Lecture 2: Exploitation Algorithms and Safe Opponent Exploitation

CSCE 631 — Intelligent Agents: Computational Game Solving

Alan Kuhnle

November 13, 2025

Today's Roadmap

Goal: Turn opponent models into profitable, robust decisions.

- **Part I:** From models to decisions—computing best responses
- **Part II:** Why naive exploitation fails (model error, drift, abstraction)
- **Part III:** Safety mechanisms—blending & constraints
- **Part IV:** Restricted Nash Response (RNR)—robust exploitation
- **Part V:** Safe subgame exploitation—putting it all together
- **Case study:** Leduc-mini walk-through
- **Practical:** Tuning hyperparameters and common pitfalls

By the end, you will be able to design, implement, and critique opponent-exploitation algorithms that balance profit with robustness.

Quick Recap: Opponent Modeling Foundations

Last lecture we built the *model* of an opponent. Today we use it to *act*.

Key Components from Last Time

- **Opponent models:**
  - Dirichlet/tabular: $\alpha(a|I)$ counts per infoset
  - Parametric: softmax / log-linear policies $\sigma_\theta(a|I)$
- **Public Belief State (PBS):** coarse public information (board, actions) + belief over private types
- **Ranges:** probability distributions over opponent's private information at each PBS
- **Posterior-predictive policy:**

$$\hat{\sigma}_{-i}(a|I) = \mathbb{E}[\sigma_{-i}(a|I) \mid \text{data}]$$

averages over parameter uncertainty.

Recap: Computing a Best Response

Best Response (BR) in a subgame:

- Given $\hat{\sigma}_{-i}$ and ranges ρ_{-i} at current PBS
- Build depth-limited subgame rooted at current state
- Use *backward induction* (dynamic programming on game tree)
- At each node h where we act:

$$v_i(h) = \max_{a \in A(h)} \mathbb{E}_{\text{next } h'}[v_i(h') \mid a, \hat{\sigma}_{-i}]$$

- Complexity: $O(|Z_{\text{subgame}}|)$ terminal nodes

Key point: BR is optimal *if* $\hat{\sigma}_{-i}$ is correct. What if it's not?

Reflection Question

If our opponent model $\hat{\sigma}_{-i}$ has high uncertainty,

should we still play a pure best response?

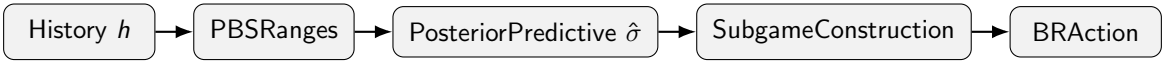
Reflection Question

If our opponent model $\hat{\sigma}_{-i}$ has high uncertainty,

should we still play a pure best response?

Short answer: No—model errors can lead to catastrophic mistakes.
Today's mission: Build *safe* exploitation algorithms that hedge against uncertainty.

The Decision Pipeline: High-Level View



1. **Observe** history h (actions, board cards, etc.)
2. **Update** PBS and ranges ρ_{-i}
3. **Compute** posterior-predictive $\hat{\sigma}_{-i}$
4. **Build** subgame rooted at current PBS
5. **Solve** for BR and select action

Step 1–2: History → PBS and Ranges

Public Belief State (PBS):

- Encodes all public information: board cards, bet sequence, pot size
- Partitions game tree into subgames

Ranges ρ_{-i}:

- Probability distribution over opponent's private cards (or types) consistent with observed actions
- Updated via Bayes' rule:

$$\rho_{-i}(c \mid h) \propto \rho_{-i}(c \mid h_{\text{parent}}) \cdot \hat{\sigma}_{-i}(a \mid I(c, h_{\text{parent}}))$$

- Example (poker): opponent raises $\Rightarrow$ range shifts toward stronger hands

Step 3: Posterior-Predictive Policy

Definition:

$$\hat{\sigma}_{-i}(a|I) = \mathbb{E}_{\theta \sim p(\theta|\text{data})}[\sigma_\theta(a|I)]$$

- Averages over uncertainty in parameters θ
- For Dirichlet model:

$$\hat{\sigma}_{-i}(a|I) = \frac{\alpha(a|I)}{\sum_{a'} \alpha(a'|I)}$$

  where $\alpha(a|I) = \alpha_0(a|I) + N(a|I)$ (prior + counts)
- For parametric models: Monte Carlo samples or moment-matching approximations

Why this matters: Uncertainty in θ translates to uncertainty in action probabilities; we can't ignore it.

Step 4: Subgame Construction

Subgame: portion of the full game tree reachable from current PBS.

- **Root:** current PBS node
- **Depth limit:** computational budget or horizon (e.g., next 2 streets in poker)
- **Frontier (leaf) nodes:** evaluate with hand strength, pot odds, or learned value function
- **Ranges at root:** ρ_{-i} computed in Step 2

Key subtlety: subgame abstractions (bucketing hands, coarse actions) must align with the model's infoset structure.

