GO Master AI Final Report

Yang Li yl5456,
Yuxuan Li yl5438,
Zhiyuan Guo zg2481

1. Synopsis

Our project, "Master Go," represents a significant advancement in the development of Go AI technology, aiming to deliver a playing experience that not only matches but potentially surpasses the strategic depth and understanding of top human players. At the core of "Master Go" are cutting-edge deep reinforcement learning techniques combined with sophisticated neural network architectures and Monte Carlo Tree Search algorithms, Zobrist Hashing algorithm, Beta-Pruning algorithm etc. This powerful blend enables our AI to analyze and respond to game dynamics with unprecedented precision.

Innovation lies at the heart of "Master Go." Unlike traditional Go AIs, our model incorporates several unique features that both refine the development process and greatly enhance user interaction. A pivotal innovation is the introduction of a resign threshold by setting the value in the command-line, which allows the AI to concede the game when victory becomes statistically unlikely, thus mimicking a human-like understanding of when a match is effectively lost. Additionally, we have integrated a set level function that dynamically adjusts the AI's difficulty based on the player entered skill level, using different pre-trained game models to provide a tailored challenge.

Moreover, "Master Go" includes a robust after-game analysis feature, which not only provides a concise summary but also displays a detailed graph of the game progression. This graph highlights key moves and win rates, helping players understand the AI’s strategy and improve their own skills. The comprehensive Move Tree visualizes each game's full trajectory, offering insights into both player and AI decisions throughout the match.

The availability of "Master Go" to our prospective user community is facilitated through an accessible online platform, ensuring that enthusiasts and professionals alike can easily engage with and benefit from this advanced AI tool. By pushing the boundaries of what Go AIs can achieve, "Master Go" not only enhances the playing experience but also contributes valuable insights into the evolving landscape of artificial intelligence in strategic game playing.

2. Research Questions

- What neural network architectures best evaluate Go board states and predict optimal strategies?

  Transformers are the main architecture for our design. They have shown good performance in sequence-to-sequence modeling tasks in Game AI research. In the context of Go AI specifically, transformers can be used to model the behavior patterns of opponents, predict future moves, and capture the long-range dependencies present in the game.

  In the coding process, we used Reinforcement learning and Transformers as our base models of the neural network architectures. The Go AI can learn how to make optimal decisions by receiving rewards or penalties for the actions in the game (simply winning or losing). The game is formulated as a Markov Decision Process and the RL algorithms can then be applied to improve the strategies. Nowadays, Go Ais can achieve superhuman performance by combining different techniques such as convolutional neural networks and residual networks for pattern recognition and then Monte Carlo Tree Search for exploring future game states, and finally RL for strategy optimization.

- In what ways can Monte Carlo Tree Search algorithms be refined to improve the AI's decision-making?

  The Monte Carlo Tree search is a heuristic algorithm that was first proposed by Remi Coulom in his paper "Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search." He successfully combined the negamax algorithm and Monte Carlo methods to create an implementation that only needs minimal Go-specific knowledge. Since its introduction, MCTS has undergone continuous refinements and enhancements, some of the most famous versions are traditional UCT (Upper Confidence bounds Applied to Trees) method and the RAVE (Rapid Action Value Estimation). For traditional UCT, the iterative update can be divided into 4 steps.

  1. Selection: From the root node, a leaf node, in other words terminal node, is identified using a specific selection algorithm. In traditional MCTS, the UCT formula is used as the basis for selecting the child node with the

highest UCT value, and the search continues down this path until a leaf mode (with no value assigned yet) is reached.

2. Expansion: The selected leaf nodes will generate a new child node which represents the legitimate moves from the current leaf node.
3. Simulation: Then use the MCTS to calculate the score of the selected leaf node from the first step. Only one simulation will be performed, and the returned value is either winning or losing.
4. Backup: Along with the selected path, we backup and iterate through the modes and then update the score based on the current node's color (black stone or white stone), finally incrementing the visit count for each backuped node.

