Erklärung zur Abschlussarbeit gemäß § 22 Abs. 7 APB TU Darmstadt

Hiermit erkläre ich, *Felix Helfenstein*, dass ich die vorliegende Arbeit gemäß § 22 Abs. 7 APB TU Darmstadt selbstständig, ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt habe. Ich habe mit Ausnahme der zitierten Literatur und anderer in der Arbeit genannter Quellen keine fremden Hilfsmittel benutzt. Die von mir bei der Anfertigung dieser wissenschaftlichen Arbeit wörtlich oder inhaltlich benutzte Literatur und alle anderen Quellen habe ich im Text deutlich gekennzeichnet und gesondert aufgeführt. Dies gilt auch für Quellen oder Hilfsmittel aus dem Internet.

Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Falle eines Plagiats (§38 Abs.2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

English translation for information purposes only:

Thesis Statement pursuant to § 22 paragraph 7 of APB TU Darmstadt

I herewith formally declare that I, Felix Helfenstein, have written the submitted thesis independently pursuant to § 22 paragraph 7 of APB TU Darmstadt without any outside support and using only the quoted literature and other sources. I did not use any outside support except for the quoted literature and other sources mentioned in the paper. I have clearly marked and separately listed in the text the literature used literally or in terms of content and all other sources I used for the preparation of this academic work. This also applies to sources or aids from the Internet.

This thesis has not been handed in or published before in the same or similar form.

I am aware, that in case of an attempt at deception based on plagiarism (§38 Abs. 2 APB), the thesis would be graded with 5,0 and counted as one failed examination attempt. The thesis may only be repeated once.

For a thesis of the Department of Architecture, the submitted electronic version corresponds to the presented model and the submitted architectural plans.

Datum / Date: 12.11.23

Unterschrift/Signature: F. H.

Game phase specific models in AlphaZero

Spielphasenspezifische Modelle in AlphaZero
Master thesis by Felix Helfenstein
Date of submission: November 16, 2023

1. Review: Prof. Dr. Kristian Kersting
2. Review: Johannes Czech
3. Review: Jannis Blüml

Darmstadt

TECHNISCHE UNIVERSITÄT DARMSTADT

Computer Science Department

Machine Learning Lab (AIML)

Erklärung zur Abschlussarbeit gemäß § 22 Abs. 7 APB TU Darmstadt

Hiermit erkläre ich, Felix Helfenstein, dass ich die vorliegende Arbeit gemäß § 22 Abs. 7 APB der TU Darmstadt selbstständig, ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt habe. Ich habe mit Ausnahme der zitierten Literatur und anderer in der Arbeit genannter Quellen keine fremden Hilfsmittel benutzt. Die von mir bei der Anfertigung dieser wissenschaftlichen Arbeit wörtlich oder inhaltlich benutzte Literatur und alle anderen Quellen habe ich im Text deutlich gekennzeichnet und gesondert aufgeführt. Dies gilt auch für Quellen oder Hilfsmittel aus dem Internet.

Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Falle eines Plagiats (§ 38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

Darmstadt, 16. November 2023

F. Helfenstein

Abstract

The game of chess is usually divided into three distinct phases to make it more accessible: the opening, the midgame, and the endgame. This separation reduces the game's complexity by introducing subtasks that are easier to understand. Different players excel at different game phases, leading to specialized phase experts.
However, modern chess engines do not make such distinctions when evaluating chess positions. We examine whether adopting the concept of phase experts can also prove beneficial for chess engines. In addition to analyzing the intricacies of typical chess game phase definitions and their transferability to chess engines, we propose and evaluate multiple approaches for combining Mixture of Experts techniques with Monte Carlo Tree Search. We show that having multiple specialized expert networks can lead to significant performance gains compared to the typical single-network approach.

Keywords: Chess, Game Phases, Monte-Carlo Tree Search, Mixture of Experts, Deep Learning

Contents

Glossary

Bullet	Bullet chess describes chess games played with a time control of less than three minutes per player. The most popular bullet time control is one minute for each player without increment.
Centipawn	Centipawn is a unit to measure the advantage in a specific chess position. One centipawn is equivalent to 1% of a pawn.
Chess960	Chess960, also known as Fischer random chess, is a chess variant with 960 possible semi-randomized starting positions.
Crazyara	Open-source neural network chess variant engine [2].
Cute Chess	Library and command-line interface for playing and simulating chess games [3].
Elo	Rating system used to measure the relative strength of a player.
Lichess	lichess.org is a free/libre, open-source chess server powered by volunteers and donations [18].

Acronyms

EM	Expectation-Maximization
GPU	Graphics processing unit
MCTS	Monte Carlo Tree Search
ME	Mixture of Experts
std	Standard Deviation

1 Introduction

1.1 Motivation

Chess is a very complex game that poses various challenges for both beginners and veteran players. To make it more accessible, chess is usually divided into three distinct phases: the opening, the midgame, and the endgame. Each phase has its own difficulties and characteristics, which is why a plethora of theoretical literature covering specific phases exists. Different players excel at different phases of the game, leading to specialized phase experts. Chess engines, however, do not make such distinctions. Modern chess agents such as *AlphaZero* [20] or *Stockfish* use a single neural network for evaluating all positions no matter in which phase the positions appear. We suspect that dividing chess into phases and tackling them individually by training specialized phase expert networks can also benefit engines. Because of the division, each expert has to solve a smaller subspace of the chess game, reducing complexity.

1.2 Problem Formulation

Since chess inherently has no game phases, meaning that there is no fundamental change in the rules of the game after a certain point is reached, we first have to investigate how to divide the game into phases appropriately. Our first research question, therefore, is how to specify typical human approaches to classifying chess positions and whether these criteria can be transferred to chess engines.
Apart from that, it is not clear how to train multiple experts and incorporate them into the typical MCTS procedure. On the one hand, we want to encourage expert localization, but on the other hand, we do not want the experts to overfit on their training set. Instead, we want each expert to be robust to positions it has not seen during training.
We will investigate several approaches to tackle these problems and answer the question

of whether these approaches can achieve better performances than using a single network for all game positions.
Of secondary importance, we want to find out whether the approaches can easily be applied to the chess variant Chess960 that might have different characteristics and demands.

1.3 Outline

This work is structured as follows.
In Chapter 2, we first give an overview of the two relevant methods needed for the experiments in this thesis, namely Monte Carlo Tree Search (MCTS) and Mixture of Experts (ME). Subsequently, we cover related chess agents as well as typical criteria for identifying chess game phases and provide other background information needed to understand the remaining chapters.
Chapter 3 introduces the methods used for our experiments, including the used datasets, the examined approaches, and the evaluation techniques.
In the following Chapter 4, we present the results of our experiments and point out interesting aspects.
Chapter 5 recaps our insights, assesses their reliability, and discusses answers to our research questions.
The final Chapter 6 concludes the thesis by adding final remarks, pointing out limitations, and describing possible future work.

2 Background and Related Work

2.1 Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) is a search and planning algorithm for environments with discrete action spaces. It searches through a tree of the game space, where nodes correspond to states s, and edges correspond to possible actions a. Its goal is to determine the best action in a given state (the root node). In the chess setting, states represent current board positions, while an action corresponds to a chosen ply (a half move) in the current position. MCTS, therefore, learns a so-called *policy* $\pi(a \mid s) = \Pr(a \mid s)$ describing the probability of taking action a in state s. It does so by approximating the expected value for all possible actions by averaging the outcome of rollouts.
A rollout consists of the following four steps.
Selection:
Traverse the search tree using an internal strategy of choosing the actions on the path until an unvisited node or a terminal node is reached.
Expansion:
Expand the search tree with the node reached in the previous step (if it was not a terminal node).
Simulation:
Estimate the value v of the newly found node. In traditional MCTS, this is achieved by doing random simulations (choosing random actions until a terminal node is reached) from this point of the tree and averaging their outcome. Games like chess have three possible outcomes: win (value 1), draw (value 0), or loss (value -1). The values are always written from the perspective of the player whose turn it is in the current node. More modern versions of MCTS (e.g., [21]) use a neural network to estimate the value of the current node instead of doing simulations.
Backpropagation:
Propagate the found value v back along the path to the root node, updating the statistics

of all encountered nodes. For each node, MCTS saves the Q-value (the average or rolling average of v values propagated through the node) and the number of times n this node has been visited.
The rollouts are repeated until the MCTS is stopped when a given maximum execution time or a maximum number of node visits or rollouts has been reached.

A modern strategy for selecting moves during step two is the so-called Polynomial Upper Confidence Bound for Trees (PUCT). It is defined as follows:

$$PUCT(s,a) = Q(s,a) + U(s,a) \quad \text{where} \quad U(s,a) = c_{\text{puct}}\, P(s,a) \frac{\sqrt{\sum_b N(s,b)}}{1 + N(s,a)}.$$

Here, Q is defined as above and represents the exploitation part (playing greedy and using what is currently estimated as best) of the move selection, and U controls the amount of exploration (trying new options to potentially find something better than the current best action). $N(s,a)$ describes the amount action a has been chosen in state s, b represents the possible actions in state s, c_{puct} is a parameter to control the amount of exploration and $P(s,a)$ is the prior probability of selecting action a in state s. This prior probability can be the output of a neural network. Nodes not yet visited are treated as losses and are given a Q-value of -1. During the selection step, MCTS selects the action a that maximizes $PUCT(s,a)$.

After the MCTS is finished, we typically select the action leading to the most visited node.

An extension to MCTS is Monte-Carlo Graph Search (MCGS) [5], which reuses parts of the tree that can be reached through multiple action paths in order to save memory and computation time.

2.2 Mixture of Experts

The term Mixture of Experts (ME) was first introduced in [13]. It is a technique for supervised learning tasks that delegates a prediction to several subsystems called ”experts” and consequently aggregates the individual results to produce a final prediction. The approach is especially beneficial if a natural task division into subtasks exists [13]. Delegation and aggregation are carried out by a gating network denoted as g. For each input, the gating

network decides which experts to use and to which extent each expert should contribute to the final prediction. Weight changes are only applied to the experts used for the specific input (and to the gating network) so that the weights of the different experts do not interfere with each other [13]. This localization leads to experts specializing in specific subareas of the input space. In order to further encourage localization, [13] propose to have the gating network make "a stochastic decision about which single expert to use on each occasion" [13] instead of linearly combining the results of several experts as in [11]. A single prediction is then made as follows:

$$\hat{y} = \sum_i g_i(x, v) \cdot \hat{y}_i \left(x, w_i\right),$$

where $\hat{y}$ is the output, x is the input, w_i are the weights for expert i, $\hat{y}_i$ is the output of expert i, v are the gate weights, and g_i is the gate output for expert i (the probability P_i of choosing expert i) [25].

A typical mean squared error loss can then be calculated as usual over all n samples:

$$\text{MSE} = \frac{1}{n} \sum_{k=1}^{n} \left(y_k - \hat{y}_k\right)^2$$

Figure 2.1 schematically depicts the general procedure of the ME approach. Since its introduction, the approach has been modified and extended in several ways. The experts themselves can be any function approximator ranging from linear models to support vector machines, Gaussian processes, or nonlinear Neural Networks. Nowadays, ME models are usually trained in an iterative fashion using the Expectation-Maximization (EM) algorithm [8]. This algorithm is used to find the maximum likelihood parameters in cases where latent variables are present. Latent variables are variables that influence other variables but are hidden. The EM algorithm consists of two steps:
1. Do an estimation of the missing variables in the dataset
2. Using the estimated values, maximize the parameters of the model
These steps are repeated until an acceptable solution is found. As an initialization, different clustering techniques, such as kmeans [14], can be used.
If the separation into subtasks is natural, training the gating network can be omitted, as we already know which sample to assign to which expert.

