Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

GNN-enhanced AlphaZero for two-player games

Semester Thesis

Andrei-Horia Pacurar

hpacurar@ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Peltonen Sakari, Dauncey Sam
Prof. Dr. Roger Wattenhofer

June 29, 2025

Acknowledgements

I would like to express my sincere gratitude to my mentors, Sakari Peltonen and Sam Dauncey, for offering me the opportunity to pursue this semester project under their supervision. Their guidance, support, time, and feedback throughout the process have been invaluable to the contributions of this project.

Abstract

Two-player games such as Go and Chess are highly complex and difficult to optimize, yet learning models have steadily improved at mastering them—often surpassing human performance. Despite this progress, most models evaluate game states by simulating sequences and summarizing the outcomes as a single scalar value, where higher values indicate better positions. This compresses strategic insight and limits the model's ability to learn from the structure of the search process. AlphaZero, one of the most successful frameworks, follows this approach by combining a deep convolutional neural network with Monte Carlo Tree Search (MCTS). In this thesis, we enhance AlphaZero by introducing a Graph Neural Network (GNN) that allows message passing between related game states. We represent MCTS trees as graphs—with nodes as board positions and edges as transitions—enabling the model to learn from both individual states and their structural relationships. We propose two major contributions: (1) during self-play, we periodically capture search trees at different depths and train the GNN to predict improved outcomes by comparing shallow and extended searches, and (2) the system alternates between standard AlphaZero training and GNN refinement, allowing it to gradually integrate relational reasoning. Our experiments show that the GNN-enhanced model consistently outperforms the standard version in direct matches, demonstrating that leveraging search structure improves both strategic understanding and overall performance.

Contents

Chapter 1

GNN-enhanced AlphaZero

1.1 Introduction

Two-player games like Chess, Go, and Shogi have long been used to evaluate progress in artificial intelligence due to their clear rules, strategic depth, and high complexity. Over the years, models have steadily improved—from early rule-based programs to deep learning approaches that heavily rely on reinforcement learning tactics. This shift has led to breakthroughs such as AlphaZero, which learns strong policies and value estimates directly from experience, with no manual intervention. However, AlphaZero evaluates a sequence of game states by assigning a scalar value to the final outcome, and propagates that value backward to all states in the sequence. If the outcome is favorable, all preceding states receive a high value, and vice versa. This compresses nuanced strategic reasoning into a single number, preventing the model from understanding why certain decisions are good or bad. In contrast, a richer representation—one that passes context between states—could help the model learn more effectively. This idea is illustrated in Figure 1.1, where one move is rewarded and another penalized, each for different relational reasons with are explicit. A mechanism like message passing with Graph Neural Networks (GNNs) enables such reasoning by allowing game states to communicate through graph-based self-attention. This makes it possible for the model to learn not just from individual positions, but from their relationships within the overall structure of the search.

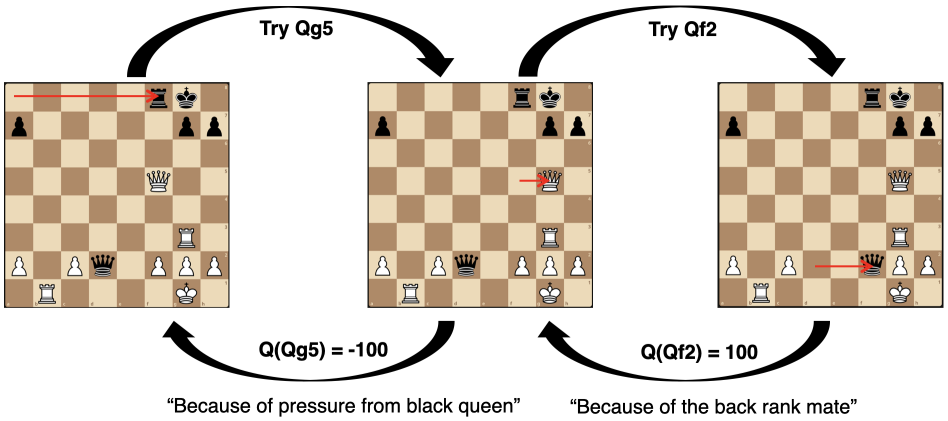


Figure 1.1: An illustration of message passing in a chess scenario. The move Qg5 is penalized due to pressure from the black queen, while Qf2 is rewarded because it enables a back rank mate. Each explanation reflects relational reasoning across pieces and positions.

