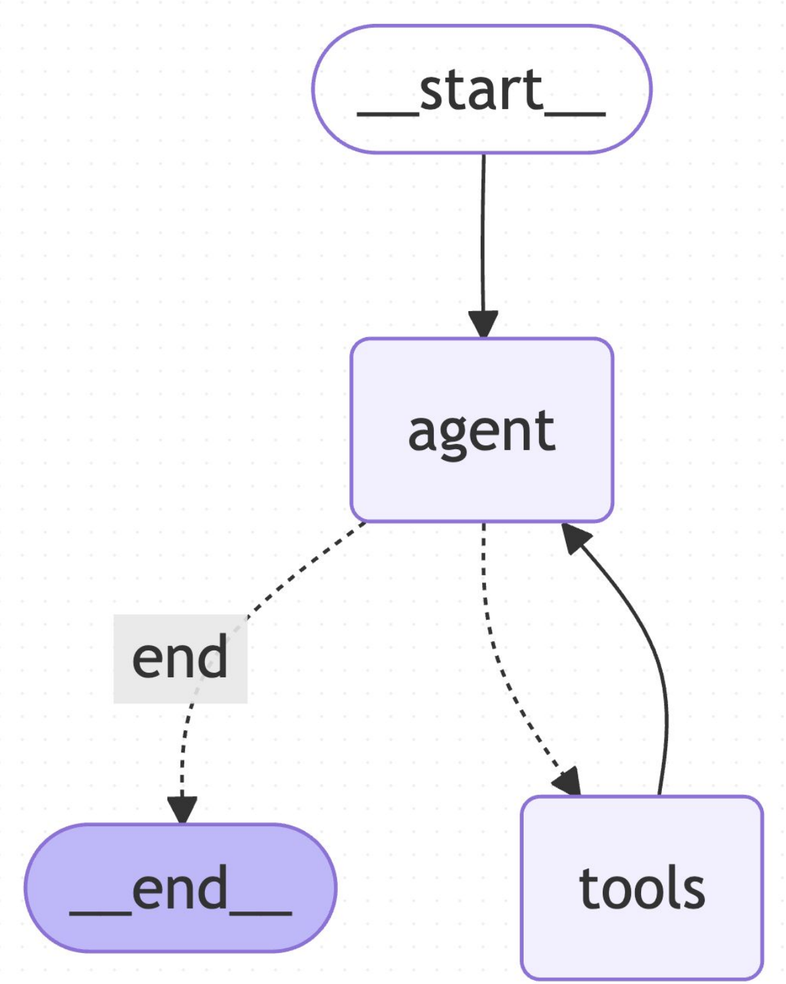
ReAct: Reasoning + Acting:

- Agent observes situation
- Reasons about next action
- Executes tool call
- Observes result
- Repeats until task complete



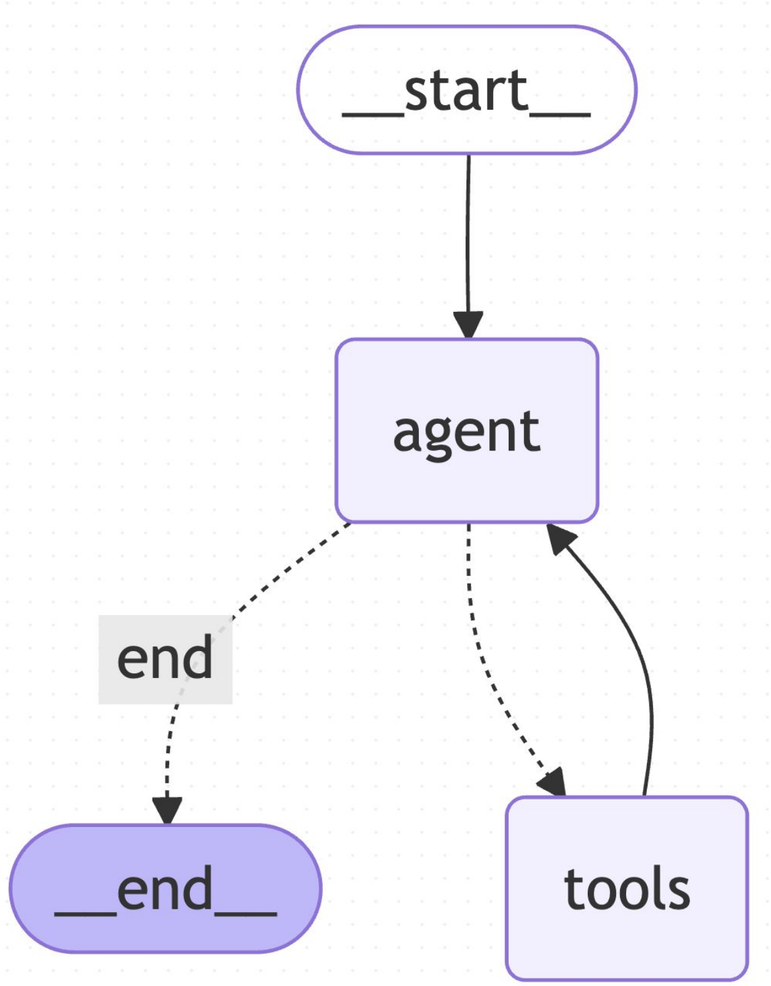
Autonomous Workflows with ReAct Agents

```python
from langgraph.prebuilt import create_react_agent

# Create ReAct agent with tools
agent = create_react_agent(
    model=llm,
    tools=[
        get_security_rules,
        get_threat_logs,
        analyze_traffic,
        create_security_rule,
        get_network_topology
    ],
    state_modifier="""
    You are an expert network security agent.

    When investigating issues:
    1. Gather relevant information first
    2. Analyze the data
    3. Consider multiple approaches
    4. Explain your reasoning
    5. Propose solutions with rationale

    Always validate before making changes.
    """
)

# Agent autonomously decides what to do
result = agent.invoke({
    "messages": [
        ("user", "Investigate suspicious traffic from IP 192.0.2.100")
    ]
})
```



