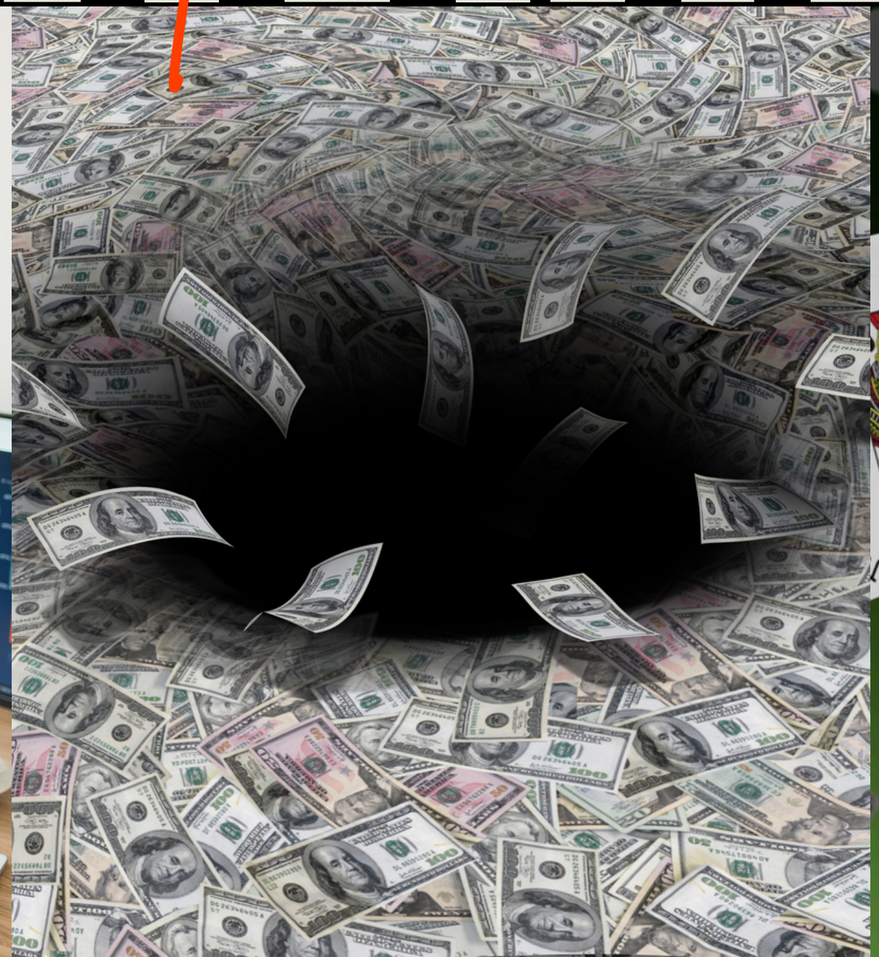
GTO POKER

Presentation by - Arne, Ido & Wes

MOTIVATION

MOTIVIATION

Project



THE “WHY?”

PREVIOUS RESEARCH

LIBRATUS BOT (2017)

latin for balanced
heads-up (1v1) no-limit Texas Hold'em
fixed strategy + counterfactual regret minimization

PLURIBUS BOT (2019)

latin for many
6-player no-limit Texas Hold'em
used reinforcement learning (similar to AlphaZero)