Step 5: Best Response via Backward Induction

Algorithm sketch:

1. Start at terminal/frontier nodes; assign values (pot size or evaluation function)
2. Work backward through tree:
   - **Opponent nodes:** expected value under $\hat{\sigma}_{-i}$

$$v_i(h) = \sum_{a \in A(h)} \hat{\sigma}_{-i}(a|I(h)) \cdot v_i(h \cdot a)$$

   - **Our nodes:** maximize over actions

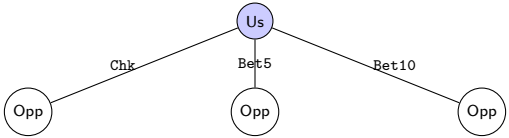
$$v_i(h) = \max_{a \in A(h)} v_i(h \cdot a)$$

3. At root, select action achieving max (or mixed strategy for exploration)

Output: deterministic or mixed BR policy in the subgame.

Example: Toy Poker Subgame

Current PBS: `Flop, Pot=10, Opp checked`. We hold `AK`.
Opponent range: 60% weak pairs, 40% strong hands.
Actions: `Check, Bet 5, Bet 10`.



BR calculation:

- Fold equity, pot odds, opponent's call/fold frequencies $\Rightarrow$ EV per action
- Suppose `Bet 10` has highest EV against $\hat{\sigma}_{-i} \Rightarrow$ BR action

Monitoring Exploitation Profit

Exploitability metric:

$$\text{Profit} = v_i^{\text{BR}}(\hat{\sigma}_{-i}) - v_i(\sigma^{\text{NE}})$$

- $v_i^{\text{BR}}(\hat{\sigma}_{-i})$: our value playing BR vs. modeled opponent
- $v_i(\sigma^{\text{NE}})$: our Nash equilibrium value (baseline)
- Positive profit $\Rightarrow$ we exploit opponent's mistakes
- **Caveat:** profit assumes $\hat{\sigma}_{-i}$ is accurate; if wrong, realized profit can be negative!

Practical tip: track *realized* profit over actual games to detect model drift or error.

Discussion Checkpoint

What could go wrong if $\hat{\sigma}_{-i}$ is inaccurate?

Discussion Checkpoint

What could go wrong if $\hat{\sigma}_{-i}$ is inaccurate?

- **Overconfident bluffs:** opponent doesn't fold as often as model predicts
- **Trap scenarios:** opponent slow-plays strong hands; model underestimates their strength
- **Adaptation:** opponent changes strategy; model becomes stale

Next section: formalize these failure modes and design countermeasures.

Risks of Naive Exploitation: Overview

Naive exploitation: blindly play BR to $\hat{\sigma}_{-i}$ without hedging.

Five Major Failure Modes

1. **Model error:** sparse data or misspecified features $\Rightarrow$ biased $\hat{\sigma}$
2. **Concept drift:** opponent adapts over time; model lags
3. **Statistical overfitting:** extrapolating from few samples
4. **Abstraction mismatch:** infosets in model differ from real game
5. **Multiplayer dynamics:** BR in N-player games can destabilize

We'll unpack each with examples.

Failure Mode 1: Model Error from Sparse Data

Scenario: opponent has checked twice in infoset I; never bet.
Counts: $N(\text{Check}|I) = 2$, $N(\text{Bet}|I) = 0$.

- Dirichlet posterior: $\hat{\sigma}(\text{Bet}|I) \approx \frac{\alpha_0(\text{Bet})}{\alpha_0(\text{Check})+2+\alpha_0(\text{Bet})}$
- If α_0 is small, $\hat{\sigma}(\text{Bet}|I) \to 0$
- BR strategy: *always* bluff (opponent "never" bets $\Rightarrow$ always folds)
- **Reality:** opponent occasionally check-raises; we lose big

Lesson: small sample sizes $\Rightarrow$ high variance in $\hat{\sigma}$ $\Rightarrow$ fragile BR.

Failure Mode 2: Concept Drift

Definition: opponent's true policy $\sigma_{-i}^{(t)}$ changes over time.

- Early in session: opponent plays tight (folds often)
- We exploit by bluffing frequently; accumulate profit
- Opponent notices and adjusts: starts calling/raising more
- Our model $\hat{\sigma}_{-i}$ (trained on early data) is now stale
- BR strategy continues aggressive bluffs $\Rightarrow$ *counter-exploited*

Real-world analogy: AlphaGo playing a human who studies its games and adapts mid-match.
Mitigation preview: use recency weighting, detect shifts, or blend with equilibrium.

Failure Mode 3: Statistical Overfitting

Problem: fitting noise rather than signal.

- Parametric model with many features: log-linear $\sigma_\theta(a|I) \propto \exp(\theta^\top \phi(I, a))$
- Training on limited data $\Rightarrow$ θ overfits to spurious correlations
- Example: "opponent bets big when board has three clubs" (true twice by chance)
- BR exploits this pattern aggressively $\Rightarrow$ fails in new samples

Classical ML lesson: regularization, cross-validation, Bayesian priors help.

Failure Mode 4: Abstraction Mismatch

Abstraction: grouping similar states/actions to reduce complexity.