1.2 Previous Work

AlphaZero [1] marked a major advancement in reinforcement learning for two-player games. It showed that strong performance could be achieved without any human supervision, domain-specific heuristics, or opening books—using only the rules of the game and self-play. By applying the same general algorithm to games as different as Chess, Shogi, and Go, AlphaZero demonstrated that deep reinforcement learning can master highly complex, strategic environments through repeated interaction and self-improvement.

At the core of AlphaZero is a deep convolutional neural network combined with Monte Carlo Tree Search (MCTS). The neural network encodes each board position into a high-dimensional feature space using a series of residual convolutional layers. From this shared representation, two separate heads are used: a `policy head`, which predicts a probability distribution over legal moves, and a `value head`, which estimates the expected outcome of the game from the current position. During MCTS, these predictions are used to guide exploration by maintaining statistics at each node—visit count, total and mean value, and prior probability. Promising moves are chosen based on a combination of exploitation and exploration, and new nodes are evaluated by the network instead of random simulations. This results in a tightly coupled interaction between the neural model and the search process.

A single iteration proceeds in three main stages. First, AlphaZero generates training data through self-play, using its current model to play games against itself via MCTS. For each move, it records the board state, the policy derived

from visit counts, and the final game result. Second, it updates the neural network by minimizing a loss that combines policy and value prediction errors using this data. Finally, in the evaluation phase, the newly trained model is pitted against the previous best version. The new model is retained only if it consistently wins more games. This loop—self-play, learning, and evaluation—drives continuous improvement and allows the system to learn strategies that are both deep and general across different games.

1.3 Our Method

While AlphaZero's architecture yields strong performance, it does not explicitly reason and model the relational structure of game states. Information from one position is compressed into a scalar value and passed forward, preventing the model from learning how different search paths relate or contribute to overall strategy. To address this, we introduce a Graph Neural Network (GNN) between the CNN backbone and the policy/value heads. After the board is processed through convolutional layers, we construct a graph in which each spatial location is a node, and edges represent either learned or predefined relationships. This allows the model to perform message passing between distant but strategically related parts of the board. Figure 1.2 illustrates the difference between the original AlphaZero pipeline and our proposed architecture with an added GNN module. The GNN receives high-level CNN features and processes them with multiple layers of graph attention or aggregation. This enables reasoning over broader spatial contexts before producing the final policy and value outputs.

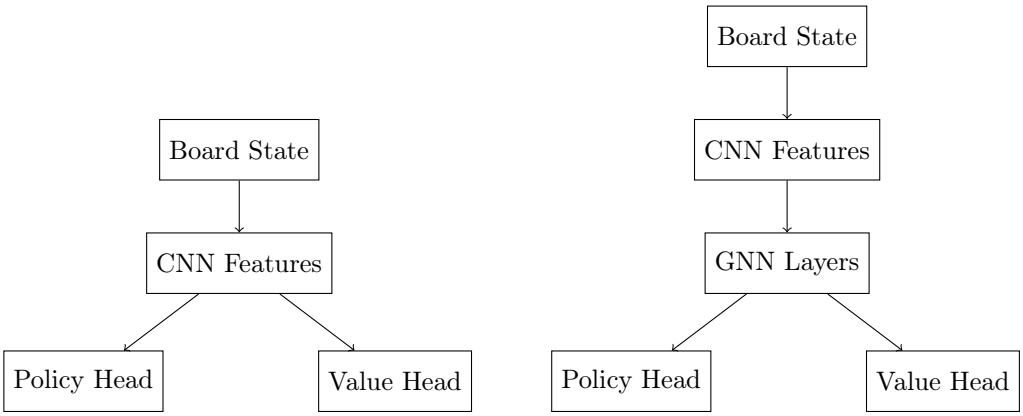


Figure 1.2: Comparison of standard AlphaZero architecture (left) with GNN-enhanced version (right), showing the placement of GNN layers between CNN feature extraction and policy/value heads.