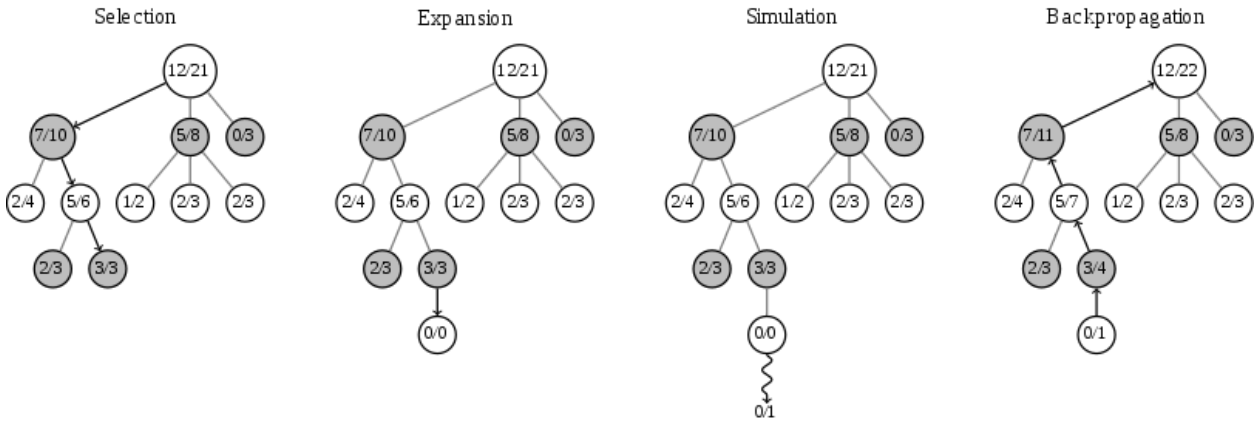


Figure 1: Traditional UCT

In the year 2017, AlphaGo Zero proposed a refined MCTS algorithm with two main differences from the traditional method, Adding Strategic Values to UCT, and removing the random simulation process. This refinement only repeats three steps:

1. Selection: From the root node, select the child node with the highest PUCT value according to the PUCT algorithm, and continue searching downwards until reaching a leaf node (a terminal node with no value assigned yet.)
2. Expansion: The selected leaf node will now generate a new child node which represents the legitimate moves from the current leaf node, and the neural network policy values will be assigned to the new child node.
3. Backup: Along with the selected path, we backup and iterate through each node in sequence, updating the neural network value based on the current node's color (black stone or white stone), and incrementing the visit count for each backuped node.

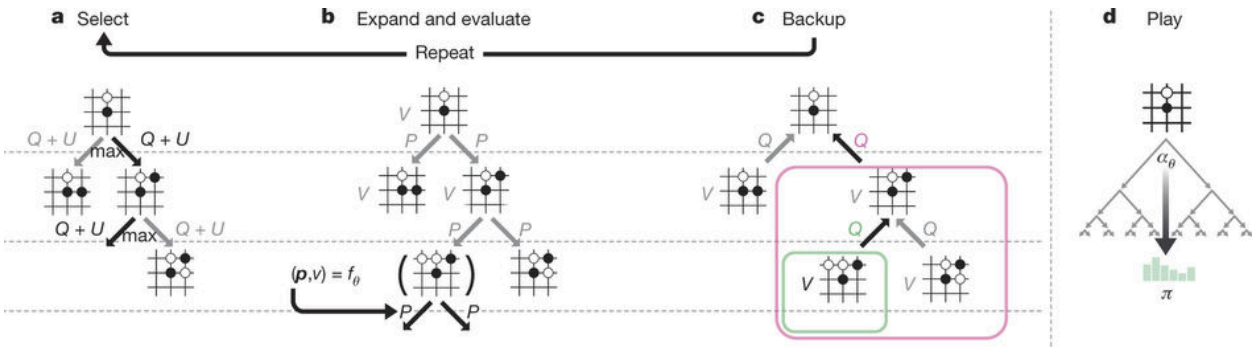


Figure 2: Refined PUCT

Since this refinement doesn't have the random simulation step, the calculations and results would be the same for the same game.

- What computational challenges arise in the development of Go AI, and how can they be effectively addressed?

  One of the most challenging development problems we faced in this project is the deep learning system of the different Weights of skills of the AI. Below is how we get the different trained weights.