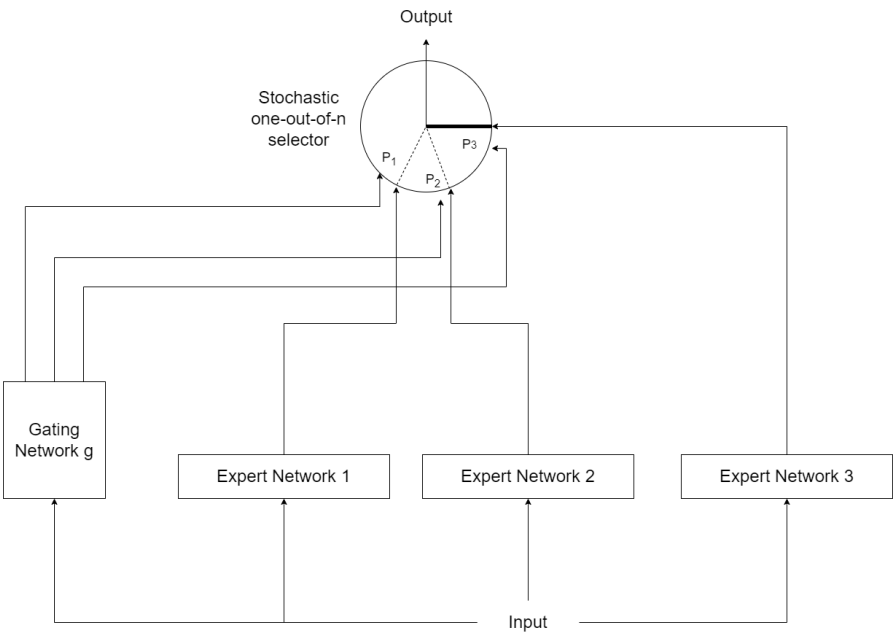


Figure 2.1: Overview of the general Mixture of Experts (ME) structure when using three experts (inspired by [13]). The gating network assigns probability P_i to expert i and the selector makes a stochastic decision about which expert output to pass through (bold) and which ones to reject (dotted).

2.3 Chess

Chess is a deterministic, turn-based, two-player, zero-sum and fully observable board game. A game can end in a win, a draw or a loss.

2.3.1 Neural Chess Agents

The first usage of neural networks in the context of evaluating positions for chess engines was *Giraffe* [15]. However, they aided evaluation with hand-crafted features and did not reach Grandmaster level play. *DeepChess* [7] extracted high-level features of positions using unsupervised learning and used them in combination with a fully connected feed-forward neural network.

In 2017, Google Deepmind published *AlphaZero* [20]. Building upon the successes in Go [21] and [22], *AlphaZero* managed to achieve superhuman performance in chess and other challenging games. It relied exclusively on experience gained from self-play without any domain or expert knowledge.

In the following years, the ideas of *AlphaZero* have been replicated in several open source projects, including *Leela Chess* (now *Lc0* - one of the strongest open source chess engines [17]) and Crazyara (see Section 2.5 for more information).

The state-of-the-art open-source chess engine *Stockfish* initially exclusively relied on hand-crafted features and MiniMax search with alpha-beta pruning without using neural network evaluations. After the introduction of *Efficiently Updatable Neural Networks* (NNUE) in [19], which represent neural networks with fewer parameters that can efficiently be run on CPUs, *Stockfish* was adjusted to incorporate NNUEs, increasing its strength significantly.

The *Ender* network [9] is specifically trained on endgame chess positions. It was originally trained on positions containing a sum of 6, 5, 4, or 3 pieces and pawns to specialize in endgames and surpass other engines that rely on tablebases to solve these ultra-late game positions. They later added positions with a total piece and pawn count of 12. After further adjustments, the engine managed to win the Queen vs Rook endgame. Even though this endgame is theoretically solved, the technique required is very complicated, and *Ender* was the first neural network to learn it. They also performed experiments where a regular chess engine played the early parts of the game, and *Ender* took over when the piece count dropped to 16 or less. This combination called *Franken-Ender*, achieved notable results against other popular chess engines.

Scorpio [24] is another strong chess engine. In its 3.0 release, it added the option of using multiple networks that specialize in certain game phases. Because of good results, this option became the default.

[16] try a new approach of using multiple networks during MCTS. Small networks have faster inference times and can, therefore, lead to more possible simulations and a more extensive tree search. Bigger networks, on the other hand, provide more accurate value estimations. Using both types of networks and finding a balance between their advantages can lead to overall performance gains as shown in [16].

2.3.2 Game Phases of Chess

Chess inherently does not have any game phases, meaning that there are no rule changes or other fundamental differences starting from a certain point in the game. Nevertheless, chess is still often split up into three distinct parts to break down the game's complexity and make it more approachable for players and viewers. These three parts are the opening, the midgame, and the endgame.

However, there is no clear and agreed-upon definition of when the transitions between the phases actually happen. Instead, people attribute general ideas and plans to each phase.

The **opening** usually revolves around controlling the center with pawns, developing the minor pieces from their starting squares, castling the king to safety, and connecting the rooks. Since the start of the opening is fixed, contrary to the start of the other phases, people have acquired heavy theoretical knowledge in terms of learning the correct responses to the opponent's moves as positions will naturally repeat themselves over and over again. Especially because of the easy access to superhuman chess engines and opening databases, people can nowadays carry out excessive preparation for the first phase of the game.

The **midgame** characteristics are more challenging to quantify. People will make long-term strategic plans to improve the structure of the position, start attacks on either side of the board, break up central tension with pawns, and try to exploit weaknesses in the position to win material. As both the start and the end of midgames are not clearly defined, it becomes harder to prepare for this phase in particular, as positions may vary significantly from game to game. One way of characterizing certain midgame types is by checking the

overall pawn structure and position imbalances.

The **endgame** resembles the last phase of the game. It usually involves fewer pieces and shifts the focus to new objectives. The value of specific pieces may change during the endgame, and pawns can become more relevant because they are closer to promotion. The further an endgame progresses, the more likely it is that position setups that are theoretically solved are reached, meaning that the outcome of the game is predefined as long as the best moves are played. For these setups, so-called tablebases exist, which are extensive position databases containing the best move and best reply in all paths the game can take from that point on. The biggest tablebases solve all positions containing seven or fewer pieces, including the kings and pawns.

Typical transition criteria

Move Number:
One typical way of trying to define the start of a phase is by checking the number of moves that have been played until this point of the game. The advantage of this criterion is that it is easy to understand and can easily be applied to both midgame and endgame starts. The disadvantage is that it is not very precise. Games can play out in very different ways, and a position at a specific move number might look totally different in one game than in another. Additionally, chess includes many transpositions, meaning equivalent board positions that can be reached via several move sequences. For example, by going back and forth with pieces or by taking longer routes to a particular square, the same positions can arise but with a different move number.

Materical Count:
Another criterion for game phase identification is the current material or piece count. People usually assign specific relative values to pieces and pawns as a heuristic to judge whether a particular combination of moves will be beneficial for them or not. The typical values are 1 for pawns, 3 for knights and bishops, 5 for rooks, and 9 for queens. Summing up the values of all pieces and pawns currently on the board results in the total material count of a position. It is then possible to define phase transitions according to specific material count thresholds that have to be reached. This criterion is more adjusted to the actual situation in the game than using strict move numbers. However, it is not good at

identifying phase transitions for games with few piece trades that involve long strategic maneuvering instead.

Lichess game phase definitions

Lichess is an open-source chess server for online play, study, and analysis. State-of-the-art chess engines power their analysis section, which is why players of all skill levels, from beginner to master, use them on a daily basis. Submitting a game for engine analysis results in getting a report of the game development, including a separation of the game in the three typical phases to teach people where they went wrong and how they could improve. The site uses a more sophisticated system to determine game phases by incorporating several transition criteria.

Endgame definition:
According to the Lichess implementation, a position belongs to the endgame if the total count of major and minor pieces (queens, rooks, bishops, and knights) is less than or equal to 6. Note that while this resembles a material count criterion, it does not use different relative values for the pieces and instead values them all equally as 1. A potential reason for this approach could be that the complexity of a position is not tied to the relative value of its pieces but rather to their total amount. Furthermore, the pawn count is irrelevant to the decision.

Midgame definition:
A position counts towards the midgame if it does not qualify as an endgame position and if one of the following three criteria is fulfilled. The number of major and minor pieces is less than or equal to 10, the backrank of at least one player is sparse, or the total mixedness score of the position is bigger than 150. Here, backrank sparseness is defined as having less than four total pieces on rank 1 (for white) or 8 (for black), including the king. The mixedness score describes how close black and white pieces are to each other. It is calculated by going through all two by two squares of the chess board, starting from the square a1, b1, a2, b2 and ending at the square g7, h7, g8, h8. For each of those two by two squares, we count the number of black and white pieces inside it and assign a score based on the result and the square's location. We then sum up all square scores to get the final mixedness score of a position. The exact implementation can be found in [10].

Opening definition:
All remaining positions are classified as opening positions.

Using the previously described phase definitions may lead to transitions to previous phases (e.g., going back to the opening because the mixedness score has increased again). Therefore, to do a strict separation into three sections, Lichess forbids such transitions and only allows transitions to later phases.

2.3.3 Chess960

Chess960, also called *Fischer random chess*, is a popular chess variant that uses the same board and the same pieces as regular chess. Its main difference is the randomized starting positions. The pieces in each player's back rank are set up in semi-random order, with the only restriction of having one rook on each side of the king. This limitation leads to 960 possible starting positions, which gives the mode its name and makes memorizing opening theory nearly impossible. The rules are identical to regular chess with the exception of special castling rules.

2.4 Elo Ratings

The Elo rating system, invented by Arpad Elo, is used to measure relative skill differences between players of a game. Everyone starts with the same arbitrary starting value, and the ratings are adjusted from that point on based on the outcome of finished games. Winning games increases the Elo rating, while losing decreases it. Using the rating of two players A and B, it is possible to calculate the expected score of player A (and player B by replacing $R_\mathrm{B} - R_\mathrm{A}$ with $R_\mathrm{A} - R_\mathrm{B}$):

$$E_\mathrm{A} = \frac{1}{1 + 10^{(R_\mathrm{B} - R_\mathrm{A})/400}}. \tag{2.1}$$

2.5 Crazyara

Crazyara is an open-source chess engine originally trained in supervised style for the Crazyhouse chess variant, beating the 2017 Crazyhouse world champion Justin Tan [6].

It was later adapted to incorporate *Reinforcement Learning* and reimplement the ideas of *AlphaZero*. The Crazyara repository now features multiple versions like *ClassicAra* (for chess and Chess960) and *MultiAra* (for all Lichess variants). The implementation for this thesis builds upon the *ClassicAra* repository to add multiple expert neural networks. If not explicitly specified otherwise, we use the hyperparameter settings found in the repository's main branch as of September 2023 [1].

2.6 Cute Chess

Cute Chess is a set of cross-platform tools for working with chess-engines [3]. Its command-line interface *cutechess-cli* can be used to simulate chess matches between two or more engines. Several options that specify the characteristics of the simulation exist. The amount of time available for each engine is set via the *tc* parameter.

[1] https://github.com/QueensGambit/CrazyAra/commit/b566c045a7da852d0bdbc9960225e00616fd434f

3 Methodology

This chapter describes our experiments, including the used datasets, model architectures, evaluation criteria, and chosen hyperparameters for both the training and the MCTS.

3.1 Chosen Game Phase Definition

We base our game phase separation on the implementation of Lichess introduced in Section 2.3.2. Their definition, in our opinion, comes closest to what players intuitively use while judging a board position, as it incorporates several criteria for leaving the opening phase (board mixedness, back-rank sparseness, and piece count) that all contribute to the common interpretation of chess game phase transitions. In addition to that, the site is used by millions of chess players every day to analyze their games, which is why this definition is well-known and established in the chess community.