- Model uses coarse infoset partition (e.g., bucket all medium-strength hands)
- Real game has finer distinctions (suited vs. offsuit, etc.)
- BR computed in abstract game may prescribe action that's poor in real game
- **Example:** abstract infoset "medium pair" lumps 88 and JJ; optimal vs. abstract opponent differs for each

Consequence: value estimates in abstract subgame don't transfer; BR can be exploitable in ground game.

Failure Mode 5: Multiplayer Complications

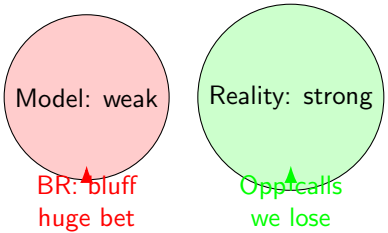
Two-player zero-sum: BR is well-defined; exploits one opponent.
Multiplayer / non-zero-sum:

- No unique BR notion; depends on *all* opponents' strategies
- Aggressive exploitation of one player can trigger retaliation or coalition
- **Example (3-player poker):** over-bluffing against loose player makes tight player adjust; net profit drops
- Equilibrium concepts (correlated equilibrium, coarse correlated) less stable

Implication: safety mechanisms even more critical in multiplayer settings.

Concrete Example: The "Trap" Scenario

Setup: opponent holds AA (pocket aces); flop comes Q92.
Opponent *checks* (slow-play). Model sees one check $\Rightarrow$ infers weak hand.
BR: shove all-in as a bluff.
Reality: opponent snap-calls; we lose stack.



Key insight: single observation $\Rightarrow$ high posterior uncertainty $\Rightarrow$ aggressive BR = risky.

Summary: Why Naive BR Fails

Failure Mode	Root Cause
Model error	Sparse data, misspecified features
Concept drift	Opponent adapts; model stale
Overfitting	Noise vs. signal in small samples
Abstraction mismatch	Coarse infosets $\neq$ real game
Multiplayer	No stable BR; retaliation risk

Common thread: *model uncertainty + environment non-stationarity*.
Solution preview: blend with equilibrium, add robustness constraints, or optimize over uncertainty sets (RNR).

Safety Mechanisms: Motivation

Goal: retain profit from exploitation while bounding downside risk.

Two Main Approaches

1. **Blending:** interpolate between BR and Nash equilibrium
2. **Constraints:** impose value floors or regularization penalties

Philosophy: "Don't put all your eggs in the exploitation basket."
Trade off some EV for robustness to model error.

Approach 1: Blending with Equilibrium

Idea: model opponent as a *mixture* of your estimate and Nash.

$$\sigma_{-i}^{\text{blend}} = \lambda\, \hat{\sigma}_{-i} + (1 - \lambda)\, \sigma_{-i}^{\text{NE}}$$

- $\lambda \in [0, 1]$: exploitation parameter
  - $\lambda = 1$: full trust in $\hat{\sigma}$ (naive BR)
  - $\lambda = 0$: ignore model, play Nash
- Compute BR against $\sigma_{-i}^{\text{blend}}$
- Guarantees: if opponent plays Nash, we also play Nash (safe); if opponent deviates, we exploit partially

Choosing λ: Confidence-Based Tuning

Heuristics for setting λ:

1. **Sample size:** more data $\Rightarrow$ higher λ

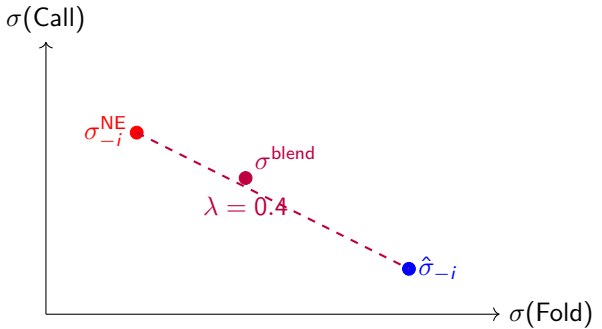
$$\lambda = 1 - \exp\left(-\frac{N_{\text{total}}}{\tau}\right)$$

   τ: threshold parameter (e.g., 100 samples)

2. **Dirichlet concentration:** low $\sum \alpha(a|I) \Rightarrow$ low λ
3. **Validation likelihood:** hold-out log-likelihood of $\hat{\sigma}$; poor fit $\Rightarrow$ lower λ
4. **Infoset-specific:** vary $\lambda(I)$ per infoset based on local data count

Practical note: start conservative ($\lambda \approx 0.3$), increase as model stabilizes.

Blending: Geometric Intuition



Blended opponent lies on line segment between Nash and model estimate.

BR against blend hedges: exploits deviation but stays near equilibrium.

Example: Blending in Leduc Poker

Scenario: infoset I with actions $\{\texttt{Fold}, \texttt{Call}, \texttt{Raise}\}$.
Nash: $\sigma^{\text{NE}} = (0.1, 0.6, 0.3)$
Model: $\hat{\sigma} = (0.3, 0.5, 0.2)$ (opponent folds more)
Blending with $\lambda = 0.5$:

$$\sigma^{\text{blend}} = 0.5 \cdot (0.3, 0.5, 0.2) + 0.5 \cdot (0.1, 0.6, 0.3) = (0.2, 0.55, 0.25)$$

BR decision:

- Against $\hat{\sigma}$: always raise (exploit high fold frequency)
- Against σ^{blend}: raise somewhat less often; occasionally call

Result: lower profit ceiling but much lower risk if model is wrong.