  Advanced:

  Reduce the number of blocks (block_size): This will decrease the number of layers in the network, thereby reducing the total number of parameters and computational complexity.

  Reduce the number of channels in each block (block_channels, policy_channels, value_channels): This directly reduces the number of parameters per layer.

  Do not use the Transformer attention mechanism (if use_policy_attention is set): Transformer layers are relatively complex, removing this part can significantly reduce complexity and computational demands.

  Intermediate:

  Simplify module structure:Removed the residual block (ResBlock) and related SE (Squeeze-and-Excitation) modules to simplify the model's structure. This reduces the model's complexity and the number of parameters, making the model easier to understand and maintain.

  Reduce the number of layers: The original code may contain multiple levels and more complex stacks; the simplified code reduces the use of layers, retaining only the basic structure of the input convolution layer, policy head, and value head.

  Simplified configurations and options:

Removed advanced configuration options (such as the use of attention mechanisms, complex expansion heads, etc.) that may have been present in the original code, focusing on core functionalities.
Unified and simplified function calls: Ensured that all network layers and functions have direct and clear calling methods, such as unifying the initialization and forward propagation calling methods of each layer.
Beginner:
Network structure simplification: The number of layers in the network is reduced, and the treatment of convolution blocks is unified, removing batch normalization and selective activation functions, directly using ReLU activation.
Parameter simplification: The network's initialization parameters are simplified, reducing the number of configurable external parameters, making the model easier to understand and operate.
Function focus: Focus on basic strategy and value outputs, using Softmax and tanh to process outputs.

- How do our proposed features (Move Tree, after-game analysis, AI match rate) improve upon existing Go AIs in terms of player learning and engagement?

  Instead of the Move Tree, which is very unfriendly to new players, we used a diagram to show the real-time winning rate. In this way, players can analyze which step they went wrong in the middle of the game. The second feature is that the user can set up the level of the AI. These levels correspond with the traditional Chinese ranking divisions, from 30k to 1k. If the user is a Go beginner, one can set a low-ranking AI to improve the skills and then level up. If the user is a Pro Goer, one can just directly set the ranking to a higher ranking to compete with AI to advance further. The after-game analysis is presented with a win rate graph and important changes throughout the game for easier reinforcement of strategies and every move's decision.

3. Deliverables

- Repository Link: https://github.com/poppyindarrk/GOAI1

4. Results and Discussion

The figures below are the comparison between our GUI interface and another Go application in the Market. Our interface is more concise, and on the other hand

can offer more analysis because of the white information panel. Also the other one can't change the level of the AI model which offers more versatility.

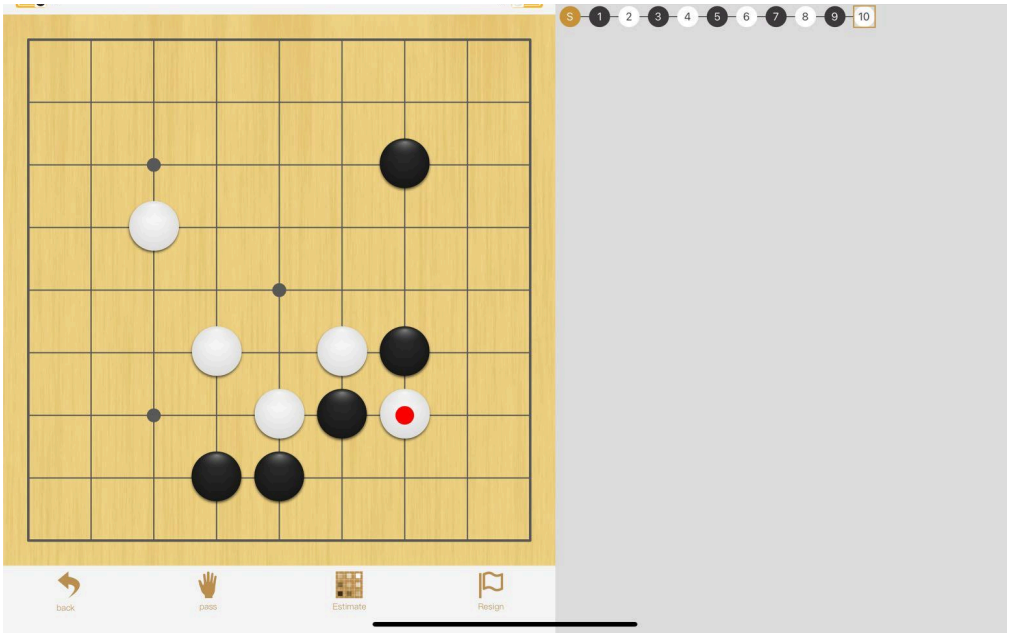


Figure 3: the UI interface of KataGo app.

