Regressing Rewards

From RLVR to Agents

CS 2824: Foundations of Reinforcement Learning | Spring 2026

Outline

This talk follows one idea — squared-regression losses for RL —

from foundations (REBEL → A*PO) to off-policy training (OAPL) to agents (KARL).

Part 1: Algorithm Lineage — REBEL → A*PO

Part 2: Why Off-Policy? — practical challenges in real systems

Part 3: OAPL — embracing off-policy training

Part 4: RL for Agents — tool use, credit assignment, learned compaction

Part 5: KARL — putting it all together at scale

Part 1

Algorithm Lineage: REBEL → A*PO

Recall: KL-Regularized RL and REBEL

$$\max_\pi \; \mathbb{E}[\, r(x,y) - \beta \cdot \mathrm{KL}(\pi \,\|\, \pi_{\text{ref}}) \,] \qquad \pi^* \propto \pi_{\text{ref}} \cdot \exp(r/\beta)$$

KL-regularized RL — closed-form optimal policy (from last lecture)

$$\text{REBEL:}\;\; \min_\pi \; \mathbb{E}_{\{y,y' \sim \pi_t\}}\left[\left(\beta(\ln \pi(y|x)/\pi_t - \ln \pi(y'|x)/\pi_t) - (r(y) - r(y')) \right)^2 \right]$$

Why it works: $\beta \ln \pi^*/\pi_{\text{ref}} = r - V^*$, so pairwise difference eliminates V^* (cf. DPO's logistic loss)

REBEL: What Can We Improve?

Two limitations motivate the next algorithm:

1. Needs pairs of generations (y, y') per prompt

   → At least 2 generations per prompt (vs. 1 for A*PO online)

2. Advantage relative to current policy π_t

   → V^{π_t} must be re-estimated every iteration,
   requiring multiple generations or a learned critic

Can we avoid pairs? Can we avoid re-estimating the value every iteration?

A*PO

Optimal Advantage Regression

A*PO: The Core Insight

$$V^*(x) = \beta \cdot \ln E_{y \sim \pi\_ref} [\exp(r(x,y) / \beta)]$$

$V^*(x)$ is fixed — expectation under π_ref, not the current policy

$$\hat{V}^*(x) = \beta \cdot \ln \left((1/N) \Sigma_i \exp(r(x, y_i) / \beta) \right) \qquad (N = 8 \text{ in practice})$$

Estimate offline with N samples from π_ref; filter prompts with 0% pass rate

A*PO: Two-Stage Algorithm

Stage 1 — Offline

Sample N responses per prompt from π_{ref}

Compute $\hat{V}^*(x)$ via log-sum-exp

Fully parallelizable, no gradients

Use vLLM for fast inference

Done once, reused throughout training

Stage 2 — Online

Single generation per prompt from π_t

Regress the optimal advantage:

$$\min \left(\beta \ln \pi(y|x)/\pi_{ref}(y|x) - A^*(x,y) \right)^2$$

where $A^*(x,y) = r(x,y) - \hat{V}^*(x)$

No pairs, no critic, no clipping

A*PO: Theory (Intuition)

Key theoretical message:

> KL-regularized RL with $\beta > 0$ can be solved without explicit exploration, as long as π_ref has non-zero probability of solving each prompt.

No optimism-in-the-face-of-uncertainty needed

No critic, no complex exploration mechanisms

Just regression + the right target (V*)

*Practically: if π_ref has nonzero pass rate on a prompt, A*PO can likely learn to solve it.*

From REBEL to A*PO

	REBEL	A*PO
Generations / prompt	2 (pairs)	1 (online stage)
Value baseline	Cancels via pairs	V* estimated offline
Reference policy	π_t (changes each iter)	π_ref (fixed)
Critic network	No	No
Clipping	No	No

A*PO: Practical Gains

INSERT: Figure 2 from apo.pdf — accuracy vs training time, memory, KL

2× faster than GRPO/PPO · 30%+ less memory · Lowest KL · Comparable accuracy

A*PO Solves the Pairing Problem. But...

A*PO eliminates pairs and the moving baseline.

Stage 2 still assumes on-policy samples: $y \sim \pi_t(\cdot|x)$

In practice, is this assumption ever satisfied?

RL training systems are rarely truly on-policy.

Let's look at real training systems to see why.

