Lecture 4: Tool Use & Simple Orchestrators

MCP-like tool registries, guardrails, and verification

University of Chicago

December 29, 2025

What you will build today

- A tiny tool registry (MCP-like): names, descriptions, input schemas
- A minimal orchestrator loop: plan $\rightarrow$ tool calls $\rightarrow$ verify
- Guardrails: allow-lists, step limits, and basic injection resistance
- A research brief deliverable grounded in local docs + a CSV

Business problem

Scenario: An analyst needs a research brief from messy internal docs + a dataset.

Goal: Produce a structured brief with:

- key findings (grounded in sources)
- risks and assumptions
- recommended next steps
- a small table from the dataset (Python tool)

Inputs (provided in `lecture_4/data`)

- `docs/`: internal docs (including a malicious prompt-injection example)
- `market_signals.csv`: small dataset to summarize
- `tool_registry.json`: tool definitions (names + schemas)

New workflow primitive introduced

- **Tool use**: model chooses among tools, but only within constraints
- **Orchestration**: plan/execute/verify with a step budget
- **Defense**: treat retrieved text as untrusted input

Deliverable

- Notebook: `notebooks/lecture_4_tooling_orchestrator.ipynb`
- Output: `data/outputs/research_brief.md`

Extensions / Optional challenges

- **Verification step**: add a checker that validates claims against tool trace + sources
- **Planning robustness**: re-plan on tool errors; add stopping conditions and max-step budgets
- **Strict tool schemas**: validate tool-call JSON against schemas; reject malformed calls
- **Security hardening**: demonstrate injection and add filters (treat docs as data; never execute doc instructions)
- **Swap-in real MCP**: replace the in-notebook tool registry with an MCP server later