Autonomous Workflows with ReAct Agents

```text
User: "Investigate suspicious traffic from IP 192.0.2.100"

Agent Thought: "I should first check what traffic this IP is generating"
Agent Action: get_threat_logs(source_ip="192.0.2.100")
Observation: [logs showing port scanning attempts]

Agent Thought: "This shows scanning behavior. Let me check existing rules"
Agent Action: get_security_rules(source="192.0.2.100")
Observation: [no blocking rules found]

Agent Thought: "No rules blocking this IP. Let me check network topology"
Agent Action: get_network_topology(ip="192.0.2.100")
Observation: [IP is external, hitting public-facing servers]

Agent Thought: "External IP scanning our servers - this is a threat"
Agent Action: create_security_rule(
    source="192.0.2.100",
    destination="any",
    action="deny",
    rule_name="block_scanner_192.0.2.100"
)
Observation: [rule created successfully]

Agent Response: "I've investigated the IP 192.0.2.100 and found it was
performing port scanning against our public servers. I've created a
blocking rule. Summary of findings: [detailed report]"
```

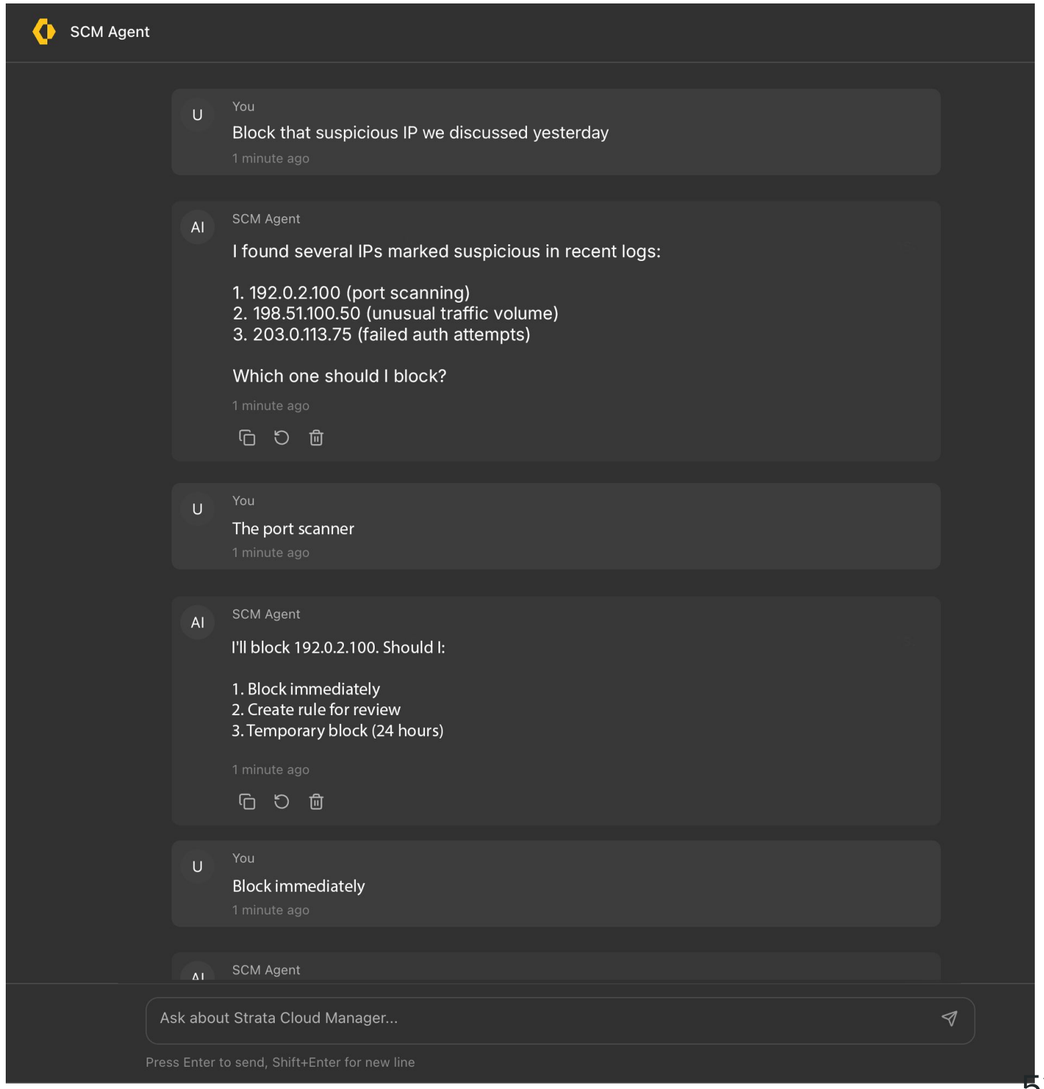


Natural Language Processing in Action

NLP: The Conversational Interface

NLP Capabilities:

