Module 5: Advanced Topics & Autonomous Agents

CSCE 631 — Algorithmic Game Theory meets LLMs

Week 5: June 23 (Mon) – June 29 (Mon, last day of classes)

Contents

Learning Objectives

By the end of this module, you should be able to:

1. Explain how Deep CFR replaces tabular regret storage with neural network function approximation and describe the two-network architecture (advantage network + strategy network).
2. Frame LLM-based agents as acting in a POMDP and map agent components (memory, tools, planning) to the POMDP formalism.
3. Analyze multi-agent debate as an extensive-form game, connecting debate protocols to cheap-talk signaling games and identifying when debate improves factual accuracy.
4. Model red-teaming as a two-player zero-sum game and compare co-evolutionary frameworks (MAGIC, Purple Agent, ACE-Safety, SSP, RvB) using game-theoretic equilibrium concepts (SPNE, Stackelberg).
5. Synthesize the connections across the entire course: how normal-form games, regret minimization, extensive-form games, CFR, abstraction, and modern LLM applications form a coherent intellectual arc.

1 Lecture 18: Deep CFR (59 min)

Tabular CFR, as developed in Modules 3–4, stores counterfactual regrets explicitly for every information set. This works beautifully in games of moderate size—Kuhn poker, Leduc Hold'em, even abstracted versions of limit Hold'em—but it collides with a fundamental scalability wall when applied to games with billions of information sets. Full no-limit Texas Hold'em, for instance, has on the order of 10^{161} game states; even Libratus's aggressively abstracted game had approximately 10^{12} information sets, requiring terabytes of storage and a supercomputer. Deep CFR addresses this by replacing the tabular regret and strategy representations with neural networks that generalize across information sets.

1.1 The Scalability Wall

The core problem is storage. Tabular CFR maintains, for each information set I and each action a, a running sum of counterfactual regrets $R^T(I,a) = \sum_{t=1}^{T} r^t(I,a)$. When the number of information sets exceeds what fits in memory, we can either abstract the game (as in Module 4) or approximate the regret function itself. Deep CFR takes the second path: it trains a neural network to predict regrets as a function of information-set features, enabling generalization to unseen states.

1.2 Two-Network Architecture

Definition: Deep CFR Training Loop

Deep CFR maintains two neural networks:

1. **Advantage network** V_θ: estimates the counterfactual regret advantage

$$\hat{r}_i(I,a) = v_i(I,a) - v_i(I)$$

   at each information set I, where $v_i(I,a)$ is the counterfactual value of playing action a and $v_i(I)$ is the expected counterfactual value under the current strategy. Trained on sampled trajectories stored in a reservoir buffer.
2. **Strategy network** Π_ϕ: trained to predict the *average strategy* over all iterations. Since

CFR's convergence guarantee applies to the average strategy (not the last iterate), this network is the final output.

The training loop proceeds as follows:

1. Sample game trajectories via external sampling.
2. Compute sampled counterfactual values along those trajectories.
3. Store (I, a, r) tuples—information set, action, regret—in the advantage buffer.
4. Periodically retrain V_θ on the buffered data.
5. Use V_θ to derive the current strategy via regret matching.
6. Train Π_ϕ on the current strategy profile to approximate the running average.

The advantage network approximates what tabular CFR would compute exactly: at each information set, the regret for each action relative to the expected value. By training on sampled trajectories rather than enumerating the full tree, Deep CFR avoids the exponential tree traversal of vanilla CFR. The strategy network is essential because reconstructing the average strategy from scratch would require re-traversing the entire history of play—a prohibitively expensive operation.

1.3 Reservoir Sampling and Iteration Weighting

A fixed-size buffer cannot store all training examples from all iterations. **Reservoir sampling** maintains a representative subset: each new example replaces a random existing example with probability proportional to the buffer capacity divided by the total number of examples seen so far. Deep CFR further applies *linear CFR weighting*, which overweights recent iterations (iteration t receives weight proportional to t). This mirrors the linear averaging scheme that accelerates convergence in tabular CFR.