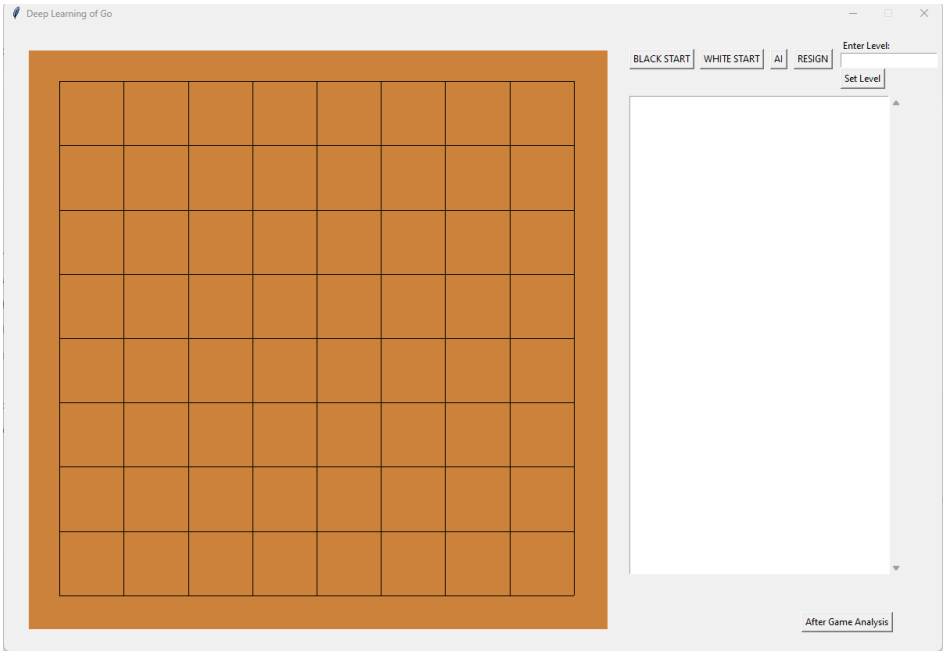


Figure 4: the UI interface of our Go AI.

After building the application, we tried multiple games with the Go AI model. We divided the experiment groups by play styles. We have a pro Goer, a Beginner, and another Go AI from the app market to play with our AI model. The game results are presented in the Appendix. We can see that the pro Goer stamped the beginner AI, the beginner almost drew with the model, and the other Go app was beaten by our model. The experiment shows that our model can cope with different levels of users and outperforms other AI models in strategies and optimal moves.

After comparing with other Go AI models, ours prevails in multiple aspects.The real-time win rate and the after game analysis are more user friendly than other AI models. The level changing feature adds more flexibility to our model, and this feature stands out to be the most innovative one among the Go AIs. We can learn from other models as well. We lack the feature of checking each individual move, also we should let the users pass their round to the AI which is a better choice than the users biting the bullet for certain situations.

5. Self-Evaluation

Yang Li: In the "Master Go" project, I was responsible for defining our goals and research questions, sourcing foundational literature and code, and implementing key features. I developed the set level function to adjust AI difficulty based on player skill and introduced an after-game analysis feature that provides statistical insights and visual game progression. Additionally, I enhanced our Monte Carlo Tree Search algorithms to include real-time win rate displays and resolved critical bugs, including network connectivity issues and end-game winner determination policies.

Zhiyuan Guo: In the “Master Go” project, I was responsible for the GUI interface on the application aspect. The GUI changes included the Go board, real-time win rate display, real-time move language display and the end game analysis panel. Then in the final report aspect, I answered all the research questions. I also lifted the experiment idea, collected data and analyzed the result in the discussion part of the paper.