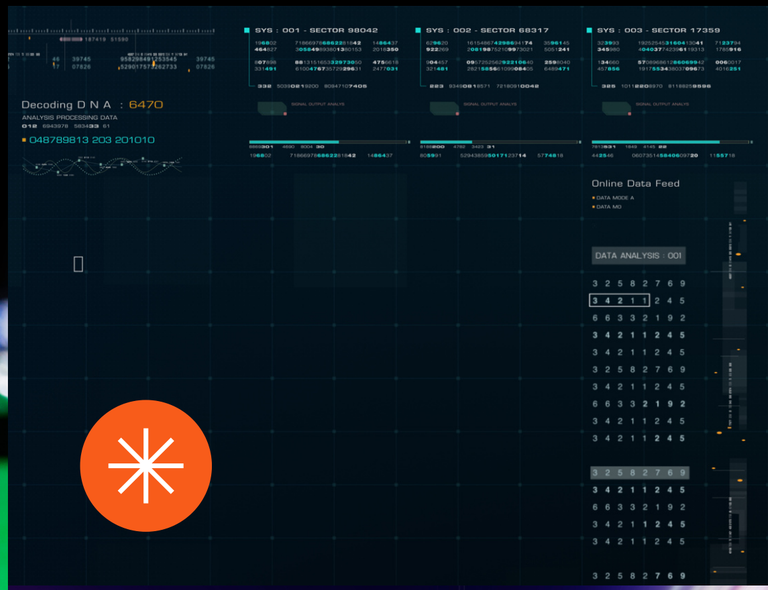
VISION & GOALS

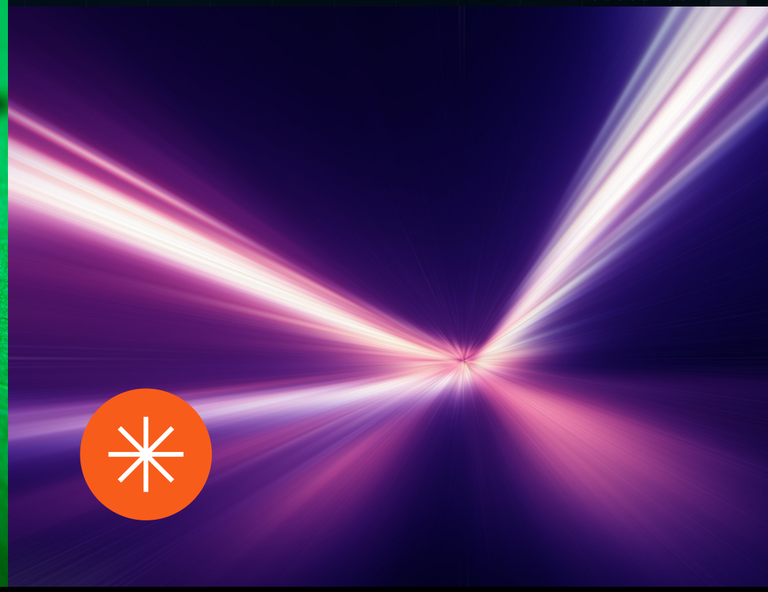
Reinforcement Learning Model

Develop an accurate model that can play end-to-end poker

Fixed Strategy

Play RF Model against a fixed strategy model





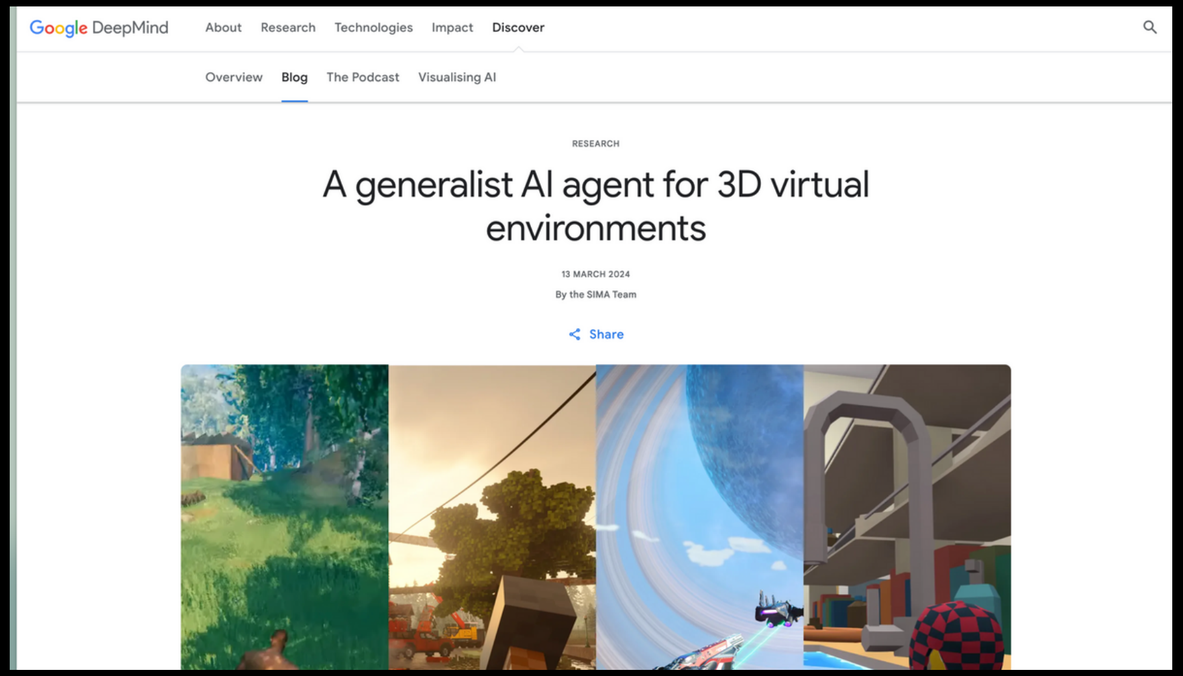
2 CHALLENGES

&

no-limit texas hold’em is not solved &
a game of imperfect information
(unlike chess, go)

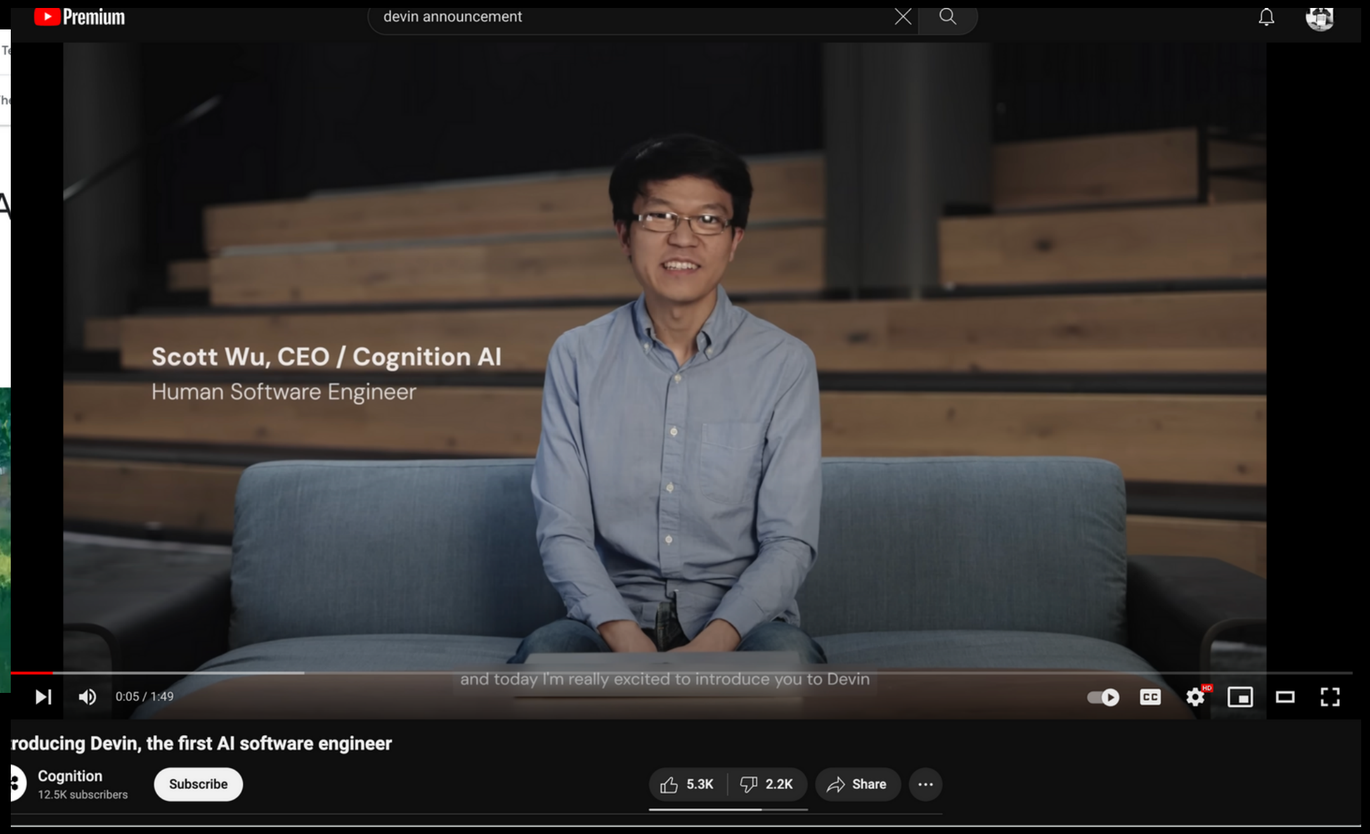
dynamic environment - psychological
element and components like bluffing

APPLICATIONS



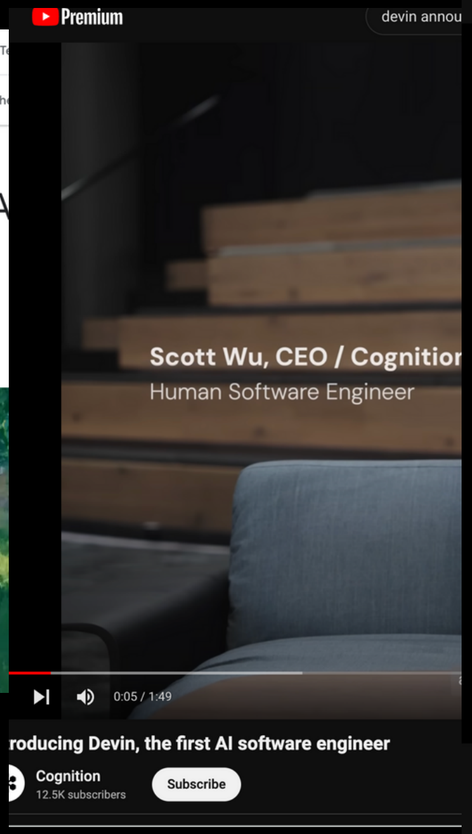
WITHIN THE LAST 48 HOURS...