Approach 2: Constrained Decisions

Idea: add penalties or hard constraints to BR optimization.

Regularized Objective

$$\max_{\sigma_i} \left\{ u_i(\sigma_i, \hat{\sigma}_{-i}) - \beta\, D(\sigma_i \,\|\, \sigma_i^{\mathrm{NE}}) \right\}$$

D: divergence (KL, ℓ_2, etc.); $\beta > 0$: safety weight.

- Penalizes deviations from Nash $\Rightarrow$ smooths BR strategy
- Large $\beta \Rightarrow$ stay near Nash; small $\beta \Rightarrow$ aggressive exploitation
- Interpretation: risk-averse decision-making

Value-Floor Constraints (Safe Re-Solving)

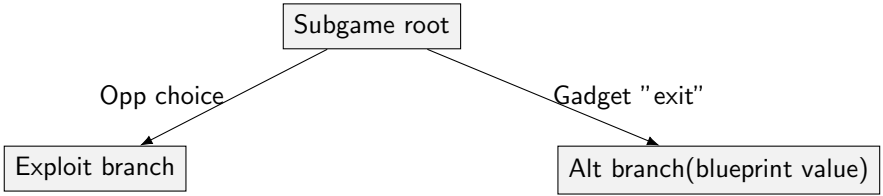
Blueprint strategy: precomputed Nash equilibrium for full game.
Constraint: ensure opponent's value in subgame $\geq$ blueprint value minus slack.

$$v_{-i}(\text{subgame root}) \geq v_{-i}^{\text{blueprint}} - \delta$$

- Prevents giving opponent a "gift" via our exploitation attempt
- $\delta \geq 0$: safety margin (typically small, e.g., 0.01 pot)
- Implemented via *gadget game* or Lagrange multipliers in subgame solver

Benefit: even if our model is wrong, opponent can't punish us beyond blueprint worst-case.

Safe Re-Solving: Gadget Game Diagram



EV $\geq v^{\text{blueprint}} - \delta$

Enforces floor

Opponent can always choose to "exit" and get blueprint value; we optimize subject to that threat.

Comparing Blending vs. Constraints

Aspect	Blending	Constraints
Implementation	Mix opponent models	Add penalty/floor to solver
Tuning	Choose λ	Choose β or δ
Interpretation	"Opponent might play Nash"	"Don't deviate too far"
Guarantees	Bounded exploitability	Value-floor guarantee

Hybrid: can combine both—blend opponent model *and* add regularization.

Choice depends on domain, computational budget, and risk tolerance.

Discussion: When to Use Which?

Blending or constraints—which is better?

Discussion: When to Use Which?

Blending or constraints—which is better?

Blending:

- Simple, interpretable
- Good when Nash strategy is strong baseline
- Less control over worst-case loss

Constraints:

- Explicit worst-case guarantees
- More complex to implement (modify solver)
- Better when blueprint is very strong (e.g., endgame solver)

In practice: many systems use blending for speed, constraints for high-stakes spots.

Restricted Nash Response (RNR): Motivation

Observation: both blending and constraints hedge *implicitly*.
Question: Can we *explicitly* model uncertainty and optimize robustly?

Key Idea

Define **uncertainty set** S_R around $\hat{\sigma}_{-i}$:

$$S_R = \{\sigma_{-i} : D(\sigma_{-i} \,\|\, \hat{\sigma}_{-i}) \leq \varepsilon\}$$

Optimize our strategy against the *worst-case* opponent in S_R:

$$\max_{\sigma_i} \min_{\sigma_{-i} \in S_R} u_i(\sigma_i, \sigma_{-i})$$

Intuition: find strategy that performs well even if opponent deviates from $\hat{\sigma}$ within credible bounds.

RNR Definition (Formal)

Restricted Nash Response (RNR):

$$\sigma_i^{\text{RNR}} \in \arg\max_{\sigma_i} \min_{\sigma_{-i} \in S_R} u_i(\sigma_i, \sigma_{-i})$$

- S_R: convex set of plausible opponent policies (e.g., KL-ball, ℓ_2-ball, Dirichlet credible region)
- $\varepsilon > 0$: radius parameter (larger $\varepsilon \Rightarrow$ more conservative)
- Produces a **robust exploiting** strategy: good EV on average, bounded worst-case

Contrast with BR: BR assumes $\sigma_{-i} = \hat{\sigma}_{-i}$ exactly; RNR hedges over S_R.

