Notes: A Practical Guide to Building Agents

1. What is an Agent?

An agent is a system that independently accomplishes tasks on a user's behalf by leveraging Large Language Models (LLMs), tools, and decision-making logic. Unlike simple LLM integrations, agents can control workflow execution, recognize completion, and proactively correct actions or halt execution and transfer control to the user.

- Uses an LLM to manage workflow execution and make decisions.
- Has access to various tools for interacting with external systems.
- Recognizes when a workflow is complete and can correct or halt actions.

2. When to Build an Agent

Agents are most appropriate when traditional deterministic or rule-based automation falls short. Identify workflows that exhibit one or more of the following characteristics:

- Complex decision-making: Involving nuanced judgment, exceptions, or context-sensitive decisions (e.g., refund approval).
- Difficult-to-maintain rules: Situations where extensive rule sets become unwieldy (e.g., vendor security reviews).
- Heavy reliance on unstructured data: Scenarios requiring natural language interpretation or document processing (e.g., processing insurance claims).

3. Agent Design Foundations

Every agent comprises three core components that work together to execute workflows effectively:

- Model: The LLM powering the agent's reasoning and decision-making.
- Tools: External functions or APIs the agent can use to gather context or take actions.
- Instructions: Explicit guidelines and guardrails defining agent behavior.

4. Selecting Models

Model choice affects task complexity handling, latency, and cost. Follow these guidelines:

- Establish a performance baseline using the most capable model for all tasks.
- Focus on meeting accuracy targets before optimizing for cost or latency.
- Experiment by replacing larger models with smaller ones to identify acceptable tradeoffs.

5. Defining Tools

Tools extend an agent's capabilities by enabling access to data and actions. Categorize tools into:

- Data Tools: Retrieve context or information (e.g., querying databases, reading PDF documents, web search).
- Action Tools: Execute tasks such as sending emails, updating records, or initiating processes (e.g., initiate refunds).
- Orchestration Tools: Agents can call other specialized agents as tools (e.g., a refund agent or research agent).

Well-defined tools should have standardized definitions, clear parameters, thorough documentation, and version management to ensure reusability and discoverability.

6. Configuring Instructions

High-quality instructions reduce ambiguity and improve agent reliability. Best practices include:

- Use existing documents: Leverage SOPs, support scripts, or policy documents to create clear routines.
- Prompt agents to break down tasks: Decompose complex instructions into smaller, explicit steps.
- Define clear actions: Ensure each step corresponds to a specific, testable action or output.
- Capture edge cases: Anticipate variations (e.g., missing information or unexpected user queries) with conditional branches.

7. Orchestration Patterns

Orchestration determines how agents execute workflows. Begin with a single-agent system and evolve to multi-agent systems as needed.

7.1 Single-Agent Systems

A single agent with appropriate tools and instructions can handle many tasks. Execution typically follows a loop until an exit condition is met, such as:

- Invoking a final-output tool.
- The model returns a response without additional tool calls.
- Reaching a maximum number of turns or encountering an error.

Use prompt templates to manage complexity. Dynamic prompt templates allow variables to be injected, reducing the need for multiple static prompts.

7.2 Multi-Agent Systems

When a single agent with tools becomes insufficient (e.g., overly complex logic or tool overload), consider multi-agent architectures. Two primary patterns are:

- Manager Pattern: A central 'manager' agent orchestrates specialized agents via tool calls, maintaining overall context and control.
- Decentralized Pattern: Agents hand off tasks to each other directly, operating as peers without a single controller.

8. Manager Pattern (Multi-Agent)

In the manager pattern, a central agent delegates tasks to specialized agents (as tools). This ensures a cohesive user experience with specialized capabilities on-demand. Ideal when:

- One agent maintains workflow control and user interaction.
- Different tasks require domain-specific expertise that specialized agents provide.

The manager instructs which specialized agents to call based on user requests, aggregates their outputs, and returns a unified response.

9. Decentralized Pattern (Multi-Agent)

In a decentralized pattern, agents pass control to one another via handoffs. Each agent takes over execution based on its specialization. This is optimal when:

- Peer-to-peer task delegation without a central orchestrator.
- Workflow steps map cleanly to distinct specialized agents (e.g., triage, technical support, sales, order management).

Handoffs transfer conversation state and control. Optionally, agents can hand control back if further tasks arise.

10. Guardrails

Guardrails ensure agents operate safely, protect data privacy, and align with brand guidelines. Implement multiple, layered guardrails rather than relying on a single protection.

10.1 Types of Guardrails

- Relevance Classifier: Flags off-topic inputs to keep interactions focused.

- Safety Classifier: Detects and blocks jailbreak attempts or prompt injections.
- PII Filter: Prevents unnecessary exposure of personally identifiable information.
- Moderation API: Captures harmful or inappropriate content (hate speech, violence).
- Tool Safeguards: Assign risk ratings (low/medium/high) to each tool; pause or escalate before invoking high-risk functions.
- Rules-based Protections: Use blocklists, regex filters, and input length limits to prevent known threats.
- Output Validation: Ensure responses align with brand values and policies.

10.2 Building Guardrails

Effective guardrail development follows these steps:

- Focus on data privacy and content safety first.
- Add new guardrails based on real-world edge cases and failures.
- Balance security with user experience; refine guardrails as the agent evolves.

Guardrails can be implemented as functions or as specialized agents. In optimistic execution, guardrails run concurrently with the main agent, raising exceptions if violations occur.

11. Human-in-the-Loop (HITL)

Human intervention is crucial for handling edge cases, identifying failures, and ensuring safety during early deployment. Incorporate mechanisms for escalation when:

- Failure thresholds are exceeded (e.g., multiple retries fail).
- High-risk actions are requested (e.g., large refunds, account cancellations).

Agents should gracefully transfer control to a human or return to the user when these conditions are met.

12. Conclusion

Agents represent an advanced form of workflow automation, capable of reasoning through ambiguity and handling multi-step tasks end-to-end. To build reliable agents:

- Combine capable models with well-defined tools and structured instructions.
- Start with a single-agent system; progress to multi-agent only when complexity demands it.
- Implement layered guardrails to address privacy, safety, and policy compliance.
- Include human-in-the-loop processes for high-risk or failure-prone scenarios.
- Iterate incrementally: start small, validate with real users, and expand functionality over time.