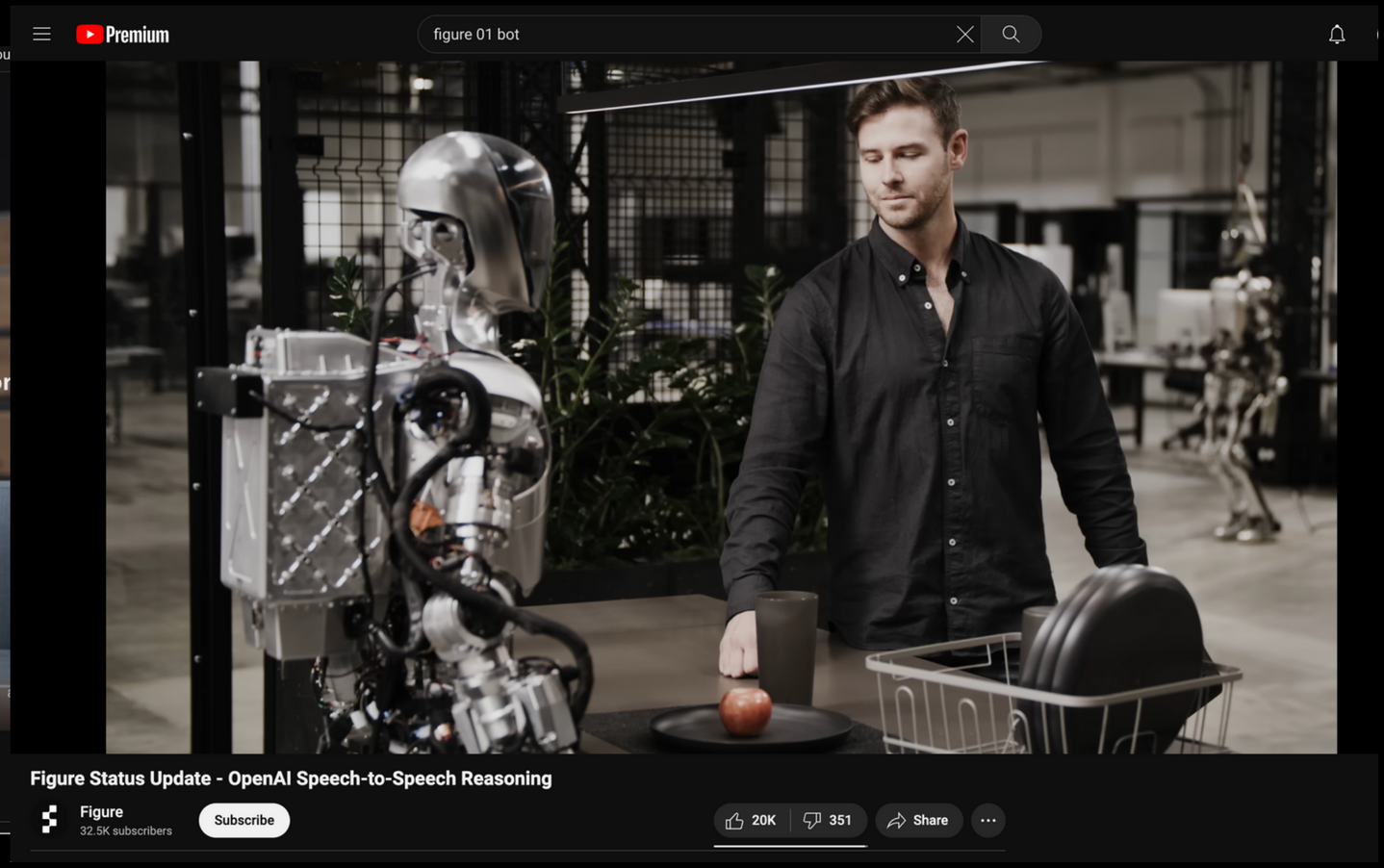
APPLICATIONS



WITHIN THE LAST 48 HOURS...