Uncertainty Set S_R: KL-Ball Example

KL divergence:

$$D_{\mathrm{KL}}(\sigma \| \hat{\sigma}) = \sum_{a} \sigma(a|I) \log \frac{\sigma(a|I)}{\hat{\sigma}(a|I)}$$

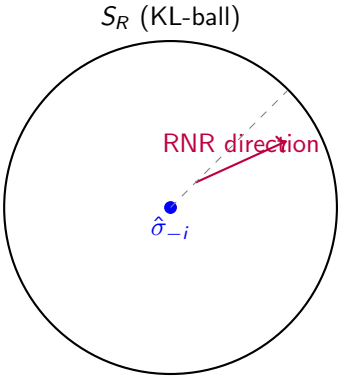
Uncertainty set per infoset I:

$$S_R(I) = \{\sigma(\cdot|I) : D_{\mathrm{KL}}(\sigma(\cdot|I) \,\|\, \hat{\sigma}(\cdot|I)) \leq \varepsilon\}$$

- All distributions within KL-radius ε of $\hat{\sigma}$
- Larger $\varepsilon \Rightarrow$ bigger set $\Rightarrow$ more robust (and less exploitative)

Alternative: ℓ_2 distance, Wasserstein, or Bayesian credible region (e.g., 95% highest posterior density from Dirichlet).

RNR: Geometric Picture



RNR strategy hedges: moves toward exploitation but not as aggressively as pure BR.

Stays robust across all $\sigma \in S_R$.

Solving the RNR Problem

Challenge: min-max optimization over continuous sets.

Approach 1: Iterative Best Response

1. Fix σ_i; solve $\min_{\sigma_{-i} \in S_R} u_i(\sigma_i, \sigma_{-i})$ (convex in σ_{-i})
2. Fix σ_{-i}; solve $\max_{\sigma_i} u_i(\sigma_i, \sigma_{-i})$ (standard BR)
3. Iterate until convergence

Approach 2: Solve Restricted Game with CFR

- Treat restricted set S_R as modified action spaces or payoffs
- Run CFR/MCCFR; converges to approximate RNR equilibrium
- Data-biased robust counter strategies (Johanson & Bowling, 2017)

Choosing Radius ε

Heuristics:

1. **Statistical:** set ε to cover, e.g., 90% posterior credible region from Dirichlet
2. **Validation regret:** tune ε on held-out data; larger ε if validation error is high
3. **PBS-conditional:** vary $\varepsilon(I)$ per infoset based on data count $N(I)$

$$\varepsilon(I) = \varepsilon_0 \cdot \exp\left(-\frac{N(I)}{\tau}\right)$$

   more data $\Rightarrow$ tighter ball

4. **Cross-validation:** grid search over ε to maximize profit minus risk metric

Practical range: $\varepsilon \in [0.01, 0.5]$ in KL divergence for poker domains.

RNR vs. Blending: Comparison

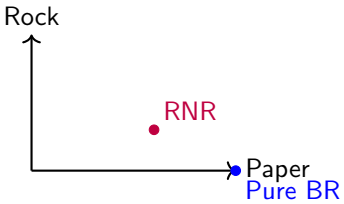
Method	Mechanism	Pros / Cons	
Blending	Linear mix of $\hat{\sigma}$ and σ^{NE}	Simple; no explicit worst-case	**Empirical**
RNR	Max-min over S_R	Explicit robustness; harder to solve	

finding: RNR often smoother profit curves; less variance over opponent realizations.
Cost: more computation (iterative or CFR in restricted game).

Example: RNR in Rock-Paper-Scissors

Setup: opponent's empirical frequencies: Rock=0.5, Paper=0.3, Scissors=0.2.
Uncertainty: S_R is 10% ball around $\hat{\sigma} = (0.5, 0.3, 0.2)$. **Pure BR:** always play Paper (beats Rock).

RNR: consider that opponent might play up to 0.4 Scissors (within ball); hedge by playing some Rock.
Result: RNR mixed strategy: Paper=0.6, Rock=0.3, Scissors=0.1.



Insight: RNR sacrifices some EV for robustness.

Advanced: Data-Biased CFR

Idea (Johanson & Bowling): modify CFR regret updates to restrict opponent to S_R.

- At each iteration t, opponent's strategy constrained: $\sigma_{-i}^{(t)} \in S_R$
- Projection step: if regret-matching pushes outside S_R, project back
- Converges to RNR equilibrium in restricted game

Practical note: projection can be expensive; approximate methods (e.g., truncate probabilities) often sufficient.

Putting It All Together: Safe Subgame Exploitation

Synthesis: combine opponent modeling, RNR, and blueprint safety.

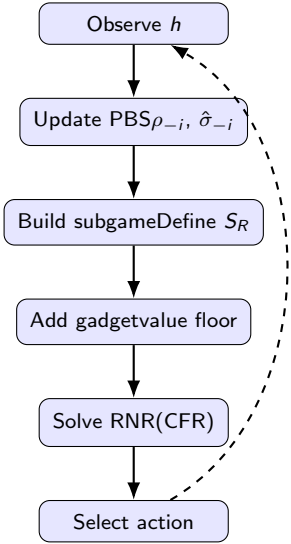
Algorithm Outline

1. Maintain blueprint strategy $\sigma^{\text{blueprint}}$ (precomputed Nash)
2. Online: observe history h; update PBS, ranges ρ_{-i}, and model $\hat{\sigma}_{-i}$
3. Construct subgame at current PBS; define S_R around $\hat{\sigma}_{-i}$
4. Add gadget "exit" node enforcing $v_{-i}^{\text{blueprint}} - \delta$ floor
5. Solve RNR in augmented subgame (CFR or iterative)
6. Execute RNR action; repeat as game progresses

Key properties:

- Exploits deviations within S_R
- Guarantees opponent value $\geq$ blueprint $-\delta$ (safe)
- Adapts to new data each decision point

Safe Subgame: Detailed Pipeline



Loop continues throughout the session, re-solving at each decision node.