- Intent parsing and understanding
- Entity extraction (IPs, networks, devices)
- Ambiguity resolution through clarification
- Context maintenance across conversation
- Natural response generation



Natural Language Processing in Action

NLP: The Conversational Interface

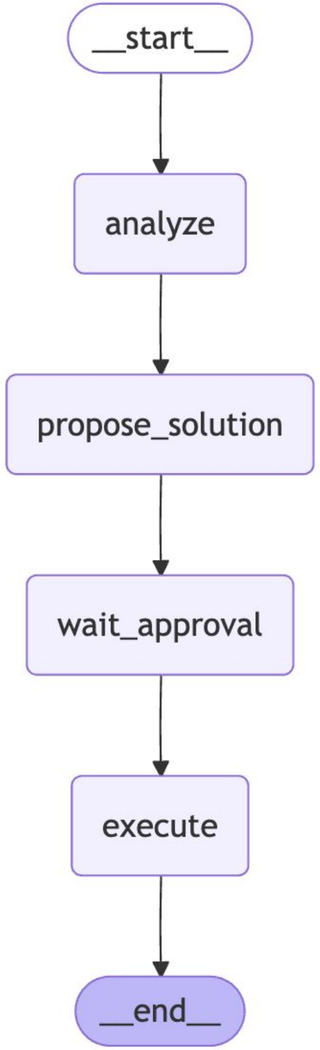
Implementation Pattern:

- Natural conversation, not rigid commands
- Agent asks for clarification when needed
- Maintains context across entire conversation
- Enables complex workflows through dialogue
- Accessible to anyone who can chat

```python
# Agent maintains conversation history in state
def chat_agent(state: ConversationState):
    # Previous messages provide context
    response = agent.invoke({
        "messages": state["messages"],
        "context": state["context"]  # What we've discussed
    })

    return {
        "messages": state["messages"] + [response],
        "context": update_context(state["context"], response)
    }
```

Human-in-the-Loop Patterns

Intelligent automation with human oversight

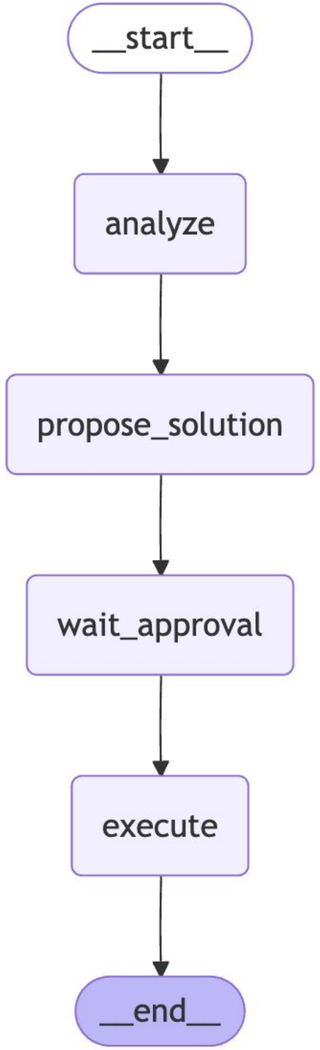


Requiring Human Approval:

- Workflow pauses at approval checkpoints
- Human reviews agent's proposed action
- Human can approve, modify, or reject
- Workflow resumes with human decision
- Full audit trail maintained

Human-in-the-Loop Patterns

Intelligent automation with human oversight

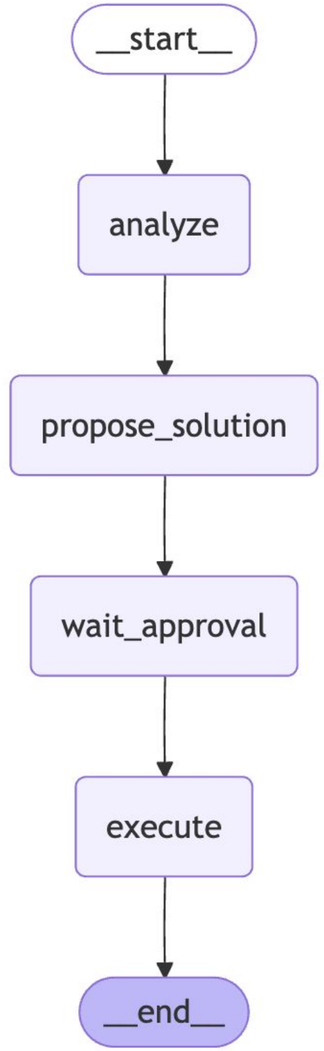


When to Require Human Approval:

- High-impact changes (production deployments)
- Security-sensitive operations
- Regulatory compliance requirements
- Learning phase (building trust)
- Ambiguous situations requiring judgment

Human-in-the-Loop Patterns

```python
from langgraph.checkpoint import MemorySaver

# Enable checkpointing for pause/resume
memory = MemorySaver()

# Define workflow with approval gate
workflow = StateGraph(AutomationState)
workflow.add_node("analyze", analyze_request)
workflow.add_node("propose_solution", create_plan)
workflow.add_node("wait_approval", interrupt("approval_required"))
workflow.add_node("execute", deploy_changes)

workflow.add_edge("analyze", "propose_solution")
workflow.add_edge("propose_solution", "wait_approval")
workflow.add_edge("wait_approval", "execute")

# Compile with checkpointing
app = workflow.compile(checkpointer=memory)

# Run until approval needed
config = {"configurable": {"thread_id": "change-123"}}
result = app.invoke(initial_state, config)

# Human reviews and approves
app.update_state(
    config,
    {"approved": True, "approval_note": "Approved by security team"}
)

# Resume execution
final_result = app.invoke(None, config)
```



