Solving Rubik's Cube with Reinforcement Learning

DS-GA 3001 Final Project

By Group Deepcube:
Zihan Liu, Mengqi Liu, Michelle Tong, Siyi Wang

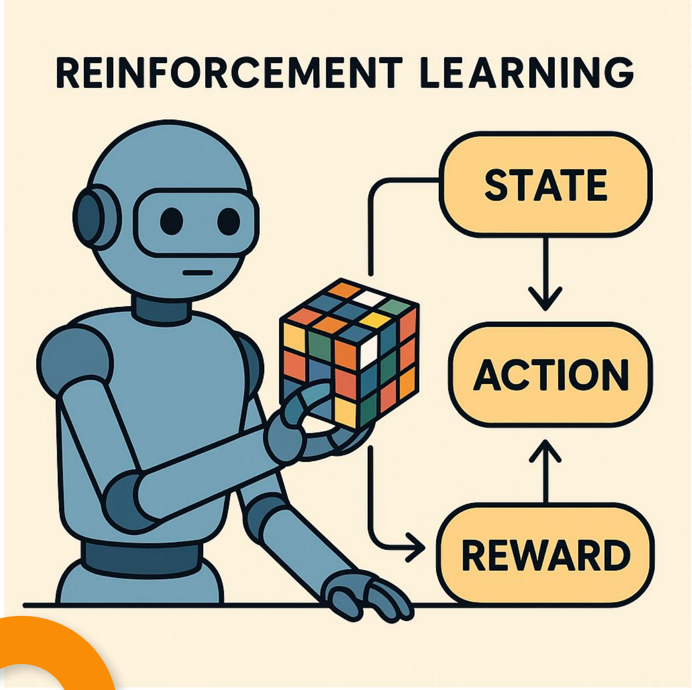
Table of contents

01 Objectives

Create a **reinforcement-learning agent** capable of reliably solving any **3×3×3 Rubik's Cube** scrambled with **≤ 5** random face-turns (~5.8 million reachable states). The goal is to train the agent so that, starting from any such scramble, it can choose the **correct sequence of moves** to return the cube to its **solved state** with high success rate and in as few moves as possible.

Why Rubik's Cube?

- 4.3×10^{20} possible configurations → extreme complexity
- Classic benchmark for spatial reasoning and sequential decision-making
- Single well-defined goal state yet multiple solution paths



02 Environment Setup

State Representation

480-dimensional one-hot encoding

- **For corner pieces:**
  - Each corner has 8 possible positions on the cube
  - Each corner has 3 possible orientations at each position, identified by its unique set of 3 colors
  - 8 corner pieces × 3 = 24 orientations per position
- **For edge pieces:**
  - Each edge has 12 possible positions on the cube
  - Each edge has 2 possible orientations at each position, identified by its unique set of 2 colors
  - 12 edge pieces × 2 = 24 orientations per position
- **20 pieces** total and **24 possible states per piece**
- Encoded as a **(20×24) array**, flattened to **480** dimensions

Action Space

12 possible moves in standard notation: F, F′, B, B′, L, L′, R, R′, U, U′, D, D′

- Each representing a 90° rotation of a face
- the apostrophe (′) after a letter indicates a counterclockwise rotation of that face, when looking directly at the face.

For example:

- F = Front face clockwise rotation (90°)
- F′ = Front face counterclockwise rotation (90°)

Environment Methods

- **step(move_idx):** Execute move, return new state and reward
- **reset(moves):** Return to solved state, optionally scramble
- **scramble(moves):** Apply random sequence of moves with specified number of steps
- **is_solved():** Check if current state matches solved state

Methodologies & Results 03

Methodologies

- **DQN Approaches**
  - **Plain DQN**
  - **Dueling Double DQN with PER**
  - **Deep Q-learning from Demonstrations (DQfD)**
- **Advantage Actor–Critic (A2C) with Curriculum Learning**

  A single policy-value network is trained on-policy, and a curriculum scheduler automatically raises scramble depth each time the agent's success rate exceeds 60-80%.
- **Policy-Value Networks and MCTS**

  An AlphaZero-style neural-MCTS approach: a deep policy–value network guides Monte Carlo Tree Search, and the improved search targets in turn train the network via self-play.

