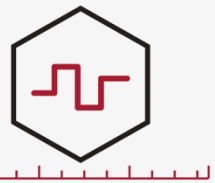
https://cs2680.com

Modern AI Systems: Agents and System Optimizations

Overview

Juncheng Yang

Harvard John A. Paulson
School of Engineering
and Applied Sciences



HARVARD UNIVERSITY
MaDSys
Measurement and Design of Systems Lab

Plan for today

- Course overview
- Policy and logistics
- AI systems and agentic systems

About me

Juncheng Yang

- Assistant Professor at SEAS
- Graduated from CMU in 2024
- Previously, AWS, Snowflake, Cloudflare, Twitter

Harvard MaDSys (Measurement and Design of Computer Systems) Group

Efficiency

less resources

higher utilization

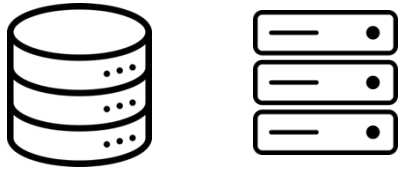
Reliability

less downtime

more robust

Storage Systems

Caching
Sustainable storage



Agent Serving Systems

Context management
LLM serving optimizations

Performance

faster

more scalable

Sustainability

less carbon emission

**People find solutions, we find problems;
we measure, design, and operate real systems**

Me and TFs

Juncheng Yang

Office hour: by appointment

Yiyu Liu

Office hour: by appointment

Yunjia Zheng

Office hour: by appointment

Course staff email

cs2680-staff@g.harvard.edu

I will try to remember everyone’s name

Please say your name first before asking or answering questions

Next, Your Turn

Introduce yourself to others at your table

5 min

- name and where you are from
- program, year and background
- research interest
- why you take this class and what you hope to learn

Each table selects a leader to introduce everyone at the table

The course

What you will learn, the assignments,
and the project

What is this course about

Agentic AI Systems

- Agent design
- LLM / agent serving systems

A graduate-level computer systems course

- Expected background: one graduate-level systems course (OS, networking, databases, architecture)
- Experience with real systems

What you will learn in this course

Foundations

First few classes

- Introduction to LLMs and agents
- How to use Claude Code
- How to design a coding agent
- How to optimize your coding agent

Systems for LLMs & agents

Majority of the classes

- GPU programming and kernels
- Batching, scheduling, memory management
- Load balancing and routing
- KV-cache and prefix-cache
- Agent serving systems and other optimizations

Five assignments (1–3)

1

Build a web UI for Claude Code

- Experience the magic of coding agents
- Understand the lifecycle of coding agent

2

Build a simple coding agent

- Solve a set of simple problems
- Create new problems for the assignment 3 leaderboard

3

Optimize your coding agent

- Open-ended: context, memory, test-time scaling, routing, sub-agents, tool call
- Metrics: accuracy, cost, latency

Five assignments (4–5)

4 Serve your agent

- From a commercial API to a small model
- Co-tune your agent and the serving engine for performance

5 Optimize the full stack

- The hardest assignment — harder tasks
- Open-ended: quantization, routing, fine-tuning, engine–agent co-design

Scored on the leaderboard

Assignments 3 & 5: leaderboard ranking + baseline improvement

Do NOT take this class if you care about GPA

Expect 10+ hours every week

Project

- Open-ended, individual project + AI
- Poster and demo session during reading period
- Grading: code + report + poster / demo peer review

1

Build a tool

Find a gap and build a tool you will actually use

2

Research

A research project — talk to the TFs first

A lot of learning happens after class

Class time is limited

Learn the skills to learn — we provide pointers

Hands-on class: learn by doing the assignments

Grading, policy and logistics

The parts worth reading carefully

Grading

Five assignments 70%
1, 2, 4: 10% each · 3, 5: 20% each

Project 16%

Presentation 10%
paper + assignment 1

Participation 4%

Bonus 10%
course feedback + problems that AI cannot solve

Assignment 3 and 5 sharing sessions are based on leaderboard ranking

4 Late days throughout semester 20% penalty per day for late submission

Alert: 20% A grading policy enforced · an A or even A- is hard to earn

Enrollment

40

planned and budgeted capacity

80

enrolled today

68+10+6

prospective, cross-registration and waiting list

Cap has increased to 80

However, computing resources might not increase proportionally

It is likely that you may not be able to get a GPU on the assignment due date

Auditing

Allowed only if Harvard policy permits and a seat is available

Policy: using AI in this course

Using AI is encouraged

However, you should reason about AI-generated code, and you are responsible for all code you submit