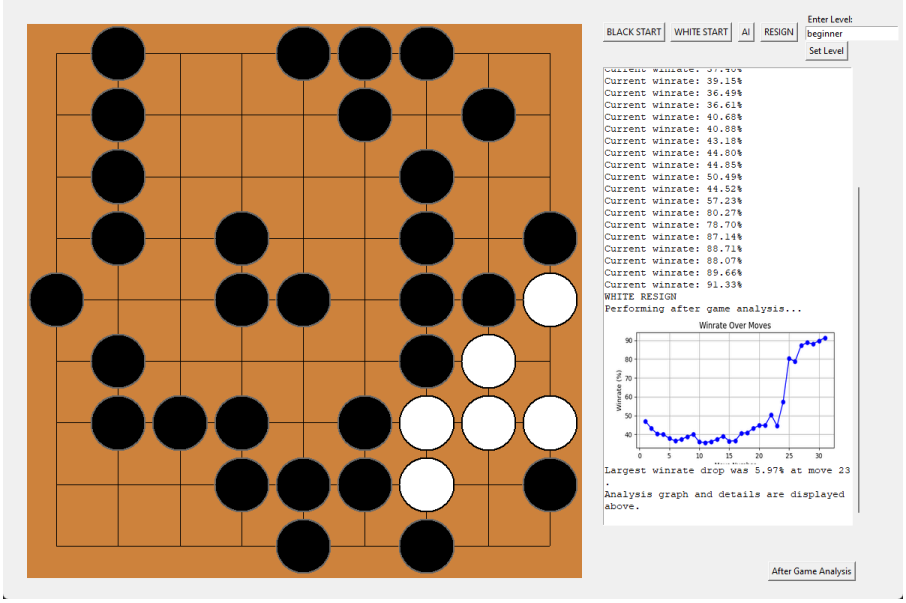
Yuxuan Li: In the “Master Go” project, I was responsible for achieving the function — supporting different AI levels. By changing the block size and block channels, decreasing the layers of the model, and simplifying the network structure, I get 4 different trained weights for the system which allow users to change different game difficulties. The most challenging part is designing new models and networks to achieve different trained results. Through this project, I have learned the details of how to train a

GO model and which methods are efficient in achieving the different levels of AI but have similar functionality.

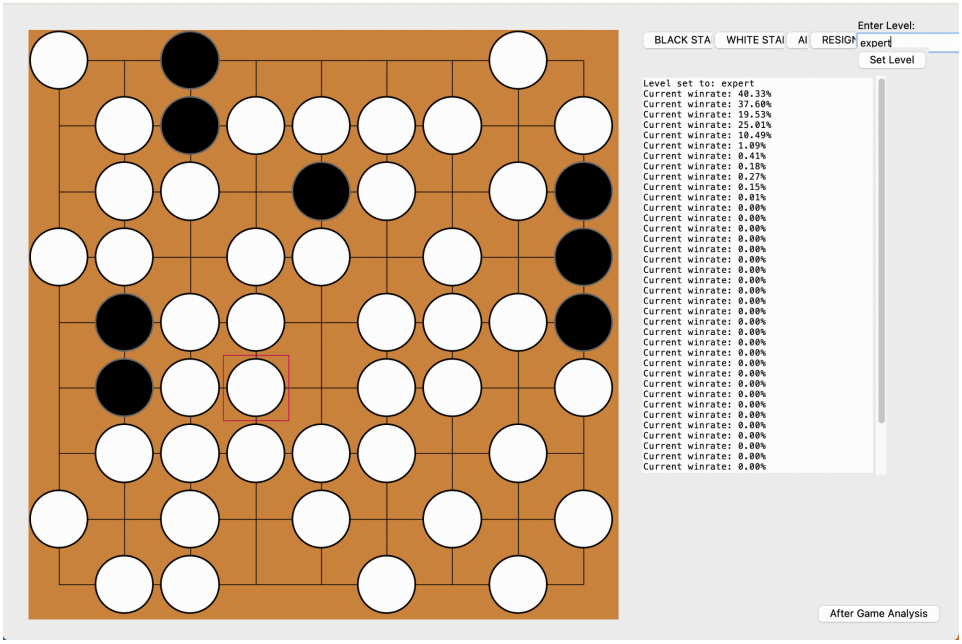
6. References

Our project builds on established datasets, software, and methodologies to enhance the efficiency and effectiveness of "Master Go." We have utilized the open-source repository found at https://github.com/CGLemon/pyDLGO as the foundation for our source code. This reuse of existing resources helps avoid redundancy and accelerates our development process.

