CSCE 631 – Algorithmic Game Theory meets LLMs

Lecture 5.1: Agent Architectures — LLMs as Decision-Making Agents

Alan Kuhnle

Summer 2026

Texas A&M University

Today's Question

How can a text generator be a **decision-maker**? And what does that have to do with the game theory you just learned?

The plan:

1. **The bridge** — information sets, behavioral strategies, and payoffs from Weeks 1–4 carry over directly to LLM agents.
2. **Design choices** — given a POMDP, what memory, actions, and loop structure does the agent use? We'll walk through ReAct, Reflexion, ToT, and others as different answers to the same design question.
3. **The payoff** — DeepSeek-R1's RL training uses policy optimization, outcome-based payoffs, and emergent strategies under partial observability.

From Text Generator to Decision-Maker

The puzzle: you ask an LLM to debug a program. It can't do it by generating text alone. It needs to:

1. Read the code and error messages (*observe*)
2. Reason about possible causes (*think*)
3. Edit a file or run a test (*act*)
4. See the result (*observe again*)
5. Repeat until the tests pass

That loop — **perceive, reason, act, observe, repeat** — is what makes something an *agent*.

> *The moment a system acts on an environment and conditions future behavior on observations, it is an agent — and we already have a formalism for that.*

From Poker to Code: The Same Mathematics

Poker player (Week 3)		**LLM coding agent**
Doesn't see opponent's cards — only hand + betting history	→	Doesn't see the full codebase state — only compiler output + test results
Decision rule maps history to action (fold, call, raise)	→	Policy maps observation history to action (edit file, run tests, search docs)
Behavioral strategy: distribution over actions at each info set	→	**Policy**: $\pi(a_t \mid h_t)$ — next-token distribution conditioned on context window
Payoff: chips won	→	**Reward**: $+1$ when tests pass

Both are sequential decisions under partial observability.

The POMDP Bridge to Weeks 1–4

POMDPs are the single-agent specialization of the extensive-form games from Weeks 1–4:

Weeks 1–4 concept		**POMDP equivalent**
Information sets (Week 3)	$\rightarrow$	Observation histories h_t
Behavioral strategies (Week 3)	$\rightarrow$	Policies $\pi(a_t \mid h_t)$
Payoff functions (Week 1)	$\rightarrow$	Reward signal $R(s, a)$
Regret minimization (Week 4)	$\rightarrow$	Policy optimization

The LLM's **context window** implements h_t. The next token sequence is a_t. The reward is task success, user satisfaction, or alignment.

The formalism from Weeks 1–4 describes these systems directly.

POMDP Tuple Recap

A **Partially Observable Markov Decision Process**: $\mathcal{M} = (S, A, O, T, R)$

- S — state space (world + agent)
- A — action space (tool calls, text)
- O — observations (tool outputs, messages)
- $T(s' \mid s, a)$ — transition dynamics
- $R(s, a)$ — reward function

Example: coding agent

S: codebase + test results
A: edit file, run tests
O: compiler errors, output
R: $+1$ when tests pass

Partial observability requires **history conditioning**: $\pi(a_t \mid h_t), \quad h_t = (o_1, a_1, \ldots, o_t)$

Any tool-using LLM is a POMDP agent by construction: the **context window** implements h_t; the next token sequence is a_t — whether or not the agent interleaves reasoning.

Part 2: Design Choices for POMDP Agents

What memory, what actions, what loop structure?

Each architecture is a different answer to the same design question.

Three Design Dimensions of a POMDP Agent

Given a POMDP, the agent designer must choose:

1. **Memory:** what does the agent remember beyond the current context?

Type	Role	Concrete example
Working	Current context	The file you're editing + last 5 compiler errors
Episodic	Past experiences	"Last week I fixed a null pointer by checking the constructor"
Semantic	World knowledge	Python lists have `.append()`, not `.push()`
Procedural	Learned skills	Writing binary search without thinking about it

Taxonomy from Sumers et al. (2023), "Cognitive Architectures for Language Agents."

Actions and the Decision Loop

2. **Actions:** what can the agent do at each step?