Integrating AI Agents Into Your Existing Automation

Practical Integration: Adding AI to Your Stack

Python Library Integration

```python
# Use in existing scripts
from langgraph.prebuilt import create_react_agent

agent = create_react_agent(llm, tools)
result = agent.invoke({"messages": [("user", request)]})
```

Practical Integration: Adding AI to Your Stack

REST API

```python
# Expose as FastAPI service
from fastapi import FastAPI

app = FastAPI()

@app.post("/api/security/block-ip")
async def block_ip(request: BlockIPRequest):
    result = security_agent.invoke({
        "messages": [("user", f"Block IP {request.ip}")]
    })
    return {"status": "success", "details": result}
```

Practical Integration: Adding AI to Your Stack

CLI Tool

```python
# Command-line interface
$ network-agent "investigate traffic spike on FW-01"
$ network-agent --autonomous "respond to security alert SA-12345"
```

Practical Integration: Adding AI to Your Stack

Chat Platform Integration

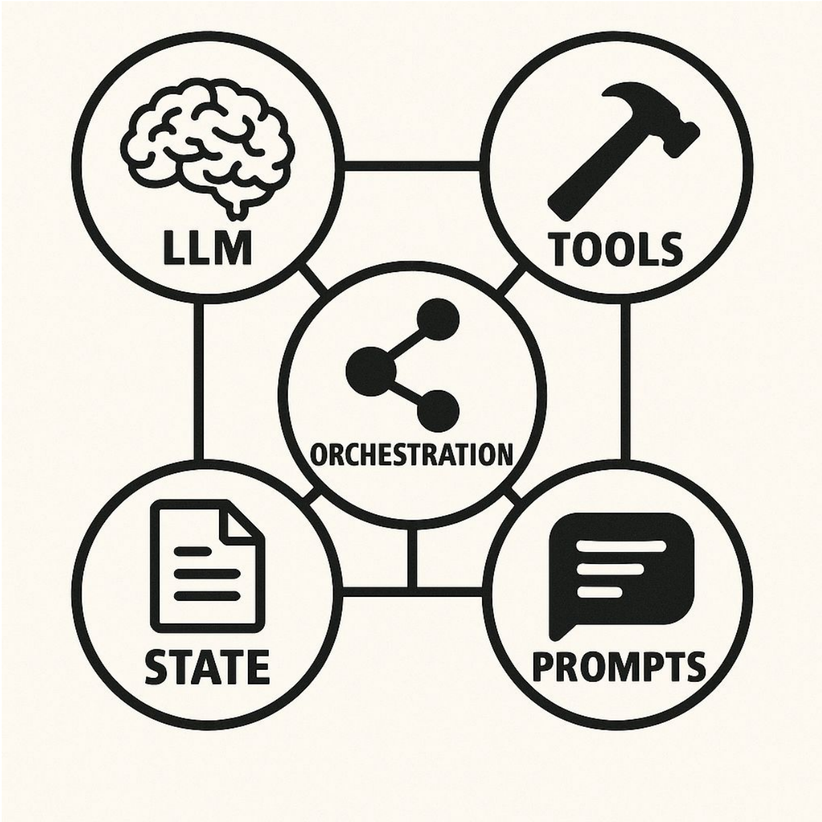
```python
# Slack bot
@slack_app.event("message")
def handle_message(event):
    response = agent.invoke({"messages": [("user", event["text"])]})
    slack_client.chat_postMessage(channel=event["channel"], text=response)
```

Practical Integration: Adding AI to Your Stack

Gradual Adoption Path

Phase 1: Pilot

- Single use case (e.g., security investigation)
- ReAct agent with limited tools
- Human approval on all actions
- Small team validation