Plain DQN

Environment

- **Max steps**: 100
- **Reward**: Sparse
  - -0.1 per move
  - +100 for solving

Network & Training

- **Architecture**: MLP 480 → 512 → 256 → 128 → 12 (Q-values)
- **Hyperparameters**:
  - Optimizer: Adam (lr = 0.001)
  - Discount factor (γ): 0.99
  - ε-greedy: decay from 1 → 0.1
- **Training setup**:
  - Replay buffer: 100k
  - Batch size: 64
  - Target network update: every 1k steps
  - Total data collected: ~3k transitions (3 seeds × 10 episodes × 100 steps)

Plain DQN - Results

- **Average reward:** Plateaus at -2.0
- **Success rate:** Remains at 0%
- **Loss:** Rapid collapse to near-zero

Interpretation:

- Agent rarely receives positive rewards
- Q-values converge prematurely due to lack of meaningful signal
- DQN + TD loss alone inadequate for this **combinatorial task**

Deep Q-learning from Demonstrations (DQfD)

RL boosted by 30,000 expert demonstrations (inverse scrambles)

- Pre-training (20 epochs) + fine-tuning (2,000 episodes) with 4 parallel environments
- Dense rewards: +10 for solved state, partial credit for correct pieces, -0.1 per move
- Curriculum learning increases difficulty as performance improves
- Double Q-learning prevents value overestimation

Dueling Double DQN with PER

Architecture: Dueling network with separate value & advantage streams

- 480→512→256 (w/LayerNorm) → Value(128→1) & Advantage(128→12)
- Max steps: 50, Dense rewards (+100 solved, -0.1/move)
- Huber loss, SumTree-based PER (100k buffer), 64 batch size

Key Features::

- Double-DQN targets reduce overestimation
- 3-step returns accelerate reward propagation
- Prioritized sampling with importance weight correction (β: 0.4→1.0)
- 1000 episodes with progressive difficulty (1-5 scrambles)

Enhanced DQN Methods - Results

Performance:

- Both methods achieved **~20%** solve rate on simple scrambles (1-3 moves)
- Both failed on more complex scrambles (4-5 moves)

Insights:

- Learning is occurring but at an impractically slow rate for this project
- Rubik's Cube state space is too vast for pure DQN approaches
- Even with demonstrations and advanced techniques, progress plateaus quickly

A2C with Curriculum Learning

Neural Network

- Shared representation layer (256 units)
- Dual-head output:
  - **Actor:** Policy head outputs action logits → softmax probabilities
  - **Critic:** Value head estimates state-value function V(s)
- Combined loss: policy_loss + 0.5 × value_loss - 0.02 × entropy [typical coefficients]

Training Optimizations

- N-step returns (t_max = 5) for temporal credit assignment
- Gradient clipping (0.5) to stabilize learning
- Entropy regularization (0.02) to balance exploration/exploitation
- Learning rate schedule: exponential decay (3e-4 → 9e-5)

A2C with Curriculum Learning

Curriculum-Based Training

- Progressive difficulty: 1 → 5 scramble moves
- Advancement criteria:
  - Version 1: **50%** success rate over 50-episode window, 5000 episodes total
  - Version 2: **60%** success rate over 300-episode window, up to 16500 episodes total

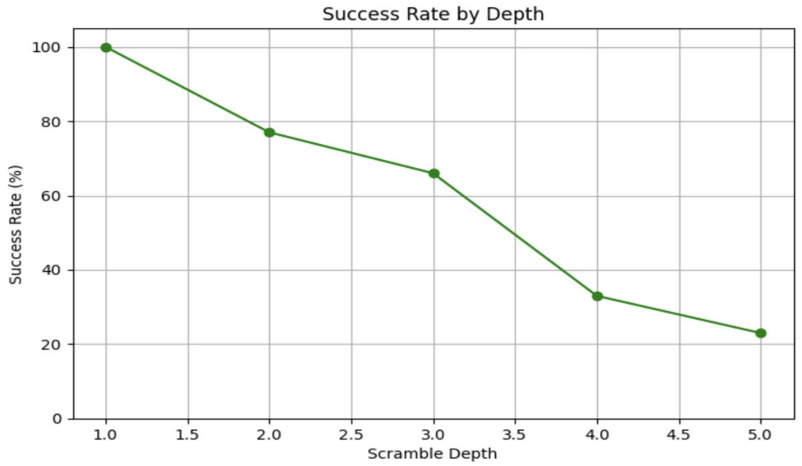
Reward Engineering

- **Version 1**:
  - Terminal reward (+1.0) when solved
  - Small bonus for partial progress (0.01 × correct pieces)
  - Penalty (-1.0) for each step
- **Version 2**:
  - Terminal reward (+100.0) when solved
  - Piece-position correctness (0.001 × correct pieces/20)
  - Step penalty (-1.0 per step)