Action type	What it does	Concrete example
Grounding	Interact with tools/env	Run the test suite, read errors
Retrieval	Access memory	Look up function signature in docs
Reasoning	Internal thought	"The null is on line 42"
Learning	Update memory	"Remember: check constructors first"

3. **Loop structure:** how does the agent decide what to do next?

$$\textbf{Observe} \rightarrow \textbf{Retrieve/Reason} \rightarrow \textbf{Decide} \rightarrow \textbf{Act} \rightarrow (\text{repeat})$$

Each architecture we'll see is a different answer to: *what memory, what actions, what loop?*

Precursors: Chain-of-Thought & Self-Consistency

Chain-of-Thought (Wei et al., 2022):

- Single-trajectory rollout in an implicit internal world model
- Not yet an "agent" — no environment interaction

Self-Consistency (Wang et al., 2023):

- Sample multiple reasoning paths, take majority vote
- Conceptual seed: *exploring multiple trajectories beats committing to one*

These establish the foundation that agent architectures build on:

$$\text{CoT} \rightarrow \text{multiple paths} \rightarrow \text{environment grounding} \rightarrow \text{memory} \rightarrow \text{search}$$

ReAct: Augmenting the Action Space

Yao et al. (2023, ICLR) — Synergizing Reasoning and Acting

Augmented action space: $\hat{A} = A \cup L$

- A — tool calls (grounding actions)
- L — reasoning traces (natural language thoughts interleaved with actions)

The **Observe** $\rightarrow$ **Think** $\rightarrow$ **Act** loop: $\pi(a_t \mid h_t), \quad h_t = (o_1, a_1, \ldots, o_t)$

POMDP design: working memory only. The contribution is *empirical*: interleaving CoT with tool calls outperforms either alone (HotpotQA, ALFWorld, WebShop).

The LLM produces both thoughts and actions from the **same context window**. Each is a natural language sequence; the distinction is semantic, not architectural.

Why the Coupling Works

Approach	Strength	Weakness
CoT only	Planning	Hallucinates facts
Act only	Grounding	No plan, thrashes
ReAct	**Both**	**Best of both worlds**

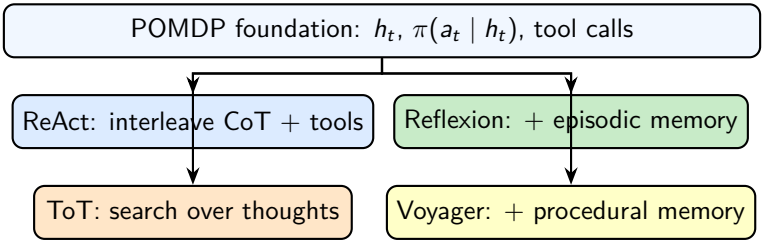
Key empirical results:

- **HotpotQA**: grounding reduces hallucination
- **ALFWorld**: $\sim$71% success ($+$34 pts over Act-only)
- **WebShop**: $+$10 pts over baselines

Design Choices on the Same POMDP

Any tool-using LLM is already a POMDP agent: context window $= h_t$, generated tokens $= a_t$.

Each architecture makes **different** choices within that structure:



These are **not layers**. They are independent design choices — some can be combined (e.g., LATS = tree search + grounding).

Reflexion: Verbal Memory as Policy Improvement

Shinn et al. (2023, NeurIPS)

Three roles:

- **Actor** M_a — generates trajectories
- **Evaluator** M_e — scores outcomes
- **Self-Reflection** M_{sr} — produces verbal reflections

Verbal reflections stored in **episodic memory** m. Policy becomes: $\pi(a_t \mid h_t, m)$

Example reflection

"I tried fixing the null pointer by adding a null check, but it still failed — the object was never constructed. Next time: verify the constructor is called first."

POMDP design: the policy now conditions on more than the current observation history: $\pi(a_t \mid h_t, m)$. Episodic memory *extends the information set* across episodes.

Reflexion as RL over Contexts

RL Concept	Reflexion Equivalent
Policy update	Appending text to prompt
Value function	Evaluator M_e
Experience replay	Episodic memory m
Gradient step	*None* (no weight updates)

Empirical highlights:

- ALFWorld: +22 pts HotPotQA: +20 pts HumanEval: 91% pass@1

Key insight: "Reinforcement learning" via natural language, without ever touching model weights.

