AI Agents

Concepts, Projects, and Security Challenges

[Matthew Schwartz](https://github.com/schwartz1375/)
https://github.com/schwartz1375/

What is an AI Agent?

Autonomous systems that reason, plan, act, and self-correct.

Agents vs LLM vs RAG

LLM = Brain

RAG = Feeds Fresh Data

Agent = Orchestrator + Decision-maker

LLM & MLLM Deep Dive

- LLMs (Large Language Models): Trained on massive text corpora; can generate, reason, summarize; limited to text-only knowledge.
- MLLMs (Multimodal LLMs): Extend beyond text – process images, audio, video, sensor data. Enable richer reasoning (vision + text Q&A, multimodal search).

RAG Deep Dive

- Retrieves external knowledge to enhance LLM answers
- Pipeline: User query → Retriever (vector DB/search) → Context injection into LLM → Output
- Supports hybrid retrieval: structured (databases) + unstructured (docs, web)
- Keeps models current without retraining

Agent Deep Dive

Agents add autonomy: decide when to call tools, search web, summarize, or store info.

Behave like orchestrators – managing tasks rather than just responding.

Building Blocks – Overview

Six principles for effective agents:

- Role-playing – Assign specific roles for sharper reasoning.
- Focus – Narrow tasks reduce hallucinations
- Tools – APIs, DBs, search, code exec extend capabilities.
- Cooperation – Multi-agent collaboration improves accuracy
- Guardrails – Rules, validation, fallback keep agents aligned.
- Memory – Short/long-term + entity memory enable continuity

Role-playing & Focus

Role-playing: A 'legal analyst' agent vs. a 'generic assistant' → more context-aware answers

Focus: Specialized agents outperform generalists (e.g., marketing agent sticks to tone, not pricing)

Tools & MCP

Tools: Extend reach with web searches, API calls, DB queries, code execution

MCP (Model Context Protocol): Enables reusable, shared tools across agents for consistency and scalability

Cooperation, Guardrails, Memory

Cooperation: Data agent + Risk agent + Report writer = collaborative financial analysis

Guardrails: Legal assistant must avoid outdated laws; checkpoints prevent drift

Memory: Tutoring agent recalls past lessons → continuity & personalization

Agentic Design Patterns

Five common design patterns:

- Reflection
- Tool Use
- ReAct
- Planning
- Multi-Agent

Reflection Pattern

Agent reviews and iterates on its own output until quality is acceptable.

Tool Use Pattern

LLM augments reasoning with:

- Querying vector DBs (compliance docs)
- Executing scripts (analytics)
- Invoking APIs (real-time stocks)

ReAct Pattern

- Loop: Thought → Action → Observation → Answer
- Frameworks like CrewAI use this

Planning Pattern

Agent decomposes tasks:

- Subdivides into steps
- Outlines objectives
- Executes strategically

Multi-Agent Pattern

Several specialized agents coordinate tasks, delegate work, and refine outputs

Levels of Autonomy

Progression of control:

- Basic Responder – Human guides all steps; LLM only replies
- Router – LLM selects among predefined functions.
- Tool Calling – LLM chooses tools + arguments
- Multi-Agent – Manager oversees sub-agents.
- Increasing Autonomous (potentially fully autonomous in the future) – LLM writes/executes code independently

Higher autonomy = greater capability and risk → requires stronger guardrails

Agentic RAG Case Study

- Retriever + Writer agents operating in a managed multi-agent system
- Fetches context from DB/web and produces insights

Enables dynamic, real-time knowledge retrieval

Financial Analyst Agent

- Multi-agent via MCP system generates stock insights and visualizations
- Stack: Orchestration framework, LLMs, and AI-native IDE

Automated financial research with safe sandbox execution

Brand Monitoring System

- Scrapes mentions from X, YouTube, Instagram, web
- Agents analyze + summarize insights

Real-time brand intelligence for marketing & risk

Why Agent Security is Different

Not reducible to 'alignment only' or 'cybersecurity only'

Agents blur data + instructions, unlike traditional systems

Prompt Injection Risk

Agents can't separate data from instructions → inherently injection-prone

Adversaries can hijack workflow with malicious inputs

Practical Insight: No Miracles Needed

Prompt injection - *largely mitigable with current controls*:

- Input validation & sanitization
- Role/task constraints
- Layered defenses + monitoring
- Does not require ML breakthroughs

Trust Zone Model

- Align agent via fine-tuning, scaffolding, guardrails
- Operate inside least-privilege trust zone
- IAM, PKI, taint analysis ensure agent cannot act outside zone

Lessons from History

- PCs → arbitrary code execution → patched later
- Internet → unsafe code → probabilistic defenses
- Lesson: Don’t oversell weak defenses; design for resilience

Fusion of Alignment + Security

- Security assumes misalignment
- Alignment work aims to maximize the chance that the model acts "aligned" in every interaction
- Fusion provides resilience and reliability

Progressive Trust / Guided Autonomy

Expand trusted region gradually:

- Use red teaming
- Continuous evals
- Real-world feedback

Security Stack for Agents

Enterprise integration:

- AM, PKI, tool auth
- Static/taint analysis on outputs
- Provenance in decisions
- Detection/response for reasoning traces
- Red-teaming pipelines

Practical Path Forward

Defense-in-depth against injection

Guardrails + monitoring

Continuous integration with enterprise security stack

Closing & Q&A

Agentic AI is powerful but risky

Future requires fusing ML safety, AI alignment, and cybersecurity