Disclosure required

What worked and did not work

Upload your session log

Required with each assignment

Read the assignment submission page on the course website

Policy: attendance and laptops

Attendance required

In-person attendance is expected

No video recording

Class is not recorded

Laptops allowed

If you plan to keep yours open, please sit at the back

Guest lecturers

Meeting sign-up opens a week before

See the website for the full policy

Other logistics: computing

Claude Code

API access

api.cs2680.com

GitHub

Private repo, shared with staff

Class HPC

- Everyone with Canvas access is already on the system
- SLURM-based; RTX 6000 PRO Blackwell
- Course work only

AWS

- **Sign up and fill in the Google Form before Friday**
- **Terminate or turn off your instance when not in use**

Read the computing page on the website

Schedule may change during the semester

The first time offering this course: expect a bumpy ride ahead!

Grading may change as well (paper reading)

Week 1 **Week 14**

A few questions first

To help me understand everyone's background

How many computer systems courses have you taken?

- **Operating systems**
- **Distributed systems**
- **Databases**
- **Computer networking**
- **Computer architecture**
- *... anything else*

How many of these topics are you familiar with?

- ☐ Transformer architecture
- ☐ TP / PP / DP / EP
- ☐ iteration-level scheduling / pagedAttention
- ☐ Linear attention
- ☐ GPU programming
- ☐ Linux kernel

How have you used AI?

1

Chat

2

Coding agent

MCP, Skills / Plugins, subagent, parallel agent

3

Customized agent

Agentic Coding Levels

No AI — hand-written code

Assist — AI autocompletion

Supervised — AI writes functions, you review

Scoped — hand off a task, review the code

Goal-driven — AI writes all, tests verify

Parallel agents — sub-agents split the work

Part I — AI Systems

How they differ from traditional computer systems

What is an AI System?

Many moving parts: scheduling & orchestration, state management, communication, caching, fault-tolerance, security, observability…

AI / ML

"What model should we use?" / "How should I design the model"

- Pick a model and write a Python script to solve the task
- Optimizes for accuracy and capability

AI Systems

"How do we train and serve it at scale?"

- Maximum GPU utilization (performance)
- Minimal resource waste (efficiency)
- Minimal downtime (reliability)
- Lower carbon footprint (sustainability)

* AI systems a broad term, this course narrowly focuses on LLM systems.

Why should you care about AI systems?

This is where capability becomes product

... and it is harder than the AI itself

Critical digital infrastructure

Understand model design

Design better products

More job opportunities

How AI systems differ: compute

Heterogeneous computation

- CPU: tokenization, kernel launch, data movement, tool calls — GPU: matmul
- Mixed GPU generations and xPUs

Distinct phases of computation

- Prefill is compute-bound; decode is memory-bandwidth-bound
- The same deployment hits different bottlenecks: compute, bandwidth, capacity

Unpredictable execution time

- Output-length dependent, which complicates scheduling

How AI systems differ: non-deterministic compute

Why it happens

- Floating-point addition isn't associative — rounding at every operation
- Butterfly effect: a 7th-decimal shift in a matmul flips the next token

Where it comes from

- **Runtime**: warp/block scheduling, memory ordering, async streams, batch size
- **Software**: fused multiply-add vs two ops, autotuning, subnormals
- **Hardware**: CUDA vs tensor core, Hopper FP8 vs Blackwell micro-scaling tiles

Approximate computing is acceptable

Distributivity also fails: a * (b + c) != a * b + a * c, but not used widely
Transitivity: a == b, b == c does not imply a == c

How AI systems differ: memory, storage and communication

Huge state

- Model weights: GBs to TBs
- KV-cache: 10s–100s KB per token
- Prefix cache: KV reuse across requests
- Data movement is the bottleneck

Multi-GPU communication

- Pipeline / tensor / expert / context parallelism
- Collectives: NCCL, UCCL
- Topology: NVLink, Ethernet/InfiniBand
- CPU and kernel bypass: RDMA

How AI systems differ: fast-moving models and workloads

Rapidly-evolving model architecture

- MHA, GQA, MLA, NSA, linear attention...

