Machine Learning Systems

Build efficient and scalable ML services through the vertical integration of algorithms, system software, and hardware

Li Shang
lishang@slai.edu.cn

Acknowledgement

Machine learning systems is a broad and rapidly evolving field. The course material has been developed using a broad spectrum of resources, including research papers, lecture slides, blog posts, research talks, tutorial videos, and other materials shared by the research community.

The Evolution of Multi-Agent AI

From Prompts to Digital Workforces: A History of Multi-Agent AI

Era 1 - The Cognitive Spark

Chain-of-Thought (CoT) and the "Inner Monologue"

- The Paradigm: Moving from direct answering to step-by-step reasoning.
- The Breakthrough: Prompting models to explicitly write out intermediate reasoning steps (e.g., "Let's think step by step") before arriving at a final answer.
- The ReAct Framework: Combining reasoning with basic actions (Reason + Act).
- The Agentic Seed: By separating "thinking" from "answering," researchers realized models could evaluate their own thoughts and formulate plans. This was the foundational cognitive engine required for future agents.

Era 2 - The Orchestration Era

LangChain and "Long-Chain" Execution Pipelines

- **The Paradigm: Tool-Use and API Orchestration.**
- **The Mechanism: Frameworks like LangChain allowed developers to build "long-chain" scripts that connected LLMs to external APIs, calculators, and search engines.**
- **The Architecture: Directed pipelines where the text output of one model automatically became the prompt input for the next.**
- **First-Gen Agents: This era gave rise to early, highly experimental agents like AutoGPT.**
- **Limitation: Early chains were rigid and fragile. If a single external API threw an error on step 4, the entire chain usually crashed rather than self-correcting.**

Era 3 - The Autonomous Revolution

OpenClaw and Persistent, Local Ecosystems

- **The Paradigm: From run-once scripts to always-on digital entities.**
- **The Breakthrough: The explosive rise of open-source, local agent frameworks like OpenClaw, which moved agents out of developer terminals and integrated them into daily messaging apps (WhatsApp, Telegram).**
- **Persistent Memory: Maintaining 24/7 context across sessions and executing background "cron jobs" without a human prompt.**
- **Multi-Agent Teams: OpenClaw popularized local multi-agent swarms. A main "Orchestrator Agent" can spawn isolated "SubAgents" (e.g., a web scraper or a data analyzer) that share a local memory folder but run in parallel to solve complex tasks.**

Era 4 - Enterprise & Native Operation

Claude Code, AutoGen, and "Harness Engineering"

- **The Paradigm: Deep environment integration and strict mechanical scaffolding.**
- **Native Control: Moving beyond simple API wrappers. Systems like Anthropic's Claude Code and Computer Use are embedded natively into terminal environments and visual operating systems.**
- **Separation of Concerns: The realization that a single omnipotent agent is a liability. Modern frameworks utilize Generator-Evaluator Loops (e.g., one agent writes code, a completely separate AI acts as the QA tester to prevent self-evaluation bias).**
- **The Focus: Harness Engineering—building strict sandboxes, Model Context Protocol (MCP) tool routing, and human-in-the-loop (HITL) approval gates to safely control these powerful autonomous swarms.**

Harness Engineering: The Claude Code Design Guide

Building Boundaries, Continuity, and Control for AI Coding Agents

The Core Problem

Why Harness Engineering is NOT just "Prompt Engineering"

- **The Past:** Prompt engineering focuses on what to ask a model to generate text.
- **The Present:** Harness engineering focuses on how a system operates when an AI is given execution privileges.
- **The Equation:** Agent = Model + Harness
- **The Goal:** Building a robust scaffolding that maintains boundaries, ensures operational continuity, and controls the consequences of AI actions.

The Control Plane

Prompts Are Not Input Boxes

- **Changing the UI Paradigm:** A prompt is a deterministic control surface, not just a chatbot text box.
- **Injecting State:** The harness automatically injects system states, available tools, memory, and environment variables into the context window before the AI even "thinks."
- **Mechanical Invariants:** Using system files (like CLAUDE.md) to establish non-negotiable rules, project architecture, and coding standards that govern the model's environment.

Continuous Execution

The Query Loop - Taking Over the Workflow

- **Legacy AI: Question → Answer (Stateless)**
- **Agent Architecture: Query → Stream → Tool-Call → Loop (Stateful)**
- **Autonomous Operation: The agent reads an objective, streams its reasoning, executes a terminal command (e.g., npm test), reads the output, and recursively loops until the condition is met.**

Sandboxing the AI

Tools, Permissions, and Interrupts

- Tool Orchestration: Giving the agent "hands" (File I/O, Bash execution, MCP servers, Web Search).
- Permission Boundaries: The model decides what to attempt; the harness decides what is allowed.
- Interrupt Hooks: Implementing strict PreToolUse and PostToolUse lifecycle hooks to pause execution.
- Human in the Loop (HITL): Demanding explicit human approval for destructive commands (e.g., rm -rf, git push, dropping databases).

Context Anxiety & Memory

Surviving Long-Term Tasks Without Degradation

- **The Threat: As sessions drag on, models hallucinate, lose the architectural plot, and forget instructions (Context Overflow).**
- **The Golden Rule: If state isn't saved to the disk, it doesn't exist.**
- **Micro-compression: Summarizing past execution loops to shrink the active context window.**
- **On-Demand Knowledge: Pulling in specific reference files only when relevant to save tokens and reduce model confusion.**

Resilience by Design

Error and Recovery as the Runtime Norm

- **Mindset Shift: AI agents will write bad code. Optimize for graceful recovery, not zero-shot perfection.**
- **Sensors & Feedback Loops:**
  - **Linters, type-checkers, and compilers run automatically.**
  - **stderr (Standard Error) logs are fed directly back into the LLM context as an "Observation."**
- **Anti-Looping Mechanics: The harness detects if an agent has tried the same failing action repeatedly and forces a hard stop.**

Multi-Agent Validation

Don't Let the System Referee Itself

- The Flaw: "Self-Evaluation Bias"—LLMs will confidently tell you their broken code works perfectly.
- Separation of Concerns:
  - Generator Agent: Writes the implementation.
  - Evaluator Agent: A separate, skeptical sub-agent spawned to test the implementation.
- Objective Verification: Verifying success via external systems (e.g., passing a test suite) rather than asking the LLM to read its own code.

Scaling to the Enterprise

The Engineering Litmus Test

- From "Vibe Coding" to Systems: Turning individual prompt-hacking into a repeatable organizational capability.
- Shared Invariants: Establishing team-wide configurations so the agent behaves exactly the same for a Junior Dev as it does for a Senior Dev.
- How do you know if your AI setup is an engineering system?
  - Are rules mechanically enforced, not just written in text?
  - Is state persisted across sessions?
  - Are permissions decoupled from the AI's reasoning?

Conclusion

The Future of Autonomous Development

- **Summary: Effective AI development is 20% model intelligence and 80% Harness Engineering.**
- **Final Thought: The underlying AI model provides the horsepower; the harness provides the steering wheel, brakes, and navigation.**
- **Questions?**