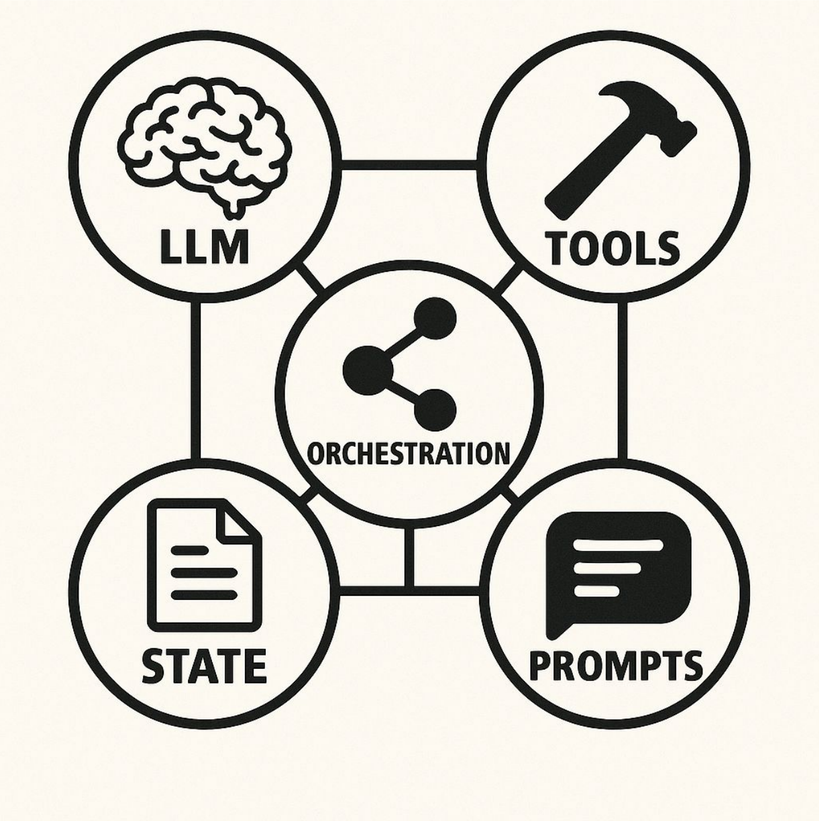


Practical Integration: Adding AI to Your Stack

Gradual Adoption Path

Phase 2: Expand

- Add more use cases
- Deterministic workflows for standard ops
- Selective approval gates
- Broader team access

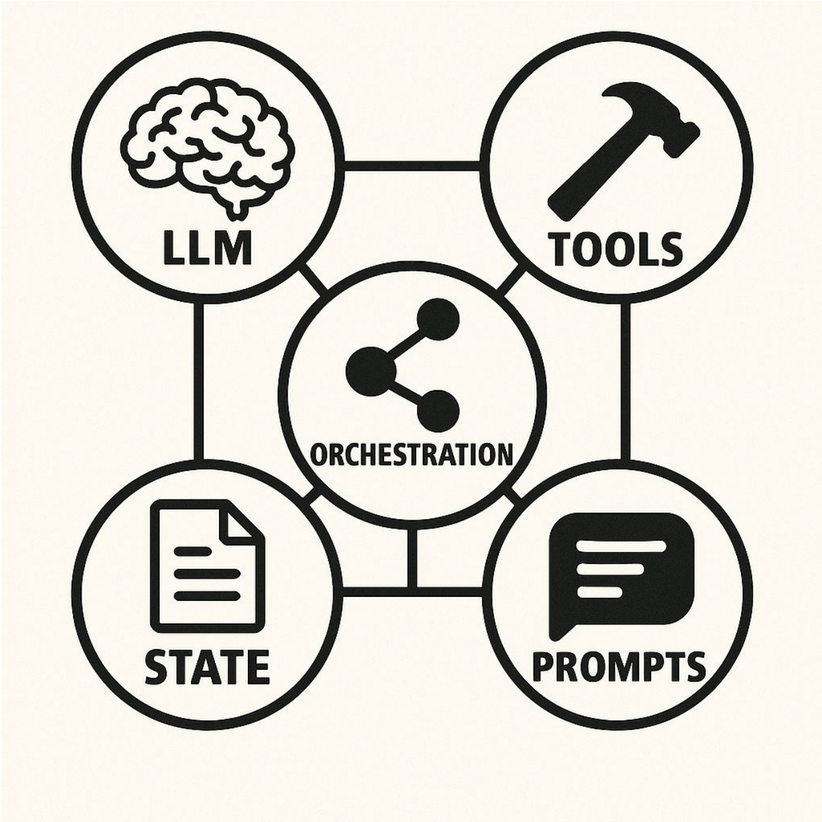


Practical Integration: Adding AI to Your Stack

Gradual Adoption Path

Phase 3: Production

- Core workflows automated
- Mix of autonomous and deterministic
- Service-oriented architecture
- Organization-wide deployment



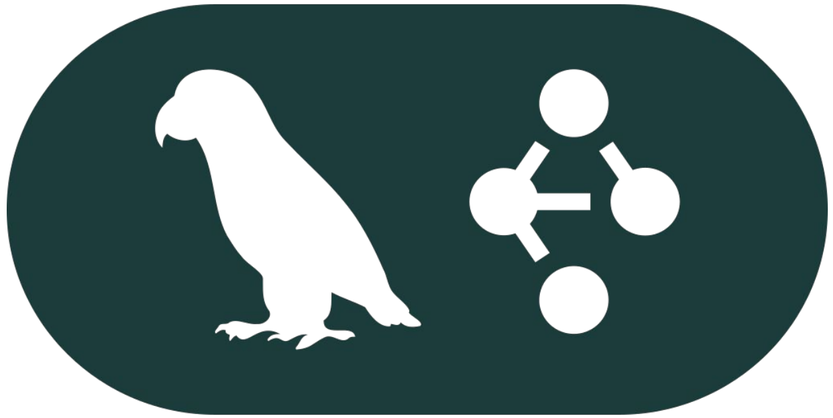
LangGraph Introduction

LangGraph: Bringing It All Together

The framework for production AI agents

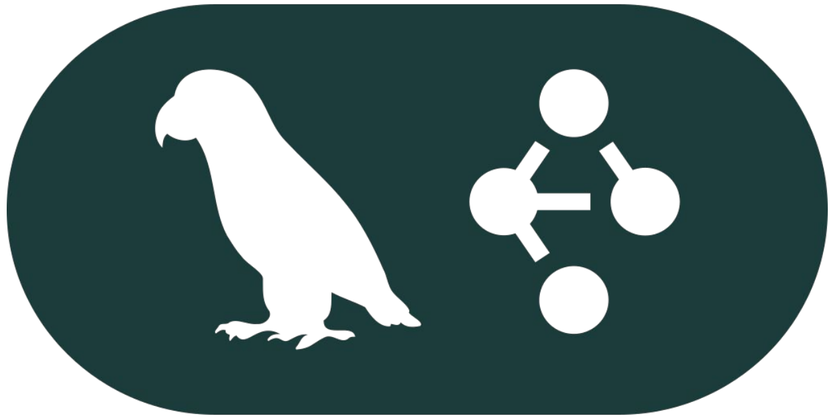
Why LangGraph:

