DSI: AI Development Program

Part 1: Introduction to AI

University of Chicago

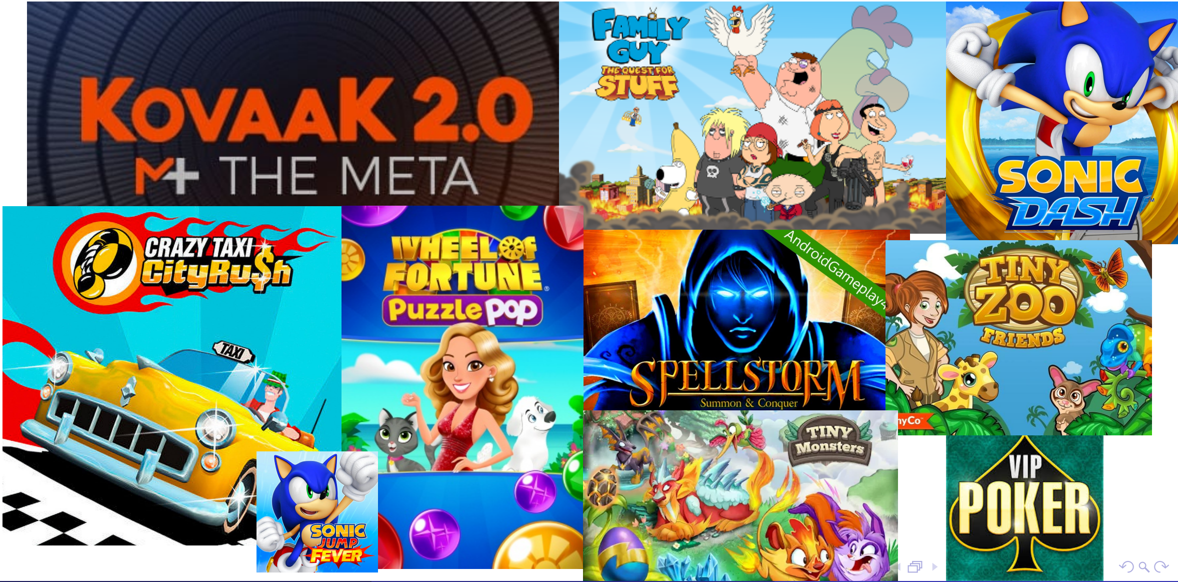
December 29, 2025

- Who am I
- Learning objectives for the course.
- Today: A brief introduction to the concepts around AI

Who am I

- Director Data Science Clinic
- Undergrad: UC Berkeley, PhD: UCLA
- Worked in Consulting, Video Games, AI

Some Games I've Worked on



What is an LLM?

- A **Large Language Model (LLM)** is an artificial intelligence system trained on massive amounts of text data (often trillions of words)
- These models learn patterns in language and can:
  - Generate human-like text
  - Answer questions
  - Translate languages
  - Summarize documents
  - Write code
  - And much more
- Think of an LLM as a very sophisticated autocomplete system that has read a significant portion of the internet

Foundation Models

- A **foundation model** is a large-scale machine learning model trained on broad data (generally using self-supervision at scale) that can be adapted to a wide range of downstream tasks
- **Key characteristics:**
  - Broad training: Trained on diverse, large-scale datasets
  - General-purpose: Can be adapted to many different tasks
  - Transfer learning: Knowledge learned from training can be applied to new tasks
  - Base for specialization: Can be fine-tuned or used as-is for specific applications

Foundation Models vs LLMs

- **Foundation Model** is the broader categoryit includes models for vision, audio, code, etc.
- **LLM (Large Language Model)** is a type of foundation model focused specifically on language
- **Examples of foundation models:**
  - **Language**: GPT-4, Claude, Llama (these are LLMs)
  - **Vision**: CLIP, DALL-E
  - **Code**: Codex, CodeLlama
  - **Multimodal**: GPT-4V (vision + language), Gemini (multimodal)

Key Terminology

- **Foundation Model**: A large-scale model trained on broad data that can be adapted to many tasks
- **LLM (Large Language Model)**: A type of foundation model specifically designed for language tasks
- **Token**: The basic unit of text that an LLM processes (word, part of word, or punctuation)
- **Prompt**: The input text you give to an LLM
- **Completion/Response**: The output text generated by the LLM
- **Context Window**: The maximum number of tokens an LLM can process in a single conversation

Key Terminology (continued)

- **Temperature**: A parameter that controls randomness (Lower = more deterministic, Higher = more creative)
- **Fine-tuning**: Training an LLM on specific data to improve performance on particular tasks
- **Inference**: The process of generating text from an LLM (as opposed to training)

Context windows: what they are (and why you care)

- The **context window** is the model's working memory for a single request
- Everything the model can use must fit inside it:
  - system prompt + developer instructions
  - user messages
  - tool outputs / retrieved documents
  - the model's own previous messages (in chat)
- When you exceed the window, something gets dropped or truncated