We also deliberately do not rely on the current move counter to determine the game phase of a specific position, which makes how the position was reached irrelevant. This decision is consistent with the commonly chosen input representation of chess boards (see Section 3.3), which also does not include the number of moves made until this point in the game, as it should be irrelevant for evaluating a particular position.

The fact that we want to treat every position equally no matter how it was reached leads to a slight difference in our definition compared to that of Lichess. The website does not allow for transitions back to a previous phase in order to ensure the game can be fully divided into three separate sections. For our purpose, we do not make such restrictions as positions are treated independently and not in the context of the game they appeared in. Apart from that, we closely follow the criteria introduced in Section 2.3.2.

We assume that using these criteria splits the available positions into three approximately equally sized buckets. We will verify the correctness of this assumption and analyze the distribution of game phases in Section 4.1.

3.2 Dataset Preparation

Our chess datasets are built based on the Kingbase Lite 2019 database [1]. This database includes over one million chess games from players rated at least 2200 Elo. We filter out all games shorter than five moves in total as these games are either quickly arranged draws or errors in the database and, therefore, unreliable sources. In order to create the training and evaluation data for our experiments, we build four datasets. The first dataset, which we will refer to as ”no-phases”, contains all positions left after filtering the database as described above. For the remaining three datasets (”opening”, ”midgame” and ”endgame”), we strictly split the database into one part for each of the three phases defined in Section 3.1. This splitting is carried out by going through every game in the dataset and removing all board positions that do not belong to the phase for which we are currently preparing the dataset. This procedure leads to some games being completely excluded for a specific part if they contain no position of the particular phase. For example, about 32 percent of games ended before the endgame began, therefore contributing no position for the endgame dataset. More detailed analyses of the distribution of the different phases and the resulting dataset sizes can be found in Section 4.1.2.

For the Chess960 datasets, we export and combine the Lichess monthly game databases from August 2013 to July 2020. To ensure a reasonable game quality, we only include games where both players had a rating of at least 1950. This criterion approximately corresponds to the top ten percent of all active Chess960 players on Lichess in terms of playing strength. As Chess960 is a much more complicated environment with a considerably larger variety of positions due to the 960 different possible starting positions, we decided to include the games of all available time controls to cover as many positions from high-level games as possible. This decision, however, also lowers the overall quality of moves since players make more mistakes when given less time to think. Especially Bullet games often end in a time scramble phase where both players play extremely fast, with the primary purpose of winning on time rather than trying to win on the board. The four Chess960 datasets (”no-phases”, ”opening”, ”midgame” and ”endgame”) are created analogously to the way described in the previous paragraph. We also use the same game

[1] https://archive.org/details/KingBaseLite2019

phase definitions as for regular chess, making the assumption that the chosen criteria are still appropriate as Chess960 games play out in an approximately comparable style to regular chess games, especially the further a game proceeds. There may, however, be more adequate game phase definitions for Chess960 that are specially customized to incorporate the intricacies of this variant.

We split each of the four types of datasets into a training, an evaluation, and a testing set to be able to evaluate the models on unseen data and to monitor the amount of overfitting. The testing and evaluation parts each contain 1000 games of one distinct month of game data, while the training part contains the games of all remaining months. There might be slight distribution shifts when comparing the game data of different months caused by changes in the chess meta. Examples of this might be newly discovered opening lines or recent successes in particular lines that lead to more players playing them in the following month. However, we assume that these trends are too minuscule to bias the evaluation and training sets in a meaningful way.

3.3 Model Architecture and Input Representation

We use the RISEv3.3 model architecture for all approaches, which is an improved version of the RISEv2 architecture introduced in [6]. It is a convolutional neural network with residual blocks and two output heads (a value head and a policy head). Compared to its previous version, RISEv3.3 makes use of 5x5 convolutions in deeper layers and adjusts the structure of the channels. Detailed information can be found in the CrazyAra wiki [23]. We did not change our input representation compared to the default version 3 in the CrazyAra repository [2], as this representation is already established and leads to good results as examined in [4].

3.4 Experiment Types

In order to test the effectiveness of using the ME technique for MCTS, we evaluate three main approaches on both regular chess and Chess960. We compare our approaches against the typical setup, which we will use as a baseline and refer to as "Regular Learning", of

[2] September 2023, commit: https://github.com/HelpstoneX/CrazyAra/commit/b566c045a7da852d0bdbc9960225e00616fd434f

training a single network on the full dataset of game positions and using this network for all predictions.

3.4.1 General Mixture of Experts Procedure

For each of our approaches, explained in the following, we generally aim to have one specialized expert for each of the three game phases defined in Section 3.1. The advantage of choosing strict and fixed definitions for the inputs each expert belongs to prior to the training is that we do not need to learn this delegation on the fly. Therefore, we do not need to employ any EM techniques (compare Section 2.2) for iteratively arrowing out, how to split the input space of game positions, and how to determine adequate expert probabilities for a given input. Instead, the decision about which phase a game position belongs to can be made with absolute certainty by following the chosen definition. Therefore, our ME gate is no trained classifier but instead a deterministic algorithm that strictly follows the game phase definitions and always assigns 100% probability to one of the experts and 0% to the others. Note that our gate still matches the general ME procedure introduced in Section 2.2 and Figure 2.1. We can view it as a classifier that outputs a discrete probability distribution in its most extreme form by putting all the probability mass on one of the experts:

$$P_i = 1 \cdot \delta_{il}, \quad \delta_{il} := \begin{cases} 1 & : i = l \\ 0 & : i \neq l \end{cases}$$

where P_i describes the probability of expert i being used and l describes the phase of the current position.

By choosing a fixed definition of phases, we eliminate the possibility of the gate assigning responsibility to the wrong experts. This predetermined splitting of the input space simplifies the learning process. We do, however, in return, lose the possibility of learning criteria for the different phases that would be even more appropriate and lead to even better solutions.

The three approaches we will describe in the following subsections only differ in the way they are trained and in the dataset parts they use. The general setup of having a deterministic gate that always assigns full responsibility to a single expert stays the same.

3.4.2 Approach 1: Separated Learning

The first approach, which we call ”Separated Learning”, refers to strictly separating the three expert networks by only training them on positions of the specific game phase they belong to.

The goal is to make learning easier for each expert network by reducing the whole input space of all possible positions to a smaller subspace of positions this expert has to cover. The drawback of this approach is that each expert network has no information about the intricacies of the other two phases, which can lead to problems when transitioning from one phase to another during the MCTS, as will be explained in Section 3.5.

Since we strictly split the whole dataset into distinct parts for each expert, the number of total positions used for training remains the same compared to the ”Regular Learning” approach. The experts can be trained entirely separately, and the complete training of all three experts takes approximately as much GPU time as training the single network for the baseline approach.

3.4.3 Approach 2: Continual Learning

Our second approach, denoted as ”Continual Learning”, describes the idea of first training a network on the positions of one phase and then, subsequently, using the resulting network weights as an initialization for a consecutive training run on the next phase. This process is repeated until the network has gone through all three phases. The last phase the network is trained on determines which expert (”opening”, ”midgame” or ”endgame”) this network resembles. For example, for the endgame expert, we first train the network on all opening positions, then use the final weights as a warm start for a training run on all midgame positions, and then, finally, use these resulting weights for initializing a final training run on all endgame positions. After each run, we reset the optimizer state and only keep the network weights. Consequently, each individual run has the same learning rate schedule, and no momentum calculations are carried over between runs. We also execute each run for the same number of epochs (see Section 3.7 for an overview of the most relevant chosen hyperparameters).

The general motivation behind this approach lies in feeding the experts knowledge about the other two phases while still keeping them primarily focused on the main phase they belong to (the last one they are trained on). Even though the training set is changed for

expert	train set 1	train set 2	train set 3
opening	endgame	midgame	opening
midgame	opening	endgame	midgame
endgame	opening	midgame	endgame

Table 3.1: The order in which the different experts of the Separated Learning approach use the different phase training sets.

each run, we expect the resulting experts not to forget everything they learned in previous runs, making them more robust to the positions of other phases.

The order in which we execute the three training runs for each expert is based on the similarity of the three phases. We start with the phase the expert's main phase is furthest away from and continue with the closest phase. For opening and endgame experts, the order of training runs is natural based on the order of the game phases (opening $\longrightarrow$ midgame $\longrightarrow$ endgame). For the midgame expert, it is not obvious whether to start with the endgame or with the opening run. We made the assumption that endgame and midgame positions are more similar to each other than opening and midgame positions and, therefore, started with the opening run for the midgame expert. All three chosen training orders can be seen in Table 3.1.

For this approach, training a single expert takes just as long as training the single baseline network since each expert has gone through all available positions after its three training runs are completed. Therefore, the total amount of GPU time is tripled in this approach.

3.4.4 Approach 3: Weighted Learning

Our third approach, "Weighted Learning", is based on the idea of giving each sample a different weight according to the phase its position belongs to. By doing so, we allow each expert to learn based on all available positions in the dataset while still incentivizing it to specialize in its specific phase by weighting this phase higher. We, therefore, do not weigh a sample based on the class/output it belongs to, as it is commonly done for class-imbalanced datasets, but rather weigh it based on its input. The weighting is implemented by adjusting the loss of each batch based on the game phases occurring in the batch. In particular, we multiply the individual loss of each sample with its weight

and, subsequently, take the arithmetic mean to get the overall loss for a specific batch:

$$L_{weighted} = \frac{1}{B} \sum_{n=1}^{B} w_n L(x_n),$$

where B is the batch size, w_n is the weight of a particular sample, and $L(x_n)$ is the regular sample loss. Note that the regular (unweighted) batch loss of the baseline network is essentially a specialized version of the weighted batch loss with $w_n = 1.0$ for all samples.

We set the weights in a way that all samples belonging to the main phase of each expert are a times as important as the samples of the other two phases:

$$w_{\text{main}} = a \cdot w_{\text{other}}, \quad a \geq 1.0, \tag{3.1}$$

where w_{main} specifies the weight of the main phase samples, and w_{other} specifies the weight of the samples of the other two phases.

Furthermore, we normalize the phase weight values in a way that they do not significantly alter the overall magnitude of the loss, meaning that the average sample weight should still be about 1.0, just as for the baseline network. Assuming each phase has approximately the same amount of samples, we only have to normalize the overall phase weights by ensuring the following equation holds:

$$\frac{w_{\text{main}} + 2 \cdot w_{\text{other}}}{3} = 1.0 \tag{3.2}$$

Inserting Equation 3.1 in 3.2, we get:

$$\Leftrightarrow \quad \frac{a \cdot w_{\text{other}} + 2 \cdot w_{\text{other}}}{3} = 1.0 \tag{3.3}$$

$$\Leftrightarrow \quad \frac{(a+2) \cdot w_{\text{other}}}{3} = 1.0 \tag{3.4}$$

$$\Leftrightarrow \quad w_{\text{other}} = \frac{3}{a+2} \tag{3.5}$$

and, consequently, by inserting 3.5 in 3.1

Table 3.2: Sample weights for the Weighted Learning Approach

a	**expert main phase**	$w_{\textbf{main}}$	$w_{\textbf{other}}$	$w_{\textbf{opening}}$	$w_{\textbf{midgame}}$	$w_{\textbf{endgame}}$
4	opening	**2**	0.5	**2**	0.5	0.5
	midgame	**2**	0.5	0.5	**2**	0.5
	endgame	**2**	0.5	0.5	0.5	**2**
10	opening	**2.5**	0.25	**2.5**	0.25	0.25
	midgame	**2.5**	0.25	0.25	**2.5**	0.25
	endgame	**2.5**	0.25	0.25	0.25	**2.5**

$$w_{\text{main}} = \frac{3 \cdot a}{a+2}, \quad a \geq 1.0. \tag{3.6}$$

We carry out experiments for two different a values ($a \in \{4, 10\}$) to see whether a big weight difference (more expert specialization) or a small weight difference (more expert similarity) leads to better overall performance. The values of the resulting normalized sample weights can be found in Table 3.2.