- Built specifically for agent workflows
- Production-grade state management
- Graph-based orchestration
- Built-in persistence and checkpointing
- Human-in-the-loop support
- LangSmith integration for observability
- Active development and community



LangGraph: Bringing It All Together

The framework for production AI agents

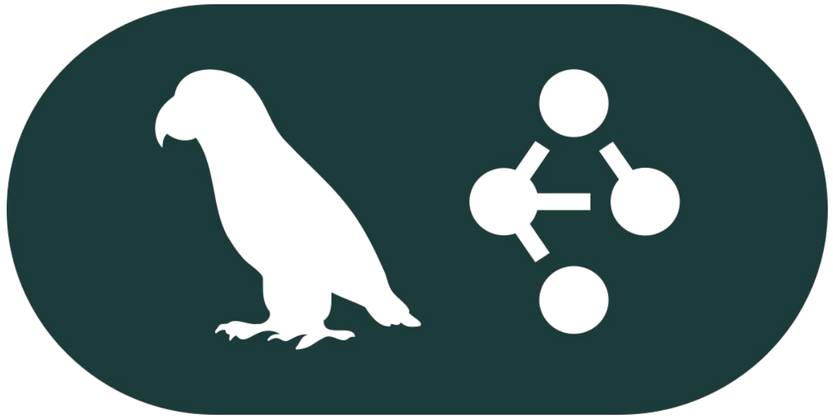


Key Features:

- **StateGraph**: Build deterministic workflows
- **create_agent**: Quick autonomous agent setup
- **Checkpointing**: Pause/resume workflows
- **Subgraphs**: Modular workflow composition
- **Streaming**: Real-time output
- **Async support**: High-performance execution

LangGraph: Bringing It All Together

The framework for production AI agents



Ecosystem:

- **LangChain:** Tool library and integrations
- **LangSmith:** Observability and tracing
- **LangGraph Studio:** Visual workflow builder
- **LangGraph Cloud:** Managed deployment

LangGraph: Bringing It All Together

Getting Started

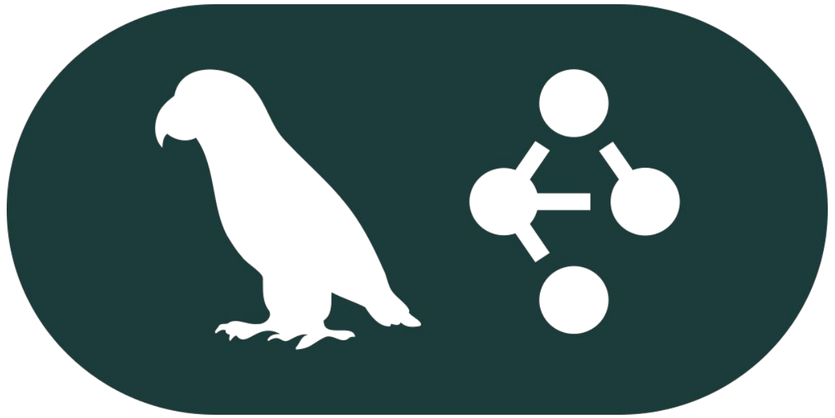
```python
# Install
pip install langgraph langchain-anthropic

# Simple agent in ~10 lines
from langchain_anthropic import ChatAnthropic
from langgraph.prebuilt import create_react_agent

llm = ChatAnthropic(model="claude-3-5-sonnet-20241022")

agent = create_react_agent(
    llm,
    tools=[my_tool_1, my_tool_2]
)

result = agent.invoke({
    "messages": [("user", "your request here")]
})
```

Key Takeaways

Key Takeaways: AI Agents for Automation

Problems Are Real and Costly:

- High barrier to coding prevents adoption
- Lack of accessibility creates bottlenecks
- Missing reasoning and state management



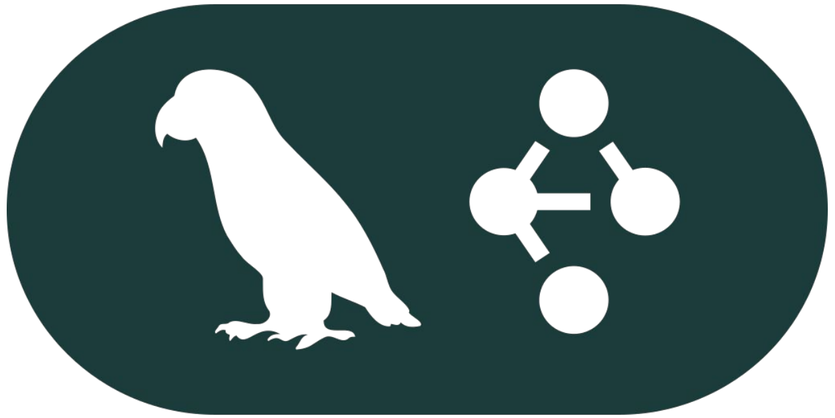
AI Agents Provide Solutions:

- Natural language democratizes access
- Intelligent reasoning enables adaptation
- Built-in state management for complex workflows
- Flexible tool integration leverages existing work

Key Takeaways: AI Agents for Automation

Two Modes for Different Needs:

- Deterministic workflows for standard operations
- Autonomous agents for exploratory tasks
- Choose based on use case requirements



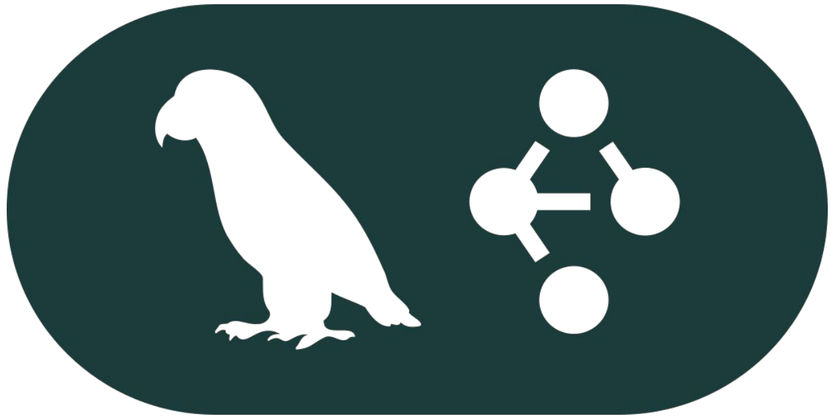
Practical Integration Path:

- Start small with pilot use case
- Wrap existing automation as tools
- Gradual adoption reduces risk
- Multiple integration patterns available

Key Takeaways: AI Agents for Automation

Production-Ready Today:

- LangGraph provides robust framework
- Real companies using in production
- Quantifiable ROI and business value
- Lower costs, faster response, fewer errors



Live Demo Time

Building A Secure AI Agent Architecture

Job Risks and Security

The Job Market Reality Check

The Numbers Don't Lie:

- Bureau of Labor Statistics revised 2024-2025 job growth down by **900,000 jobs**
- These positions were counted but *never actually created*
- **7.4 million Americans** actively searching for work (4.3% unemployment rate)
- Job market experiencing unprecedented tightening amid economic uncertainty

***Bureau of Labor Statistics, March 2025 revision; Official unemployment rate Q3 2025*

The Profit Paradox

Corporate Health ≠ Job Security:

- Markets Soaring
  - **S&P 500**: **+35%** rally since spring 2025
  - **Dow Jones**: First-ever close above **47,000 points** (October 2025)
  - **Corporate profit margins**: **13%** (record levels, Q3 2025)
- Jobs Plummeting
  - **1 million jobs cut** in 2025 (highest since 2020)
  - **1.1 million layoffs** announced year-to-date through October
  - **+65% year-over-year** increase in layoffs

***CNN Business (Dow milestone); Bank of America Research (profit margins); CBS News (2025 job cuts); Challenger, Gray & Christmas (layoff tracking)*

AI Isn't Coming, It's Already Here

The Scale of AI-Driven Displacement:

- Current Impact (2025):
  - **77,999 tech job losses** directly attributed to AI/automation
  - **~500 people/day** losing jobs to AI (H1 2025)
  - Major tech companies (Amazon, Microsoft) leading cuts
- Projected Impact (by 2030)
  - **92 million jobs displaced** globally by AI/automation
  - **300 million full-time jobs** affected worldwide (Goldman Sachs)
  - **30% of work hours** in US could be automated (McKinsey)

***World Economic Forum Future of Jobs Report 2025; Goldman Sachs AI impact analysis; McKinsey generative AI research; Bloomberg automation task analysis*

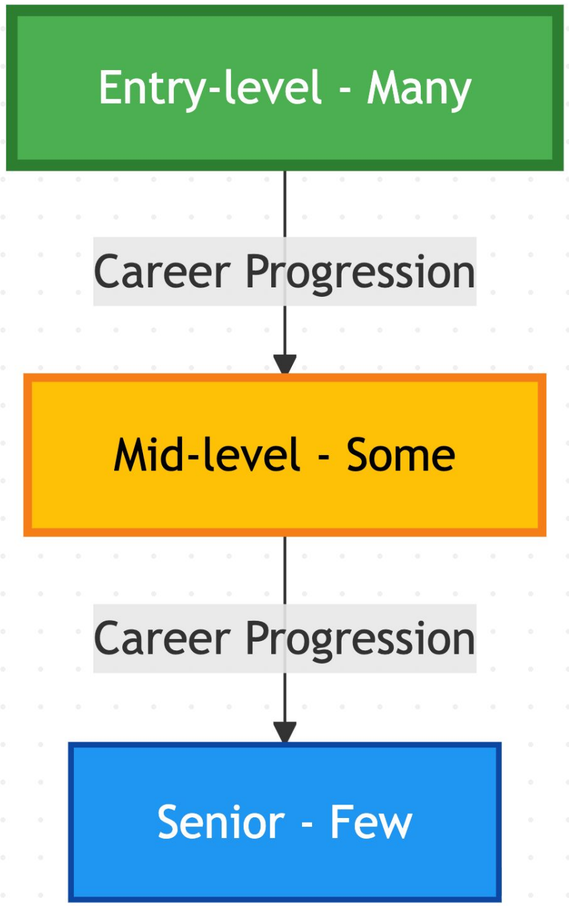
Network Automation's Vulnerability

Infrastructure Roles Are Not Immune:

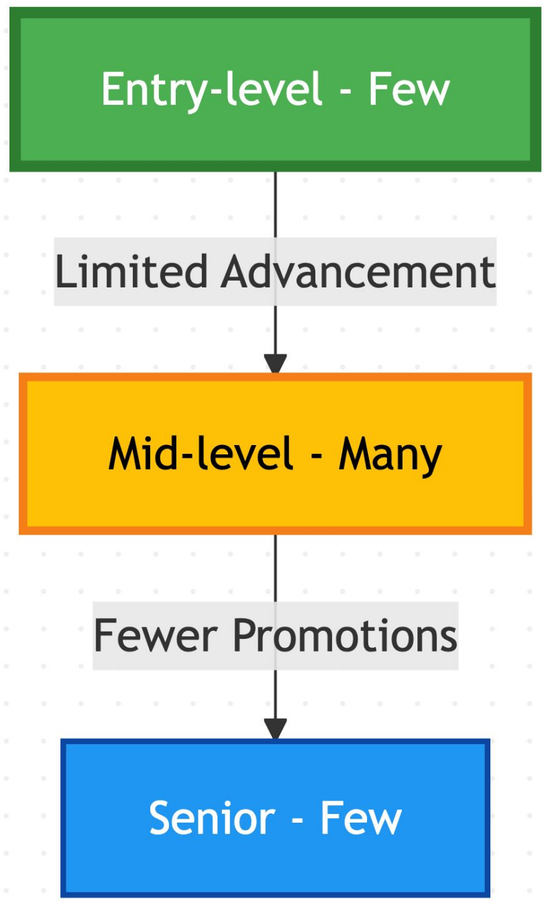
- Traditional Network Engineering Tasks at Risk:
  - Configuration management → AI-driven intent-based systems
  - Troubleshooting → ML-powered root cause analysis
  - Security policy creation → Generative AI policy generation
  - Capacity planning → Predictive AI analytics
  - Documentation → Auto-generated from network state
- The Automation Paradox:
  - We've been automating networks for years, but now AI is automating the automation
  - Entry-level network positions vanishing
  - Mid-career roles requiring AI fluency to survive

The Career Ladder Is Breaking

Traditional Career Bottleneck:



New Career Bottleneck:



The Career Ladder Is Breaking

The Numbers:

- *61% of "entry-level" jobs* now require 2-3 years experience (SHRM Research)
- Average 2025 graduate carries *$39,000 in student debt* (Federal Reserve)
- *Experience paradox*: Can't get job without experience, can't get experience without job

Impact on Network Engineering:

- Junior network engineer roles disappearing
- Help desk → Network admin pipeline broken
- Fewer pathways to gain hands-on experience
- Industry certifications alone no longer sufficient

The Opportunity Equation

- Jobs Displaced vs. Created
  - **92 million jobs displaced** by AI/automation
  - **170 million new roles created** in emerging sectors
  - Net gain of **+78 million jobs globally** (World Economic Forum)
- Where New Jobs Emerge:
  - AI development and machine learning engineering
  - AI agent architecture and workflow design
  - Digital infrastructure and cloud systems
  - Green technology and sustainable infrastructure
  - Cybersecurity and AI safety

The Opportunity Equation

- The Skills Gap Challenge
  - **59% of workers** will require upskilling/reskilling by 2030 (National University)
  - Workers with in-demand certifications see **+42% interview callback rate** (Coursera)
  - **12-14% of workers** need to transition to entirely new occupations (McKinsey)
- Critical Insight
  - The jobs of the future exist today, but the workers of today may not yet have the skills for those jobs

***World Economic Forum Future of Jobs Report 2025; National University research; Coursera Skills Report; McKinsey occupational transition analysis*

Domain Expertise + AI Skills = Defensible Career

- Domain Knowledge is Irreplaceable
  - Understanding of network protocols, security models, infrastructure constraints
  - Years of troubleshooting experience and architectural patterns
  - Context AI alone cannot replicate: regulatory requirements, business constraints, vendor ecosystems
- Automation Fluency as Foundation
  - Already comfortable with Infrastructure-as-Code
  - Understand APIs, integrations, system orchestration
  - Experience with state management and declarative configuration
- AI Agents Need Domain-Specific Design
  - Generic AI can't safely modify production networks
  - Requires guardrails, validation, rollback mechanisms
  - Who better to design network-aware AI agents than network automation engineers?

The Path Forward

- The Reality
  - Job market is transforming (we've seen the data)
  - AI is displacing traditional roles (including network engineering)
  - New opportunities exist (170M jobs created)
  - Skills gap is the barrier (59% need reskilling)
  - You have domain advantage (network expertise + automation)
- What You'll Build Today
  - AI agents that understand state and context
  - Workflows with conditional logic and error handling
  - Integration with real network automation platforms
  - Foundation to build domain-specific solutions

The Path Forward

- What You'll Take Away
  - LangGraph framework proficiency
  - Patterns for production AI systems
  - Portfolio project (Palo Alto SCM automation)
  - Career positioning as AI agent developer
- Your Action Plan
  - **Today**: Complete workshop, build working AI agent
  - **This week**: Apply patterns to your own network automation challenges
  - **This month**: Build 2-3 portfolio projects showcasing AI agent skills
  - **This quarter**: Position yourself for AI-augmented network engineering roles
  - **This year**: Be on the creation side of the 78M job growth equation

Demos And Lab Workshop

Thank You