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

Technical Approach

Unsloth and GRPO

Unsloth is an open-source Python framework that speeds up the process of fine-tuning and accessing large language models (LLMs). It does so through the following methods:
Optimized Computation Kernels
Memory Efficiency and Reduced Overhead
Integrated Fast Inference

How We Use It:

GRPO (Group Relative Policy Optimization), 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). Through group-based comparison, scoring, and relative reinforcement, this process helps the model learn the underlying reasoning process and not just the final answer.

PyPokerEngine

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:

  $$\Delta W = BA$$

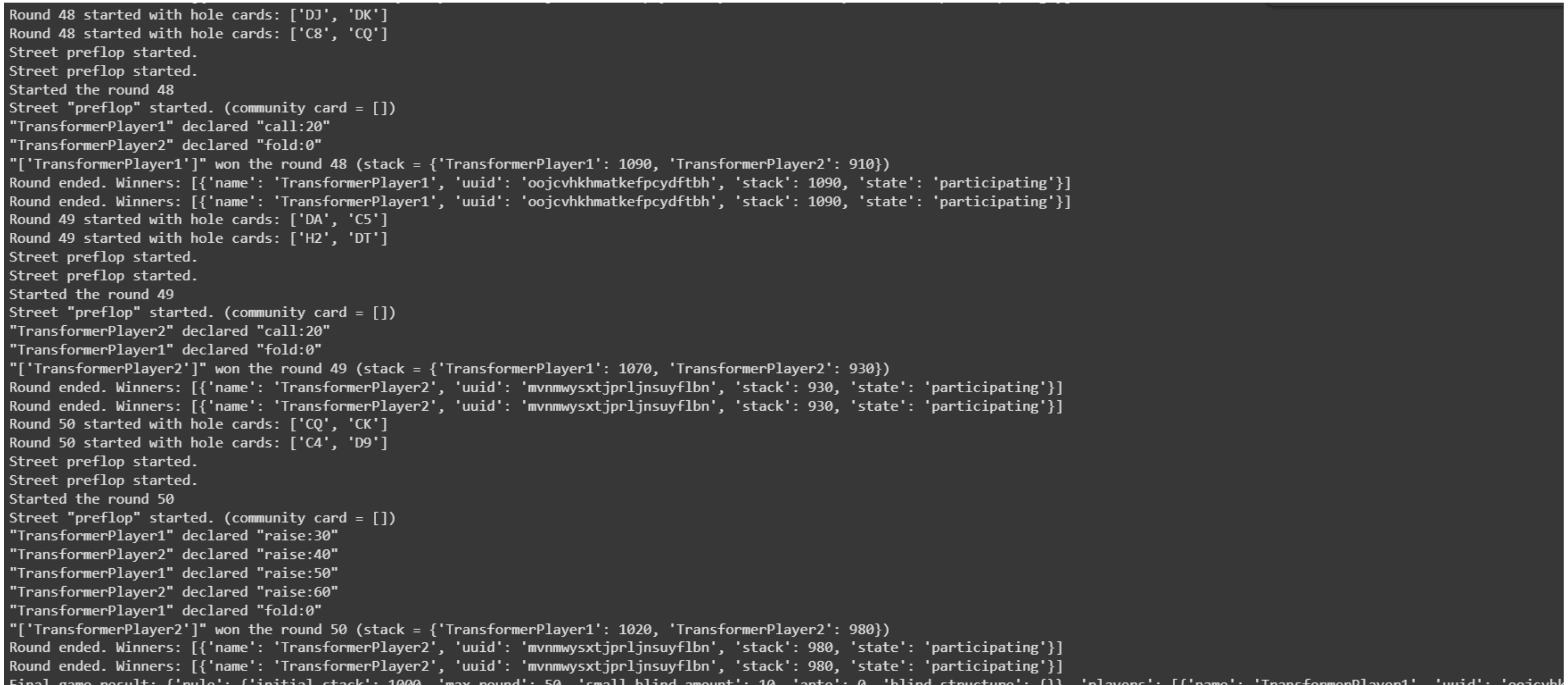
  where:
- $B \in \mathbb{R}^{d \times r}$
- $A \in \mathbb{R}^{r \times k}$
- $r \ll \min(d, k)$
- **Adapted Weights:** The updated weight matrix becomes: $W' = W + BA$

Key Benefits & Applications

- **Parameter Efficiency:** Fine-tune models with fewer trainable parameters.
- **Reduced Computational Cost:** Lower memory and compute requirements.
- **Flexibility:** Easily integrated into various neural network architectures.
- **Wide Adoption:** Applied in large language models and other deep learning systems.