Rapidly-evolving workloads

- Multi-turn chat: interactive, latency-sensitive
- Problem solving: reasoning with long output
- RAG: retrieve doc before answering, long context
- Agents: long context, prefix-cache critical, routing and load balancing

Adapt model to hardware ↔ design hardware for model architecture

How AI systems differ: reliability, power and energy

Reliability

- 9% AFR, 50,000hr MTBF
- One GPU fails every 3 hours in a 16K-GPU cluster

Power and energy

- CPU 100–300W; GPU 700–1000W, 4–8 GPUs per server
- Server power 800W → 6,000W, an 8x increase
- How to avoid power fluctuation

Part II — Agentic AI Systems

What an agent is, and why it is a systems problem

What is an AI agent?

An AI agent is an autonomous software system that uses an AI ***model*** to reason about a goal, use external ***tools***, and execute ***multi-step actions*** until a ***task*** is finished

1. **plan** — break the task into sub-steps
2. **act** — call external tools (APIs, databases)
3. **observe** — interpret the results of those tools
4. **adapt** — revise the plan on new information
5. **deliver** — return the final result

Why do we need an AI agent?

LLMs are strange computers: *superhuman and incompetent at the same time*

Superhuman at

- Recalling vast factual knowledge
- Language
- Pattern matching
- Generating code
- Operating across many domains

Surprisingly weak at

- Persistent memory
- Reliably understanding context
- Avoiding hallucinations
- Knowing what they don’t know
- Security and adversarial input

https://www.youtube.com/watch?v=LCEmiRjPEtQ&t=874

Agents as a new kind of computer

CPU	→	**LLM**
RAM / working memory	→	**Context window**
Peripherals	→	**Tools**
OS	→	**Agent Harness**
Application	→	**Agent**

An LLM is more than a smarter CPU

a new kind of programmable computer — with a completely different programming model and failure model

Why are agents a systems problem?

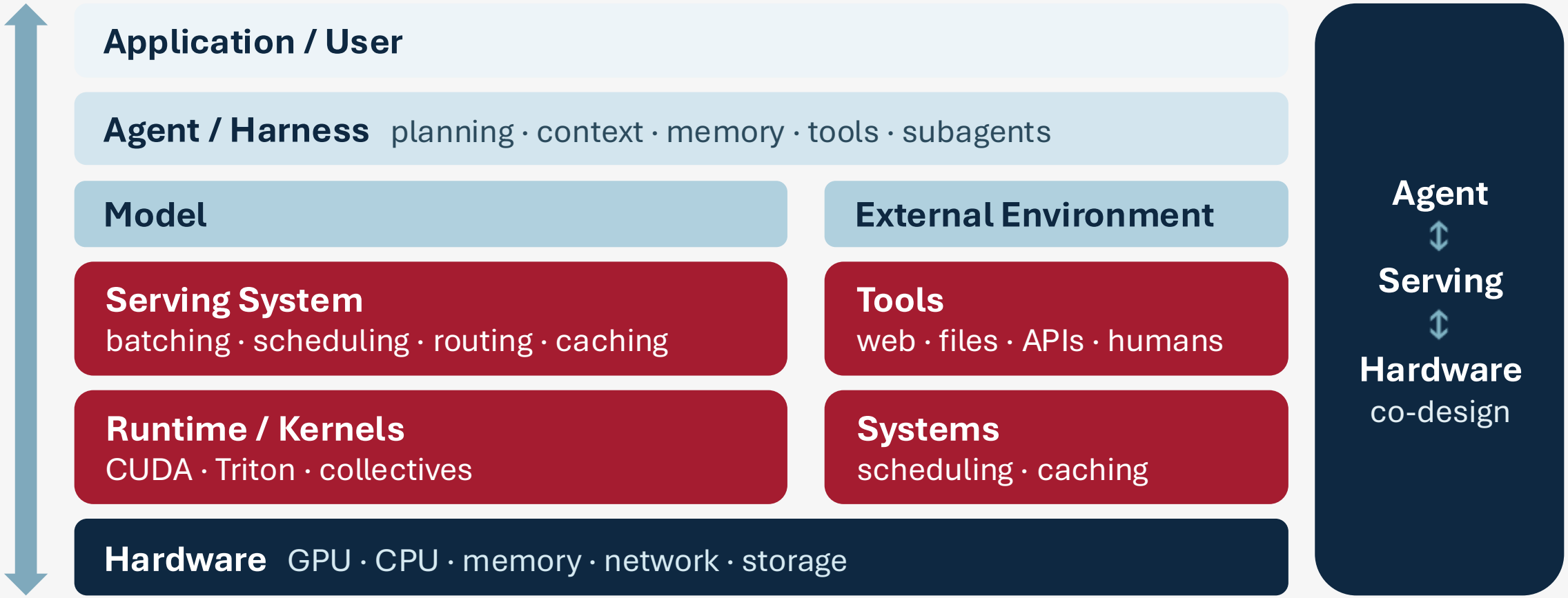
Many turns: the critical path spans GPU, CPU, storage, network and external services.