Frontier Evaluation in Subgames

Problem: subgame depth-limited; must evaluate non-terminal frontier nodes.

- **Option 1:** use blueprint strategy value at frontier

$$v_i(\text{frontier}) = v_i^{\text{blueprint}}(\text{frontier state})$$

- **Option 2:** learned value function (neural network) trained on blueprint data
- **Option 3:** rollout with simplified policies

Trade-offs:

- Blueprint values: safe but may not exploit opponent
- Learned evaluator: can be biased; needs validation
- Rollouts: expensive but flexible

Continual Re-Solving

Key advantage: opponent model improves over time $\Rightarrow$ better exploitation.

- Each hand/round: new observations $\Rightarrow$ update $\hat{\sigma}_{-i}$
- Re-solve subgame with updated model and ranges
- Strategy adapts to opponent's tendencies

Computational challenge: solving subgames on the fly.
Mitigations:

- Cache solutions for similar PBS (clustering)
- Use shallow subgames (1–2 streets)
- Approximate CFR with fewer iterations

Example: Safe Subgame in No-Limit Hold'em

Scenario: Turn card dealt; we are in-position with top pair.
Opponent has checked. Pot = 100 chips, effective stacks = 200. **Steps:**

1. **PBS:** board texture, pot, stacks
2. **Ranges:** opponent's distribution over holdings after preflop + flop actions
3. **Model:** $\hat{\sigma}_{-i}$ says opponent check-calls 60%, check-folds 30%, check-raises 10%
4. S_R**:** allow $\pm 5\%$ variation in each action frequency
5. **Subgame:** turn + river (2-street lookahead)
6. **Gadget:** opponent can exit to blueprint value (e.g., 48 chips)
7. **RNR solve:** CFR for 1000 iterations $\Rightarrow$ strategy: bet 70% pot 80% of time, check 20%

Result: exploit opponent's high fold frequency but hedge against check-raise risk.

Leduc Poker: Quick Overview

Leduc Hold'em: simplified poker variant for research.

- Deck: 6 cards (2 ranks $\times$ 3 suits), e.g., {J,J,J,Q,Q,Q}
- Two betting rounds: preflop and flop (one community card)
- Actions: Fold, Call, Raise (fixed bet sizes)
- Small game: $\approx$10k decision points

Why Leduc? Tractable for exact Nash computation; good testbed for exploitation algorithms.

Leduc-Mini Exploitation Setup

Opponent model: Dirichlet with prior $\alpha_0(a|I) = c\,\sigma^{\mathrm{NE}}(a|I)$.

- $c = 10$: moderately informative prior
- Online: observe opponent actions, update counts $N(a|I)$
- Posterior predictive:

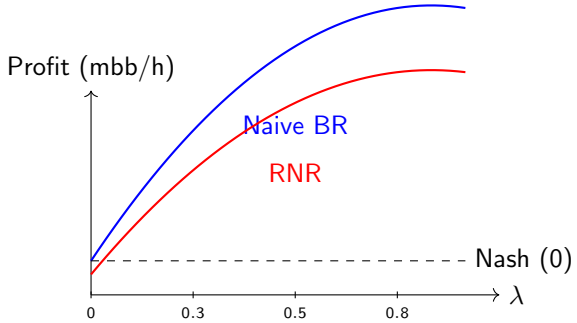
$$\hat{\sigma}(a|I) = \frac{c\,\sigma^{\mathrm{NE}}(a|I) + N(a|I)}{c + \sum_{a'} N(a'|I)}$$

Loop:

1. Play hand; observe opponent actions
2. Update model
3. Form blended opponent: $\sigma^{\mathrm{blend}} = \lambda\,\hat{\sigma} + (1-\lambda)\,\sigma^{\mathrm{NE}}$
4. Compute BR via backward induction (full game tree feasible)
5. Measure profit vs. Nash baseline

Leduc-Mini: Profit vs. λ

Experiment: vary $\lambda \in [0, 1]$; measure average profit over 10k hands vs. fixed non-Nash opponent.



Observation:

- Naive BR (blue): higher peak profit but volatile (variance not shown)
- RNR (red): smoother curve; lower peak but more consistent
- Both beat Nash ($\lambda = 0$) when opponent deviates

Leduc-Mini: Model Calibration

Calibration: does predicted $\hat{\sigma}(a|I)$ match observed frequencies?

- Metric: log-likelihood on held-out actions

$$\ell = \sum_{(I,a)\in\text{test}} \log \hat{\sigma}(a|I)$$

- Higher $\ell \Rightarrow$ better model fit
- **Result:** Dirichlet with $c = 10$ achieves $\ell \approx -0.8$ nats/action (better than $c = 1$ or $c = 100$)

Implication: prior strength c matters; tune via cross-validation.