Part 2

Why Off-Policy? Practical Challenges in Real Systems

The On-Policy Assumption Rarely Holds

Textbook RL: data comes from your current policy π.

In LLM post-training, this assumption is violated at every level of the stack:

Two major sources of off-policyness:

1. Training-inference mismatch
2. Asynchronous pipelines

(Plus: multi-policy replay buffers, stale data from earlier iterations)

INSERT: Use: LlamaRL Figure 2 (arxiv:2505.24034) — side-by-side sync on-policy vs async off-policy RL. Also: OpenRLHF Figure 1 (arxiv:2501.03262) showing GPU allocation split between rollout engines (vLLM) and actor engines (DeepSpeed)

Source 1: The Training-Inference Mismatch

Same weights, different log-probs.
The inference engine (vLLM/SGLang) and trainer (FSDP/Megatron) use:

- Different attention kernels
- Different precision (BF16 vs FP16)
- Different batching & padding

Qi et al. (2025): BF16 introduces exponentially larger mismatch with longer sequences. For some tokens, $\pi_{vllm}(a) \approx 1$ while $\pi_{fsdp}(a) \approx 0$.

Not a bug — an inherent engineering trade-off in every real system.

INSERT: Use: Qi et al. 'Defeating the Training-Inference Mismatch via FP16' (arxiv:2510.26788) Figure 2 — left: token-level probability divergence, right: sequence-level log-prob ratio distributions. BF16 vs FP16 comparison

Source 2: Async Pipelines

Synchronous RL wastes GPUs — inference devices idle during training, and vice versa.

AReal (Fu et al., 2025):
Fully async generation & training. Inference engine falls many gradient steps behind trainer.

PipelineRL (Piche et al., 2025):
In-flight weight updates — each sequence sees a different θ.

The faster you train, the more off-policy your data becomes.

INSERT: Use: AReal Figure 1 (arxiv:2505.24298) — execution timeline: sync (left) vs async (right) showing idle GPU time. Also: PipelineRL Figure 1 (arxiv:2509.19128) — conventional alternating vs concurrent pipeline

What Happens When You Ignore Off-Policyness?

GRPO / PPO assume on-policy data.

When violated:

- Entropy collapse (DAPO)
- Prosperity → crash (Zheng et al.)
- Gradient norm spikes from IS

Standard patches:

- Token-level IS → ESS collapse
- FP16 matching → helps, not enough
- Delete stale data → wasteful

Errors compound with sequence length

(IS ratio = product of T token ratios).

Root cause: π (trainer) ≠ π_vllm

INSERT: Two-panel figure. Left: DAPO Figure 2(b) (arxiv:2503.14476) — entropy collapse in GRPO. Right: 'Prosperity before Collapse' Figure 1 (arxiv:2510.01161) — GRPO under 256-step staleness: initial gains then sharp collapse. Also: VCPO Figure 1 (arxiv:2602.17616)

Part 3: OAPL

Embracing Off-Policy Training

Key Concept: π_vllm — The Inference Engine Policy

Two copies of the model exist:

π (trainer)
Optimized via gradient descent

π_vllm (inference engine)
Frozen snapshot for generation
Periodically synced with π

Data always comes from π_vllm.

OAPL's key idea:

KL-regularize to π_vllm —
the actual data-generating policy.

INSERT: Use: veRL docs diagram (verl.readthedocs.io/en/latest/algo/rollout_corr.html) showing rollout policy (vLLM) vs actor policy (training) vs reference policy. Or draw: π_vllm generates → buffer D → π trains → periodic sync π_vllm ← π

OAPL: The Objective

$$\max_{\pi} \; \mathbb{E}[\, r(x,y) - \beta \cdot \mathrm{KL}(\pi \,\|\, \pi_{\text{vllm}}) \,]$$

Same as A*PO but with $\pi_{\text{ref}} := \pi_{\text{vllm}}$ — the inference engine is the reference policy

$$\min_{\pi} \; \Sigma \left(\beta \ln \pi(y_i|x)/\pi_{\text{vllm}}(y_i|x) - (r(x,y_i) - \hat{V}^*(x)) \right)^2$$

Single β here; dual-β variant (β_1 for V*, β_2 for KL) discussed in Unifying View

OAPL: The Algorithm

1. Synchronize π and π_vllm
2. For t = 1 to T:
   - π_vllm generates data, stores in buffer D
   - Trainer π optimizes squared regression loss on D
   - Every L steps: sync π_vllm ← π, clear D
3. Between syncs: fully off-policy, fully async

L = 50 for math (periodic sync, modest lag)

Code generation: fully offline for ~400 steps per epoch

No importance weighting. No clipping. No data deletion.