Since the samples are shuffled, the game phase distribution differs from batch to batch. Therefore, the average sample weight in each batch is not necessarily close to 1.0, which could lead to losses changing significantly from batch to batch. This behavior would hinder the learning process, e.g., by incorrectly assuming a low batch loss was caused by a good network update, even though the actual reason was that the batch contained significantly fewer samples of the main phase than other batches. However, we assume that due to the chosen high batch size of 2048 (see Section 3.7), the batch phase distributions will approximately follow the overall phase distribution so that the loss will not significantly differ between different batches.

The weights are applied to both the training and the evaluation sets. The test set remains unweighted to be able to compare the final performance of the trained agents with the other approaches.

Since each expert network is trained on all samples in the dataset (albeit with different sample weights), every expert training run takes as much time as the training of the baseline network. Therefore, this approach triples the amount of GPU time needed.

3.4.5 Expert Impact

In order to determine which of the three expert networks has the most impact on playing strength, we perform additional evaluations that mix baseline and expert networks. Instead of combining all three phase experts and comparing the resulting ME to the baseline network, we only use a single expert and replace the other two experts with the baseline network. For example, to test the impact of the opening expert, all opening positions will be delegated to the opening network as usual, but all remaining positions will be delegated to the baseline network that was trained on all phases. We, therefore, create an ME consisting of the opening expert (for opening positions), the baseline network (for midgame positions), and again the baseline network (for endgame positions). This ME is then compared to the baseline approach, and the procedure is repeated accordingly for all three phases.

3.5 Mixture of Experts in MCTS

For each of the three ME approaches described in the previous section, we end up with one model for each of the three defined game phases. Ideally, we only query the network that corresponds to the current position of the MCTS. This strategy, however, does not work when using MCTS batch sizes bigger than 1, meaning that we buffer multiple positions before doing neural network inference. The reasoning behind using bigger batch sizes is that it improves GPU utilization. In return, it decreases the update frequency of the node statistics. For MCTS batch sizes bigger than 1, we analyze all positions in the current batch and determine the most occurring phase. Subsequently, we query the expert that corresponds to this majority phase. Consequently, we do not have to adjust the general workflow of the MCTS, but in return, it will be possible to have some positions processed by the wrong expert network. These situations should, however, primarily occur when the game transitions from one phase to another, meaning that these positions are at least close to belonging to the expert they were (incorrectly) processed by.

3.6 Evaluation

3.6.1 Training Evaluation

During training, we evaluate the model after every 500 iterations, where one iteration describes passing one batch and doing backpropagation. In addition to an evaluation on the evaluation dataset of the currently trained expert, we perform an additional evaluation on all four test sets (opening, midgame, endgame, and no-phases, see Section 3.2) to see how each expert's expertise on the individual subsets develops during the training process. We save a checkpoint of the current model state if the evaluation loss is better than the best previously measured evaluation loss. Our final model corresponds to the last saved checkpoint, i.e., the model state that performed best on the evaluation set.

We reset the model to a previous state during training if the loss on the evaluation set increased significantly from one evaluation cycle to the next. In particular, these loss spike recoveries are carried out if $L_{\text{prev}} \cdot s < L_{\text{curr}}$, with $s = 1.5$ being the spike recovery threshold, L_{prev} being the loss during the previous evaluation and L_{curr} being the current loss.

3.6.2 MCTS Evaluation

After the experts have been trained, we integrate them in the MCTS as described in the previous section. The resulting agent is then matched against the baseline "Regular Learning" approach using the Cute Chess command-line interface (see Section 2.6). We perform a 1000-game match and calculate the relative playing skill of the currently evaluated approach and the baseline approach based on the resulting score of the match. Wins award 1 point, draws are worth 0.5 points, and losses count as 0 points. By using Formula 2.1 defined in Section 2.4, we can derive the relative Elo difference by substituting the expected score with the actual score achieved in the match and solving for the rating difference $R_{\text{B}} - R_{\text{A}}$.

We also provide 95% confidence bounds for the Elo differences based on the number of games played and the resulting scores. The exact implementation is adapted from the implementation used in the Cute Chess repository (see [12]).

Opening suites: In order to provide a diversified playing ground, the games do not start with the initial board position of chess. Instead, we make use of an opening suite containing 31526 chess positions that feature positions appearing after the first several moves have already been played. These positions are intentionally chosen in a way that each position is imbalanced, with a slight edge for either side to reduce the amount of resulting draws.

For each match, we randomly sample 500 opening positions from this opening suite and let our agents play against each other starting from there. In order to take the imbalances of the positions into account, we use each starting position twice so that each agent is playing once as white and once as black.

For the Chess960 experiments, we use an opening suite that consists of all 960 possible starting positions. We again randomly sample 500 positions and assign both colors to each agent, resulting in 1000 games per match.

Game endings: A game can end in several different ways. Games result in a draw or win/loss according to the regular chess rules. Checkmates and illegal moves lead to a win/loss, while draws are caused by 3-fold repetition, insufficient mating material, stalemate, or the fifty-move rule. Additionally, a game is counted as a draw if each engine's evaluation of the position is within 20 Centipawns from zero for at least four consecutive moves. The consecutive move counter is reset after each capture or pawn move. This rule only applies after at least 30 moves have been made. Losses are also awarded if an engine's position evaluation is at least 600 Centipawns below zero for five consecutive moves. As our MCTS outputs resemble expected scores rather than Centipawn advantages, we convert them as follows:

$$\text{cp} = -\frac{v}{|v|} \cdot \log \frac{1 - |v|}{\log \lambda},$$

where $\lambda = 1.2$ [6].

Time control: The available time for both agents is not limited via the Cute Chess tc (time control) parameter but instead managed by explicitly specifying either a fixed available move time per move or by specifying the number of nodes that can be visited before a move has to be made. We test our agents with different move times and node values (see Section 3.8).

Misc:
We do not give our agents access to any tablebases to also test their strength in theoretically solved endgames.

All experiments are executed on a Tesla V100 GPU.

3.7 Training Configuration

To provide a fair comparison, we train our models for seven epochs (20 epochs for chess960) regardless of their phase. We use a batch size of 2048 and randomize the order of our samples. Our model weights are updated via stochastic gradient descent with Nesterov momentum [1]. All remaining hyperparameters are set to their default values in the train_cnn notebook of the Crazyara repository in September 2023 [3].

3.8 MCTS Configuration

We disable reinforcement learning to examine the direct impact of our ME approaches for the supervised learning part. We evaluate our approaches with different values for the following parameters. The *Batch_Size* parameter (explained in Section 3.5) is set to 1, 8, 16, 32, and 64. The *Nodes* parameter (values of 0, 100, 200, 400, 800, 1600, 3200) limits the number of nodes that the MCTS can visit per move during its search. The *Simulations* parameter describes the number of simulations per move and is set to double the *Nodes* value. When the *Nodes* parameter is 0, we instead limit the extent of the search by a fixed move time in milliseconds (*Fixed_Movetime* values of 100, 200, 400, 800, 1600). The *Search_Type* is set to *mcts*.

[3] https://github.com/QueensGambit/CrazyAra/commit/b566c045a7da852d0bdbc9960225e00616fd434f

4 Experimental Results

4.1 Dataset and Game Phase Analysis

4.1.1 Game Phase Statistics

This section analyses the different phase transition criteria and evaluates the resulting game phase distribution. All Figures are based on the data in the chess test set.

Figure 4.1 shows the development of the mixedness score throughout a game. The data resembles all games in the chess test set, and the positions are aggregated every five moves. As defined in Section 2.3.2, the mixedness score measures how close black and white pieces are in all two by two subareas of the chess board: the more intertwining, the higher the resulting mixedness score. We see that the average mixedness quickly ramps up during the first 15 moves of the game, which is what we expect from a typical game where people develop their pieces and try to control the center with pawns. This increase continues until around move 30 and then slowly declines, which can be attributed to the fact that the players will trade more and more pieces and pawns, which leaves less material on the board. Notably, the variance of the mixedness score is very high throughout the game, and many outlier positions exist. This problem becomes irrelevant in the game's later stages as the phase transition most likely already happened. Nevertheless, we conclude that this score alone, due to its variance, would not be a good indicator for transitioning to the midgame. However, we think that mixedness is a reasonable inclusion in cases where the other two phase transition criteria (pieces left and backrank sparseness) fail to detect that the game is not in the opening anymore. Examples would be very closed and strategic positions where few pieces have been traded, and the back rank might not be sparse yet. By common judgment, these positions also count as midgames, which the mixedness criterion ensures.

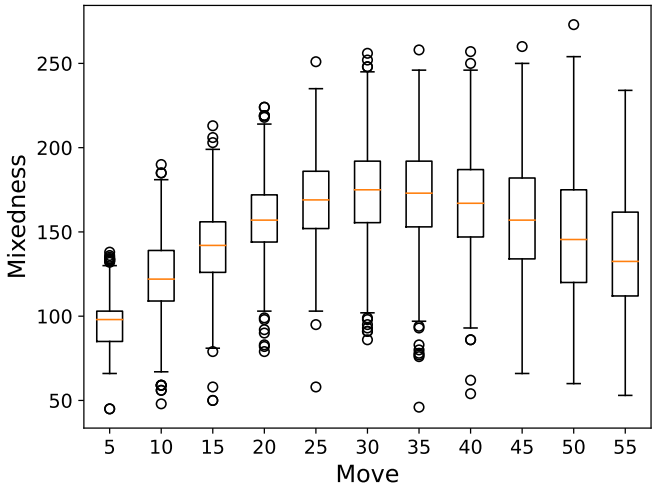


Figure 4.1: The mixedness scores (according to the definition in Section 2.3.2) of chess game aggregated by move number. The data is based on all positions of the chess test set.

Similar conclusions can be drawn by looking at the second phase transition criterion: Figure 4.2 depicts the progression of major and minor piece counts throughout a chess game. The positions stem from the chess test dataset and are aggregated every five moves. As expected, the median piece count monotonically decreases as the game processes. Contrary to the mixedness score, the variance of this criterion does not continuously increase with higher move numbers but instead decreases again during the game's later stages. Nevertheless, the overall variance is still high apart from the first move bucket. Consequently, there is a substantial amount of positions with 15 to 25 moves played where no or nearly no pieces have been exchanged. According to this criterion alone, these positions would still count as opening positions, which is not sensible. Apart from that, we believe this criterion is a meaningful first indicator to judge a position's game phase as it is closely tied to the human judgment of positions and is much less complicated than the mixedness score.

As explained in Section 2.3.2, the last midgame transitioning criterion, backrank sparseness, is of boolean nature. We iterate over all move numbers and calculate the percentage of positions where the backrank of at least one player is sparse. Figure 4.3 shows the resulting graph. We can see that the chance of backrank sparseness monotonically increases as a chess game progresses. There are zero positions with a sparse backrank from move

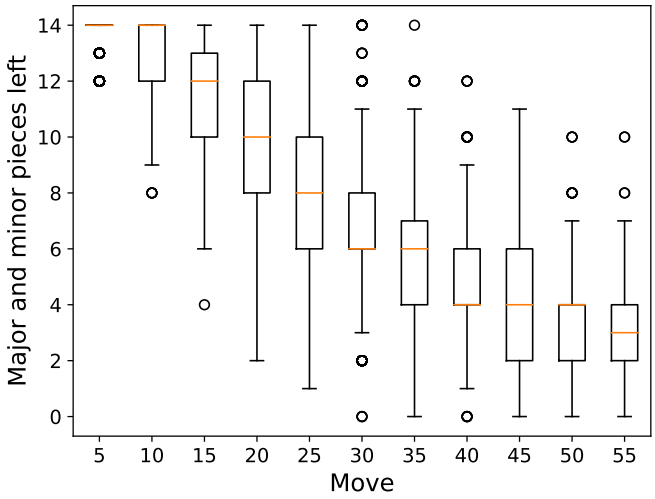


Figure 4.2: The average number of major and minor pieces left aggregated by move number. The data is based on all positions of the chess test set.

zero to move eight. Consequently, these positions would never be classified as midgames if only the backrank criterion was used. There are, however, famous opening variants that involve heavy piece trades before move eight like the Berlin endgame. Thus, this criterion should also not be used in isolation.