1.4 Variants: Single Deep CFR, Dream, and NFSP

Single Deep CFR (Steinberger, 2019) simplifies the architecture by using a single network for both regrets and strategy. The average strategy is computed as a running average of the regret-matched outputs, reducing the number of networks to train from two to one. This is conceptually simpler and cheaper to train, and performs comparably to full Deep CFR on standard benchmarks, though it forgoes the separate approximation of the average strategy that the strategy network provides.

Dream (Deep Regret minimization with Advantage baselines and Model-free learning) improves on Deep CFR through variance reduction via baseline subtraction. By subtracting an estimated baseline from counterfactual values, Dream reduces the variance of gradient estimates and improves sample efficiency.

Neural Fictitious Self-Play (NFSP) takes a different approach altogether. Rather than approximating CFR, it interleaves two learning processes: a best-response learner (using reinforcement learning) and an average-strategy learner (using supervised learning). The RL component learns the best response to the opponent's current average strategy; the supervised component tracks the evolving average. NFSP is less theoretically grounded in CFR but has proven empirically competitive, particularly in settings where the RL component can leverage deep Q-learning.

1.5 DeepStack

DeepStack takes a hybrid approach: it uses deep neural networks as *value functions at depth limits* during real-time subgame solving. Rather than training offline to approximate the full game,

DeepStack solves subgames in real time but uses a neural network to estimate values at the leaves of its limited-depth search tree. This approach was applied to heads-up no-limit Hold'em and achieved superhuman performance before Libratus, albeit in a more limited testing format. DeepStack connects directly to the subgame-solving techniques from Module 4: the neural network replaces the blueprint strategy as the source of leaf values.

1.6 Important Results

Deep CFR achieved competitive exploitability on Leduc Hold'em and was applied to simplified no-limit Hold'em variants. DeepStack was the first system to achieve superhuman performance in heads-up no-limit Hold'em. However, a fundamental tradeoff remains: neural approximation introduces error that can prevent convergence to an exact Nash equilibrium. Unlike tabular CFR, where exploitability decreases at a provable $O(1/\sqrt{T})$ rate, Deep CFR's convergence depends on the approximation quality of the networks. Bounding this error rigorously remains an open problem.

Common Pitfalls

- Deep CFR's convergence guarantees are *weaker* than tabular CFR's. Function approximation error can accumulate across iterations, and there is no guarantee that exploitability decreases monotonically.
- Do not confuse Deep CFR (which approximates CFR regrets using neural networks) with deep RL in games (which learns policies directly via reward maximization). They have different theoretical foundations: Deep CFR inherits from regret minimization; deep RL inherits from policy-gradient or value-based methods.
- The strategy network provides a direct approximation of the average strategy in standard Deep CFR. Single Deep CFR replaces it with a running average of the regret-matched outputs, which is simpler but relies on the advantage network's accuracy for both current-iteration strategy and the final averaged output.

Connection: Module 3–4

Deep CFR builds directly on the CFR framework from Lectures 12–13 and the subgame-solving and abstraction techniques from Lectures 15–17. The advantage network approximates the same counterfactual regrets that tabular CFR computes exactly. DeepStack's use of depth-limited solving with learned value functions is a neural extension of the safe subgame-solving procedure used in Libratus.

2 Lecture 5.1: Agent Architectures — LLMs as Decision-Making Agents (58 min)

The transition from game-solving algorithms to LLM-based agents may seem like a leap, but it is grounded in the same conceptual framework: an agent must make sequential decisions under uncertainty, choosing actions that maximize long-term value given partial information about the world. This lecture formalizes that connection by casting LLM agents as POMDP solvers and surveying the rapidly evolving landscape of agent architectures.