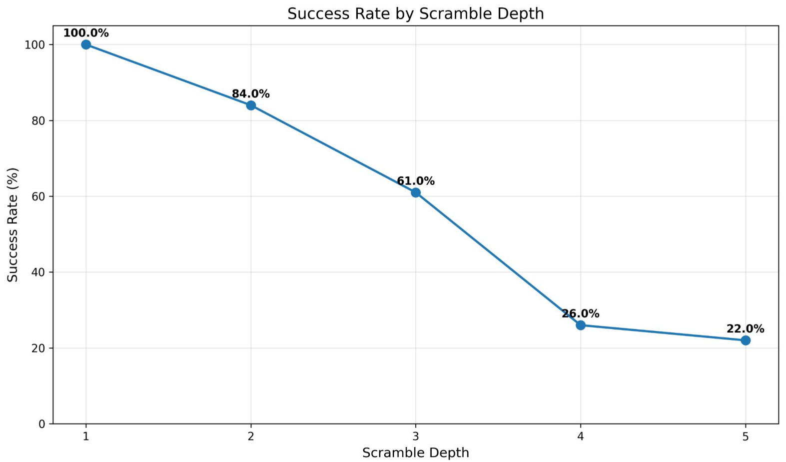
A2C - Results

Depth	Version 1	Version 2
1	100% (100/100)	100% (100/100)
2	77% (77/100)	84% (84/100)
3	66% (66/100)	61% (61/100)
4	33% (33/100)	26% (26/100)
5	23% (23/100)	22% (22/100)

Version 1



Version 2



Policy-Value Networks and MCTS - Environment & Cube Agent

- **Sparse reward** system:
  - +1 when cube is solved
  - -1 otherwise
- Maintains a **grid of CubeEnv instances**:
  number_of_cubes × batches environment matrix
- In each row: the first cube is manually scrambled (or reset to a baseline)
  - Each subsequent cube copies the previous state and applies **one more random move**
- Enables **curriculum-like training**:
  - Data goes from easy → hard
  - Later cubes are deeper in the scramble tree
- Helps model learn **progressive solving strategies**

Policy-Value Networks and MCTS - Neural Network

Model: "functional"

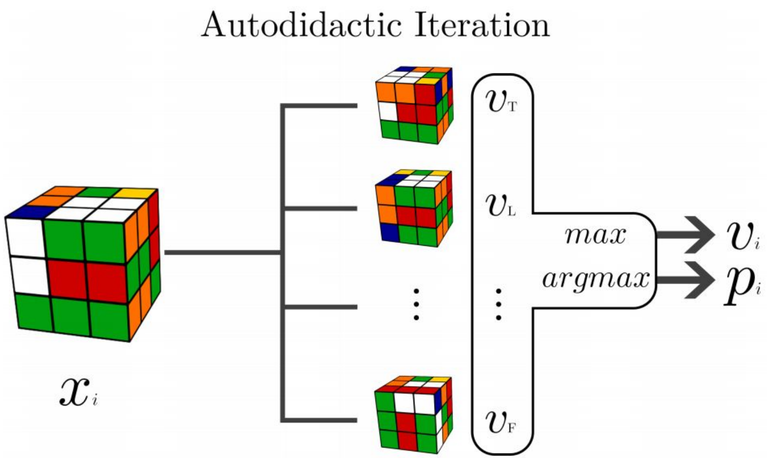
Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 480)	0	-
dense (Dense)	(None, 4096)	1,970,176	input_layer[0][0]
dense_1 (Dense)	(None, 2048)	8,390,656	dense[0][0]
dense_2 (Dense)	(None, 512)	1,049,088	dense_1[0][0]
dense_3 (Dense)	(None, 512)	1,049,088	dense_1[0][0]
output_value (Dense)	(None, 1)	513	dense_2[0][0]
output_policy (Dense)	(None, 12)	6,156	dense_3[0][0]