Voyager: Self-Evolving Agents

Wang et al. (2023) — GPT-4 Minecraft agent

Three components — all operating through **natural language**:

- Automatic curriculum (proposes next challenges, in text)
- Skill library (**procedural memory**, stored as code)
- Iterative prompting with self-verification

Results: 3.3× more unique items; skills transfer across worlds.

POMDP design: adds **procedural memory** (a library of reusable strategies). Contrast with Reflexion's **episodic memory** (lessons from past failures). Both expand the agent's information beyond the current episode — but different *kinds* for different purposes.

PRAct: From Reflections to Principles

Liu et al. (2024) — Principled Reasoning and Acting

Augments the agent with explicit natural-language *principles* for each action.

Reflective Principle Optimization (RPO):

$$\text{Execution} \rightarrow \text{Reflection} \rightarrow \text{Principle Optimization} \rightarrow (\text{repeat})$$

Advances verbal RL from **unstructured episodic memory** (Reflexion) to **reusable behavioral constraints**.

A **principle** is a natural language statement of reusable behavioral guidance, e.g., "When searching for products, always check availability before comparing prices." Contrast with Reflexion's episode-specific reflections.

On WebShop: PRAct-B (batch RPO) outperforms Reflexion (0.5904 vs. 0.5539 with GPT-3.5).

Tree of Thoughts: Search over Internal States

Yao et al. (2023, NeurIPS)

Each partial solution is a node: $s = [x, z_1, \ldots, z_i]$. The LLM serves two roles:

- **Generator**: proposes successor thoughts
- **Evaluator**: scores promise $f(n)$ of each thought

Search strategies: BFS, DFS, beam search.

Not token-level search. ToT searches over **natural language thoughts** — the LLM evaluates their *semantic* promise, not token probabilities.

POMDP design: expands the **action space** from a single reasoning trace to a *tree*. Memory stays the same (working); the innovation is in **how the policy explores** — branching and evaluating, like game-tree search.

Game of 24: The Power of Search Structure

Task: combine four numbers with $+, -, \times, \div$ to make 24.

Method	Success
Standard CoT	4%
ToT (BFS, b=5)	**74%**

Thought at one node

"$8 \times 3 = 24$, need to zero out 4 and 6. Try $6 - 4 = 2$, then $24/2 = 12$ — dead end."

The advantage is **structural**: branching, evaluating, backtracking. Each node is a **natural language thought**, not a token. Classical search primitives (A*, MCTS) become dramatically more powerful when the heuristic function is an LLM.

BFS with $b = 5$ to depth 3: $\sim$155 LLM calls per problem. The structural advantage comes at substantial compute cost.

LATS: Language Agent Tree Search

Zhou et al. (2023) — combines tree search with environment grounding.

Integrates **MCTS** into the agent loop: LLM-powered value functions, self-reflections as feedback, external environment interaction during search.

Key difference from ToT: each candidate action is *executed* in the environment. ToT is purely internal — LATS grounds it.

Key result: LATS with GPT-3.5 nearly doubles ReAct on HotPotQA (0.61 vs. 0.32 EM).

POMDP design: combines an expanded **action space** (tree search) + **grounding** (environment execution) + **episodic feedback**. Composes across all three design dimensions.

RAP and AdaPlanner

RAP (Hao et al., 2023): LLM as explicit world model + MCTS

- Separates policy (what action) from model (what happens)
- Both operate on **natural language state descriptions**
- LLaMA-33B + RAP beats GPT-4 + CoT by 33% on Blocksworld

AdaPlanner (Sun et al., 2023): generates **natural language plans** and refines them based on environment feedback.

Planning type	System	What the LLM produces
Open-loop	CoT	A single reasoning trace (text)
Closed-loop, fixed	ReAct	Interleaved thoughts + actions
Closed-loop, refine	AdaPlanner	A revisable plan (text)

POMDP design: restructures the **decision loop** — separates planning from execution. The world model acts as **semantic memory** for lookahead.

He et al. (2024): One Formal Lens

ICML 2024. Formalizes LLM agents as a hierarchical POMDP with **Planner–Actor–Reporter** decomposition. Three theorems:

1. Pretraining $\approx$ Bayesian Aggregated Imitation Learning (BAIL)
2. Naive imitation $\rightarrow$ **linear regret**
3. ε-greedy exploration $\rightarrow$ **sublinear regret**

Example: planning a trip

Planner: "fly to Chicago." Actor: books the flight. Reporter: "flight booked, departs 3pm."