1.3.1 Self-Play with Graph Data Collection

One of our main contributions is an extension to the original self-play stage. To enable graph-based learning, we extend the standard AlphaZero self-play routine to periodically extract structural data from the Monte Carlo Tree Search (MCTS). At regular intervals during gameplay, the agent performs a shallow search from the current position, producing initial policy and value estimates $(\text{init } \pi, \text{init } v)$. This is followed by a deeper search phase, in which additional simulations are performed to refine these estimates, yielding a more informed set of targets, denoted as the expanded policy and value estimates $(\exp \pi, \exp v)$. This two-step procedure exposes how deeper exploration changes the model's predictions, and provides a natural learning signal for the graph neural network. As shown in Figure 1.3, the search tree expands significantly between the shallow and deep phases, allowing the GNN model to reason by observing extended trajectories of game states.

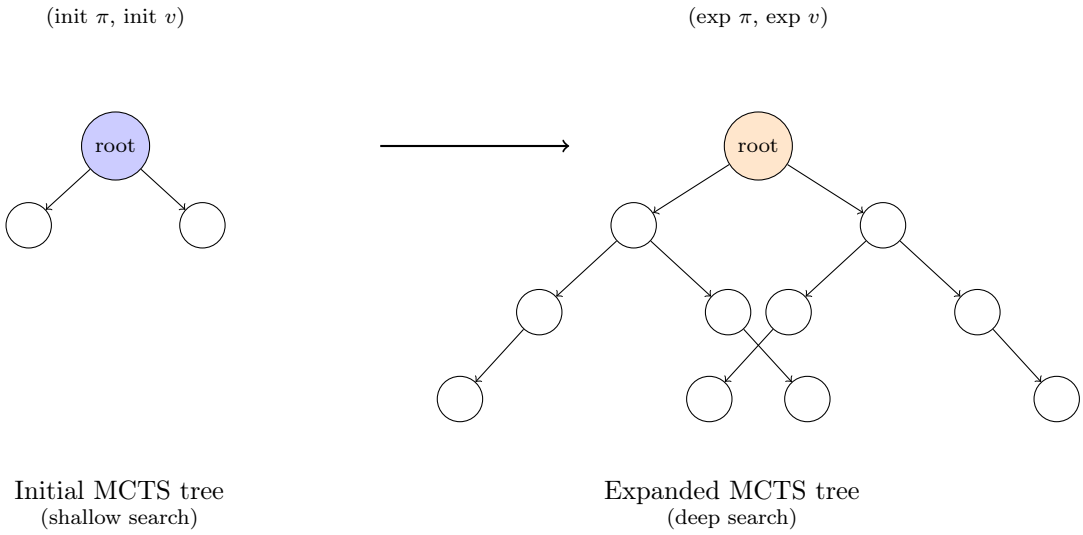


Figure 1.3: MCTS tree expansion during self-play, illustrating how the search tree grows from an initial shallow search (left) to a deeper expanded search (right), with corresponding changes in policy (π) and value (v) estimates.

Each collected MCTS tree is converted into a graph where nodes correspond to unique game states and edges represent legal move transitions explored during search. These edges are stored as index pairs in a $2 \times E$ tensor, where E is the number of edges. Each training sample contains five key elements: the root board state, the initial and expanded policy and value predictions, and the full graph structure derived from search. These graph-structured examples allow the GNN to learn from both the content of individual positions and their relationships

in the search space. The parameters that govern when and how these samples are collected—such as how often to expand the tree and how many additional simulations to perform—are summarized in Table 1.1.