Context windows: practical implications

- **Tradeoffs**: more context can improve grounding, but increases cost/latency
- **Recency and placement matter**: newer text is usually attended to more
- **Failure modes** to watch for:
  - forgetting earlier constraints
  - losing critical details due to truncation
  - quoting the wrong source when too many docs are included

System prompts: role and best practices

- The **system prompt** sets the highest-level behavior (tone, rules, safety, format)
- Use it to encode **non-negotiables**:
  - output format requirements (e.g., JSON only)
  - refusal and safety rules
  - grounding requirements (cite sources; do not invent facts)
- Keep it **short, explicit, testable**; avoid vague goals like "be helpful"

Context engineering (a.k.a. make the model successful)

- **Context engineering** = designing what goes into the context window so the model can reliably do the task
- Common building blocks:
  - task definition + success criteria
  - constraints and policies (what is allowed/not allowed)
  - examples (few-shot) and edge cases
  - retrieved evidence (RAG) with citations
  - tool schemas for structured interaction with code

A simple context template (recommended)

1. **System**: rules, safety, output format
2. **Developer**: task-specific policies (what to do, what not to do)
3. **User**: the request + required inputs
4. **Evidence**: only the relevant retrieved snippets (not entire documents)
5. **Tools**: schemas + allowed actions
6. **Final**: "Return JSON only" / "Cite sources" / "Ask clarifying questions if needed"

Example LLM Models: Commercial

- **GPT-4** (OpenAI)
  - One of the most capable models
  - Available via OpenAI API
  - Powers ChatGPT Plus
- **Claude 3.5 Sonnet** (Anthropic)
  - Strong reasoning and coding capabilities
  - Available via Anthropic API
- **Gemini Pro** (Google)
  - Google's flagship model
  - Available via Google AI Studio

Example LLM Models: Open Source

- **Hugging Face** is a platform hosting thousands of open-source models
- Popular models include:
  - **Llama 3** (Meta)
  - **Mistral** (Mistral AI)
  - **Phi** (Microsoft)
  - **Gemma** (Google)
- You can use these models via:
  - Hugging Face Transformers library (Python)
  - Hugging Face Inference API
  - Local deployment

Model Aggregators

- **Open Router** provides access to many models through a single API
  - Unified API for 100+ models
  - Compare models side-by-side
  - Pay-per-use pricing
  - Easy to switch between models

Token economics: why tokens matter

- Most LLM APIs charge by tokens:
  - **input tokens** (prompt, system prompt, retrieved docs, tool outputs)
  - **output tokens** (the model's response)
- Cost and latency generally scale with total tokens processed
- **Context engineering is cost engineering**:
  - tighter prompts $\rightarrow$ lower cost and often higher reliability
  - smaller context when possible $\rightarrow$ faster, cheaper

Pricing references: `https://openai.com/api/pricing/`

Cost math (simple model)

- Typical pricing is "$ per 1M tokens"
- Approximate cost per call:

$$\text{cost} \approx \left(\frac{T_{in}}{10^6} \cdot p_{in}\right) + \left(\frac{T_{out}}{10^6} \cdot p_{out}\right)$$

- Engineering levers:
  - reduce retrieved context (better retrieval, smaller snippets)
  - constrain output length (max tokens, structured output)
  - pick the cheapest model that meets quality requirements
  - cache repeated context (where supported)

Pricing references: `https://openai.com/api/pricing/`

Real-world pricing examples (text)

Example (OpenAI GPT-4.1 family; per 1M tokens)

Model	Input	Output
GPT-4.1	$2.00	$8.00
GPT-4.1 mini	$0.40	$1.60
GPT-4.1 nano	$0.10	$0.40

- Decision implication: use smaller/cheaper models for simpler tasks (classification, routing, QC), reserve larger models for harder steps

Source: `https://openai.com/index/gpt-4-1/`

Multimodal pricing: what changes?

- Multimodal models may price **each modality differently**:
  - text tokens (input/output)
  - image generation (often priced by image or image tokens)
  - audio tokens (speech in/out) and realtime usage
  - video generation (often priced per second/resolution)
- Decision implication:
  - use multimodal only when it adds value (OCR, visual QA, audio transcription)
  - consider hybrid pipelines (cheap OCR $\rightarrow$ text-only LLM) to control cost

Pricing references: `https://openai.com/api/pricing/` and `https://platform.openai.com/docs/pricing`

LLM Limitations

While LLMs are powerful, they have important limitations:

1. **Knowledge Cutoff**: They only know information from their training data up to a certain date
2. **Hallucination**: They can generate plausible-sounding but incorrect information
3. **No Real-World Actions**: They can't directly interact with external systems (databases, APIs, file systems)
4. **Context Limits**: They have maximum context window sizes
5. **Static Knowledge**: They can't learn new information after training without fine-tuning or retrieval

What is an AI Agent?