Why Squared Regression Tolerates Off-Policyness

Policy gradients estimate $E_\pi[...]$ — need data from π

- Off-policy data → biased gradient → instability

Regression minimizes squared error:

- When $\hat{V}^* = V^*$, the global minimizer is π^* regardless
- of which distribution generated the data

Jensen's caveat: $E[\hat{V}^*] \leq V^*$ (ln is concave, so

- $E[\beta \ln(1/N \Sigma \exp(r/\beta))] \leq \beta \ln E[\exp(r/\beta)] = V^*$)
- Downward bias is bounded and shrinks with N and pass rate
- Compare: IS ratio variance can be unbounded for long sequences

In OAPL, $\pi\_ref := \pi\_vllm$ — same $\hat{V}^*$ formula, same estimation procedure

- Use π_vllm's log-probs directly — no IS ratios, no MoE router replay

OAPL: Results on Competition Math

INSERT: Figure 1 from oapl.pdf — HMMT 25, AIME 25, BRUMO 25

Outperforms GRPO+IS across all benchmarks and Pass@k metrics

OAPL: Entropy & Test-Time Scaling

INSERT: Figure 3 + Figure 4 from oapl.pdf — entropy curves and Pass@k

GRPO entropy collapses; OAPL maintains it · Pass@k improves for ALL k

OAPL: Code Generation — Extreme Off-Policy

INSERT: Figure 5 from oapl.pdf — LiveCodeBench Pass@k + sample efficiency

Matches DeepCoder with 3× fewer samples · 400+ steps lag · No IS

Can This Work for Multi-Step Agents?

Squared-regression losses handle off-policy single-turn well:

- Math reasoning (A*PO, OAPL)
- Code generation (OAPL, extreme off-policy)

But these are all contextual bandits: prompt → one generation → reward

- (Even though y has token-level sequential structure — Part 2)

Can the same loss family handle multi-step agents?

- Long trajectories with tool use
- Expensive rollouts we want to reuse
- Real deployment data from older models

Part 4

RL for Agents: Tool Use, Credit Assignment, Learned Compaction

Example: A Knowledge Agent Trajectory

Step 1 Agent: search("SARS-CoV-2 mRNA vaccine efficacy")

Step 1 Tool: Retrieved 20 documents...

Step 2 Agent: search("BNT162b2 neutralization Omicron")

Step 2 Tool: Retrieved 20 documents...

...

Step 8 Agent: <summarize findings so far>

...

Step 15 Agent: "Based on the evidence, mRNA vaccines..."

Reward: Judge scores completeness → 0.85 (outcome only)

What Makes Agents Different for RL?

1. Tool use — non-model tokens in the trajectory

Agent calls search, code execution, web browsing

Tool outputs are observed, not generated — non-differentiable

→ Must mask tool tokens from the RL loss

2. Long horizons — trajectories exceed the context window

Report synthesis: ~15 steps. Deep entity search: 200+ steps

Context management becomes a trainable decision (more on this soon)

3. Sparse credit assignment — outcome reward over 50+ steps

Which search query was critical? Did step 3 or step 47 matter?

Current approach: assign outcome reward to entire trajectory

Works in practice; richer per-step signals are an open frontier

Why Off-Policy Matters Even More for Agents

Agentic rollouts are expensive:

- Each rollout = real tool calls (search, retrieval, execution)
- BrowseComp+ rollout: 200 steps, 150K+ characters
- A single rollout can cost 10-100× a math problem

You cannot afford to throw away data.

- Reuse expensive rollouts across multiple gradient updates
- Amortize data collection cost across HP sweeps

This is exactly KARL's setup.

What adaptations does the objective need?

Adapting OAPL for Agents (as done in KARL)

Three practical adaptations:

1. Log-probability masking
   - Mask out tool outputs, retrieved docs
   - Only optimize model-generated tokens

2. Compression segments
   - Split long rollouts at compression points
   - Each segment gets outcome reward

3. E2E compression training
   - Include compression in RL optimization
   - Model learns what to remember
   - Base: 44% → RL-trained: 57%

INSERT: Draw: annotated trajectory showing (1) grayed-out tool tokens (masking), (2) segment boundaries at compression points, (3) outcome reward arrow spanning full trajectory. Cf. KARL paper Section 4.2 for details

Case Study: Why Learned Compaction Matters

Why do naive approaches fail?