APPLICATIONS

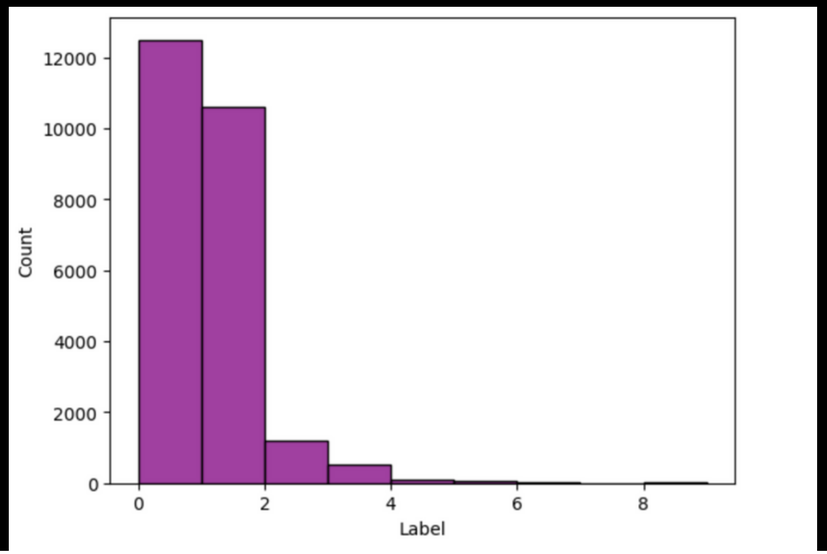


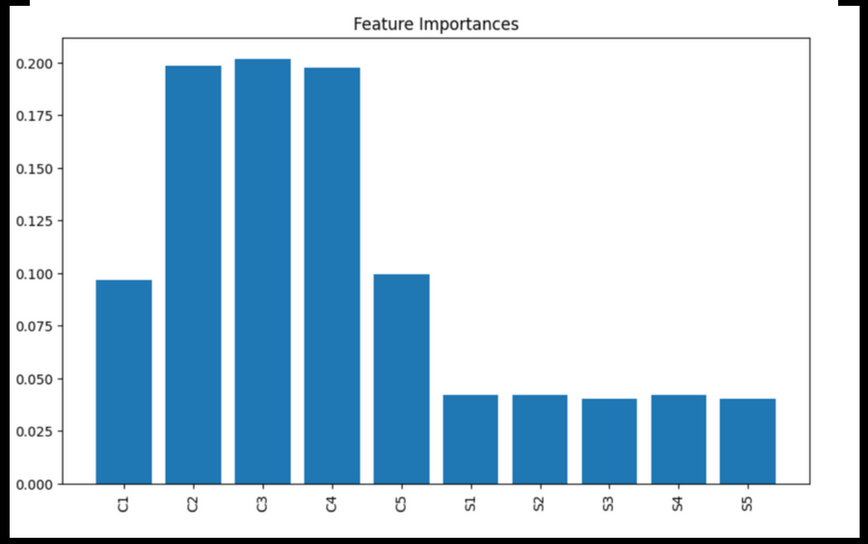


WITHIN THE LAST 48 HOURS...

IMPLEMENTATION

THE "HOW"





SOURCE

- UCI dataset : 25,010 poker hands with labeled hand strength
- 10 integers representing 5 cards and their suits
- 0-13 : represent the card
- 0-4 : represent the suit (0 represents a missing card)

PREPARATIONS

- Testing and Training Data, Reorder data, normalize attributes, ...

SOME PROBLEMS

- Data consists of 5 hand plays (may not account for dynamics of Texas Hold'em)
- Skewed right (there are less labeled "Rare" hands)

poker_hand_classification.ipynb | poker-hand-testing.data | agent

hand_classification > data > poker-hand-testing.data

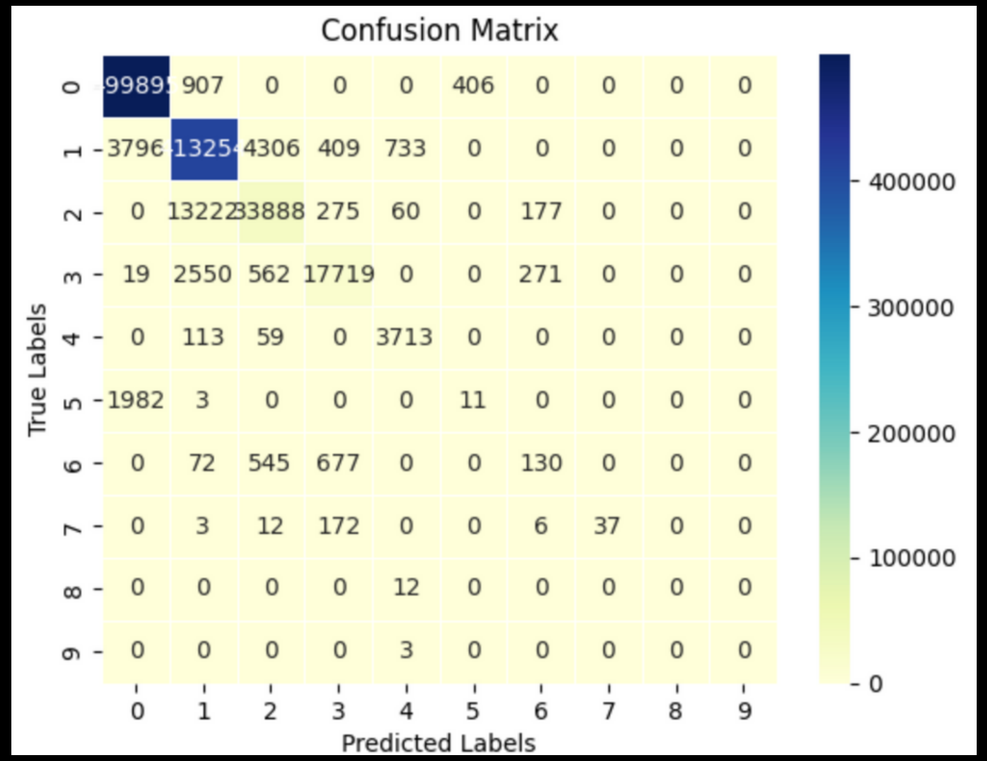
```
1,1,1,13,2,4,2,3,1,12,0
3,12,3,2,3,11,4,5,2,5,1
1,9,4,6,1,4,3,2,3,9,1
1,4,3,13,2,13,2,1,3,6,1
3,10,2,7,1,2,2,11,4,9,0
1,3,4,5,3,4,1,12,4,6,0
2,6,4,11,2,3,4,9,1,7,0
3,2,4,9,3,7,4,3,4,5,0
4,4,3,13,1,8,3,9,3,10,0
1,9,3,8,4,4,1,7,3,5,0
4,7,3,12,1,13,1,9,2,6,0
2,12,1,3,2,11,2,7,4,8,0
4,2,2,9,2,7,1,5,3,11,0
1,13,2,6,1,6,2,11,3,5,1
3,8,2,7,1,9,3,6,2,3,0
2,10,1,11,1,9,3,1,1,13,0
4,2,4,12,2,12,2,7,3,10,1
4,5,2,2,4,9,1,5,4,1,1
2,3,3,9,2,1,2,6,4,10,0
1,7,2,11,4,1,2,9,3,13,0
4,12,1,6,3,1,2,2,1,8,0
2,5,3,1,3,13,4,13,3,8,1
1,3,4,8,2,1,1,12,3,5,0
2,8,4,6,1,12,2,13,1,8,1
1,7,4,13,4,9,1,9,1,10,1
2,13,3,3,2,11,2,6,1,4,0
4,3,2,4,4,9,2,8,1,11,0
2,5,3,7,2,12,3,3,2,11,0
3,4,2,1,3,10,1,8,4,1,1
4,11,2,13,4,4,3,8,4,1,0
```