2.1 LLM Agents as POMDP Solvers

Definition: POMDP Formulation for LLM Agents

An LLM agent operates within a **partially observable Markov decision process** (POMDP), defined by the tuple $(S, A, T, R, \Omega, O, \gamma)$:

- S is the set of **world states**—the true configuration of the environment (web pages, file systems, user intent, etc.).
- A is the **action space**, which includes both *internal actions* (reasoning steps, chain-of-thought) and *external actions* (tool calls, API invocations, text generation).
- $T\colon S \times A \to \Delta(S)$ is the **transition function**, capturing both environment dynamics (how the world changes in response to actions) and the stochastic effects of tool execution.
- $R\colon S \times A \to \mathbb{R}$ is the **reward function**, encoding task success, user satisfaction, or alignment objectives.
- Ω is the set of **observations**—text strings received by the agent (tool outputs, user messages, error logs).
- $O\colon S \times A \to \Delta(\Omega)$ is the **observation function**, governing what the agent sees after taking an action.
- $\gamma \in [0, 1)$ is the **discount factor**.

The agent maintains a *belief state*—a distribution over S conditioned on the observation history—which in practice is represented by the LLM's context window.

This formulation reveals a deep connection to extensive-form games: the agent's context window plays the role of the information set, summarizing all accessible history. The key difference from the poker settings of Modules 3–4 is that the state space, action space, and observation space are all effectively infinite and structured over natural language rather than discrete tokens. The POMDP framing clarifies that every design choice in an agent architecture—what memory to maintain, what tools to expose, what planning procedure to use—corresponds to a choice about how to approximate optimal behavior in this intractably large POMDP.

2.2 The CoALA Framework

The **Cognitive Architectures for Language Agents** (CoALA) framework (Sumers et al., 2024) organizes the agent architecture landscape along two axes: *memory modules* and *action types*.

The four memory modules are:

- **Working memory**: the LLM's context window—limited, volatile, and the site of active reasoning.
- **Episodic memory**: records of past experiences and interactions, enabling the agent to learn from prior attempts.
- **Semantic memory**: factual knowledge stored externally (e.g., retrieved documents, knowledge bases).
- **Procedural memory**: executable skills or code (e.g., a library of tool-use patterns or reusable plans).

The four action types are: *reasoning* (internal chain-of-thought), *retrieval* (querying memory or external sources), *learning* (updating stored knowledge), and *grounding* (interacting with the external environment via tools). CoALA provides a taxonomy for comparing architectures: ReAct emphasizes reasoning + grounding; Reflexion adds episodic learning; Voyager builds procedural memory.

2.3 ReAct

ReAct (Yao et al., 2023) interleaves reasoning ("think") steps and acting ("tool use") steps in an alternating loop. It is the minimal operational agent architecture: it augments the LLM's native action space with tool calls, allowing the model to observe tool outputs and incorporate them into subsequent reasoning. In CoALA terms, ReAct uses working memory and grounding actions, with minimal memory persistence across episodes.

2.4 Reflexion and PRAct

Reflexion (Shinn et al., 2023) adds a self-reflection mechanism after task failure. When the agent fails, it generates a verbal analysis of what went wrong and stores this reflection in episodic memory. On the next attempt, the agent conditions on its prior reflections, implementing a form of policy improvement without gradient updates. The key insight is that natural-language self-critique can substitute for the reward signal in RL, at least for tasks where failures are identifiable.

PRAct extends this idea by distilling reflections into reusable *principles*—generalizable heuristics that transfer across tasks. Where Reflexion stores episode-specific memories, PRAct builds semantic memory from accumulated experience.

2.5 Voyager

Voyager (Wang et al., 2023) is a self-evolving agent deployed in Minecraft. Its defining feature is a persistent *skill library*: when the agent solves a novel sub-task, it encapsulates the solution as a reusable code function and adds it to procedural memory. Over time, Voyager builds an expanding repertoire of skills, demonstrating open-ended learning without human curriculum design. In CoALA terms, Voyager is the canonical example of procedural memory construction.

2.6 Search-Based Reasoning: ToT, LATS, and RAP

Tree of Thoughts (ToT) (Yao et al., 2023) structures LLM reasoning as explicit search over a tree of intermediate thoughts. At each node, the LLM proposes candidate continuations; an LLM-based evaluator scores them; and a search algorithm (BFS or DFS) selects which branches to explore. This transforms single-pass generation into deliberate problem-solving with backtracking.