Figure 4.4 shows the distribution of starting moves of each phase according to our chosen phase definitions (see Section 3.1). The opening always starts on move zero by default, so it is excluded. The data emphasizes that hard move cutoffs for phase separations would lead to a much more superficial and less sophisticated splitting of the game. In our case, the starting moves for a midgame range from move 6 to move 22, and the starting endgame moves range from 11 to 60, which shows how diversely chess games can play out. The average move number of all midgame starts is 11.89, with a standard deviation of 2.966. For the endgame starts, the average is 28.15, with a standard deviation of 7.441.

The distribution of all positions by phase is depicted in Figure 4.5a. We display the histograms for all opening (blue), midgame (orange), and endgame (green) positions separately to show the amount of overlap between the different phases for specific move counts. We can again see that a strict separation by move number would lead to a very different sample distribution for the three phases, no matter how the move cutoffs would

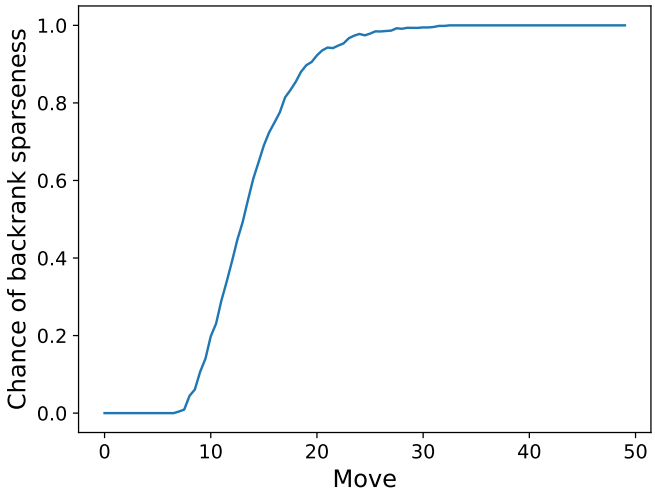


Figure 4.3: The percentage of positions with a sparse backrank by move number. The data is based on all positions of the chess test set.

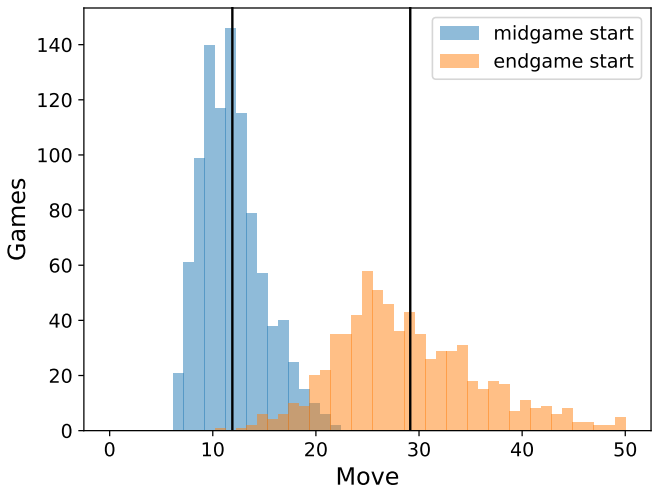
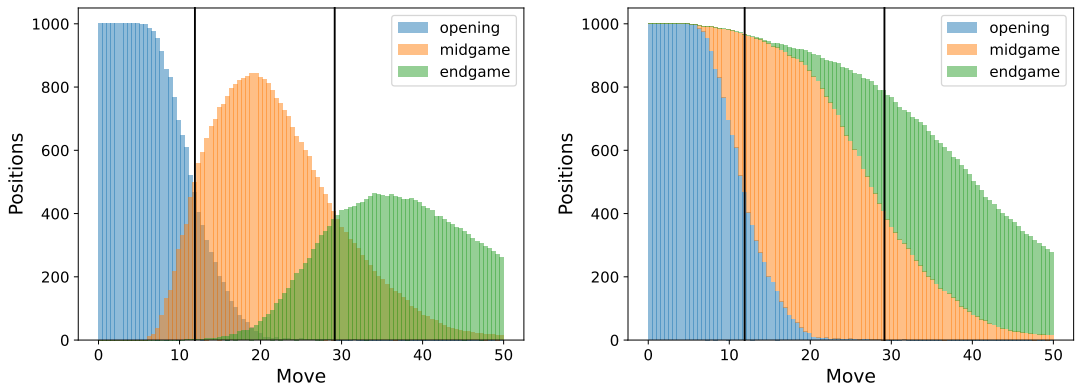


Figure 4.4: The distribution of starting moves of each phase aggregated by move number. The average starting moves of the midgame (11.89 with an std of 2.966) and the endgame (29.15 with an std of 7.441) are added as vertical lines. The data is based on all games of the chess test set.

be chosen. The overlap is especially significant between midgames and endgames, with nearly 50 different move numbers (move 11 to move 60) contributing positions for both the mid- and the endgame. Since endgame starts are decided by the number of major and minor pieces left, it is natural that the variance of this transition is very high as the piece trading behavior can differ considerably from game to game. This discrepancy is caused not only by different middle game plans of different openings but also by varying playstyles. Some players intentionally keep the tension and leave many pieces on the board to make the game more complicated, while others often try to trade down to an endgame, maybe because they excel in this phase. Notably, opening moves appear for a much smaller range of move numbers than the other two phases. The relevant move numbers are even less when excluding the first several moves, which are often irrelevant in benchmark games if an opening suite is used. Even though midgames form the vast majority of positions around move 18, no move number exclusively belongs to them, as even the opening and endgame histograms overlap. Therefore, some move numbers contain positions of all three phases (e.g., move 18).

Figure 4.5b shows the same data in a single stacked histogram. Naturally, the number of positions strictly monotonically decreases with higher move numbers. The naive approach of splitting the game purely by equidistant move number values would, therefore, lead to many more positions in the opening phase than in the other phases. An example of this would be using move 0 as the opening start, move 20 as the midgame start, and move 40 as the endgame start, assuming a typical game does not last longer than 60 moves. In our case, the decreasing sample size by move number is counteracted by a shorter overall move span of the opening.

One last thing to examine is the number of jumps back to a previous phase during a game. As explained before, the common interpretation of game phases, as well as the Lichess definition, enforce strictly splitting the game into three distinct sections. In order to always treat the same position equally no matter at which point in the game it appeared, we get rid of this restriction. Consequently, there may be cases where a game transitions from a more advanced phase back to a previous one. We assumed that the number of these transitions is relatively small and, therefore, negligible. The distribution of jumps to previous phases in the chess test set is shown in Figure 4.6. We see that 71.6% of games contain no such jumps, 20.4% of games have exactly one jump, and the percentages quickly decrease from there. There were no games with more than five jumps in the test set, and the average jump count per game was 0.38. 28.4% of all games had at least one jump, 28% of which were jumps from midgame to opening. Interestingly, there even was a game transitioning from an endgame to a midgame. This scenario can only happen if the number of major and minor pieces increases again to a value of more than 6. The way this can be achieved



(a) Individual (b) Stacked

Figure 4.5: The distribution of the number of positions for the three game phases aggregated by move number. Phase separated histograms in Fig. 4.5a and stacked histogram in Fig. 4.5b. The average starting moves of the midgame (11.89 with an std of 2.966) and the endgame (29.15 with an std of 7.441) are added as vertical lines. The data is based on all positions of the chess test set.

is through promoting a pawn. The following Lichess study shows an exemplary game with many transitions to previous phases: `https://lichess.org/study/3rI5SJXi`. There are annotations for the phase transitions, and the mixedness score is also added for the early game transitions. This game is an extreme case and was not part of our dataset.

4.1.2 Resulting Dataset Statistics

Following our game phase definitions described in Section 3.1, we can analyze the input and output distributions of the resulting datasets of each phase.

Figure 4.7 illustrates our training set's relative game outcome distributions. A game counts towards the data of a specific phase if it contributes at least one position of that phase. Consequently, the bars representing the opening phase (blue) are equivalent to bars representing all games in the dataset since every game has at least one opening position. White wins in 34.42% of all games, 40.36% of games end in a draw, and the remaining 25.22% are wins for black. We can detect that games that reach the midgame are less likely to end up as draws (38.51%). This trend continues and is even more apparent for

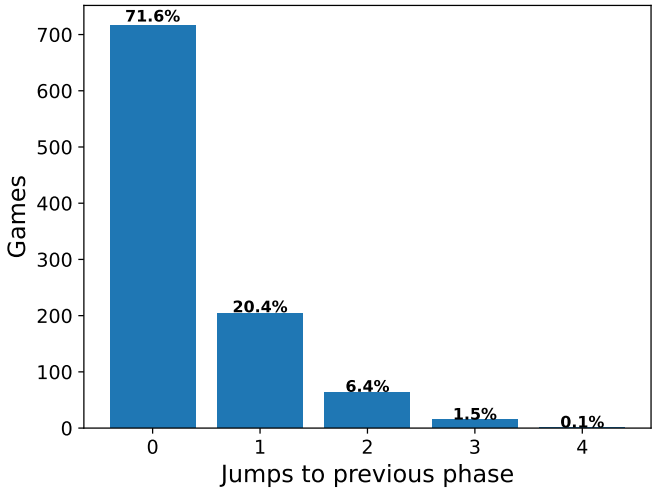


Figure 4.6: The distribution of transitions to previous game phases. The average number of such transitions per game was 0.38.

games that reach the endgame (36.09%). Conversely, the percentage of black and white wins increases in the later phases.

The game phase distributions in our training set are depicted in Figure 4.8. Almost all games (96.81%) reach the midgame for at least a single move. For the endgame, however, this number is much smaller. There are 357 716 games (32.15%) with no endgame move. This imbalance is, however, counteracted by the fact that endgames usually last substantially longer than the other two phases. As a result (see Fig. 4.8b), the training set even contains more endgame than opening positions, with 31.63% compared to 28.67%. The midgame stands out with 36 287 651 total positions, which corresponds to 39.7%.

Table 4.1 shows a comprehensive overview of the size of all phase datasets we used for our experiments. The full dataset contains 1,112,647 games leading to 91,413,951 positions. Table 4.2 summarizes the number of games that contributed to the samples of the particular datasets (at least one sample).