Leduc-Mini: Robustness to Model Error

Stress test: inject 10% noise into opponent model (perturb $\hat{\sigma}$ randomly).

Method	**Mean Profit**	**Std Dev**
Naive BR ($\lambda = 1$)	+15 mbb/h	25 mbb/h
Blending ($\lambda = 0.5$)	+12 mbb/h	10 mbb/h
RNR ($\varepsilon = 0.1$)	+11 mbb/h	8 mbb/h
Nash ($\lambda = 0$)	0 mbb/h	0 mbb/h

Takeaway: RNR and blending reduce downside risk; critical when model is noisy.

Evaluation Metrics

Assessing exploitation performance:

1. **Profit (mbb/hand):** expected value vs. opponent minus Nash baseline

$$\text{Profit} = v_i(\sigma_i^{\text{exploit}}, \sigma_{-i}^{\text{true}}) - v_i(\sigma^{\text{NE}})$$

2. **Exploitability:** our own worst-case loss vs. best response

$$\text{Exploit}(\sigma_i) = \max_{\sigma_{-i}} u_{-i}(\sigma_i, \sigma_{-i}) - v_{-i}(\sigma^{\text{NE}})$$

3. **Model quality:**
   - Test log-likelihood: $\ell_{\text{test}} = \sum \log \hat{\sigma}(a_{\text{obs}}|I)$
   - Calibration: binned frequency vs. predicted probability
4. **Robustness:** profit variance under opponent perturbations in S_R

Tuning Hyperparameters

Key knobs:

- **Blending λ:** start ≈ 0.3; increase as data accumulates
- **RNR radius ε:** typical range 0.01–0.5 in KL; tune via validation profit
- **Blueprint floor slack δ:** small (e.g., 1% pot); tighter for risk-averse play
- **Prior strength c (Dirichlet):** cross-validate on held-out likelihood
- **Subgame depth:** deeper $\Rightarrow$ better decisions but more compute

Process:

1. Grid search over parameter ranges
2. Evaluate on diverse opponent pool (Nash, exploitable bots, humans)
3. Select config maximizing profit while keeping exploitability below threshold

Practical Tuning Example

Domain: No-Limit Hold'em heads-up, 100 bb stacks.
Baseline: blueprint Nash computed offline. **Tuning grid:**

- $\lambda \in \{0.2, 0.4, 0.6, 0.8\}$
- $\varepsilon \in \{0.05, 0.1, 0.2\}$
- $\delta \in \{0.01, 0.02\}$ pot

Validation: 50k hands vs. 5 scripted opponents (loose, tight, aggressive, etc.).
Result: optimal config $\lambda = 0.5$, $\varepsilon = 0.1$, $\delta = 0.01 \Rightarrow +8$ bb/100 vs. opponents, exploitability $+1.5$ bb/100 (acceptable). **Lesson:** validation on diverse opponents crucial; overfitting to one opponent profile is risky.

Cross-Validation Strategy

1. **Train:** learn $\hat{\sigma}_{-i}$ on 80% of opponent's actions
2. **Validation:** tune λ, ε, δ on 10% (maximize profit)
3. **Test:** final evaluation on held-out 10% (report metrics)

Caution: in online play, concept drift means validation set may not reflect future opponent behavior $\Rightarrow$ use rolling window validation.

Implementation Pitfalls: Overview

Even with sound algorithms, implementation bugs and domain subtleties can sink performance.

Common Pitfalls

1. Data sparsity in rare infosets
2. Concept drift and stale models
3. Abstraction mismatch between model and real game
4. Multiplayer dynamics and retaliation spirals
5. Numerical precision issues in solvers

We'll examine each and discuss mitigations.

Pitfall 1: Data Sparsity

Problem: some infosets visited rarely $\Rightarrow$ high posterior variance.

- Example: deep river spot with specific board runout
- Dirichlet posterior wide; BR may prescribe extreme action

Mitigations:

1. **Hierarchical priors:** share information across similar infosets (e.g., Bayesian hierarchical model)
2. **Parameter sharing:** neural network features pool data
3. **Fallback to Nash:** if $N(I) < \tau_{\min}$, use $\sigma^{\text{NE}}(a|I)$ instead of $\hat{\sigma}$
4. **Stronger prior:** increase α_0 for rare infosets

Pitfall 2: Concept Drift

Problem: opponent adapts; old data misleading.

- Model trained on first 1000 hands; opponent shifts after observing our strategy

Mitigations:

1. **Exponential forgetting:** weight recent observations more:

$$\alpha(a|I) = \alpha_0(a|I) + \sum_t \gamma^{T-t} \mathbb{1}(a_t = a)$$

   $\gamma < 1$ (e.g., 0.95); older data decays
2. **Sliding window:** only use last W hands
3. **Change detection:** run statistical test (e.g., Page-Hinkley) on log-likelihood; reset model if drift detected
4. **Continual learning:** online updates with adaptive learning rate

Pitfall 3: Abstraction Mismatch

Problem: model uses coarse infoset partition; real game has finer distinctions.