To promote generalization and dataset diversity, we apply symmetry-based augmentations to both the board states and the associated graph structures. In games like Go or Connect Four, for example, mirroring or rotating the board results in equivalent but distinct inputs. When such transformations are applied, the graph's edge connections are adjusted accordingly to preserve the validity of parent-child relationships. These augmentations increase the number of useful training examples and help the GNN develop invariance to superficial changes in board layout. Together, the graph structures and their augmentations allow the model to learn not only from individual outcomes, but also from the patterns embedded in the structure of the search itself.

Parameter	**Default**	**Description**
`gnn_accumulate`	True	Enables collection of graph-based training examples during self-play
`gnn_collect_interval`	5	Number of moves between graph expansion steps
`expand_by`	10	Additional MCTS simulations performed during tree expansion
`gnn_inference_routing`	True	Routes inference through the graph neural network pipeline during gameplay

Table 1.1: Parameters controlling graph data collection during self-play

1.3.2 Dual-Phase Training Pipeline

Our training pipeline now also incorporates a series of GNN layers between the convolutional backbone and the output heads. Figure 1.4 illustrates how data is processed in this enhanced architecture, including how node features and edge indices are passed through the GNN layers. We then distinguish between two training steps: one dedicated to training the standard AlphaZero architecture and another focused on improving the graph-based layers. In the first phase, the core convolutional network is trained using examples collected through standard self-play. These examples consist of board positions along with corresponding policy and value targets. During this stage, all graph-based components remain inactive (frozen), and the network learns to evaluate positions and suggest moves using only the base structure. As shown on the left side of Figure 1.5, only the convolutional and output layers are updated, while the graph-related layers remain fixed.

The second phase activates the graph-based enhancement path. In this phase, the convolutional and output layers are frozen, and only the graph-specific layers are trained. The inputs are still board states, but this time they are sampled from the periodic expansion steps in MCTS. Features are first extracted using the frozen convolutional backbone, then projected into a lower-dimensional space, processed by a stack of graph neural network layers, and expanded back before being passed to the frozen output heads. The network is trained to match the improved policy and value targets obtained from deeper search. This stage allows the graph module to learn how to improve initial predictions by incorporating structural information gathered from extended exploration, as illustrated on the right side of Figure 1.5.