- Sliding windows → forget early context that matters later
- Truncation → lose critical details unpredictably
- Prompted summarization → generic, not task-aware

The key insight:

- Compression is not a preprocessing step — it's a decision.
- What to keep depends on what you're optimizing for.
- Train it E2E with outcome reward → task-aware compression.

Independent validation: Cursor's Composer trains self-summarization E2E,

- achieving 50% error reduction over prompted baseline using 1/5 the tokens.

KARL: E2E Compression Works

KARL trains compression E2E
with the same insight:

Compression is an RL-optimized step
(x, y) = (history, summary)
Reward = outcome of full trajectory

Cross-evaluation (KARL Table 6):

GLM search + GLM compress: 44%
GLM search + KARL compress: 54%
KARL search + GLM compress: 46%
KARL search + KARL compress: 57%

RL improves both search AND
compression independently.

INSERT: Use: Table 6 from karl.pdf (cross-evaluation matrix) + KARL Figure 6 zoomed to lifecycle plugin slot showing where compression fires in the harness

Part 5

KARL: Putting It All Together at Scale

KARL: Where Everything Converges

KARL applies A*PO's regression loss, OAPL's off-policy tolerance,
and the agent adaptations you just saw — to a real agentic system
on GLM 4.5 Air (large MoE).

1. Diverse benchmark (KARLBench — 6 tasks, 4 held out)
2. Agentic data synthesis (agents generate training data)
3. Iterative off-policy RL with OAPL (3 iterations, re-synthesizing data each time)
4. Test-time compute scaling

Result: Pareto-optimal on cost-quality vs. Claude 4.6 and GPT 5.2

KARLBench: 6 Tasks, 6 Capabilities

Task	Capability	In/Out of Distribution
BrowseComp+	Constraint-driven entity search	In-distribution
TREC-Biogen	Cross-document report synthesis	In-distribution
FinanceBench	Tabular numerical reasoning	OOD
QAMPARI	Exhaustive entity retrieval	OOD
FreshStack	Procedural reasoning over docs	OOD
PMBench	Fact search over internal notes	OOD

The Agent Harness

INSERT: Figure 6 from karl.pdf — aroll architecture. Also reference: Anthropic 'Brain/Hands' model (anthropic.com/engineering/multi-agent-research-system) as conceptual bridge

Dispatcher · Agent (LLM) · Environment (tools, rewards) · Lifecycle Plugins (compression, budgeting, tool gating)

Agentic Data Synthesis

INSERT: Figures 2-3 from karl.pdf — two-stage synthesis pipeline

Stage I: Agent explores corpus → Q&A · Stage II: Solver rollouts → filtering

KARL: Main Results

INSERT: Table 4 from karl.pdf — KARLBench across all models

Without TTC: matches Sonnet 4.5 · With N=10 parallel: matches Opus 4.6

RL Expands Capabilities, Not Just Sharpening

INSERT: Figures 10-11 from karl.pdf — Max@K curves + pass rate flow

Max@k improves at every k · 37% unsolved → solvable · Max@1 after RL ≈ base Max@8

KARL: Cost-Quality Pareto Frontier

INSERT: Figure 1 from karl.pdf — cost-quality and latency-quality Pareto

Cheaper per query than its own base model — RL-trained model solves tasks in fewer steps, reducing token cost

Test-Time Compute: Complementary to RL

Parallel Thinking:

- N rollouts in parallel → aggregator synthesizes final answer
- On PMBench (N=5), 23.7% of time aggregator beats all rollouts

Value-Guided Search:

- Small value model guides parallel tree search at each step
- Pushes KARL-BCP to 70.4 on BrowseComp+ (exceeds GPT 5's 68.3)

Key finding: RL and TTC are complementary

- Each RL iteration unlocks more gains from test-time compute

The Unifying View

Zooming Out: The Squared Regression Family

The Squared Regression Family

	REBEL	A*PO	OAPL	Kimi K2
Loss	Squared	Squared	Squared	Squared
Advantage	Relative (pairs)	Optimal A*	Optimal A*	Group-relative
Baseline	Cancels via pairs	$\hat{V}^*$ (log-sum-exp)	$\hat{V}^*$ (log-sum-exp)	Mean reward $\bar{r}(x)$
Reference	π_t	π_{ref} (fixed)	π_{vllm} (lagged)	π_{old}
On/Off-policy	On	On	Off	On ($\pi_{old} \approx \pi$)
KL control	β	β (single)	β (single)	τ (single)

