Poker Zero Lightning Presentation

Problem Statement and Motivation

Objective

Develop a reasoning model optimized for **no-limit hold'em poker**. Success in poker extends beyond monetary gain—it signifies advancements in:

- **Reasoning under incomplete information**
- **Adversarial decision-making**
- **Strategic adaptation**

Applications

Poker AI has significant implications in:

- **Game Theory**
- **Economic Modeling**
- **Negotiation**

Prior Work

Several notable AI-driven poker models include:

- **Poker Bench** – Trained LLMs to become professional poker players
- **PokerGPT** – Lightweight solver leveraging a large language model
- **Pluribus** – Demonstrated multiplayer, near-GTO strategies

Data

Instruction	Output
You are a specialist in playing 6-handed No Limit Texas Hold'em. The following will be a game scenario, and you need to make the optimal decision.	raise 18
Game Summary:	
- Small Blind: **0.5 chips**	
- Big Blind: **1 chip**	
- Everyone started with **100 chips**	
- Player positions: **UTG, HJ, CO, BTN, SB, BB**	
- **Your Position:** HJ	
- **Your Hand:** [King of Diamond, Jack of Spade]	

Technical Approach

Unsloth

Unsloth is an open-source Python framework that speeds up the process of fine-tuning and using LLMs.

- Optimized PyTorch code
- Handwriting GPU kernels to speed up inference
- Better memory utilization via typecasting
- Allows us to fine-tune smaller models like Qwen 2.5 3B on just a Colab T4 GPU

GRPO

GRPO (Group Relative Policy Optimization) is a reinforcement learning algorithm developed by DeepSeek.

- Trains a model to optimize a reward function instead of training a model solely on next-token prediction (which simply teaches it to mimic data)
- Uses group-based comparisons to improve performance instead of after every trial
- Calculates advantage and updates policy to increase likelihood of better actions
- Uses KL Divergence constraint to prevent drastic changes in policy
- Overall objective maximizes cumulative reward with stable policy updates

LoRA

Low Rank Adaptation (LoRA) is a method for fine-tuning large models efficiently.

- It works by introducing low-rank matrices into pretrained model layers.
- This approach allows significant performance gains with a minimal number of additional parameters.

Mathematical Formulation of LoRA

- **Pretrained Weight Matrix:** Let $W \in \mathbb{R}^{d \times k}$ be a pretrained weight matrix.
- **Low-Rank Decomposition:** LoRA approximates the weight update ΔW as:
  $$\Delta W = BA$$
  where:
- $B \in \mathbb{R}^{d \times r}$
- $A \in \mathbb{R}^{r \times k}$
- $r \ll \min(d, k)$
- **Adapted Weights:** The updated weight matrix becomes: $W' = W + BA$

Implementation Updates

Model Selection

- **Switched to Qwen 2.5 3B Instruct Model** - Migrated from Llama 8B to a smaller but still capable model
- More cost-effective and enables faster iteration cycles
- Better suited for limited compute resources while still exploring LLM potential for poker reasoning

Response Structure Revisions

Implemented structured output format to encourage explicit reasoning:

- Standardized response format with regex: "^(fold|call|raise \d+)$"
- Removed ambiguity by converting "check" to "call" and "bet" to "raise" for consistency
- Makes reasoning process transparent and evaluable
- Critical for both training and human verification

Enhanced Reward Functions

Redesigned reward functions to combat reward sparsity:

- **Format Compliance**: Rewards for properly using defined response format
- **Partial Correctness**: Partial rewards for "raise" values within 20% of the optimal amount
- **Higher Reward Frequency**: More lenient criteria to provide more frequent positive reinforcement
- Reduces training instability from sparse rewards

Self-Play Data Generation

PyPokerEngine Integration

- Implemented self-play data generation using PyPokerEngine
- 6-player poker games where each player is an instance of our fine-tuned LLM
- Convert winning moves into new training examples
- Pipeline: Fine-tune on PokerBench → Generate self-play data → Continue fine-tuning on new data

Training Challenges

- Computational constraints (1.5 hours for 250 GRPO steps on T4 GPU)
- Format conversion between PokerBench and PyPokerEngine outputs
- Unsloth optimizations not compatible with A100 GPUs in Colab

Next Steps

Evaluation Methods

- Implement visualization of rewards throughout training to track improvement
- Compare models fine-tuned on different datasets through head-to-head competition
- Track average chip difference over many games to measure progress

Training Efficiency

- Continue optimizing training pipeline
- Consider exploring other poker bots for competition/evaluation
- Explore integration with other poker frameworks like a modified Pluribus

Thank You!

Questions?