You, 2 weeks ago • Initial commi

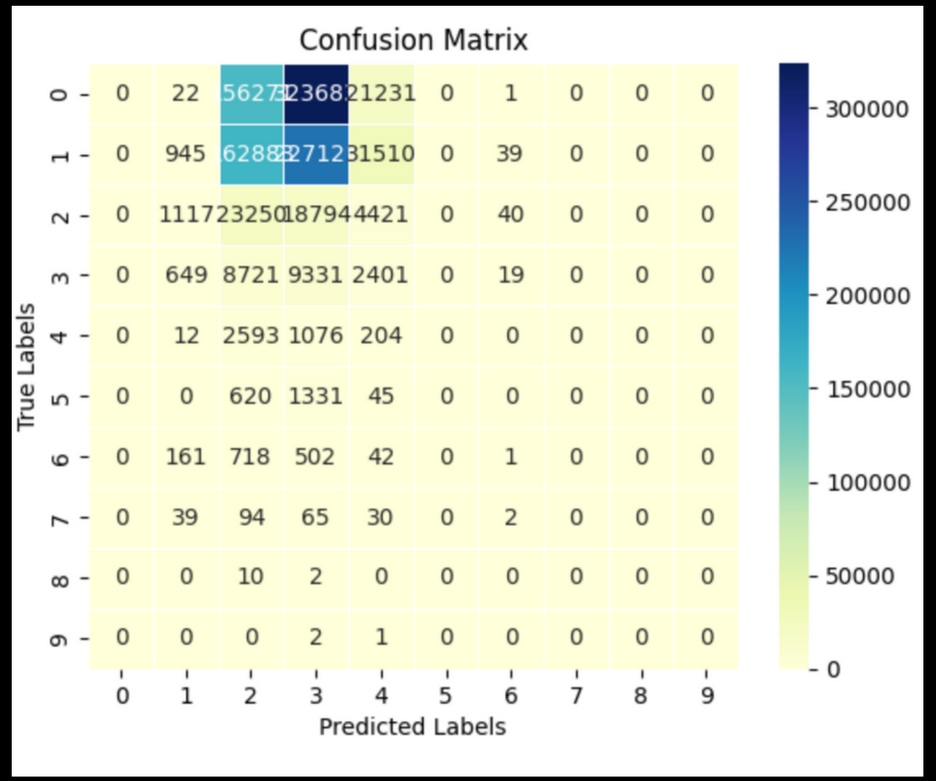
UCI DATASET IS BAD...

HOW BAD?

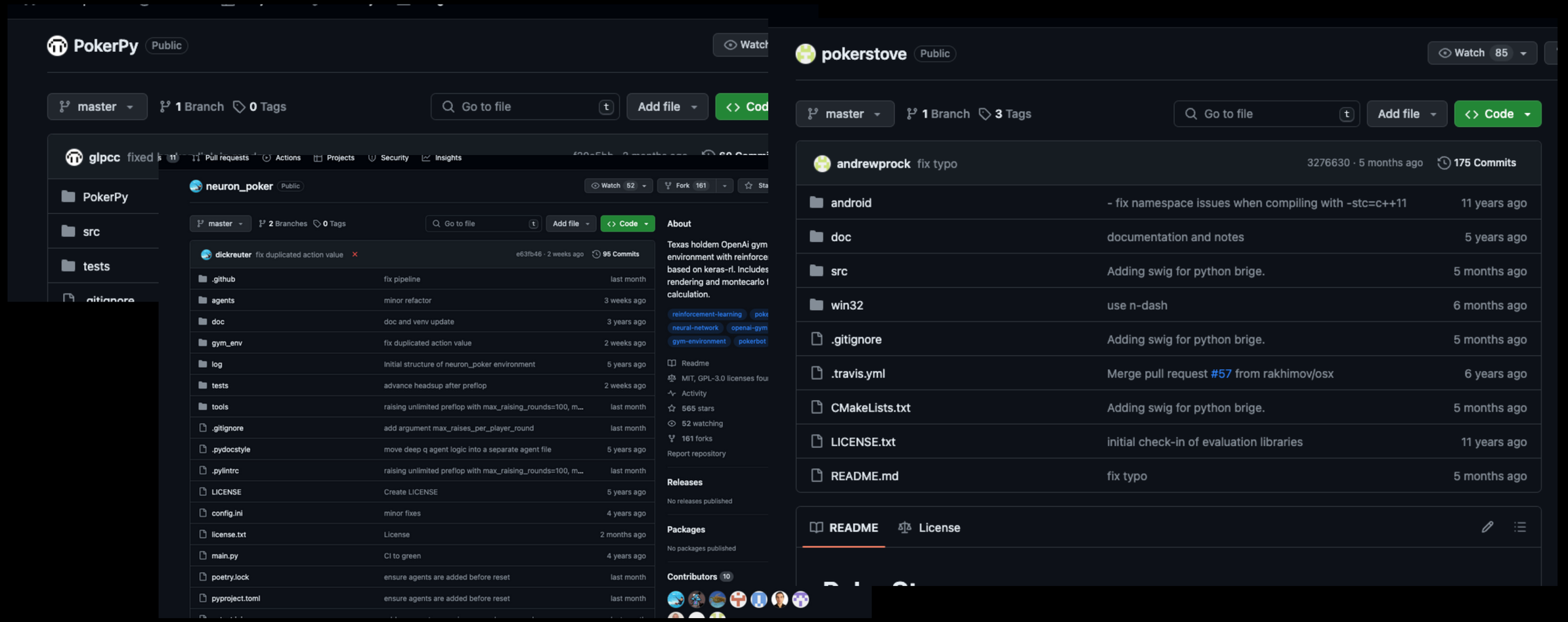
5-BEST DECISION TREES



DL MODEL WITH UCI

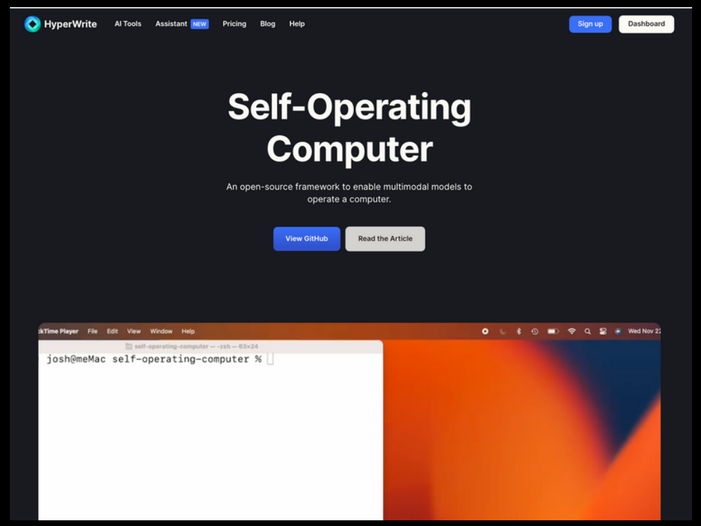


WHAT WE TRIED...