Kimi K2: Same Family, Different Design Choices

V* Estimator

$L = \Sigma\ (r(x,y_i) - \bar{r}(x) - \tau \ln \pi(y_i|x)/\pi\_old(y_i|x))^2$

Kimi: mean reward $\bar{r}(x) = (1/K)\ \Sigma\ r(x,y_i)$

→ $\beta_1 \to \infty$ limit of log-sum-exp

Simpler; may suffice at 1T scale

A*PO/OAPL: tunable β_1

$\hat{V}^* = \beta_1 \ln(1/N\ \Sigma \exp(r/\beta_1))$

$\beta_1 \to \infty$: mean · $\beta_1 \to 0$: max

Intermediate β_1: estimates true V*

KL Control

Kimi: single τ controls both

baseline smoothness and KL strength

A*PO/KARL: decouple into β_1, β_2

$L = \Sigma\ (\beta_2 \ln \pi/\pi\_ref - (r - \hat{V}^*))^2$

β_1: V* estimation (in $\hat{V}^*$)

β_2: KL regularization (in loss)

Allows conservative V* (large β_1)

+ aggressive optimization (small β_2)

Wrap-Up

Key Takeaways & Open Problems

Key Takeaways

1. Squared-regression losses are a general, scalable family

   From 1.5B math → 1T MoE (Kimi K2) → large-scale agents (KARL)

2. Real systems are rarely on-policy — embrace it

   OAPL: no IS, no clipping, no data deletion — just regression

   Stable at 400+ steps of lag, 3× more sample efficient

3. RL genuinely expands model capabilities

   Not just sharpening: Pass@k improves for all k

   Multi-task RL generalizes OOD; distillation does not

Open Problems

Credit assignment in multi-step trajectories

- 200-step rollouts with outcome reward only — can we do better?

The agent harness as optimization surface

- Compaction works E2E — what else can we train?
- Query formulation, tool selection, multi-agent orchestration

V* estimation and the β_1/β_2 split

- Is there something principled between log-sum-exp and mean?

Learning from deployment

- Deploy → collect off-policy data → improve → redeploy
- OAPL makes the RL step feasible; reward estimation remains open.

References (1/2)

REBEL — Gao et al., NeurIPS 2024

A*PO — Brantley et al., 2025

OAPL — Ritter et al., 2026

Kimi K2 — Kimi Team, 2025

KARL — Databricks AI Research, 2026

Off-policy challenges & systems:

Qi et al. 'Defeating Training-Inference Mismatch via FP16' (arxiv:2510.26788)

AReal — Fu et al., 2025 (arxiv:2505.24298)

PipelineRL — Piche et al., 2025 (arxiv:2509.19128)

DAPO — Yu et al., 2025 (arxiv:2503.14476)

'Prosperity before Collapse' — Zheng et al., 2025 (arxiv:2510.01161)

VCPO — Huang et al., 2026 (arxiv:2602.17616)

Buffer Matters (BAPO) — Wan et al., 2026 (arxiv:2602.20722)

References (2/2)

Frameworks:

OpenRLHF — Hu et al., EMNLP 2025 (arxiv:2501.03262)

veRL / HybridFlow — Sheng et al., EuroSys 2025 (arxiv:2409.19256)

LlamaRL — Meta, 2025 (arxiv:2505.24034)

Other:

Dr. GRPO — Liu et al., COLM 2025 (arxiv:2503.20783)

GSPO — Zheng et al. (Qwen Team), 2025 (arxiv:2507.18071)

RePO — 2025 (arxiv:2506.09340)

Ross & Bagnell, 'Efficient Reductions for Imitation Learning', AISTATS 2010

Cursor — cursor.com/blog/self-summarization, 2025

ReAct — Yao et al., ICLR 2023 (arxiv:2210.03629)

Anthropic — anthropic.com/research/building-effective-agents

vLLM Blog — blog.vllm.ai/2025/11/10/bitwise-consistent-train-inference.html

Thank You

Questions?