Category	Summary of Content	Keywords	Thinking Questions	Practical Examples
Introduction & Tools	Foundation Models (LLMs) are pattern prediction engines, limited to trained data. External Tools act as the agent's "eyes and hands," allowing them to access new data, interact with external systems, and take actions. Tools either let a model know something (data retrieval) or do something (API calls). Types include Function Tools , Built-in Tools (e.g., Google Search, Code Execution, URL Context in Gemini), and Agent Tools (invoking a sub-agent).	Foundation Model, AI Agent, External Tools, Tool Calling, Function Tools, Built-in Tools, Agent Tools, Know Something, Do Something.	How does the inability to access new data fundamentally limit a model's utility without tools? What are the trade-offs between using a built-in tool (where the definition is implicit) and a custom Function Tool?	Weather Agent: Uses and tools to answer a user's question, demonstrating the need for external data and calculation. function: A Function Tool with parameters (,) to control a physical device (Snippet 1).
Best Practices for Tool Design	Effective tools require clear documentation. Names should be specific (e.g.,). Descriptions should focus on the action ("create a bug"), not the implementation ("use the tool"). Tools should encapsulate a single, granular task , not be a thin wrapper over a complex API. Concise Output is vital to prevent Context Window Bloat . Use schema validation and provide descriptive error messages to guide the LLM's next steps.	Tool Documentation, Clear Name, Describe Actions, Publish Tasks, Granularity, Concise Output, Validation, Context Window Bloat, Descriptive Error Messages.	If long parameter lists confuse the model, how can developers maintain granular tools while integrating with complex enterprise APIs that require many inputs? What is the maximum acceptable data volume for a tool response before Context Window Bloat becomes a significant issue for performance/cost?	Good Tool Name: (better than). Good Documentation Example: Snippet 4 (clear name, parameters, returns, and example). Error Message Example: "API rate limit exceeded. Wait 15 seconds before calling this tool again" (Snippet 8) - gives the LLM explicit recovery instructions.
Model Context Protocol (MCP)	MCP is an open standard (introduced in 2024 by Anthropic) designed to solve the "N x M" Integration Problem (the exponential growth of custom connectors between N models and M tools). It uses a client-server model with three core components: Host (manages user experience/orchestration), Client (connects to Server), and Server (exposes capabilities/tools). Communication uses JSON-RPC 2.0 over stdio (local) or Streamable HTTP (remote). Tools are the most broadly supported primitive (99% client support).	Model Context Protocol (MCP), N x M Integration Problem, Standardization, Host, Client, Server, JSON-RPC 2.0, Streamable HTTP, Tools, Tool Definition, Tool Results.	Since only Tools have broad support, will the other primitives (Resources, Prompts, Sampling, Elicitation, Roots) ever become strategically important, or will the standard evolve to focus purely on tool orchestration? How does standardizing the interface help organizations dynamically switch between different underlying LLMs or tool providers?	MCP Architecture: Host (AI Agent) contains Client, which sends a JSON-RPC request (e.g., with method) to the Server (Figure 2). Tool Definition Example: Snippet 6 (JSON schema for a tool with clear and).

Security Risks & Mitigations	MCP introduces a new threat landscape, especially as it lacks native enterprise-grade security (Auth/Auth, Identity Mgmt). Key risks involve agent manipulation and privilege misuse.	Dynamic Capability Injection, Tool Shadowing, Confused Deputy Problem, Explicit Allowlist, Human-in-the-Loop (HIL), Least Privilege, Taint Sources/Sinks, Output Sanitization.	The document highlights the difficulty of enforcing the rule that servers "MUST NOT use elicitation to request sensitive information." What technical control could be implemented on the Client/Host side to audit and flag elicitation requests for sensitive keywords?	Dynamic Capability Injection: A poetry agent connects to a low-risk book server, which then dynamically adds a high-risk book purchasing tool (Mitigation: Use an Explicit Allowlist). Tool Shadowing: A malicious tool with an overly descriptive name ( ) tricks the agent into choosing it over the legitimate tool ( ) to exfiltrate data (Mitigation: Prevent Naming Collisions , require HIL for high-risk actions). Confused Deputy Problem: An unprivileged employee tricks the highly-privileged MCP Server (the deputy) via the AI Agent (the confused party) into creating a new repository branch containing secret code (Mitigation: Enforce Principle of Least Privilege , keep credentials out of agent context).