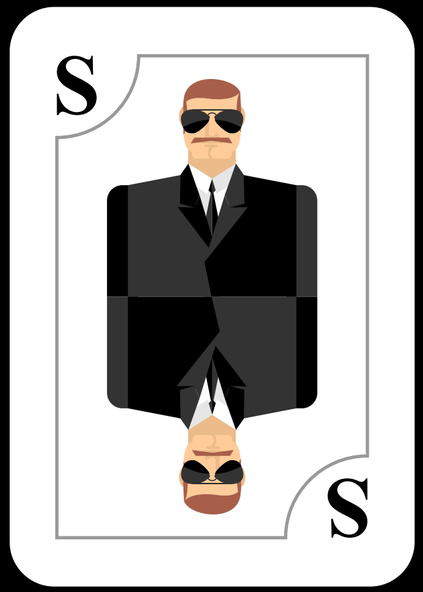
all existing datasets were bad or the library was bad

WHAT WE USED



HyperWrite's Self-Operating Computer

+



Our own DQN & fixed strat agent

=

Magic (embodied poker agent)

A RUNDOWN OF HOW IT WORKS

SCREENSHOT GAME STATE

GPT V PROCESSES THE IMG

1) WAIT
2) CONTINUE
3) MAKE MOVE

Community Cards:
7-C, 3-D, 10-H

Hole Cards:
A-C , 4-C

NOW WE MAKE A MOVE

our action options are to:

fold, check, call

NOW WE MAKE A MOVE

How we decide...

ONLINE CASH RANGES

Raise • Raise or Fold • Raise or Call

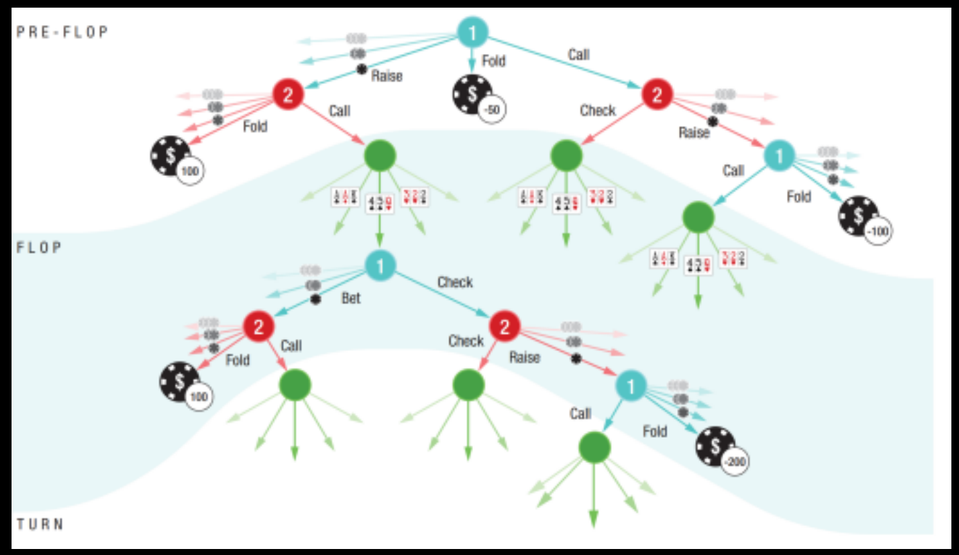
Raise, Call or Fold • Call • Call or Fold • Fold

RFI | RFI VS 3 BET | VS RFI | Show Table View

AA	AKs	AQs	AJs	ATs	A9s	A8s	A7s	A6s	A5s	A4s	A3s	A2s
AKo	KK	KQs	KJs	KTs	K9s	K8s	K7s	K6s	K5s	K4s	K3s	K2s
AQo	KQo	QQ	QJs	QTs	Q9s	Q8s	Q7s	Q6s	Q5s	Q4s	Q3s	Q2s
AJo	KJo	QJo	JJ	JTs	J9s	J8s	J7s	J6s	J5s	J4s	J3s	J2s
ATo	KTo	QTo	JTo	TT	T9s	T8s	T7s	T6s	T5s	T4s	T3s	T2s
A9o	K9o	Q9o	J9o	T9o	99	98s	97s	96s	95s	94s	93s	92s
A8o	K8o	Q8o	J8o	T8o	98o	88	87s	86s	85s	84s	83s	82s
A7o	K7o	Q7o	J7o	T7o	97o	87o	77	76s	75s	74s	73s	72s
A6o	K6o	Q6o	J6o	T6o	96o	86o	76o	66	65s	64s	63s	62s
A5o	K5o	Q5o	J5o	T5o	95o	85o	75o	65o	55	54s	53s	52s
A4o	K4o	Q4o	J4o	T4o	94o	84o	74o	64o	54o	44	43s	42s
A3o	K3o	Q3o	J3o	T3o	93o	83o	73o	63o	53o	43o	33	32s
A2o	K2o	Q2o	J2o	T2o	92o	82o	72o	62o	52o	42o	32o	22

vs RFI: BB vs CO

or



1) Fixed Strategy

2) Deep Q-Network

METHOD 1: FIXED STRAT (GTO)

in limit poker, if you know your two private cards (hole cards) and the community cards on the table, there is a mathematically optimal decision that can be made based on the probability of winning with your current hand

METHOD 1: FIXED STRAT (GTO)

```
# Check for straight flush
if is_flush(suits) and is_straight(sorted_ranks):
    return 9

# Check for four of a kind
if is_four_of_a_kind(sorted_ranks):
    return 8

# Check for full house
if is_full_house(sorted_ranks):
    return 7

# Check for flush
if is_flush(suits):
    return 6

# Check for straight
```

METHOD 1: FIXED STRAT (GTO)

```
# Make a decision based on hand strength and number of opponents
if hand_strength <= 2 and num_opponents > 2:
    return "fold"
elif hand_strength <= 3 and num_opponents > 1:
    return "check"
elif hand_strength <= 4:
    return "call"
else:
    return "raise"
```

METHOD 2: DEEP Q NETWORK

Step 1: Observation

The DQN agent watches you play the game and pays attention to what's happening on the screen. This information is like a snapshot of the game at a specific moment, which we call a "state."

```
# Define the poker environment
class PokerEnv:
    def __init__(self):
        self.action_space = [0, 1, 2]  # 0: Fold, 1: Call, 2: Raise
        self.observation_space = NUM_CARDS
        self.state = None
        self.done = False
        self.step_count = 0

    def reset(self):
        self.deck = list(range(NUM_CARDS))
        random.shuffle(self.deck)
        self.player_hand = [self.deck.pop(), self.deck.pop()]
        self.community_cards = []
        self.state = self.player_hand + self.community_cards
        self.done = False
        self.step_count = 0
        return self.state

    def step(self, action):
        self.step_count += 1

        if action == 0:  # Fold
            reward = -1
            self.done = True
        elif action == 1:  # Call
```

METHOD 2: DEEP Q NETWORK

We have a step function that updates the reward based on fold/call/raise.