Assessment: Exploration is formally necessary — a real contribution. But ReAct, Reflexion, and ToT were developed independently and do not depend on this formalism.

Brittle Foundations of ReAct

Verma et al. (2024) — ReAct's performance is **highly sensitive** to:

- Prompt format and exemplar similarity
- Minor perturbations that preserve semantic content

Teaching Message

These architectures are powerful but **fragile**. The gap between benchmark performance and robust deployment remains wide.

Example: prompt sensitivity

"Find the bug in this code" vs. "Please help me locate the defect in the following implementation" — same task, substantially different results.

The Design Space: POMDP Choices Compared

Every architecture we've seen makes different choices within the **same** POMDP framework:

Architecture	Memory	Action space	Loop	Key innovation
ReAct	Working only	Reason + Ground	Linear	Interleaving CoT with tool calls
Reflexion	+ Episodic	Reason + Ground	Multi-episode	Verbal RL without weight updates
Voyager	+ Procedural	Reason + Ground + Learn	Multi-episode	Skill library (code as memory)
ToT	Working only	Reason (tree)	Branching	Search over thoughts, not tokens
LATS	+ Episodic	Tree + Ground	Branching	Grounded tree search
RAP	+ Semantic	Reason + Plan	Plan-Execute	World model separates planning

The pattern: each architecture is a different **behavioral strategy** for the same underlying POMDP — different answers to *what memory, what actions, what loop?*

Part 3: RL-Trained Reasoning

What happens when reasoning moves inside the model?

The design questions stay the same. The answers change.

DeepSeek-R1: RL-Trained Reasoning

DeepSeek-R1 (2025): trained with **outcome-based RL** (GRPO), after a brief cold-start on curated reasoning examples.

Course concept	DeepSeek-R1 equivalent
Policy optimization (Wk 2–4)	GRPO updates the model's strategy based on outcome rewards
Payoff function (Wk 1)	$+1$ correct, 0 otherwise — must discover which reasoning strategies yield high payoffs
Behavioral strategy (Wk 3)	Emergent CoT: model *learned* step-by-step reasoning because it maximizes expected payoff
Partial observability	The "aha moment": spontaneous backtracking and self-verification

Weeks 1–4 applied: policy optimization drives GRPO, payoff functions define the reward, behavioral strategies emerge as CoT, and partial observability explains backtracking.

The Internalization Shift

The POMDP design questions haven't changed. But some answers moved **inside** the model via RL training:

2023 (External architecture)	2025–26 (Internalized via RL)
CoT prompting	Native CoT (DeepSeek-R1, o-series, Claude thinking)
Self-Consistency	Internal verification + beam search
Reflexion loops	Spontaneous self-correction ("aha moment")
Plan-and-Solve	Model plans internally before executing

What's left for the architecture layer?

- **Branching search** — MCTS gives +23% on SWE-bench (SWE-Search, ICLR 2025)
- **External verification** — compilers and test suites beat self-verification
- **Multi-model orchestration** — strong reasoner + fast executor (RP-ReAct, 2025)

The question shifts: "how does the model reason?" → "when do we let it reason, and when do we override?"

Learned Dual-Process Reasoning

The governance question includes: **how much thinking is the right amount?**

- **Dualformer** (Su et al., ICLR 2025): trains on randomized reasoning traces — sometimes full CoT, sometimes just the answer. The model learns *when* to think deeply vs. answer fast.
- **System-1.x** (Saha et al., ICLR 2025): autonomously switches between cheap heuristic planning and expensive search-based planning at each step.
- **Reasoning on a Spectrum** (Ziabari et al., 2025): mismatches between reasoning style and task structure degrade accuracy — evidence for *why* governance matters.

Key insight: The dual-process dichotomy is most powerful when **learned** rather than hard-coded — the model adapts its cognitive budget to task difficulty.

Production patterns: Srinivasan (2026) formalizes the stochastic-deterministic boundary (SDB) — a contract between model outputs and system actions. Oh & Gobet (2025) identify the prefix dominance trap (∼20% accuracy loss) and propose Monitor-Generate-Verify (MGV).

Bridge to Lectures 5.2 and 5.3

Lecture 5.2 — From single agent to multiple agents:

- Debate as a signaling game
- Does adding agents help or amplify shared errors?