- An **AI Agent** is an LLM that can use **tools** to interact with the outside world
- While a basic LLM can only generate text based on its training data, an agent can:
  - Query databases
  - Call APIs
  - Read files
  - Execute code
  - Search the web
  - Perform actions in real-time

LLM vs Agent: Key Differences

Aspect	LLM	Agent
Capabilities	Text generation only	Text + tool usage
Knowledge	Training data only	Training data + real-time data
Actions	None	Can perform actions via tools
Interactivity	One-shot responses	Can loop and iterate

The Agent Loop

$$\text{Observe} \to \text{Think} \to \text{Act} \to \text{Observe} \to \ldots$$

1. **Observe**: Receive input (user query, tool results, system state)
2. **Think**: Process information and decide what to do next
3. **Act**: Execute actions (call tools, generate responses)
4. **Observe**: See the results and continue the loop

Example Agent Systems

- **LangChain Agents**
  - Framework for building LLM applications
  - Supports multiple LLM providers
  - Tool integration and agent orchestration
- **Claude with MCP** (Model Context Protocol)
  - Claude Desktop can connect to MCP servers
  - Access custom tools and data sources
- **ChatGPT with Plugins/Code Interpreter**
- **Cursor AI**

What is a Tool?

- A **tool** is a function that an AI agent can call to interact with external systems
- Tools bridge the gap between the LLM's text generation capabilities and real-world actions
- Every tool has:
  1. **Name**: A unique identifier (e.g., `get_player_list`)
  2. **Description**: What the tool does and when to use it
  3. **Input Schema**: Parameters the tool accepts (JSON Schema format)
  4. **Implementation**: The actual code that executes when called

Common Tool Categories

- **Database Tools**: Query databases, insert/update/delete records
- **API Tools**: Call REST APIs, interact with web services
- **File System Tools**: Read/write files, list directories
- **Code Execution Tools**: Run Python code, execute shell commands
- **Web Tools**: Search the web, scrape websites

What is MCP?

- The **Model Context Protocol (MCP)** is an open protocol created by Anthropic
- Enables AI assistants to securely connect to external tools and data sources
- Provides a standardized way for AI assistants to discover and use capabilities from external systems

Why MCP?

- Before MCP, each AI assistant had its own way of connecting to external tools:
  - ChatGPT had plugins
  - Claude had custom integrations
  - Each system was proprietary
- MCP provides:
  - **Standardization**: One protocol works across multiple AI assistants
  - **Security**: Secure, controlled access to tools
  - **Discoverability**: AI assistants can discover available tools automatically
  - **Composability**: Tools can be combined and chained together

MCP Architecture

AI Assistant (MCP Client)

$\downarrow$

MCP Protocol

$\downarrow$

MCP Server

$\downarrow$

Your Application (Flask API, Database, File System)

Key MCP Components

1. **MCP Server**: Exposes capabilities (tools, resources, prompts) to AI assistants
2. **MCP Client**: AI assistant that connects to servers (Claude Desktop, Cursor, etc.)
3. **Tools**: Functions that an AI can call to interact with external systems
4. **Resources**: Data sources that an AI can read (files, database tables, API endpoints)
5. **Prompts**: Pre-defined prompt templates for common tasks

How MCP Works

1. User asks: "Get me all players"
2. MCP Client analyzes query and selects appropriate tool
3. Client calls tool: `get_player_list()`
4. MCP Server makes HTTP request to Flask API
5. API returns player data
6. Server returns tool result to client
7. Client incorporates result into response
8. User receives: "Here are all the players: ..."

Why not a REST API?

- MCP uses **Server-Sent Events (SSE)** for real-time communication
- **REST**: One request → wait → one response → connection closes
- **MCP/SSE**: Long-lived connection → multiple messages → streaming updates
- This enables:
  - Real-time feedback during tool execution
  - Streaming responses for long operations
  - Continuous communication between client and server

What are Claude Skills?

- **Claude Skills** (also called "Actions" or "Tool Use") is Claude's built-in capability to use tools
- When you give Claude access to tools, it can:
  - Automatically decide when to use tools
  - Call multiple tools in sequence
  - Combine tool results into coherent responses
- Claude Skills work with:
  - MCP servers (via Claude Desktop)
  - Custom API integrations
  - Function calling (via Anthropic API)

Complete System Architecture

User Interface

$\downarrow$

AI Assistant Layer (Claude Desktop)

$\downarrow$

MCP Protocol Layer

$\downarrow$

MCP Server Layer

$\downarrow$

Application Layer (Flask API, Database, File System)

Example: Complete Interaction Flow

Scenario: User asks "What colleges do players from Washington come from?"

1. Claude analyzes query, identifies need for player data
2. Selects `get_players` tool
3. Calls `get_players(team="WAS")`
4. MCP Server makes GET request to Flask API
5. API queries database
6. Returns player records
7. Claude processes data, extracts unique colleges
8. Returns formatted response to user

Summary

- **LLMs** are powerful text generation systems trained on massive datasets
- **AI Agents** extend LLMs with tool-using capabilities
- **Tools** bridge the gap between LLMs and real-world systems
- **MCP** provides a standardized protocol for connecting AI assistants to external tools
- Together, these technologies enable AI systems that can interact with the real world

Questions?

Thank you!