LATS (Language Agent Tree Search; Zhou et al., 2023) applies Monte Carlo Tree Search to LLM reasoning. The LLM serves as both the policy (proposing actions) and the value function (evaluating states), connecting directly to the MCTS techniques discussed in Lecture 14.

RAP (Reasoning via Planning; Hao et al., 2023) formalizes LLM reasoning as planning with a world model. The LLM is used both to simulate future states (world model) and to evaluate the desirability of those states (value function), enabling lookahead search over reasoning trajectories.

Example: MCTS in LATS vs. Poker MCTS

In poker (Lecture 14), MCTS samples game trajectories and uses outcome-based rollouts to estimate action values. In LATS, the "game tree" is the space of reasoning steps, and the LLM itself replaces random rollouts as the value estimator. The structure is identical—upper-confidence-bound selection, expansion, evaluation, backpropagation—but the domain shifts from cards to thoughts.

2.7 The Internalization Shift

A striking development of 2024–2026 is the **internalization** of reasoning strategies that were previously implemented as external scaffolding. Models like DeepSeek-R1 learn to produce chain-of-thought reasoning not because a prompt instructs them to, but because training (often via RL on reasoning tasks) has made it a learned behavior. This blurs the line between "agent architecture" and "model capability": if the model itself generates multi-step reasoning, tool calls, and self-correction internally, the role of the external scaffold shrinks.

The implications for agent design are significant. As Lanham et al. (2023) argues, the question shifts from "how does the model think?" to "when do we let it think, and when do we override?" Architecture becomes a form of governance—shaping not just what the agent *can* do, but what reasoning patterns it *tends* to follow.

Common Pitfalls

- The POMDP framing is a *conceptual* tool for analyzing agent architectures, not a claim that LLM agents perform Bayesian belief updates. The context window is a lossy, finite approximation of the belief state.
- ReAct, Reflexion, and ToT are not competing alternatives—they operate on different axes (grounding, episodic learning, search) and can be composed.
- The internalization shift does not make external scaffolding obsolete. External memory, tool access, and planning remain essential for tasks that exceed the model's context window or learned capabilities.

Connection: PA2 and Course Project

MCTS in ToT and LATS connects directly to Lecture 14 (MCTS and sampling-based CFR). The agent architecture landscape mirrors the abstraction-vs-search tradeoff from Lecture 16: building a skill library (Voyager) is a form of abstraction, while tree search (ToT/LATS) is explicit computation at decision time. Self-play in Voyager and Reflexion echoes CFR's self-play convergence. Course project topics involving agent architectures should ground their designs in this taxonomy.

3 Lecture 5.2: Multi-Agent Debate and Coordination (39 min)

When multiple LLM agents interact—debating a question, negotiating a plan, or evaluating each other's outputs—we re-enter the domain of multi-player games. This lecture analyzes multi-agent debate through the lens of extensive-form games and cheap-talk signaling, and surveys the growing empirical literature on game-theoretic behavior of LLMs.

3.1 Multi-Agent Debate

In **multi-agent debate** (Du et al., 2023), multiple LLM agents argue about a question over several rounds, each updating its answer after observing the others' arguments. A judge—possibly another LLM or a human—selects the final answer. The protocol varies: agents may debate in pairs, in round-robin fashion, or through a shared forum. The empirical finding is that debate can improve factual accuracy and reasoning quality, though the gains are sensitive to protocol design.

3.2 Debate as an Extensive-Form Game

Each round of debate constitutes a stage in a sequential game. Agents choose arguments (actions); the judge's evaluation determines payoffs. Information is imperfect: agents may not know each other's internal reasoning or the judge's evaluation criteria. The game tree captures the sequential structure—early arguments constrain later ones, and agents can condition their strategies on observed statements.

This framing yields immediate analytical leverage. If agents share the same objective (e.g., both want the correct answer), the game is cooperative and we expect informative equilibria. If agents have misaligned incentives (e.g., one is rewarded for being selected regardless of correctness), the game becomes adversarial and we should expect strategic withholding or deception—exactly the dynamics that cheap-talk theory predicts.