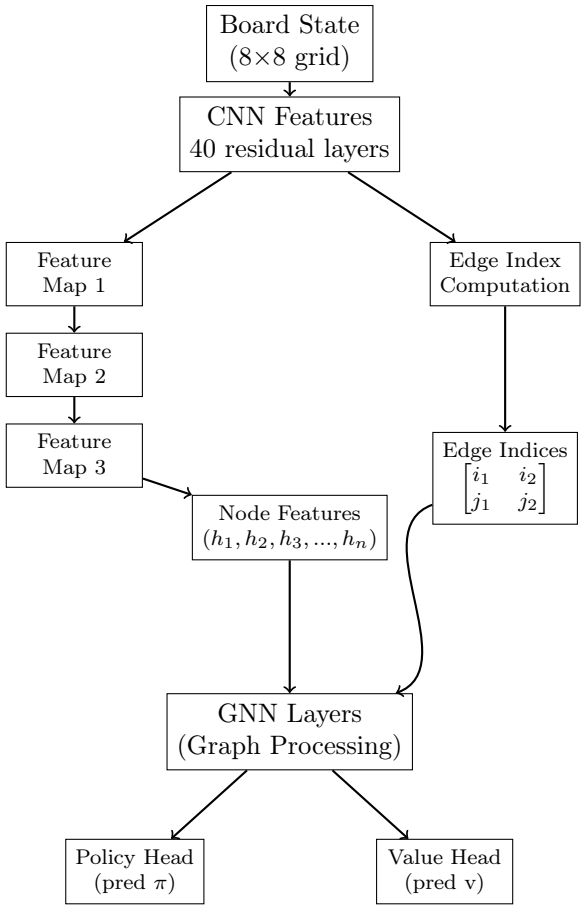
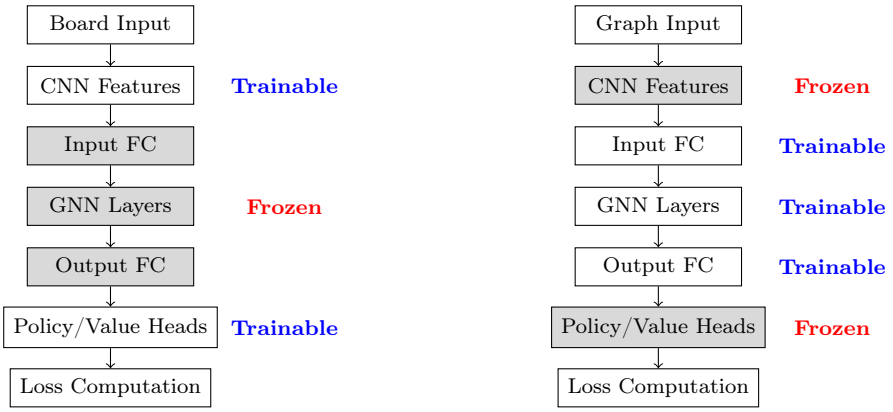


Figure 1.4: GNN architecture with edge index integration, showing how board state features are converted to node features while graph connectivity is determined through edge index computation for message passing in GNN layers.



(a) AlphaZero NN Training (b) GNN Training

Figure 1.5: Dual-phase training pipeline showing parameter freezing strategies for standard AlphaZero training (left) versus GNN enhancement training (right).

To support this training scheme, the system uses separate optimizers for each phase and trains the graph module with a reduced learning rate to ensure stability. Graph examples are processed in batches, with multiple search trees merged into a single graph and indexing managed carefully to preserve connectivity. The training loop supports mixed-precision computation and applies gradient clipping to enhance numerical stability. Various aspects of the graph enhancement are configurable, including the number of GNN layers, the dimensionality of the intermediate feature projection, and the frequency of GNN-specific training phases. It is also possible to train the entire GNN model at the very end, provided that data has been collected using the accumulation flag discussed in Table 1.1.

1.4 Results

The experimental evaluation demonstrates the effectiveness of our GNN-enhanced AlphaZero approach across multiple dimensions. Figure 1.6 illustrates the fundamental learning dynamics of both approaches through baseline comparisons. The Normal vs Normal heatmap (left) shows the expected progression inherent in AlphaZero training, where later iterations consistently outperform earlier ones, with win rates approaching 0.8–0.9 for the most recent models against initial versions. This strong diagonal pattern confirms that iterative self-play successfully strengthens model performance over time. Similarly, the GNN vs GNN heatmap (right) indicates that the GNN-enhanced models follow a comparable improvement trajectory. Later iterations achieve consistently better results, suggesting that the integration of graph-based reasoning preserves the underlying learning behavior of AlphaZero while enhancing it with relational context.

The comparison between GNN and Normal models (left panel of Figure 1.7) provides compelling evidence for the added value of graph neural network enhancement. GNN models consistently outperform their Normal counterparts, particularly in the later stages of training. For example, GNN models from iterations 51–56 achieve win rates of 0.6–0.8 against comparably trained Normal models. This advantage is most pronounced when recent GNN iterations are matched against early-to-mid Normal models, with win rates exceeding 0.7. These results demonstrate that the GNN-enhanced models are able to learn deeper strategic patterns by leveraging the structural information available in MCTS trees. The performance gap grows over time, indicating that graph-based components become increasingly effective as training progresses.