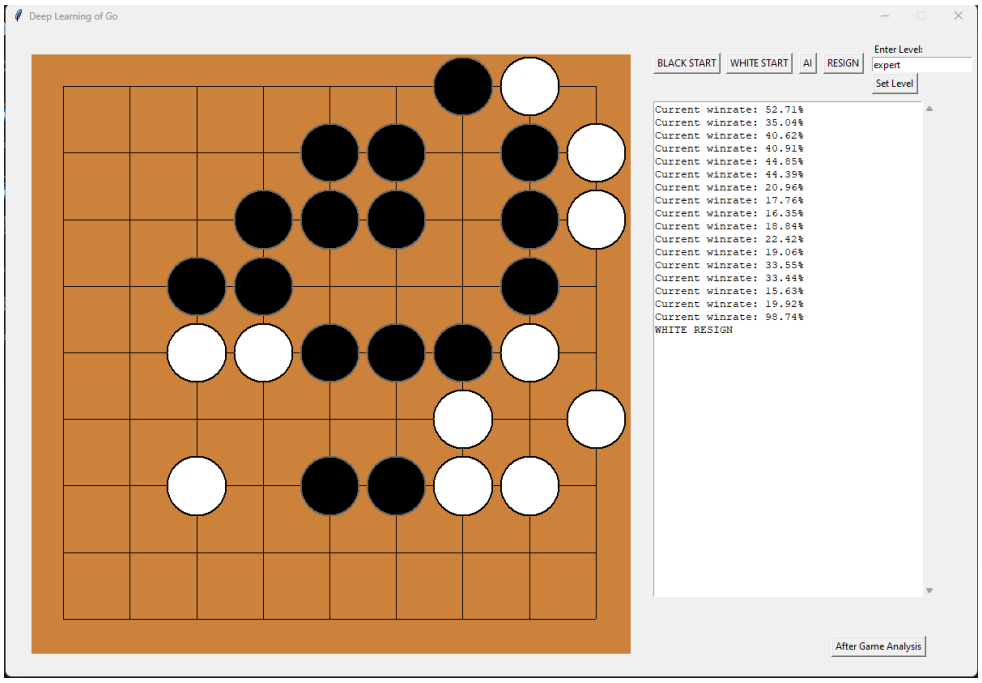
7. Appendices



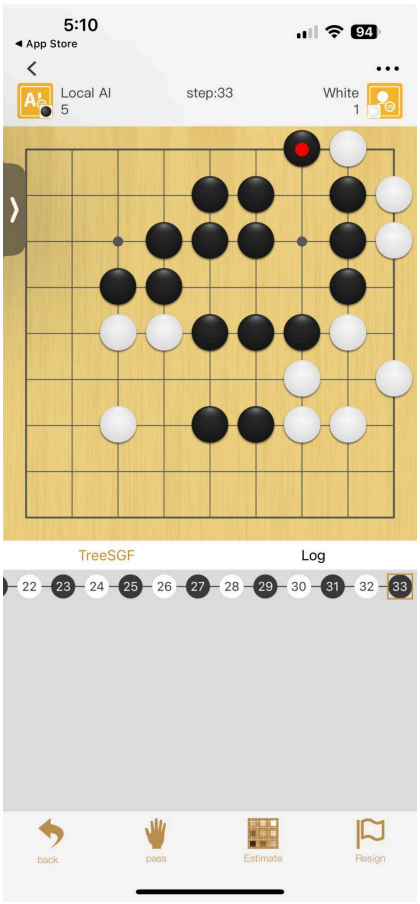
Appendix Figure 1: Pro Goer vs Our model



Appendix Figure 2: Beginner vs Our model



Appendix Figure 3: Two AI models, Our model



Appendix Figure 4: Two AI models, Master Go