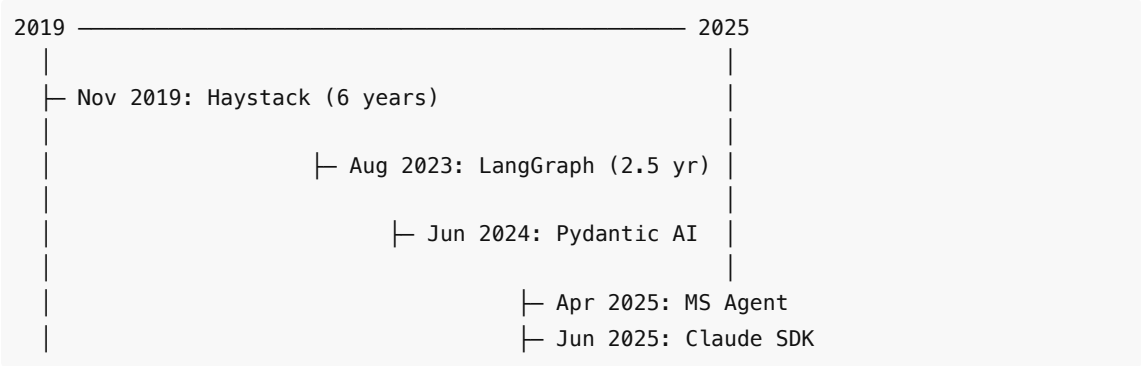
AI Agent Framework Comparison

Evaluating Frameworks for Multi-Agent Systems

Frameworks Evaluated

Framework	Release	Stars	Maintainer
Haystack	Nov 2019	24k	deepset
LangGraph	Aug 2023	24k	LangChain
Pydantic AI	Jun 2024	14.5k	Pydantic
MS Agent Framework	Apr 2025	6.8k	Microsoft
Claude Agent SDK	Jun 2025	4.4k	Anthropic

Maturity Timeline



Trade-off: Maturity vs. Modern Design

Philosophy: Haystack

"Production-Ready Pipelines"

Mental Model: Your app is a pipeline of swappable components

Key Principles:

- Composability over monoliths
- Swap providers without rewriting
- Control your data flow
- Reliability over complexity

Think: "I'm building from composable pieces. Each piece does one thing well."

Best for: RAG, document processing, production stability

Philosophy: LangGraph

"Agents as State Machines"

Mental Model: Your agent is a graph of states and transitions

Key Principles:

- Explicit over implicit
- Durable execution with checkpoints
- Human-in-the-loop built-in
- Observability first

Think: "What are my states? What triggers transitions? Where might I need human intervention?"

Best for: Complex workflows, explicit control flow

Philosophy: Pydantic AI

"Type Safety as Foundation"

Mental Model: Agents are typed, reusable components (like FastAPI routers)

Key Principles:

- Types are contracts
- Agents are reusable
- Explicit dependencies
- Observability is essential

Think: "What types go in? What types come out? If my types are right, my agent is probably right."

Best for: Regulated industries, structured outputs

Philosophy: MS Agent Framework

"Agents When Necessary"

Mental Model: Use functions when you can, agents when you need autonomy

Key Principles:

- Pragmatism first - "If you can write a function, do that instead"
- Agents + Workflows continuum
- Enterprise-grade foundations
- Multi-language parity

Think: "Does this need autonomous decision-making? If yes, agent. If no, function."

Best for: Azure environments, .NET shops

Philosophy: Claude Agent SDK

"Claude Code as Infrastructure"

Mental Model: Claude Code is your execution environment

Key Principles:

- Native capabilities (Read, Write, Bash)
- Hooks for control
- Permission-based security
- Event-driven architecture

Think: "I provide tools via MCP servers. I use hooks to enforce policies."