3.3 Cheap-Talk Signaling Games

Definition: Cheap-Talk Signaling Game

A **cheap-talk signaling game** (Crawford & Sobel, 1982) involves:

- A **sender** who privately observes a state $\theta \in \Theta$ drawn from a common prior F.
- A **message space** M from which the sender chooses a costless message m.
- A **receiver** who observes m (but not θ), then takes an action $a \in A$ that determines both players' payoffs.
- Payoff functions $u_S(\theta, a)$ and $u_R(\theta, a)$.

Because messages are costless ("cheap talk"), the sender can say anything without direct penalty. Equilibria range from *fully informative* (when sender and receiver interests are perfectly aligned) to *babbling* (when interests are sufficiently misaligned that the receiver ignores all messages).

LLM debate has cheap-talk structure: agents' messages (arguments) are costless in the sense that generating any particular text carries no exogenous cost. The informativeness of debate therefore depends entirely on incentive alignment. When all agents genuinely seek the correct answer, equilibria can be informative. When an agent is rewarded for persuading the judge regardless of truth, we should expect strategic distortion—the analog of babbling or partial-pooling equilibria in the Crawford–Sobel model.

3.4 Game-Theoretic Behavior of LLMs

A growing body of empirical work studies how LLMs behave when placed in classical game-theoretic settings:

- **Akata et al. (2023)**: tested LLMs in repeated 2×2 games (Prisoner's Dilemma, Stag Hunt, etc.) and found that LLMs exhibit a cooperation bias exceeding Nash equilibrium predictions.
- **Yao et al. (2026)**: systematically confirmed that LLMs cooperate above Nash rates in social dilemmas, even when defection is the dominant strategy.
- **Silva (2024)**: tested LLMs on games requiring mixed-strategy Nash equilibria and found that LLMs struggle to randomize correctly; performance is fragile under small game modifications, even with access to code execution.
- **Lekeas & Stamatopoulos (2026)**: provided mechanistic evidence that LLMs compute Nash-like values internally but *suppress* them in favor of cooperative responses—a "compute but suppress" phenomenon.

These findings have direct implications for multi-agent debate: if LLM agents are systematically biased toward cooperation, debate may produce consensus even when genuine disagreement would be more informative. The cooperation bias may explain Li et al.'s (2025) finding that multi-agent debate often performs no better than majority voting—agents may be *ensembling* rather than genuinely deliberating.

3.5 Debate as Oversight

Definition: Correlated Equilibrium in the Debate Context

When a debate protocol includes a mediator (the judge or a coordination mechanism), the resulting equilibrium can be analyzed as a **correlated equilibrium** (CE). The mediator sends private signals (e.g., role assignments, position assignments) to each debater, and each debater's strategy is a best response given their signal. Formally, let p be a distribution over argument profiles $(a_1, \ldots, a_n)$. This distribution is a CE if for each debater i and each pair of arguments a_i, a_i':

$$\sum_{a_{-i}} p(a_i, a_{-i}) \left[u_i(a_i, a_{-i}) - u_i(a_i', a_{-i}) \right] \geq 0.$$

The mediator can achieve outcomes unreachable by uncoordinated Nash play, potentially extracting more accurate information from the debaters.

Irving et al. (2018) proposed **debate as AI oversight**: two AI agents debate a question, and a human judge—who may lack the expertise to answer directly—selects the more convincing debater. The theoretical claim is that under certain conditions, the equilibrium of the debate game incentivizes truthful argumentation, even when the AI agents are superhuman. Constitutional AI and RLAIF pipelines incorporate debate-like structures, using adversarial or multi-perspective evaluation to improve alignment.

A critical perspective is warranted. Li et al. (2025) found that in practice, multi-agent debate often performs no better than majority voting. The gap between the theoretical promise of debate-as-oversight and its empirical performance remains significant, and closing it requires better understanding of when and how cheap-talk equilibria become informative.