Total params: 24,931,356 (95.11 MB)
Trainable params: 12,465,677 (47.55 MB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 12,465,679 (47.55 MB)

Policy-Value Networks and MCTS - Training

We use the method called Autodidactic Iteration:

- For each iteration:
  - Generate states with CubeAgent
  - Simulate all possible actions
  - Compute value + reward as Q-value
- Supervised training:
  - Best-value action → policy label
  - Max predicted value → value label
- Train model for 3 epochs per iteration (for simple scrambles, otherwise it will be long)

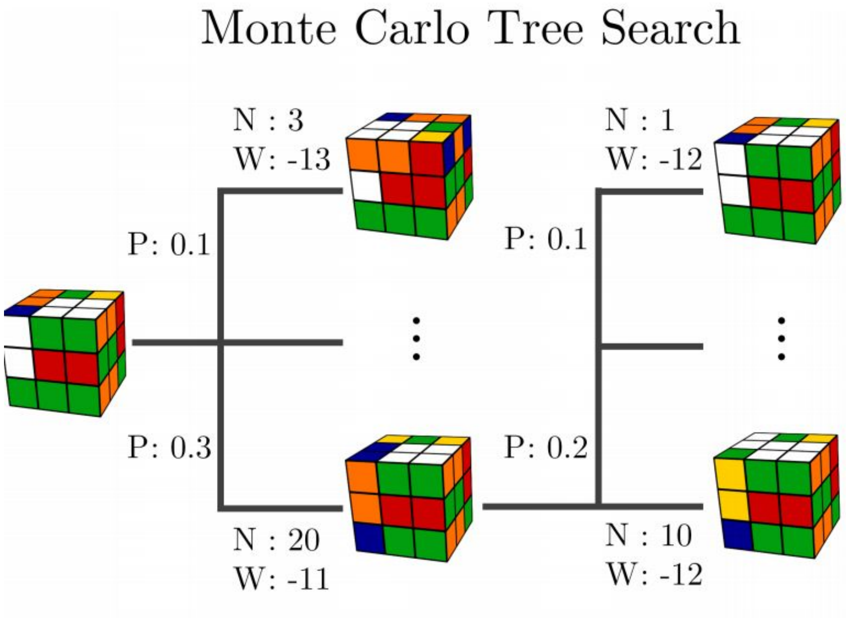


McAleer, S., Agostinelli, F., Shmakov, A., & Baldi, P. (2018). *Solving the Rubik's Cube Without Human Knowledge*. arXiv:1805.07470.

Policy-Value Networks and MCTS - Solving

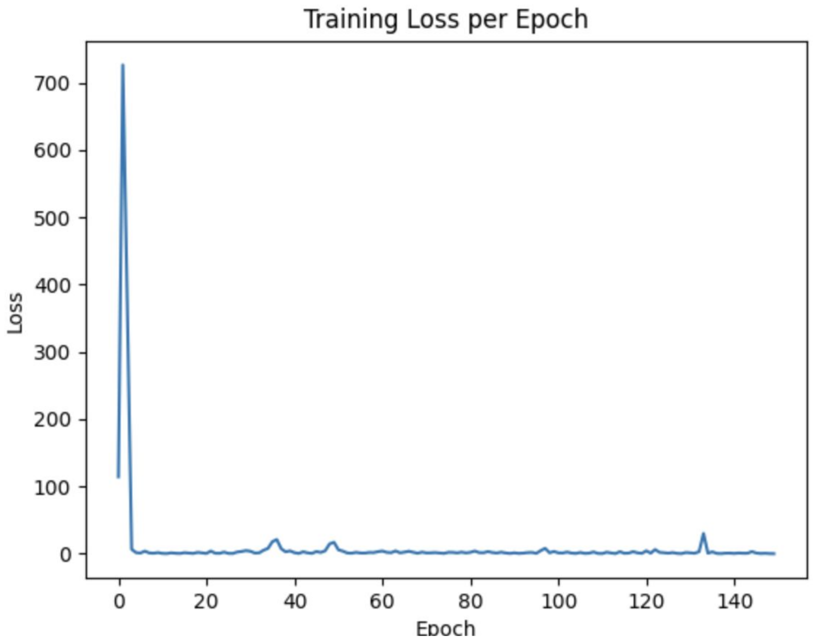
Monte Carlo Tree Search (MCTS)