Lecture 5.3 — Agent tools with flipped objectives:

- Adversarial search for inputs that break safety constraints
- The attacker's state space includes the defender's unknown policy
- The CoT itself is an attack surface (Zhao et al., 2025)

The POMDP vocabulary carries over:

The same formalism describes cooperative agents, debating agents, and adversarial agents — only the game structure changes.

References i

[1] Wei et al. (2026). *Agentic Reasoning for Large Language Models.* arXiv:2601.12538.

[2] Li (2025). *A Review of Prominent Paradigms for LLM-Based Agents: Tool Use (Including RAG), Planning, and Feedback Learning.* LREC-COLING 2024.

[3] Luo et al. (2025). *Large Language Model Agent: A Survey on Methodology, Applications and Challenges.* arXiv:2503.21460.

[4] Ferrag et al. (2025). *From LLM Reasoning to Autonomous AI Agents: A Comprehensive Review.* arXiv:2504.19678.

[5] Sumers et al. (2023). *Cognitive Architectures for Language Agents.* arXiv:2309.02427.

References ii

[6] Yao et al. (2023). *ReAct: Synergizing Reasoning and Acting in Language Models.* ICLR 2023.

[7] Shinn et al. (2023). *Reflexion: Language Agents with Verbal Reinforcement Learning.* NeurIPS 2023.

[8] Yao et al. (2023). *Tree of Thoughts: Deliberate Problem Solving with Large Language Models.* NeurIPS 2023.

[9] Zhou et al. (2023). *Language Agent Tree Search Unifies Reasoning, Acting, and Planning in Language Models.* ICML 2024. arXiv:2310.04406.

[10] He et al. (2024). *From Words to Actions: Unveiling the Theoretical Underpinnings of LLM-Driven Autonomous Systems.* ICML 2024.

References iii

[11] Verma et al. (2024). *On the Brittle Foundations of ReAct Prompting for Agentic Large Language Models.* arXiv:2405.13966.

[12] Liu et al. (2024). *PRACT: Optimizing Principled Reasoning and Acting of LLM Agent.* arXiv:2410.18528.

[13] Wang et al. (2023). *Voyager: An Open-Ended Embodied Agent with Large Language Models.* arXiv:2305.16291.

[14] Hao et al. (2023). *Reasoning with Language Model is Planning with World Model.* EMNLP 2023.

[15] Sun et al. (2023). *AdaPlanner: Adaptive Planning from Feedback with Language Models.* NeurIPS 2023.

References iv

[16] Wei et al. (2022). *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models.* NeurIPS 2022.

[17] Wang et al. (2023). *Self-Consistency Improves Chain of Thought Reasoning in Language Models.* ICLR 2023.

[18] DeepSeek-AI (2025). *DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning.* arXiv:2501.12948.

[19] Molinari & Ciravegna (2025). *Reason-Plan-ReAct: A Reasoner-Planner Supervising a ReAct Executor for Complex Enterprise Tasks.* arXiv:2512.03560.

[20] Antoniades et al. (2025). *SWE-Search: Enhancing Software Agents with Monte Carlo Tree Search and Iterative Refinement.* ICLR 2025.

References v

[21] Dong et al. (2025). *Meta-R1: Empowering Large Reasoning Models with Metacognition.* arXiv:2508.17291.

[22] Su et al. (2025). *Dualformer: Controllable Fast and Slow Thinking by Learning with Randomized Reasoning Traces.* ICLR 2025.

[23] Saha et al. (2025). *System-1.x: Learning to Balance Fast and Slow Planning with Language Models.* ICLR 2025.

[24] Ziabari et al. (2025). *Reasoning on a Spectrum: Aligning LLMs to System 1 and System 2 Thinking.* arXiv:2502.12470.

[25] Srinivasan, V. (2026). *A Methodology for Selecting and Composing Runtime Architecture Patterns for Production LLM Agents.* arXiv:2605.20173.

[26] Oh, N. & Gobet, F. (2025). *Monitor-Generate-Verify (MGV): Formalising Metacognitive Theory for Language Model Reasoning*. arXiv:2511.04341.

[27] Zhao et al. (2025). *Chain-of-Thought Hijacking*. arXiv:2510.26418.