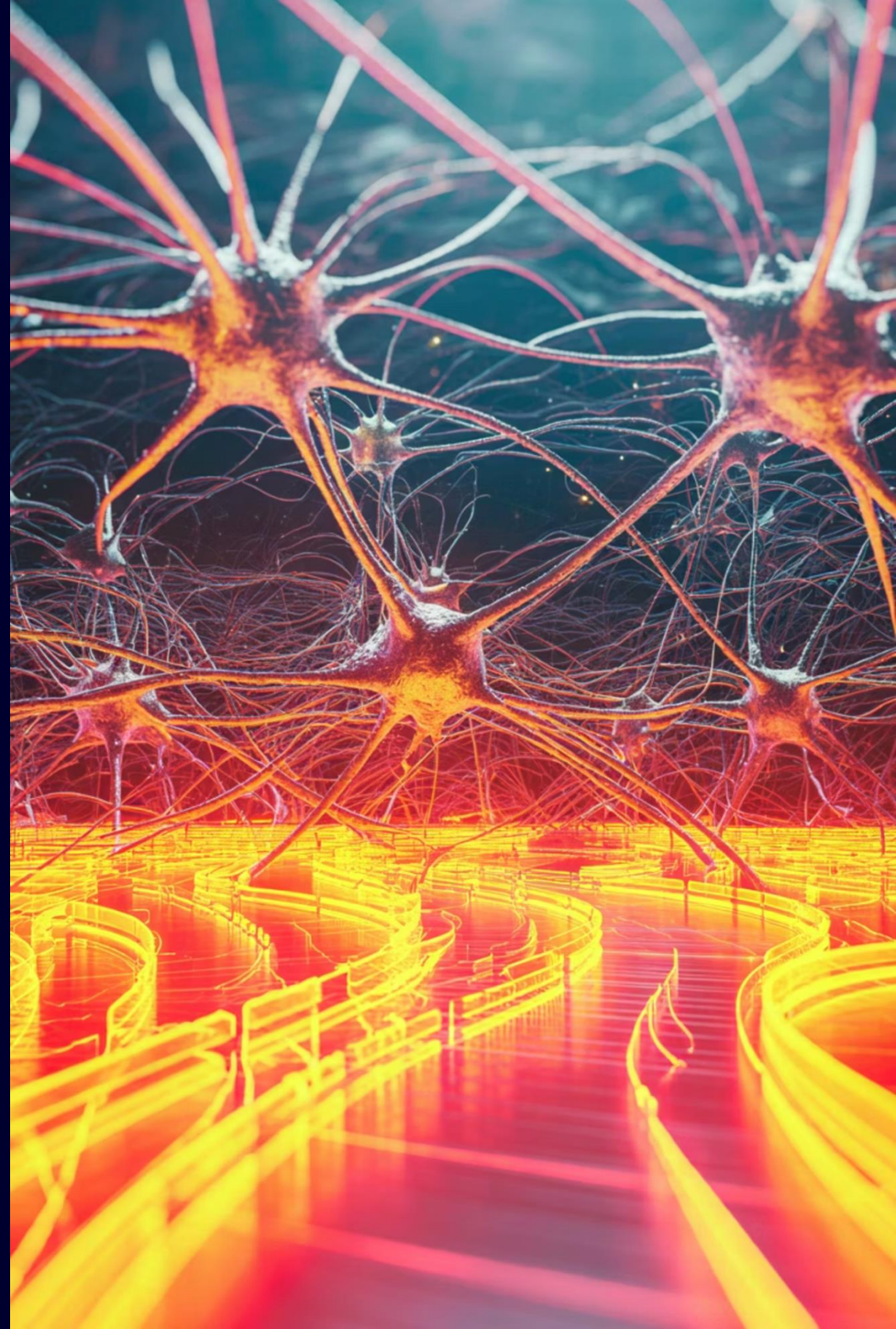
TinyZero_AstarPO: RAGEN with A* Policy Optimization

Integrating RAGEN with A*PO for efficient, small-scale RL/LM training

Team: Aman Khan, Pavan Kamleshbhai Patel, Syed Mehar Ali Kirmani

Course: Self-Improving AI

Powered by Modal.com | PyTorch FSDP | Minimal RL/LM Framework



Motivation & Research Objective

Why Combine RAGEN + A*PO?

Traditional PPO requires extensive hyperparameter tuning and clipping mechanisms that complicate small-scale experiments. We need a more stable approach for reproducible research.