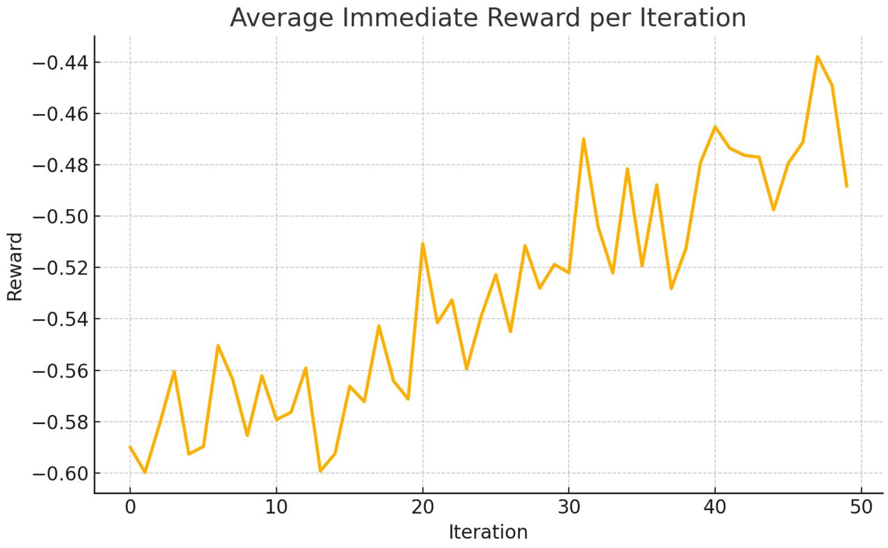
- **Model-Driven Expansion**
    - Use a neural network to predict value and policy for a given cube state.
- **Tree Traversal**
    - Traverse from root using UCB formula
- **Expansion**
    - Expand unvisited nodes by simulating all legal moves.
- **Backpropagation**
    - Update visit counts and best values along the path.



McAleer, S., Agostinelli, F., Shmakov, A., & Baldi, P. (2018). *Solving the Rubik's Cube Without Human Knowledge*. arXiv:1805.07470.

Policy-Value Networks and MCTS - Results





Policy-Value Networks and MCTS - Results (Partial)

1 move	R	U	R'	D	F
Finished	Yes	Yes	Yes	Yes	Yes
2 moves	R U	U2 R'	R' F	D2 F2	U R
Finished	Yes	Yes	Yes	Yes	Yes
3 moves	R U R'	R2 F D'	R2 F2 U'	R U R	U R F'
Finished	Yes	Yes	Took 2 minutes (not optimal: ['L', 'L', 'L', 'U', 'U', 'R', 'U', 'R'])	Yes	Yes
4 moves	R U R' U'	R2 F D' F'	R U' R F2	R U R D'	F U R F'
Finished	Yes	Yes	Yes	Yes	Yes
5 moves	R2 U R' U' F'	R U F F' R	R R' U U F	U2 R U R' F	F U R F' D
Finished	No Too Deep, cannot solve	Yes	Yes	No	No

Policy-Value Networks and MCTS - Results

We evaluated our model by testing it on **100** cases per **scramble depth (1-5)**:

Scramble Depth	Success Rate
1	1
2	1
3	1
4	0.83
5	0.43

04 Illustration

Policy-Value Networks and MCTS

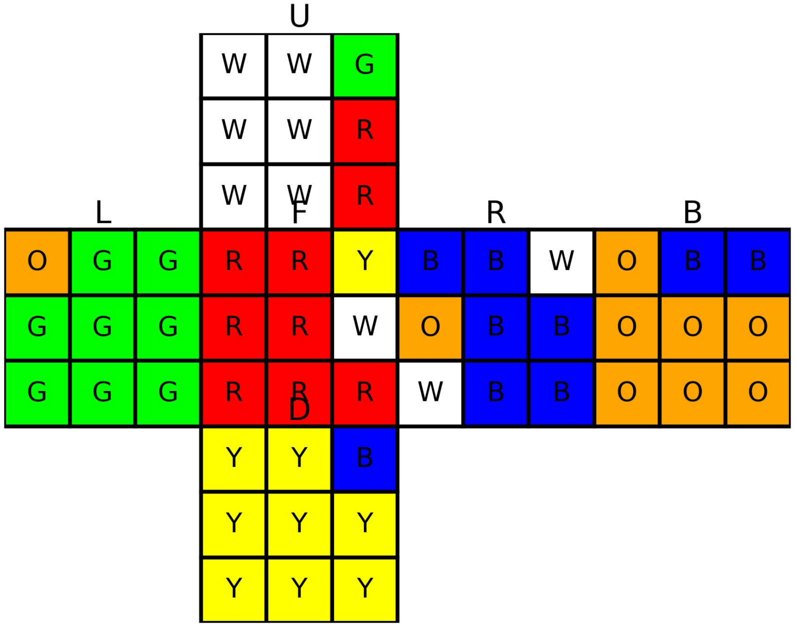
```
3.8.0
         [y][y][r]
         [y][y][g]
         [y][y][g]
[b][r][r][g][g][w][o][o][y][b][o][o]
[r][r][r][g][g][y][b][o][o][b][b][b]
[r][r][r][g][g][g][y][o][o][b][b][b]
         [w][w][o]
         [w][w][w]
         [w][w][w]

Scramble formula: R U R' U'
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 331ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
```

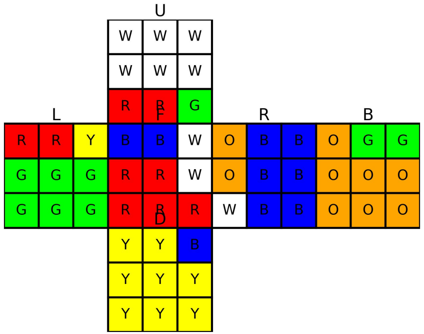
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
50.02085638046265s, naive:
['U', "B'", 'B', 'R', "U'", "R'"]
4.316840648651123s, bfs afterwards:
['U', 'R', "U'", "R'"]
```

Policy-Value Networks and MCTS

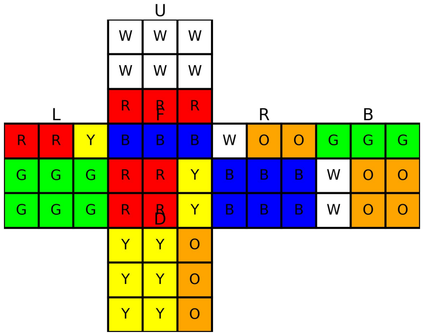
Scrambled State:



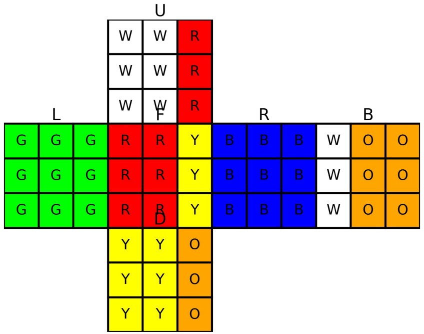
Policy-Value Networks and MCTS



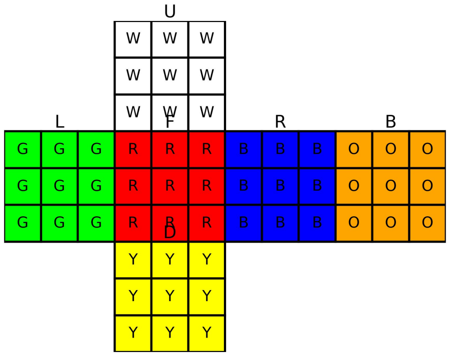
U



R



U’



R’

Conclusion & Future Work 05

Conclusion

- Pure DQN methods struggled with sparse rewards and scalability.
- A2C with curriculum learning improved early performance but plateaued on complex scrambles.
- Policy-Value networks with MCTS achieved the best results, solving up to depth-4 reliably.
- Guided search and curriculum design are key to tackling Rubik’s Cube complexity with RL.

Future Work

- Apply recurrent models (LSTM/Transformer) for better generalization
- Test on physical cube with vision-based state estimation
- Combine with real-time robotics control pipeline (e.g., using ROS)

Reference

- McAleer, S., Agostinelli, F., Shmakov, A., & Baldi, P. (2018). *Solving the Rubik's Cube Without Human Knowledge*. arXiv:1805.07470.

Thank you!