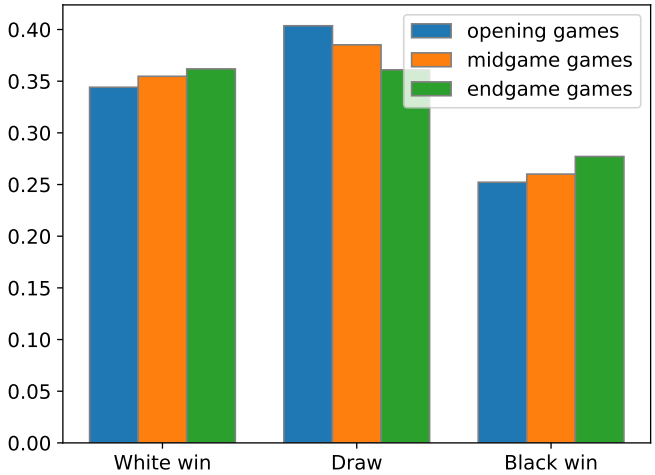
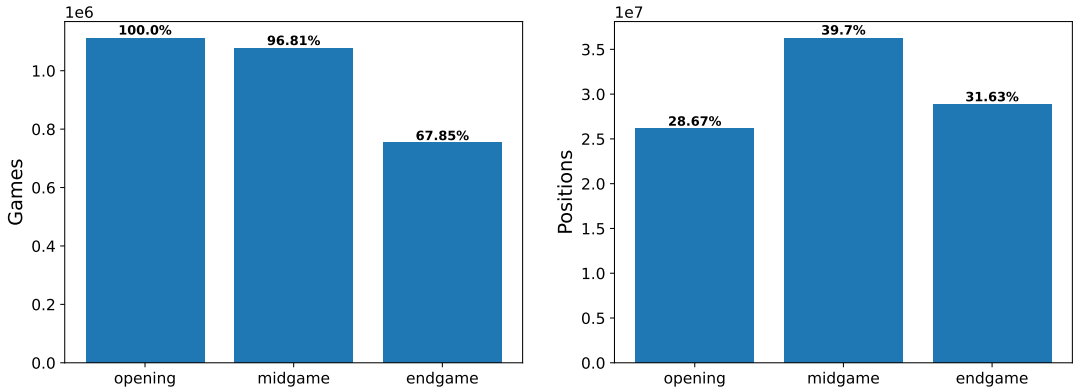


Figure 4.7: Relative distribution of game outcomes in the chess training set. The percentages are calculated based on all games with at least one move of the specific phase.



(a) Games by phase

(b) Positions by phase

Figure 4.8: Game phase distribution in the chess training set. Total games by game phase in Fig. 4.8a and total positions by game phase in Fig. 4.8b.

dataset/ phase	no-phases	opening	midgame	endgame
train chess	91 413 951	26 212 273	36 287 651	28 914 027
val chess	79 042	24 566	29 333	25 143
test chess	85 114	24 938	31 364	28 812
train Chess960	22 383 691	7 213 158	8 479 268	6 691 265
val Chess960	71 412	22 540	26 200	22 672
test Chess960	68 770	22 628	26 377	19 765

Table 4.1: Overview of all dataset sizes used for our experiments. The values correspond to the number of **positions** in a particular dataset.

dataset/ phase	no-phases	opening	midgame	endgame
train chess	1 112 647	1 112 647	1 077 136	754 899
val chess	1000	1000	947	647
test chess	1000	1000	971	717
train chess960	317 417	317 417	306 838	195 374
val chess960	1000	1000	970	621
test chess960	1000	1000	961	594

Table 4.2: Overview of all dataset sizes used for our experiments. The values correspond to the number of **games** in a particular dataset that contributed at least one position of the specific phase.

4.2 Chess Results

In this section, we examine the effectiveness of the three approaches for chess. We show how the evaluation metrics develop over the course of the training process and how much relative Elo gain each approach achieves. The Elo gains are derived based on the results of 1000 game matches against the baseline approach. We use five different batch sizes and eleven values to control the search length, leading to 55 matches in total. Additionally, we analyze each expert's impact during the MCTS and examine the stability of the training across different random seeds.

4.2.1 Baseline Approach: Regular Learning

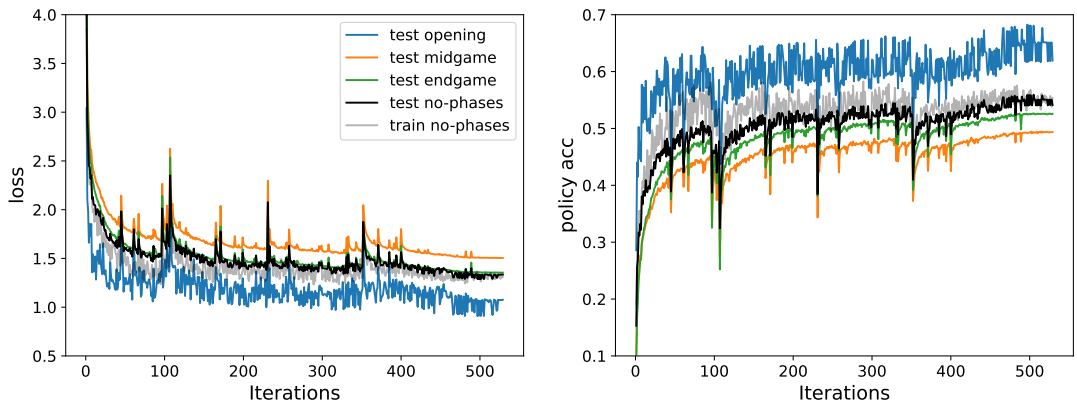
Our "Regular Learning" approach forms the baseline for our experiments. Figure 4.9a shows the development of the model's loss during training. The loss on the "no-phases" test set (black) starts at a value of around 4.0. Even though there are some loss spikes, the model always recovers from them and reaches its lowest test loss of 1.2872 near the end of the training at iteration 505. The training loss (grey) follows a similar trend and is only slightly lower throughout the training, indicating that the amount of overfitting is relatively low.

Looking at the loss curves for the test sets of the three game phases (opening (blue), midgame (orange), and endgame (green)), we see that the midgame loss is significantly higher than the losses of the other two phases. This observation can be attributed to midgames usually being more complicated than the other game phases. They include positions with more possible moves, and the position diversity is higher, making midgames harder to learn. In contrast, the opening diversity is comparatively low, with most high-level players following theoretical lines. Consequently, the loss on the opening test set is substantially lower than the other losses.

As the overall loss is primarily impacted by the policy loss, we can make the same observations when looking at the policy accuracy in Figure 4.9b. The model is considerably more accurate on opening positions, and midgame positions lead to the lowest policy accuracies. The final model checkpoint has a policy accuracy of 0.558 on the no-phases test set.

In order to allow for a better comparison between the different approaches, we add dashed lines in the metric overview figures of the following sections corresponding to the

test losses and policy accuracies of the final model checkpoint of the ”Regular Learning” approach.



(a) Loss

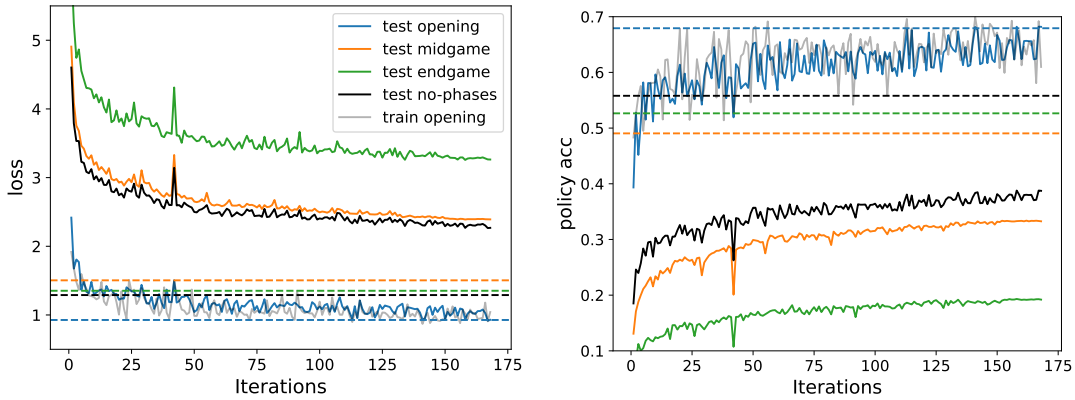
(b) Policy Accuracy

Figure 4.9: The loss (Fig. 4.9a) and policy accuracy (Fig. 4.9b) over the course of the training process for the **Regular Learning** approach. The metrics were evaluated on the opening (blue), midgame (orange), endgame (green) and no-phases (black) test set. The evaluation on the train set is shown in the background in grey. The values on the x-axis represent iterations, where one iteration is defined as doing backpropagation on one batch of training data.

4.2.2 Approach 1: Separated Learning

The ”Separated Learning” approach splits the experts strictly and only trains them on their specific subset of the dataset. In Figure 4.10a, we see the course of the different test losses for the opening expert. It clearly shows that the expert’s knowledge about the opening is far superior to its knowledge about the other phases, judging by the distance to the corresponding reference losses of the baseline model. This distance is especially considerable for the endgame test loss, indicating that opening positions provide less knowledge about endgames than midgames, which intuitively makes sense.
Nevertheless, it is also apparent that the phases are not completely without any common factors as the midgame and endgame test loss both steadily decline during training.
The opening test loss of the final model checkpoint is 0.9169, which is lower than the reference of the baseline model (0.9269) but not by much. Figure 4.10b depicts similar

trends for the development of the opening expert's policy accuracy.
Both figures show that the amount of overfitting is insignificant for the opening expert as its train and test metrics are very similar.



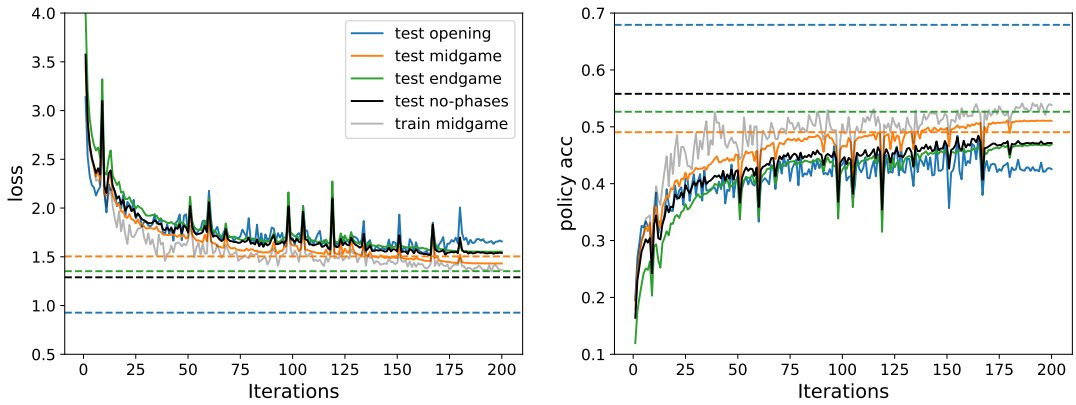
(a) Loss

(b) Policy Accuracy

Figure 4.10: The **opening expert's** loss (Fig. 4.10a) and policy accuracy (Fig. 4.10b) over the course of the training process for the **Separated Learning** approach. The metrics were evaluated on the opening (blue), midgame (orange), endgame (green) and no-phases (black) test set. The evaluation on the train set is shown in the background in grey. The values on the x-axis represent iterations, where one iteration is defined as doing backpropagation on one batch of training data. The metric values of the final model checkpoint of the Regular Learning approach are added as dashed reference lines (same colors represent the same test set).

The loss curves for the midgame expert (see Fig. 4.11a) look very similar for all test sets. However, this fact can be misleading as it does not mean that the midgame expert excels at all phases equally. The opening test loss being as high as the midgame test loss indicates that the opening knowledge is far inferior, which is supported by the distance to the dashed reference line. This discrepancy does not exist to this extent for the endgame test loss. While it does not reach the reference loss, it still comes decently close to it, indicating that midgame positions convey much information that is also useful for the endgame.
The amount of overfitting is more extensive than for the opening expert, which is clearly visible in Figure 4.11b. The train policy accuracy reaches a value of 0.538, while the final

(midgame) test policy accuracy is only 0.5106.
The test loss of the final model checkpoint is 1.4371, which is a significant improvement compared to the reference midgame test loss of the baseline model (1.5043).