Every time the DQN agent takes an action, it observes what happens next in the game and gets a reward or a punishment. It remembers this experience - the state, the action it took, the reward or punishment, and the next state. These experiences are stored in the agent's memory.

```python
def step(self, action):
    self.step_count += 1

    if action == 0:  # Fold
        reward = -1
        self.done = True
    elif
        (variable) player_score: int

        player_score = self.calculate_hand_score(self.player_hand + self.community_cards)
        reward = player_score
    else:  # Raise
        reward = 0
        self.done = False

    if not self.done:
        if len(self.community_cards) < 5:
            num_cards_to_deal = random.randint(1, min(3, 5 - len(self.community_cards)))
            self.community_cards.extend([self.deck.pop() for _ in range(num_cards_to_deal)])
            self.state = self.player_hand + self.community_cards
        else:
            self.done = True
            player_score = self.calculate_hand_score(self.player_hand + self.community_cards)
            reward = player_score

    if self.step_count >= MAX_STEPS_PER_EPISODE:
        self.done = True

    return self.state, reward, self.done, {}
```

METHOD 2: DEEP Q NETWORK

The neural network takes the state as input and outputs a number for each possible action. These numbers represent how good the agent thinks each action is in that state.

```
def _build_model(self):
    model = Sequential()
    model.add(Dense(64, activation='relu'))
    model.add(Dense(64, activation='relu'))
    model.add(Dense(self.action_size, activation='linear'))
    model.compile(loss='mse', optimizer=Adam(learning_rate=self.learning_rate))
    return model

def remember(self, state, action, reward, next_state, done):
    self.memory.append((state, action, reward, next_state, done))

def act(self, state):
    if np.random.rand() <= self.epsilon:
        return random.randrange(self.action_size)
    act_values = self.model.predict(state)
    return np.argmax(act_values[0])
```

METHOD 2: DEEP Q NETWORK

Exploring and exploiting: Sometimes, the DQN agent will choose the action with the highest number (the one it thinks is best). This is called "exploiting." Other times, it will randomly choose an action to try something new. This is called "exploring." Exploring helps the agent discover new strategies that might work even better.

METHOD 2: DEEP Q NETWORK

After playing for a while, the DQN agent takes a break and reviews its past experiences. It uses these experiences to update its neural network. The agent learns from its mistakes and successe. Repeats this process over and over, playing the game, making decisions, learning from experiences, and updating its brain. As it practices more, it gets better at the game

```python
def replay(self, batch_size):
    minibatch = random.sample(self.memory, batch_size)
    states, targets_f = [], []
    for state, action, reward, next_state, done in minibatch:
        target = reward
        if not done:
            next_state = np.reshape(next_state, [1, len(next_state)])
            target = reward + self.gamma * np.amax(self.model.predict(next_state)[0])
        target_f = self.model.predict(np.reshape(state, [1, len(state)]))
        target_f[0][action] = target
        states.append(state[0])
        targets_f.append(target_f[0])
    self.model.fit(np.array(states), np.array(targets_f), epochs=1, verbose=0)
    if self.epsilon > self.epsilon_min:
        self.epsilon *= self.epsilon_decay
```

```python
for e in range(NUM_EPISODES):
    state = env.reset()
    state = np.reshape(state, [1, len(state)])
    done = False
    score = 0
    while not done:
        action = agent.act(state)
        next_state, reward, done, _ = env.step(action)
        score += reward
        next_state = np.reshape(next_state, [1, len(next_state)])
        agent.remember(state, action, reward, next_state, done)
        state = next_state
        if done:
            print(f"Episode: {e + 1}/{NUM_EPISODES}, Score: {score}")
            break
    if len(agent.memory) > BATCH_SIZE:
        agent.replay(BATCH_SIZE)
```

REVIEW

1) SCREENSHOT
2) GPT V PROCESSES INPUTS
3a) Fixed Strat Predicted Best Action
3b) DQN Predicts Best Action

TAKE ACTION

Once we have our predicted action, we use Python to control the computer mouse and click on the button the AI suggested. Then rinse and repeat until the game is over