Common Pitfalls

- Do not assume that multi-agent debate automatically improves accuracy. Empirical evidence suggests that gains depend heavily on protocol design: the number of rounds, the judge's capability, and the topic domain all matter.
- LLMs' cooperation bias is a *default behavior*, not an architectural constraint. Self-play and adversarial training can push LLM behavior toward Nash equilibrium.
- Cheap-talk theory assumes rational agents. LLMs are not rational in the game-theoretic sense—they are trained on human-generated text that reflects prosocial norms, which is why they over-cooperate.
- A correlated equilibrium requires a trusted mediator. In debate settings, the judge fills this role, but the judge's own biases and limitations affect which equilibria are reachable.

Connection: PA2

PA2 has you build and experiment with multi-agent debate using the TAMU API. The cheap-talk analysis from this lecture provides the theoretical framework for interpreting your results: Are your debating agents genuinely deliberating (informative equilibrium) or merely ensembling (babbling equilibrium)? The correlated equilibrium concept from Module 1 (Lecture 4) reappears here in the debate context.

4 Lecture 5.3: Red-Teaming as a Two-Player Zero-Sum Game (∼45–50 min)

Red-teaming—the practice of deliberately probing an AI system for vulnerabilities—is naturally modeled as a two-player zero-sum game. The attacker (red team) tries to elicit harmful, unsafe, or policy-violating responses; the defender (the model or its safety system) tries to prevent them. This lecture formalizes the red-teaming interaction, surveys the evolution from static to co-evolutionary approaches, and connects the resulting frameworks to the equilibrium concepts from Modules 1–3.

4.1 Static Red-Teaming

Early red-teaming efforts were **static**: a fixed set of adversarial prompts was generated and tested against a fixed model. Perez et al. (2022) used one LLM to generate adversarial prompts for another. Ganguli et al. (2022) scaled this with human red-teamers. The fundamental limitation is that static attacks hit a ceiling—once the defender is hardened against known attack patterns, the same attacks stop working. In game-theoretic terms, a static red team plays a fixed strategy; the defender can best-respond and neutralize it.

4.2 Co-Evolutionary Red-Teaming

The solution is to make both attacker and defender *adaptive*: as the defender improves, the attacker evolves new attack strategies, and vice versa. This is precisely the dynamic captured by iterative equilibrium computation—both players' strategies converge toward an equilibrium through repeated interaction. Several frameworks instantiate this idea:

- **MAGIC** (Ma et al., 2026): a multi-agent game-theoretic framework that models the red-teaming interaction with subgame-perfect Nash equilibrium (SPNE) guarantees. Both attacker and defender co-evolve simultaneously through self-play.
- **Purple Agent**: a Stackelberg formulation in which the defender moves first (designing safety measures) and the attacker best-responds. This captures the realistic deployment scenario where safety training precedes adversarial probing.
- **ACE-Safety**: uses Monte Carlo Tree Search to guide attack exploration, connecting directly to the MCTS techniques from Lecture 14.
- **SSP**: self-play with reflective experience replay, where the attacker learns from its own past successes and failures to generate increasingly sophisticated attacks.
- **RvB** (Red vs. Blue): a training-free co-evolution approach using prompt-based adaptation. Neither the attacker nor the defender is fine-tuned; instead, they adapt their strategies through in-context learning over successive rounds.

4.3 SPNE vs. Stackelberg Equilibrium

The choice of equilibrium concept matters. MAGIC uses **SPNE**, which models the red-teaming interaction as an extensive-form game: both players reason about each other's strategies at every subgame, and the equilibrium is the strategy profile that is a Nash equilibrium in every subgame. The SPNE guarantee is *pointwise*—the defender must respond safely to every individual prompt, not just on average. Purple Agent uses **Stackelberg equilibrium**, which models a leader-follower structure: the defender (leader) commits to a safety strategy; the attacker (follower) observes this commitment and best-responds.

Example: SPNE vs. Stackelberg in Red-Teaming

Consider a defender choosing between two safety filters (strong, weak) and an attacker choosing between two attack types (known, novel). Under SPNE (sequential co-evolution), the equilibrium requires the defender's strategy to be optimal in every subgame—including against novel attacks. Under Stackelberg (defender commits first), the defender can commit to the strong filter, knowing the attacker will best-respond with novel attacks—but the defender's commitment power may yield a higher safety guarantee. The Stackelberg leader payoff is always at least as high as the Nash payoff.