Traditional LLM serving

Agent serving

Dependencies between steps	**Highly variable step latency**	**Blocking / waiting on tools**
Rapidly growing state	**Concurrency & speculation**	**Failures anywhere in trajectory**

The agentic systems stack



What are we optimizing?

Throughput is the wrong objective for an agent.

maximize **task success**

subject to **latency** | **cost** | **resources** | **reliability** | **security**

Useful, but insufficient

- TTFT / TPOT
- Tool-call latency
- GPU utilization

More tokens/sec can still mean a slower, costlier agent.

What we actually optimize

- Task success / accuracy
- End-to-end latency
- Tokens / task
- **$ / successful task**
- **Energy / successful task**

Agentic systems introduce new systems problems

Every property of an agentic workload names a systems problem this course will study.

Long trajectories
→ state & context management

Huge context
→ memory hierarchy & retrieval

Repeated prefixes
→ prefix caching

Branching reasoning
→ speculative execution

Tool dependencies
→ dependency-aware scheduling

Different models
→ routing

Variable tool latency
→ asynchronous execution

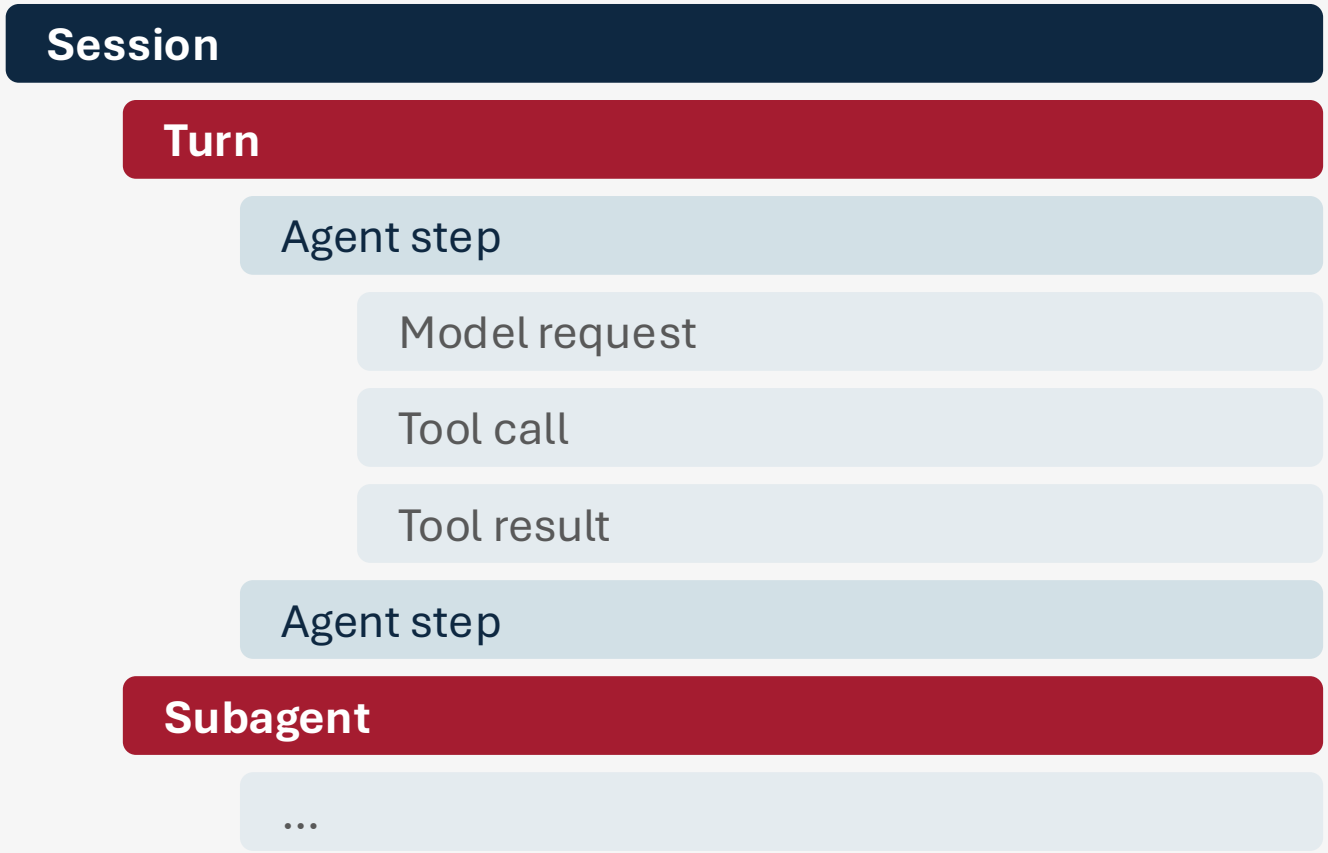
External side effects
→ security & isolation

Multiple agents
→ orchestration & communication

Long-running tasks
→ checkpointing & fault tolerance

Observability is a first-class problem

For agents, the distributed trace becomes a nested trajectory.



You cannot optimize what you cannot observe.

Questions you will think about in this course

1 Why does batching improve throughput but hurt agent latency?

2 When should an agent use a larger model?

3 Where should KV / prefix state live?

4 How should requests from the same agent be scheduled?