Best for: Claude-native applications

Tool Definition Comparison

Framework	Approach	Verbosity
Pydantic AI	`@agent.tool` decorator + docstring	Low
LangGraph	`@tool` decorator from LangChain	Low
Haystack	JSON schema or `@tool` decorator	Medium
MS Agent	Plain functions	Low
Claude SDK	`@tool` with MCP server	Medium

Tool Definition: Pydantic AI

```
from pydantic_ai import Agent, RunContext

agent = Agent('openai:gpt-4')

@agent.tool
def get_weather(ctx: RunContext, city: str) -> str:
    """Get weather for a city."""  # Docstring = description
    return f"Weather in {city}: sunny"
```

Cleanest syntax - docstring becomes tool description automatically

---

Tool Definition: LangGraph

```
from langchain_core.tools import tool

@tool
def get_weather(city: str) -> str:
    """Get weather for a given city."""
    return f"Weather in {city}: sunny"
```

```
# Bind to model
llm_with_tools = llm.bind_tools([get_weather])
```

Tool Definition: Haystack

```
# Verbose JSON schema approach
tools = [{
    "type": "function",
    "function": {
        "name": "get_weather",
        "description": "Get weather for a city",
        "parameters": {
            "type": "object",
            "properties": {
                "city": {"type": "string"}
            },
            "required": ["city"]
        }
    }
}]
```

Or use `@tool` decorator (simpler)

Observability Comparison

Framework	Primary Tool	Setup	Standard
Haystack	Langfuse, Arize, W&B	Medium	Varies
LangGraph	LangSmith	2 env vars	Proprietary
Pydantic AI	Logfire	2 lines	OpenTelemetry
MS Agent	Azure Monitor	Built-in	OpenTelemetry
Claude SDK	DIY via hooks	High	Custom

Observability: What You Can See

Aspect	Haystack	LangGraph	Pydantic AI	MS Agent
Agent runs	✓	✓	✓	✓
Tool calls	✓	✓	✓	✓
Token usage	Via integration	✓	✓	✓
Latency	Via integration	✓	✓	✓
Multi-agent	✓	✓	✓	✓

Self-hosted	Yes	No	Enterprise	Yes

Failure Handling Comparison

Framework	Recovery Mechanism
Haystack	Pipeline breakpoints, component isolation
LangGraph	Checkpoint-based resume (sync/async/exit modes)
Pydantic AI	Validation-driven reflection loops
MS Agent	Middleware stack for retry/logging
Claude SDK	`PostToolUseFailure` hooks + session resume

LangGraph: Checkpoint Recovery

```
from langgraph.checkpoint.memory import InMemorySaver

checkpointer = InMemorySaver()
graph = workflow.compile(checkpointer=checkpointer)

# Run with thread ID
config = {"configurable": {"thread_id": "my-thread"}}
result = graph.invoke(input_data, config)

# Resume from failure - same thread_id + None
resumed = graph.invoke(None, config)
```

Key: Ensure side effects are idempotent

Pydantic AI: Self-Correcting Validation

```
class WeatherResponse(BaseModel):
    temperature: float
    conditions: str

agent = Agent('openai:gpt-4', result_type=WeatherResponse)

# If LLM returns invalid data:
# 1. Pydantic catches validation error
# 2. Error sent back to LLM
# 3. LLM retries with corrected output
result = await agent.run("What's the weather?")
```

Automatic reflection - no manual retry logic needed

Complex Workflows

Pattern	Best Framework
Simple linear	Pydantic AI
Complex branching	LangGraph or MS Agent
Long-running with recovery	LangGraph with checkpoints
RAG pipelines	Haystack
Human-in-the-loop	LangGraph (first-class)
Claude Code integration	Claude Agent SDK

Real-World Testing: Our Use Case

Scenario: Australian Insurance CAT Event Verification

Agent 1 - Weather Verification:

- Geocode location (Nominatim API)
- Fetch BOM weather observations

Agent 2 - Claims Eligibility:

- Apply business rules (pure LLM reasoning)
- APPROVED / REVIEW / DENIED

Test: Brisbane, QLD, 4000 on 2025-03-07

Implementation Results

Framework	Lines of Code	Notes
Pydantic AI	~90	Cleanest, most Pythonic
Haystack	~110	Tool schema verbose
LangGraph	~120	Explicit state setup
AutoGen 0.4+	~130	Required workarounds

MS Agent Framework and Claude SDK not tested (released after)

Testing Findings: Pydantic AI

Winner for developer experience

- Cleanest code of all frameworks
- Pydantic models = automatic validation
- `@agent.tool` with `RunContext` for DI
- Async-native throughout

- Typed `.output` with IDE autocomplete

```
weather_result = await weather_agent.run(prompt, deps=deps)
print(weather_result.output.temperature)  # Typed!
```

Testing Findings: LangGraph

Good for explicit control, more boilerplate

- `StateGraph` + `TypedDict` pattern clear
- Manual tool call loop handling required
- Good for visualizing complex flows