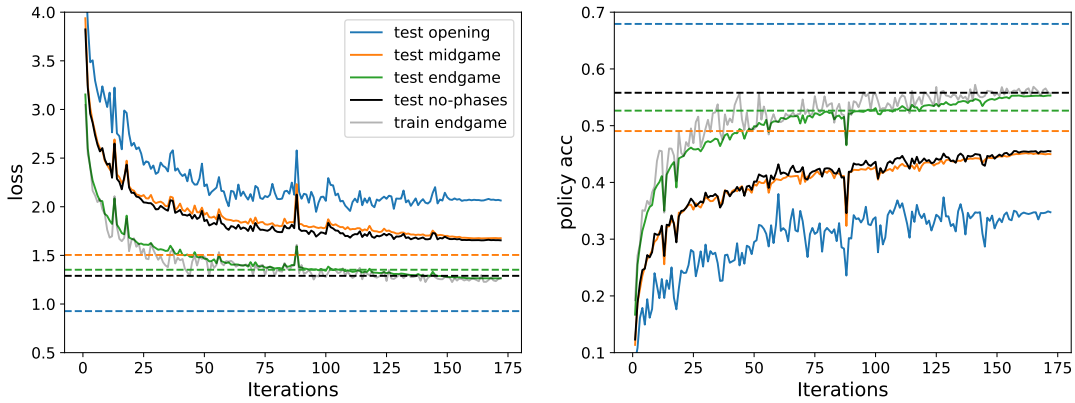


(a) Loss

(b) Policy Accuracy

Figure 4.11: The **midgame expert's** loss (Fig. 4.11a) and policy accuracy (Fig. 4.11b) over the course of the training process for the **Separated Learning** approach. The metrics were evaluated on the opening (blue), midgame (orange), endgame (green) and no-phases (black) test set. The evaluation on the train set is shown in the background in grey. The values on the x-axis represent iterations, where one iteration is defined as doing backpropagation on one batch of training data. The metric values of the final model checkpoint of the Regular Learning approach are added as dashed reference lines (same colors represent the same test set).

Figure 4.11 shows the evaluation metrics for the endgame expert. It supports the claim that midgame and endgame benefit the most from each other during the learning process because the test midgame loss and test midgame policy accuracy reach values that come close to the reference lines without the endgame expert having seen any midgame positions during training.
In contrast, the metrics for the opening test set are hugely inferior compared to the baseline model, indicating that endgames and openings have the least in common of all three phases.
The endgame test loss of the final model checkpoint is 1.2594, outclassing the reference endgame test loss of 1.3523.



(a) Loss

(b) Policy Accuracy

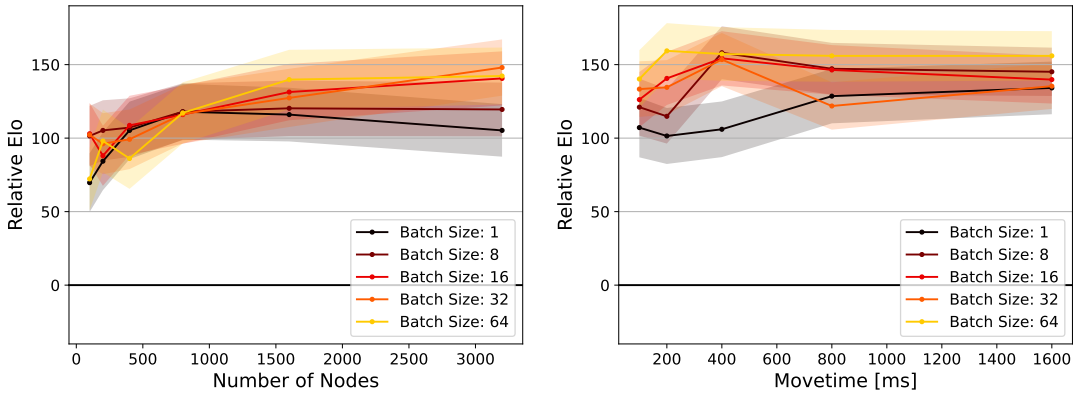
Figure 4.12: The **endgame expert's** loss (Fig. 4.12a) and policy accuracy (Fig. 4.12b) over the course of the training process for the **Separated Learning** approach. The metrics were evaluated on the opening (blue), midgame (orange), endgame (green) and no-phases (black) test set. The evaluation on the train set is shown in the background in grey. The values on the x-axis represent iterations, where one iteration is defined as doing backpropagation on one batch of training data. The metric values of the final model checkpoint of the Regular Learning approach are added as dashed reference lines (same colors represent the same test set).

By combining the three trained experts and performing an evaluation on the resulting ME model, where each sample is evaluated by its corresponding expert, we get a loss of 1.2245 on the complete no-phases dataset. The policy accuracy for this case is 0.5755. Both values are significantly better than the counterparts of the Regular learning approach (loss of 1.2897, policy acc of 0.558).

The relative Elo gains of the Separated Learning approach based on the 1000 simulated matches against the baseline method are shown in Figure 4.13. We plot the gain compared to the baseline approach with 95% confidence bounds for different batch sizes where brighter colors correspond to larger batch sizes. Even though higher batch sizes generally increase the Elo gain on average, this trend is not significant and varies for different nodes and move times. There is, however, an outlier with batch size 1, which generally leads to

the lowest gains (mean of 106.89, std of 18.29).
The most decisive benchmark match occurred for batch size 64 with a move time of 200ms. Here, the Separated Learning approach achieved a score of 714.5, winning 562 games, drawing 305 games, and losing only 133 games, which leads to an Elo gain of 159.36 ($\pm$18.87).
The improvement over the baseline approach generally increases with a higher number of nodes. The same can not be said when looking at the different move times, where the gain is relatively steady.

Overall, the Separated Learning approach is about 122.2 (std of 22.54) Elo stronger than the Regular Learning approach when averaging over all 55 experimental outcomes.



(a) Elo gain by Nodes

(b) Elo gain by move time

Figure 4.13: Relative Elo gain of the **Separated Learning** approach for different batch sizes. Node values of 100, 200, 400, 800, 1600 and 3200 in Fig. 4.13a. Move time values of 100, 200, 400, 800 and 1600 in Fig. 4.13b

4.2.3 Approach 2: Continual Learning

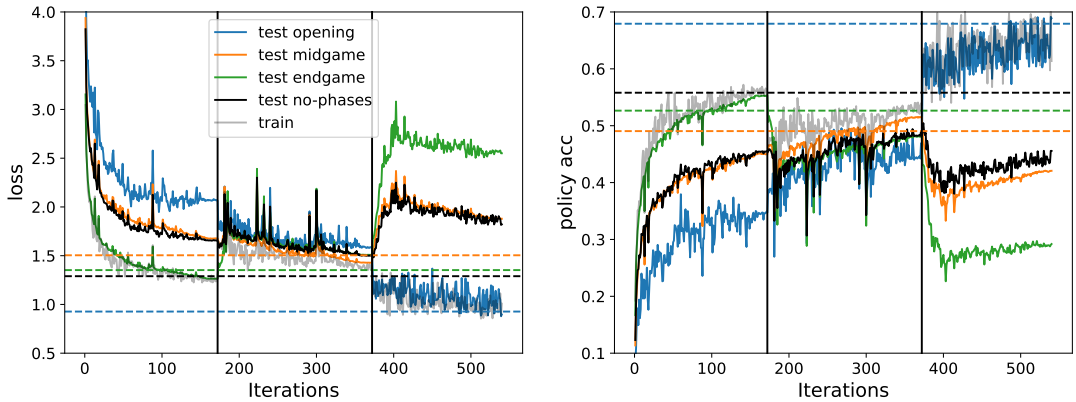
The Continual Learning approach splits the training process into three parts and always uses the resulting model parameters of the previous part as an initialization for the next part. Each part corresponds to training on a different phase dataset. We separate the following figures into three sections according to the black vertical lines to illustrate the

points at which the training set was changed to a different phase.

Figure 4.14a shows the losses for the opening expert of the Continual Learning approach. The sections correspond to training on the endgame dataset, followed by the midgame dataset and the opening dataset in the final section. As expected, we see that the losses at the start of each section are lower than those at the first iterations of the run because of the warm starts. However, for the phases not currently used for training, the loss quickly deteriorates shortly after the start of each section, indicating that most of the previously gained knowledge is quickly forgotten. Nonetheless, the training on the other phases was not entirely useless for the opening expert. This conclusion can be drawn by looking at the midgame and endgame test set losses of the final model checkpoint. With a loss of 1.8775 on the midgame test set and 2.5501 on the endgame test set the opening expert shows a substantially better understanding of these two phases than its counterpart of the Separated Learning approach (loss of 2.403 for test midgame, 3.2433 for test endgame). The final loss on the opening test set has also improved from 0.9169 (Separated Learning) to 0.87864.
Similar observations can be made for the development of the policy accuracy in Figure 4.14b. On the midgame test set, the policy accuracy improves from 0.3329 (Separated Learning) to 0.4204, and on the endgame test set, we see an increase from 0.19228 to 0.29043. The final policy accuracy on the opening test set is 0.6914 instead of 0.6803.

For the midgame expert, the catastrophic forgetting is very severe for the transition between the first two sections (see Fig. 4.14). The loss of the opening test set jumps to a higher value than during the first iterations of the run, indicating that all previously learned knowledge about the opening was immediately forgotten. Comparing the final losses on the opening and endgame test datasets to their counterparts in the Separated Learning approach, we also see that only very minor improvements were made (from 1.6064 to 1.59659 on the opening test set and from 1.5494 to 1.5293 on the endgame test set). The final midgame test loss even slightly increased compared to the midgame expert of the Separated Learning approach (from 1.4371 to 1.4437). Nevertheless, it is still significantly better than the midgame test loss of the baseline network (1.5043). The policy accuracies in Figure 4.15b support the previous findings, with no major improvements being made for the other two phases and the policy accuracy on the midgame test set slightly decreasing (from 0.5106 to 0.5088) compared to approach 1.

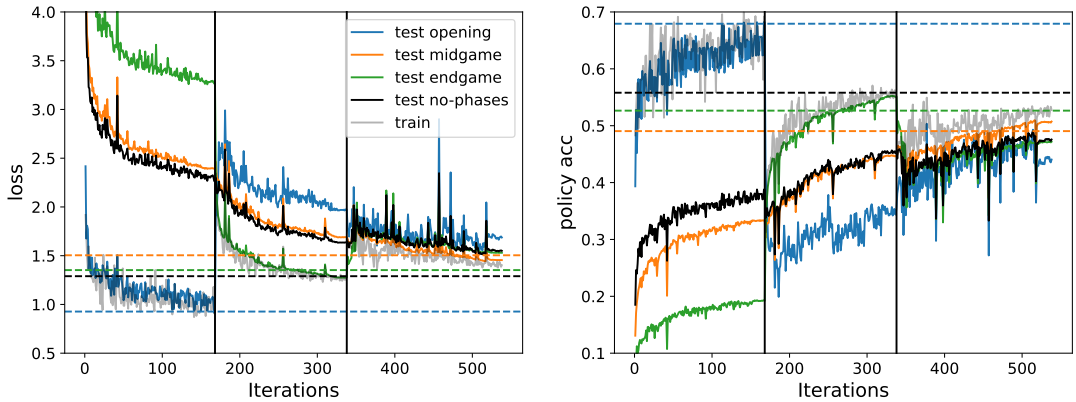
The metrics of the endgame expert show no improvement compared to their corresponding



(a) Loss

(b) Policy Accuracy

Figure 4.14: The **opening expert's** loss (Fig. 4.14a) and policy accuracy (Fig. 4.14b) over the course of the training process for the **Continual Learning** approach. The metrics were evaluated on the opening (blue), midgame (orange), endgame (green) and no-phases (black) test set. The evaluation on the train set is shown in the background. The values on the x-axis represent iterations, where one iteration is defined as doing backpropagation on one batch of training data. The vertical lines indicate the points at which the training on a new dataset started (initialized with the parameters of the last model checkpoint of the previous training stage).