Perhaps most critically, the comparison between GNN models with and without edge index information (right panel of Figure 1.7) highlights the importance of explicitly encoding MCTS tree structure. Models that utilize edge indices—allowing message passing over the graph—consistently outperform those without, achieving win rates of 0.5–0.7 across most matchups. This result confirms that modeling the parent-child relationships in the search tree provides valuable structural context that improves decision-making beyond what is possible with node features alone. The performance difference persists across all training iterations, supporting the conclusion that the graph attention mechanism effectively exploits relational patterns in the search process, rather than simply benefiting from additional network depth.

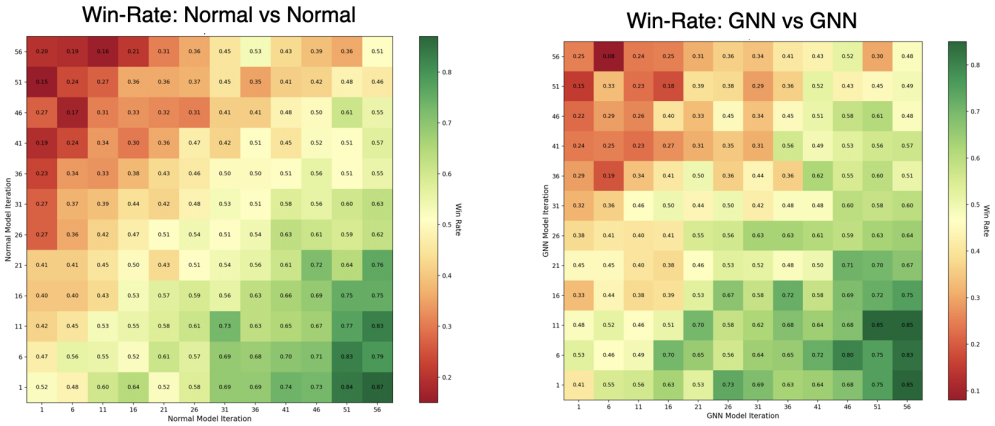


Figure 1.6: Baseline win-rate heatmaps showing Normal vs Normal models (left) and GNN vs GNN models (right), demonstrating the learning progression within each model type across training iterations.

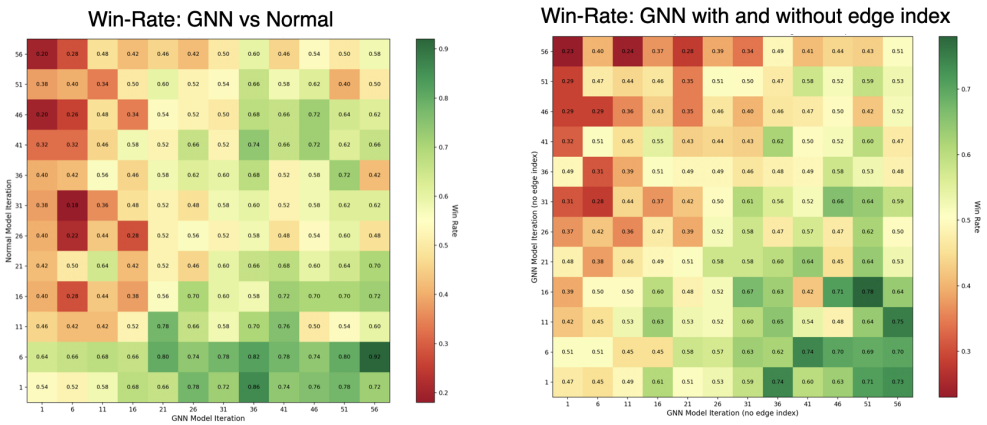


Figure 1.7: Win-rate heatmaps comparing GNN vs Normal models (left) and GNN with and without edge index (right). Darker green indicates higher win rates for the row player, while darker red indicates higher win rates for the column player.

Bibliography

[1] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.