Gotcha: You manage the tool iteration loop yourself

```
while response.tool_calls:
    # Process each tool call manually
    for tool_call in response.tool_calls:
        result = tool.invoke(tool_call["args"])
```

Testing Findings: AutoGen 0.4+

Multi-step tool calling challenges

> *"Some models struggle with multi-step tool calling in RoundRobinGroupChat"*

Workaround required:

```
# Had to call tools directly instead of
# letting agent orchestrate
geo_result = await geocode_location(...)
weather_result = await get_bom_weather(...)

# Then pass to eligibility agent
eligibility_result = await eligibility_agent.run(...)
```

Note: May be improved in new MS Agent Framework

LLM Learnability Benchmark

Question: How easily can an LLM produce working code with each framework?

Method:

- DeepSeek V3 (Dec 2024 cutoff)
- "Turns to Working Code" with error feedback
- Up to 10 iterations per attempt
- 48 total trials

Source: github.com/srepho/fwork-learnability

Learnability Results

Framework	Success Rate	Avg Turns	First-Attempt
Pydantic AI	**92%**	3.4	25%
Direct API	83%	2.4	8%
LangGraph	83%	4.7	**0%**
Haystack	75%	3.0	25%

Pydantic AI: Highest success, good first-attempt rate **LangGraph:** Zero first-attempts, most iterations needed

---

Learnability: Key Insights

1. Pydantic AI is most learnable

- Highest success rate (92%)
- LLMs pick it up fastest

2. LangGraph has steepest learning curve

- Zero first-attempt successes
- 4.7 average turns (highest)

3. Counterintuitive docs finding

- Minimal docs sometimes beat full docs
- Full docs captured marketing, not code

Error Patterns (17% failure rate)

Error Type	Count	Example
Syntax errors	40	Unterminated strings
API hallucinations	26	Fabricated `result_type` param
Logic errors	15	Wrong output, valid code

Implication: Even LLMs hallucinate API parameters

Production Readiness

Factor	Haystack	LangGraph	Pydantic AI	MS Agent	Claude SDK
Maturity	6 years	2.5 years	1.5 years	9 months	7 months
Enterprise	deepset	Klarna, Replit	Growing	Azure	Growing
Breaking Changes	Stable	Evolving	Evolving	Very new	Very new

Risk Level	Low	Low-Med	Medium	Med-High	Med-High

Framework Overhead

Factor	Haystack	LangGraph	Pydantic AI	MS Agent	Claude SDK
Dependencies	Medium	Medium	Light	Light	Light
Python	3.9+	3.9+	3.9+	3.10+	3.10+
Startup	Medium	Medium	Fast	Fast	Fast
Model Lock-in	None	None	None	Azure-opt	Claude-only

Note: LLM API costs dominate - framework overhead is minimal

Decision Framework

```
What's your primary need?
          │
     ┌────┼────┐
     ▼    ▼    ▼
  RAG?  State  Type
        Machine? Safety?
     │    │    │
     ▼    ▼    ▼
Haystack LangGraph Pydantic AI

Azure/.NET? ──► MS Agent Framework
Claude-native? ──► Claude Agent SDK
```

Recommendation Summary

Framework	Best For
Haystack	RAG, document processing, stability
LangGraph	Complex state machines, explicit control
Pydantic AI	Type safety, structured outputs, clean code
MS Agent	Azure, .NET, AutoGen/SK migration
Claude SDK	Claude-native applications

Our Recommendation

Pydantic AI

Why:

- Highest LLM learnability (92% success)
- Cleanest code (~90 LoC for our use case)
- Type safety for regulated industries
- Automatic validation + reflection
- Growing but strong community (14.5k stars)

Trade-off: Less mature than Haystack/LangGraph

If Not Pydantic AI...

Choose LangGraph if:

- Need explicit state machine control
- Complex branching/looping workflows
- Already using LangChain ecosystem
- Need checkpoint-based recovery

Choose Haystack if:

- Heavy RAG/document processing
- Need maximum stability (6 years)
- Want extensive integrations

Questions?

Resources:

- Comparison doc: `framework_comparison.md`
- Demo implementations: `*_demo.py` files
- Learnability benchmark: github.com/srepho/fwork-learnability

Key Links:

- Pydantic AI: ai.pydantic.dev
- LangGraph: docs.langchain.com/oss/python/langgraph
- Haystack: docs.haystack.deepset.ai