These equilibrium concepts connect directly to Lecture 11 (Extensive-Form Games II), where SPNE was introduced as the refinement of NE that eliminates non-credible threats. The red-teaming setting provides a compelling modern application: the "non-credible threat" of a weak defender is eliminated by requiring the safety strategy to be robust at every subgame (every possible attack scenario).

4.4 Frontier Lab Methodology

In practice, frontier labs (Anthropic, OpenAI, Google DeepMind) use structured red-teaming methodologies that combine human expertise with automated tools. System cards document known vulnerabilities and mitigations. The NIST AI 600-1 framework provides standardized evaluation criteria. DEFCON-style public events (such as the AI Village red-teaming exercises) expose models to diverse, creative attack strategies from the broader security community. These practices represent the real-world operationalization of the co-evolutionary dynamic—an ongoing cycle of attack, defense, and re-evaluation.

4.5 Expanding Attack Surfaces

As LLMs acquire new capabilities, new attack surfaces emerge. **Tool-augmented LLMs** introduce vulnerabilities such as indirect prompt injection (adversarial content embedded in retrieved documents) and tool hijacking (manipulating the model into misusing its own tools). **Multimodal models** face cross-modal attacks: adversarial images can bypass text-only safety filters by encoding harmful instructions in visual features that the text safety system does not inspect. Each expansion of the action space from the POMDP formulation (Lecture 5.1) is simultaneously an expansion of the attack surface.

4.6 Limitations of the Zero-Sum Model

A pure zero-sum framing of red-teaming has limitations. Real AI safety is not purely adversarial: over-optimizing defense can degrade model utility, making the model so cautious that it refuses

legitimate requests. The appropriate model is a *constrained game* in which the defender maximizes both safety *and* helpfulness, subject to the attacker's adversarial pressure. This tension—safety vs. capability—is the defining challenge of deployed AI safety engineering.

Common Pitfalls

- Static red-teaming provides a lower bound on safety, not a guarantee. A model that passes a fixed battery of adversarial tests may still be vulnerable to novel attacks.
- Do not conflate SPNE and Stackelberg equilibrium. SPNE models sequential co-evolution where both players' strategies must be Nash equilibria in every subgame; Stackelberg assumes the defender commits first and the attacker best-responds. They yield different strategy profiles and different safety guarantees.
- The zero-sum assumption is a simplification. Real-world safety engineering must balance adversarial robustness with model utility, which is a general-sum problem.
- Expanding attack surfaces (tool use, multimodality) mean that red-teaming a text-only model does not certify safety of the tool-augmented or multimodal version.

Connection: Course Project — Topic 6

Topic 6 of the course project (Red-Teaming with Game-Theoretic Search) asks you to formalize the red-teaming interaction and implement a game-theoretic attack strategy. The frameworks discussed in this lecture—MAGIC, Purple Agent, ACE-Safety, SSP, RvB—provide the conceptual foundation. You should choose an equilibrium concept (SPNE or Stackelberg), justify the choice, and implement at least one co-evolutionary loop. The MCTS connection (ACE-Safety, Lecture 14) is particularly suitable for a course project implementation.

5 The Course Arc

This module completes the intellectual arc of CSCE 631. The five modules form a coherent progression from mathematical foundations to modern applications:

1. **Weeks 1–2: Foundational Game Theory.** Normal-form games, Nash equilibrium, maxmin strategies, correlated equilibrium, and regret minimization. These provide the mathematical language—the syntax—for reasoning about strategic interaction.
2. **Week 3: Extensive-Form Games and CFR.** The move from simultaneous to sequential games, information sets, behavioral strategies, and the counterfactual regret minimization algorithm. CFR is the algorithmic core: the bridge between equilibrium theory and practical computation.
3. **Week 4: Abstraction and Real-World Deployment.** Game abstraction (card bucketing, action abstraction), subgame solving, and the Libratus/Pluribus systems. This is where theory meets engineering: scaling CFR to games with 10^{161} states requires principled compression and real-time refinement.
4. **Week 5: Modern Applications.** Deep CFR replaces tabular storage with neural approximation. LLM agents are framed as POMDP solvers with architecture choices mirroring the abstraction-vs-search tradeoff. Multi-agent debate is analyzed as a cheap-talk signaling game. Red-teaming is modeled as a zero-sum game with co-evolutionary dynamics.