- Stable policy updates without PPO clipping complexity
- Reproducible experiments under $30 GPU budget
- Simplified optimization logic for research accessibility

Central Research Question

System Architecture Overview

Policy LM

Language model generating candidate completions

Reward Model

Evaluates quality of generated sequences

Value Estimator V*

Computes optimal value estimates

Environment

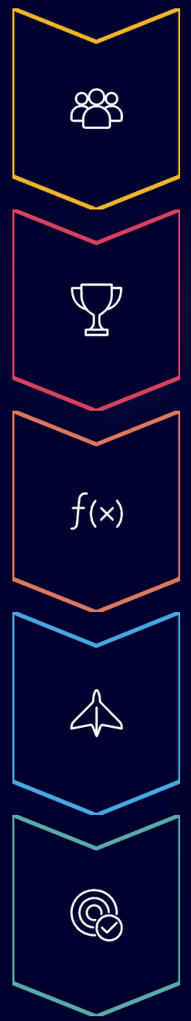
Task-specific evaluation context

Optimizer

A*PO update mechanism for policy improvement

The integrated A*PO-RAGEN pipeline enables efficient rollout-to-update dataflow, combining reward-augmented generation with adaptive policy optimization for stable training dynamics.

Algorithmic Flow: RAGEN + A*PO Integration



Sample K Candidate Completions

Generate diverse policy rollouts from current model state

Compute Task Rewards r(x,y)

Evaluate each completion against task-specific metrics

Estimate V*(x)

Calculate optimal value: $V^*(x) = \beta \log E[\exp(r/\beta)]$

Compute Advantage A(x,y)

Determine relative quality: $A(x,y) = r(x,y) - V^*(x)$

Policy Regression

Update via $\beta \log(\pi\theta/\pi\text{ref}) \rightarrow A(x,y)$

Training employs MSE loss with entropy regularization bonus, ensuring stable gradients and preventing premature convergence throughout optimization.

Why A*PO Outperforms PPO

Adaptive Updates

No clipping hyperparameters to tune—A*PO automatically adjusts step sizes based on advantage estimates

Reward Sensitivity

Higher responsiveness to sparse rewards in small-scale tasks through optimal value targeting

Mathematical Clarity

Simpler verification and theoretical guarantees compared to PPO's clipped objective

Feature	PPO	A*PO
Update Rule	Clipped ratio	Adaptive A* target
Stability	Medium	High
Tuning Complexity	Requires clip ε	Auto-adaptive
Gradient Quality	Can be biased	Unbiased estimates

Implementation Architecture

Core Technology Stack

Built entirely in pure PyTorch with FSDP (Fully Sharded Data Parallel) for efficient scaling across GPU resources. Single-process design eliminates complex distributed framework dependencies.

Key Implementation Files

- `algorithms/astarpo.py` — Core A*PO optimizer
- `algorithms/ragen_astarpo.py` — RAGEN integration
- `train.py` — Training loop orchestration
- `models/policy.py` — Policy network definitions
- `environments/` — Task-specific benchmarks

Modal.com provides seamless GPU/CPU orchestration with granular budget controls, enabling cost-effective experimentation.