VALIDATION & RESULTS

LIVE DEMO

LIVE DEMO

LIVE DEMO

LIVE DEMO

LIVE DEMO

CONTRIBUTIONS

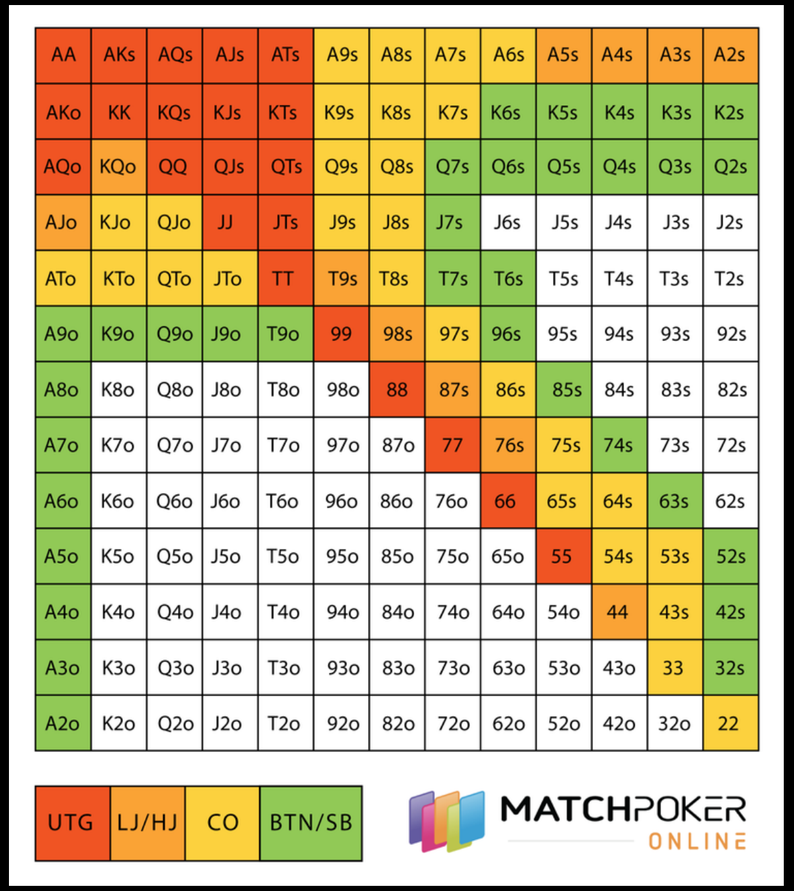
```
default_info.log
INFO - env.py - _end_round - 554 - ------------------------------
INFO - env.py - _end_round - 555 - ===ROUND: Stage.FLOP ===
INFO - env.py - _process_decision - 407 - Seat 1 (Random): Action.CHECK - Remaining stack: 534.0, Round pot: 0, Community pot: 8.0, player pot: 0
INFO - env.py - _process_decision - 407 - Seat 0 (Random): Action.CHECK - Remaining stack: 462.0, Round pot: 0, Community pot: 8.0, player pot: 0
INFO - env.py - _next_player - 611 - End round - no current player returned
INFO - env.py - _distribute_cards_to_table - 676 - Cards on table: ['9S', '2D', 'TC', 'TS']
INFO - env.py - _end_round - 554 - ------------------------------
INFO - env.py - _end_round - 555 - ===ROUND: Stage.TURN ===
INFO - env.py - _process_decision - 407 - Seat 1 (Random): Action.CHECK - Remaining stack: 534.0, Round pot: 0, Community pot: 8.0, player pot: 0
INFO - main.py - command_line_parser - 55 - Initializing program
INFO - env.py - _start_new_hand - 419 -
INFO - env.py - _start_new_hand - 420 - ++++++++++++++++++
INFO - env.py - _start_new_hand - 421 - Starting new hand.
INFO - env.py - _start_new_hand - 422 - ++++++++++++++++++
INFO - env.py - _distribute_cards - 662 - Dealer is at position 0
INFO - env.py - _distribute_cards - 670 - Player 0 got ['AH', 'KC'] and $500
INFO - env.py - _distribute_cards - 670 - Player 1 got ['QS', '3S'] and $500
INFO - env.py - _initiate_round - 503 -
INFO - env.py - _initiate_round - 504 - ===Round: Stage: PREFLOP
INFO - env.py - _process_decision - 407 - Seat 1 (Random): Action.SMALL_BLIND - Remaining stack: 499, Round pot: 1, Community pot: 0, player pot: 1
INFO - env.py - _process_decision - 407 - Seat 0 (Random): Action.BIG_BLIND - Remaining stack: 498, Round pot: 3, Community pot: 0, player pot: 2
INFO - env.py - _process_decision - 407 - Seat 1 (Random): Action.FOLD - Remaining stack: 499, Round pot: 3, Community pot: 0, player pot: 1
INFO - env.py - _next_player - 608 - Only one player remaining in round
INFO - env.py - _get_winner - 584 - Player 0 won: Only remaining player in round
INFO - env.py - _start_new_hand - 419 -
INFO - env.py - _start_new_hand - 420 - ++++++++++++++++++
INFO - env.py - _start_new_hand - 421 - Starting new hand.
INFO - env.py - _start_new_hand - 422 - ++++++++++++++++++
INFO - env.py - _distribute_cards - 662 - Dealer is at position 1
INFO - env.py - _distribute_cards - 670 - Player 0 got ['KC', '5D'] and $501
INFO - env.py - _distribute_cards - 670 - Player 1 got ['4D', '7D'] and $499
```

programmatically creating our own dataset &
will open source it

*note: running the bot for an hour costs ~$12 in GPT-V API costs

SLOW/EXPENSIVE EVALUATION

- Will play more games throughout the next week and get a percentage.
- Performance can be measured in a few different ways
  - The overall profit gained over a number of games
    - Goal is > 0
- Most of the error comes from GPT-V misreading the board



CONCLUSION

MAIN TAKEAWAYS

1 LACK OF OPTIMAL OPEN SOURCE POKER BOT

2 DEEP Q NETWORKS

3 APPLICATION FOR REAL WORLD ONLINE ENVIRONMENTS

FUTURE EXPANSION

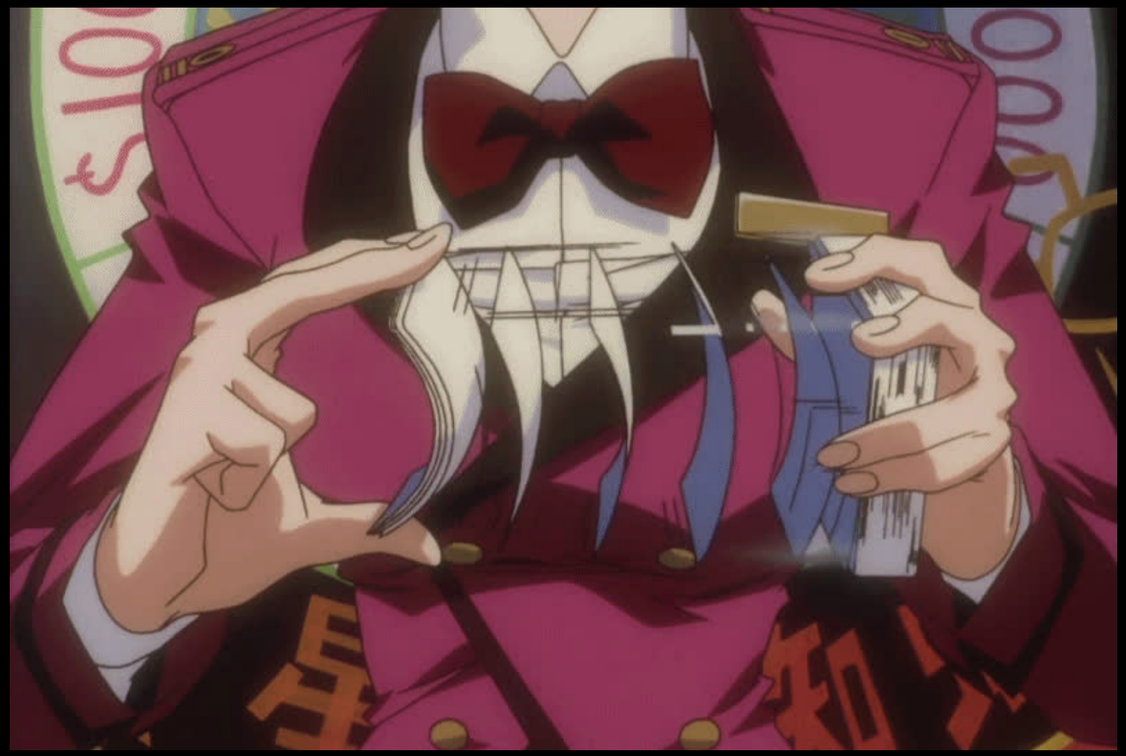
- Potentially implement betting strategies.
- Enhance opponent modeling to better predict opponents' actions and exploit their weaknesses.
- Generalize the agent to different poker variants and other online games

KEY LEARNING & COURSE RELEVANCE:

Learning:
Faster to just think of the problem from first principles and do it yourself than to rely on existing libraries

Course Relevance:
Building a narrow intelligence agent that lives on a layer above the computer that plays both a fixed strategy & DL strategy

THANKS!



github.com/arnenoori/gto-poker