5 Can tool calls overlap with inference?

6 When should we spawn a subagent?

7 How do we route across heterogeneous GPUs and models?

8 How do we recover a 30-minute agent when something fails?

9 How do we optimize $ / successful task, not tokens / sec?

By the end of CS2680, you should answer these quantitatively.

What we will not cover in this course

- Training (inference dominates compute demand)
  - data pipeline
  - checkpoint
  - pre-training, fine-tuning, post-training, RL rollout
  - cluster management
- Model architecture design
- Other topics related to inference
  - Communication
  - Power and energy
  - New hardware design

Next class

- LLM basics

Optional reading

AI agents vs. AI assistants vs. bots

	AI agent	AI assistant	Bot
Purpose	Autonomously and proactively performs tasks	Assists users with tasks	Automates simple tasks or conversations
Capabilities	Complex, multi-step actions; learns and adapts; decides independently	Responds to prompts; can recommend actions, but the user decides	Follows pre-defined rules; limited learning; basic interactions
Interaction	Proactive; goal-oriented	Reactive; responds to user requests	Reactive; responds to triggers or commands

https://cloud.google.com/discover/what-are-ai-agents

Key differences: autonomy, complexity, learning

Autonomy

Agents operate and decide independently to reach a goal. Assistants need user input and direction. Bots follow pre-programmed rules.

Complexity

Agents handle complex tasks and multi-step workflows. Assistants and bots suit simpler tasks and interactions.

Learning

Agents use machine learning to adapt over time. Assistants may learn somewhat. Bots learn little or not at all.