The key insight threading all five modules together: the tools from Weeks 1–4—Nash equilibrium, regret minimization, CFR, subgame solving, abstraction—are not merely historical artifacts of poker research. They are the mathematical and algorithmic foundations for understanding and building

modern AI systems. When we analyze whether LLM debate produces truthful equilibria, we use cheap-talk theory from Lecture 4. When we design red-teaming frameworks, we use SPNE from Lecture 11. When we build search-based reasoning agents, we use MCTS from Lecture 14. The game-theoretic lens is not an analogy—it is the correct formalism.

Key Definitions Summary

Term	Definition
Deep CFR	CFR with neural networks replacing tabular regret storage; uses advantage network V_θ and strategy network Π_ϕ
Advantage network	Neural network estimating counterfactual regret advantages $\hat{r}_i(I,a) = v_i(I,a) - v_i(I)$
Strategy network	Neural network approximating the average strategy across all CFR iterations
Reservoir sampling	Fixed-size buffer maintenance via probabilistic replacement of old samples
POMDP	$(S, A, T, R, \Omega, O, \gamma)$; framework for sequential decision-making under partial observability
CoALA	Cognitive architecture taxonomy: working, episodic, semantic, procedural memory + reasoning, retrieval, learning, grounding actions
Cheap-talk game	Signaling game with costless messages; equilibrium informativeness depends on interest alignment
Correlated equilibrium (debate)	Mediator-coordinated distribution over argument profiles with no profitable deviations given signals
Stackelberg equilibrium	Leader-follower game: defender commits to a strategy, attacker best-responds
Co-evolutionary red-teaming	Iterative attacker-defender adaptation converging toward game-theoretic equilibrium

Key Takeaways

1. Deep CFR overcomes the scalability wall of tabular CFR by approximating regrets with neural networks, but at the cost of weaker convergence guarantees.
2. LLM agents are naturally modeled as POMDP solvers; architecture choices (memory, tools, planning) correspond to approximations of optimal behavior in an intractably large POMDP.
3. Multi-agent debate has cheap-talk structure: its effectiveness depends on incentive alignment between debaters, and LLMs' systematic cooperation bias may reduce debate to mere ensembling.
4. Red-teaming is a two-player zero-sum game where co-evolutionary frameworks (MAGIC, Purple Agent, ACE-Safety) use equilibrium concepts (SPNE, Stackelberg) to drive both attacker and defender toward robustness.
5. The full course arc—from normal-form games through CFR, abstraction, and deep learning—provides the mathematical foundations for analyzing and building modern AI systems.

Suggested Reading

- Brown, Lerer, Gross, Sandholm (2019): "Deep Counterfactual Regret Minimization." *ICML*.
- Yao, Zhao, Yu, Du, Shafran, Narasimhan, Cao (2023): "ReAct: Synergizing Reasoning and Acting in Language Models." *ICLR*.
- Yao, Yu, Zhao, Shafran, Griffiths, Cao, Narasimhan (2023): "Tree of Thoughts: Deliberate Problem Solving with Large Language Models." *NeurIPS*.
- Du, Li, Torralba, Tenenbaum, Mordatch (2023): "Improving Factuality and Reasoning in Language Models through Multiagent Debate." *ICML*.
- Irving, Christiano, Amodei (2018): "AI Safety via Debate." *arXiv preprint 1805.00899*.
- Perez, Huang, Song, Cai, Ring, Aslanides, Glaese, McAleese, Irving (2022): "Red Teaming Language Models with Language Models." *EMNLP*.
- Sumers, Yao, Narasimhan, Griffiths (2024): "Cognitive Architectures for Language Agents." *TMLR*.
- Crawford, Sobel (1982): "Strategic Information Transmission." *Econometrica* 50(6).