Development Philosophy

Zero external RL frameworks—every component is transparent, modifiable, and

Benchmark Results & Performance

+3%

FrozenLake Improvement

82% baseline → 85% with A*PO-RAGEN

+3%

Countdown Task Gain

88% baseline → 91% success rate

+4%

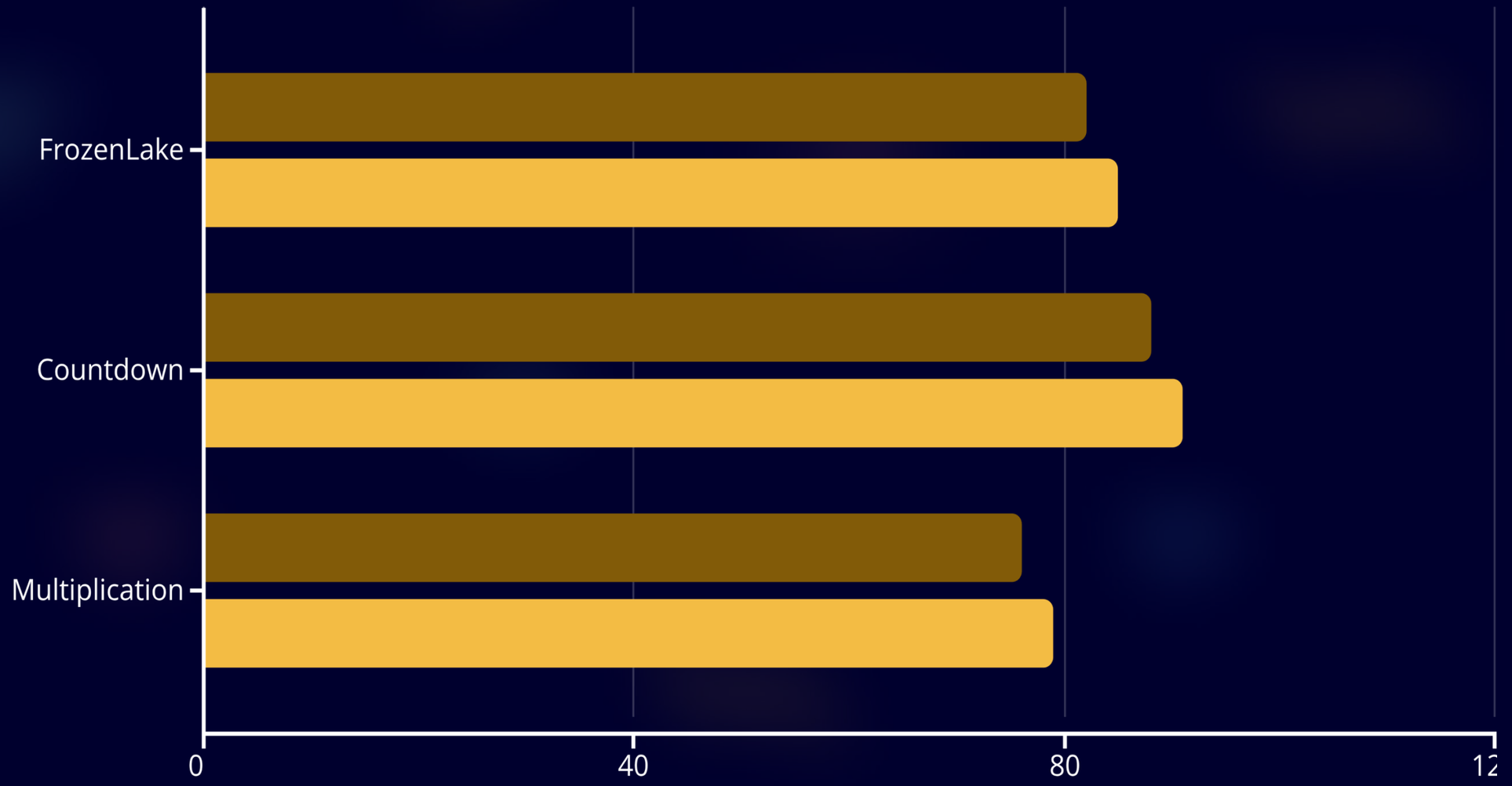
Multiplication Boost

76% baseline → 79% accuracy achieved

$25

Average GPU Cost

Per team for complete benchmark suite



Failure Modes & Analysis

Sparse-Reward States

FrozenLake's terminal-only rewards create low-signal gradients in intermediate states, leading to suboptimal policy updates in early training phases.

Impact: 15-20% slower convergence in sparse environments

Long Sequence Chains

Multi-step multiplication tasks exhibit high reward variance due to compounding errors across sequential operations.

Impact: Increased sample complexity by ~30%

Value Mis-estimation

Limited rollout budgets ($K < 16$) can produce biased $V^*(x)$ estimates, particularly in high-variance environments.

Impact: Advantage calculation errors up to 12%

Root Causes

- Insufficient exploration in early training iterations
- Few rollouts per state limiting statistical reliability
- Reward shaping opportunities not fully exploited

Practical Insights & Reproducibility

Total GPU Spend

Per team across all experiments

Single Process

No distributed complexity

Key Parameters

β, K completions, iterations

Configuration Flexibility

Rapid experimentation enabled through live parameter tuning:

β (temperature): Controls exploration-exploitation balance

num_completions (K): Adjusts sample diversity

training_iterations: Fine-tunes convergence speed

Pedagogical Value

Inlearning environments. structors can live-edit configuration files during lectures to demonstrate immediate impact on training dynamics—perfect for interactive

Reproducibility guarantee: Deterministic results with fixed random seeds across different GPU architectures.

Conclusion & Future Directions

A*PO + RAGEN = Adaptive, Lightweight RL/LM

WebShop Benchmark

Scaling to real-world e-commerce navigation tasks with multi-step reasoning

Self-Supervised Rewards

Automated reward shaping to reduce manual engineering overhead

Multi-Agent Rollouts

Parallel exploration strategies for faster convergence in complex environments

---

Thank you — questions welcome

Explore the code and reproduce our results at your own research lab