Initial Results

- **Unsloth** Successfully fine tuned our model in Colab
- **PyPokerEngine** Used PyPokerEngine to make our model play against itself



```
Round 48 started with hole cards: ['DJ', 'DK']
Round 48 started with hole cards: ['C8', 'CQ']
Street preflop started.
Street preflop started.
Started the round 48
Street "preflop" started. (community card = [])
"TransformerPlayer1" declared "call:20"
"TransformerPlayer2" declared "fold:0"
"['TransformerPlayer1']" won the round 48 (stack = {'TransformerPlayer1': 1090, 'TransformerPlayer2': 910})
Round ended. Winners: [{'name': 'TransformerPlayer1', 'uuid': 'oojcvhkhmatkefpcydftbh', 'stack': 1090, 'state': 'participating'}]
Round ended. Winners: [{'name': 'TransformerPlayer1', 'uuid': 'oojcvhkhmatkefpcydftbh', 'stack': 1090, 'state': 'participating'}]
Round 49 started with hole cards: ['DA', 'C5']
Round 49 started with hole cards: ['H2', 'DT']
Street preflop started.
Street preflop started.
Started the round 49
Street "preflop" started. (community card = [])
"TransformerPlayer2" declared "call:20"
"TransformerPlayer1" declared "fold:0"
"['TransformerPlayer2']" won the round 49 (stack = {'TransformerPlayer1': 1070, 'TransformerPlayer2': 930})
Round ended. Winners: [{'name': 'TransformerPlayer2', 'uuid': 'mvnmwysxtjprljnsuyflbn', 'stack': 930, 'state': 'participating'}]
Round ended. Winners: [{'name': 'TransformerPlayer2', 'uuid': 'mvnmwysxtjprljnsuyflbn', 'stack': 930, 'state': 'participating'}]
Round 50 started with hole cards: ['CQ', 'CK']
Round 50 started with hole cards: ['C4', 'D9']
Street preflop started.
Street preflop started.
Started the round 50
Street "preflop" started. (community card = [])
"TransformerPlayer1" declared "raise:30"
"TransformerPlayer2" declared "raise:40"
"TransformerPlayer1" declared "raise:50"
"TransformerPlayer2" declared "raise:60"
"TransformerPlayer1" declared "fold:0"
"['TransformerPlayer2']" won the round 50 (stack = {'TransformerPlayer1': 1020, 'TransformerPlayer2': 980})
Round ended. Winners: [{'name': 'TransformerPlayer2', 'uuid': 'mvnmwysxtjprljnsuyflbn', 'stack': 980, 'state': 'participating'}]
Round ended. Winners: [{'name': 'TransformerPlayer2', 'uuid': 'mvnmwysxtjprljnsuyflbn', 'stack': 980, 'state': 'participating'}]
Final game result: {'rule': {'initial_stack': 1000, 'max_round': 50, 'small_blind_amount': 10, 'ante': 0, 'blind_structure': {}}, 'players': [{'name': 'TransformerPlayer1', 'uuid': 'oojcvhk
```

Next Steps

PyPokerEngine

- **More Players** Poker environment is only heads up (2 players). Our goal is to play 6 handed, which would have more interesting applications because heads-up Texas Hold’em is a solved game

Additional Training

- **Self-Play** Model competes against previous versions, gradual improvement and adaptation to exploitative strategies
- **Training Pipeline** Feed data back into our model for reinforcement learning training. Eliminates need for a premade dataset and allows the model to learn optimal strategies by itself

Thank You!

Questions?