- Example: bucket "medium pair" includes 88 and JJ; opponent treats them differently
- BR computed in abstract game may fail in ground game

Mitigations:

1. **Finer abstractions:** increase granularity (costs memory/compute)
2. **Translation layer:** map abstract strategy to ground via probabilistic mapping
3. **Cross-abstraction validation:** test model on ground infosets; adjust if mismatch large
4. **Hybrid approach:** use abstract model for rare spots; detailed model for common spots

Pitfall 4: Multiplayer Over-Exploitation

Problem: exploiting one opponent too aggressively invites others to counter-exploit.

- 3-player poker: we bluff frequently against loose player; tight player adjusts by calling more
- Net result: profit drops or goes negative

Mitigations:

1. **Conservative blending:** lower λ in multiplayer
2. **Joint modeling:** model all opponents' strategies and correlations
3. **Equilibrium anchoring:** blend toward multiplayer Nash or correlated equilibrium
4. **Robust RNR:** expand S_R to cover coalition responses

Pitfall 5: Numerical Precision Issues

Problem: floating-point errors accumulate in iterative solvers (CFR, RNR).

- Small negative probabilities $\Rightarrow$ invalid strategies
- Value estimates drift due to rounding

Mitigations:

1. **Probability clamping:** $\sigma(a|I) = \max(0, \min(1, \sigma(a|I)))$ after each update
2. **Regret floor:** $R^+(a) = \max(0, R(a))$ in CFR
3. **Normalization:** re-normalize strategy at each iteration
4. **Higher precision:** use double precision or arbitrary-precision arithmetic for critical calculations

Debugging Checklist

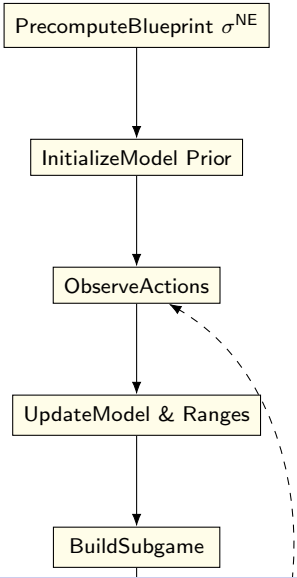
When exploitation performance is poor:

1. **Check model fit:** compute test log-likelihood; compare to baseline
2. **Inspect ranges:** are posterior ranges reasonable after Bayesian updates?
3. **Validate BR:** run BR on known opponent; verify EV matches hand calculation
4. **Profile computation:** ensure subgame solves converge (check CFR iterations)
5. **Test on scripted opponents:** known exploitable bots; should see profit
6. **Ablation study:** disable blending/RNR/constraints one by one; isolate issue

Summary: Key Takeaways

1. **Decision pipeline:** history $\rightarrow$ PBS $\rightarrow$ model $\rightarrow$ subgame $\rightarrow$ BR
2. **Naive BR fails:** model error, drift, overfitting, abstraction mismatch
3. **Safety mechanisms:**
   - Blending: $\sigma^{\text{blend}} = \lambda\,\hat{\sigma} + (1-\lambda)\,\sigma^{\text{NE}}$
   - Constraints: value floors, regularization
4. **RNR:** robust exploitation via max-min over uncertainty set S_R
5. **Safe subgame exploitation:** combine RNR + blueprint safety + continual re-solving
6. **Tuning:** cross-validate λ, ε, δ; monitor exploitability
7. **Pitfalls:** sparsity, drift, abstraction, multiplayer, numerical precision

Practical Workflow Recap



Looking Ahead: Next Lecture

Next time:

- **RNR implementation deep-dive:** projection algorithms, CFR modifications
- **Robust optimization tricks:** dual formulations, efficient constraint handling
- **Advanced opponent modeling:** temporal models, neural network policies, transfer learning
- **Case study:** full No-Limit Hold'em exploitation bot

Preparation:

- Review CFR basics (regret-matching, external regret)
- Read: Johanson & Bowling (2017), "Data Biased Robust Counter Strategies"
- Optional: implement Dirichlet opponent model in toy game (Rock-Paper-Scissors)

Questions & Discussion

Questions?

Discussion prompts:

- When would you prefer blending over RNR, or vice versa?
- How would you adapt these techniques to non-zero-sum games (e.g., negotiation)?
- What additional metrics would you track in a real-money poker bot?

Office hours: TBA. Piazza for async questions.

References & Further Reading

- Johanson, M., & Bowling, M. (2009). "Data Biased Robust Counter Strategies." *AAMAS*.
- Moravčík et al. (2017). "DeepStack: Expert-Level AI in Heads-Up No-Limit Poker." *Science*.
- Brown & Sandholm (2018). "Superhuman AI for Heads-Up No-Limit Poker: Libratus Beats Top Professionals." *Science*.
- Brown & Sandholm (2019). "Solving Imperfect-Information Games via Discounted Regret Minimization." *AAAI*.
- Zinkevich et al. (2007). "Regret Minimization in Games with Incomplete Information." *NIPS*.

Online resources:

- OpenSpiel: `https://github.com/deepmind/open_spiel`
- PokerStrategy wiki on opponent modeling