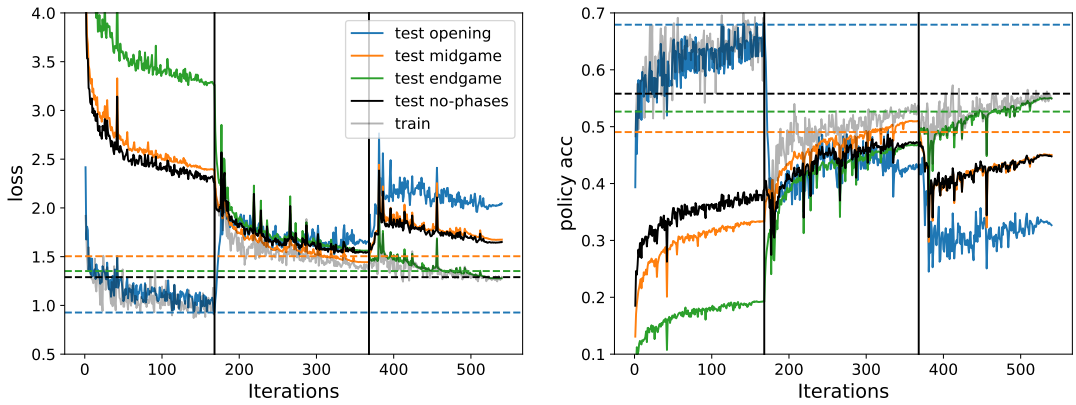


(a) Loss

(b) Policy Accuracy

Figure 4.15: The **midgame expert's** loss (Fig. 4.15a) and policy accuracy (Fig. 4.15b) over the course of the training process for the **Continual Learning** approach. The metrics were evaluated on the opening (blue), midgame (orange), endgame (green) and no-phases (black) test set. The evaluation on the train set is shown in the background. The values on the x-axis represent iterations, where one iteration is defined as doing backpropagation on one batch of training data. The vertical lines indicate the points at which the training on a new dataset started (initialized with the parameters of the last model checkpoint of the previous training stage).

values of the endgame expert in the previous section. The losses and policy accuracies (see Fig. 4.16) show the same catastrophic forgetting behavior as the ones for the midgame expert. Their final values on the endgame test set are 1.2774 (loss) and 0.55019 (policy accuracy), which is slightly worse than the values of the endgame expert of approach 1 (1.259362 and 0.5557).



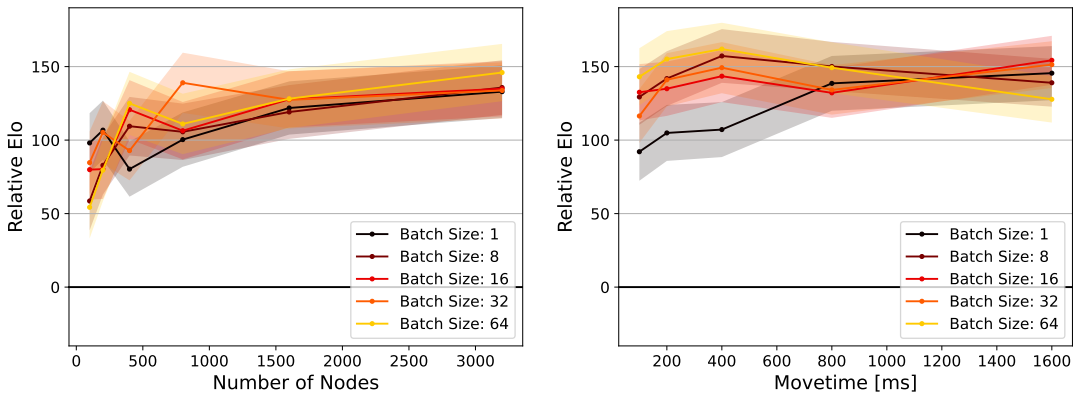
(a) Loss (b) Policy Accuracy

Figure 4.16: The **endgame expert's** loss (Fig. 4.16a) and policy accuracy (Fig. 4.16b) over the course of the training process for the **Continual Learning** approach. The metrics were evaluated on the opening (blue), midgame (orange), endgame (green) and no-phases (black) test set. The evaluation on the train set is shown in the background. The values on the x-axis represent iterations, where one iteration is defined as doing backpropagation on one batch of training data. The vertical lines indicate the points at which the training on a new dataset started (initialized with the parameters of the last model checkpoint of the previous training stage).

The ME model using all three Continual Learning experts achieves a loss of 1.2218 on the complete no-phases dataset. The policy accuracy for this case is 0.5763. Both values are significantly better than the counterparts of the Regular learning approach (loss of 1.2897, policy acc of 0.558) but similar to the ones of approach 1 (1.2245 and 0.5755).

Figure 4.17 shows the resulting Elo gains of the Continual Learning approach compared to the baseline method. Averaged over all node and move time values, larger batch sizes lead to higher gains in relative Elo. The differences are very subtle, though, and only

batch size 1 stands out with a mean Elo gain of 111.7 and a standard deviation of 20.5. The overall best Elo gain of 161.92 ($\pm$17.89) was achieved with batch size 64 and move time 400. The match ended with 539 wins, 357 draws, and 104 losses for the Continual Learning approach (score of 717.5).
Averaging over all batch sizes, the improvement over the baseline approach monotonically increases with a higher number of nodes. Although not as severe, a similar trend is observable for increasing move times.
Overall, the Continual Learning approach is about 121.11 (std of 25.68) Elo stronger than the Regular Learning approach when averaging over all 55 experimental outcomes.



(a) Elo gain by Nodes

(b) Elo gain by move time

Figure 4.17: Relative Elo gain of the **Continual Learning** approach for different batch sizes. Node values of 100, 200, 400, 800, 1600 and 3200 in Fig. 4.17a. Move time values of 100, 200, 400, 800 and 1600 in Fig. 4.17b

4.2.4 Approach 3: Weighted Learning

The "Weighted Learning" approach trains each expert on all available positions and uses different sample weights depending on the game phase to which the samples belong. It has to be stated that, contrary to the train set, the test sets use no weighting, so the evaluation metrics are comparable to the ones of the other approaches.

Figure 4.18 shows the losses and policy accuracies for the opening expert and different a values throughout the training. For $a = 4$, we see that both the loss and the policy

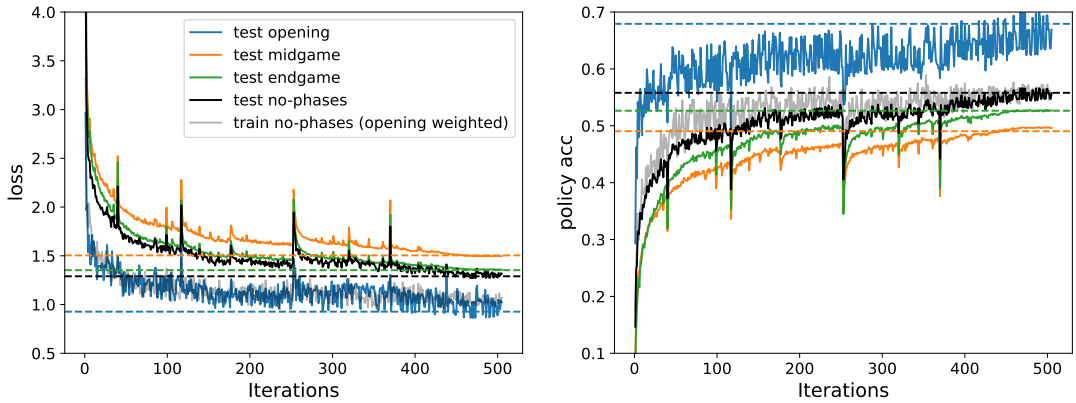
accuracy curves look almost identical to the ones of the Regular Learning approach. The approach reaches nearly the same final losses as the baseline model, indicating that the sample weighting had almost no impact. Setting a to 10 increases the specialization because the samples of the other phases are now only 10% as impactful as the samples of the expert's main phase. We can directly see the impact of this change by looking at the final metric values of the other phases. They significantly lag behind compared to the dashed baseline metric values (e.g., final endgame test loss of 1.5389 instead of 1.3523 for the baseline network). In return, the final opening test loss decreased by a decent amount compared to the baseline model (0.8887 instead of 0.9268).

The test set metrics for the midgame expert (Fig. 4.19) depict the same tendency as the ones for the opening expert. For $a = 4$, the specialization is negligible (final midgame test set loss of 1.5055 instead of 1.5043 for the baseline model). In return, this expert performs better on the opening and endgame test set than its counterpart with $a = 10$. The midgame expert with $a = 10$ specializes enough in its own phase to outdo the baseline model on the test set of that phase (final midgame test loss of 1.4834 compared to 1.5043 for the Regular Learning approach). In return, the metrics for the other phases' test sets worsened. Note that the training loss is higher than the corresponding test loss, which can be explained by the fact that the training loss is weighted, whereas the test loss is not, as mentioned before.

The metric development for the endgame expert is shown in Figure 4.20. Surprisingly, the expert trained with $a = 4$ achieves a better final loss on the endgame test set (1.2864) than the one trained with $a = 10$ (loss of 1.3036). Similarly remarkable is that the experts for both a values do not perform worse on the midgame test set than the baseline model even though midgame samples had lower weight during their training.

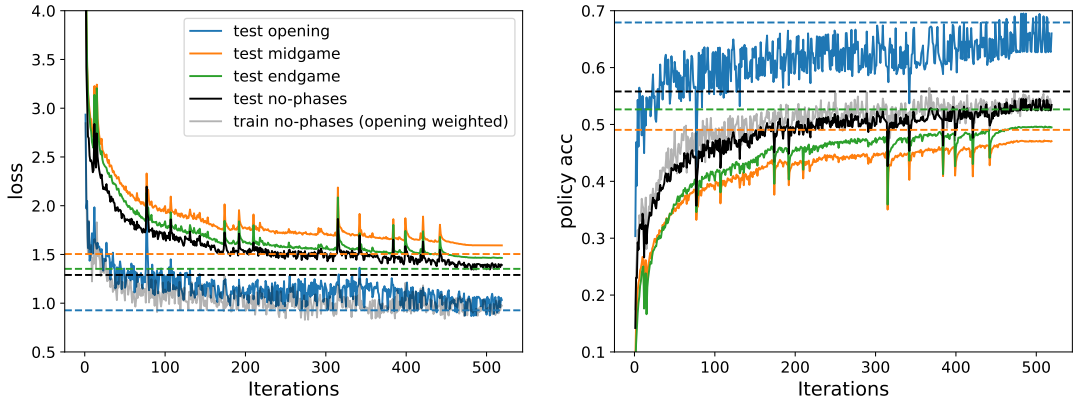
The ME model using all three Weighted Learning experts achieves a loss of 1.2432 (for $a = 4$) and 1.248 (for $a = 10$) on the complete no-phases dataset. The policy accuracies are 0.5713 ($a = 4$) and 0.5697 ($a = 10$). The values are better than the counterparts of the Regular learning approach (loss of 1.2897, policy acc of 0.558) but worse than the ones of approach 1 (1.2245 and 0.5755) and approach 2 (1.2218 and 0.5763).

Figure 4.21 shows the relative Elo gains of the Weighted Learning approach for both a values. Overall, no specific batch size performs considerably better or worse than the others, apart from batch size 1 for $a = 10$, which performs worse when limiting the tree search by move time instead of by nodes. Increasing the number of nodes or increasing



(a) Loss ($a = 4$)

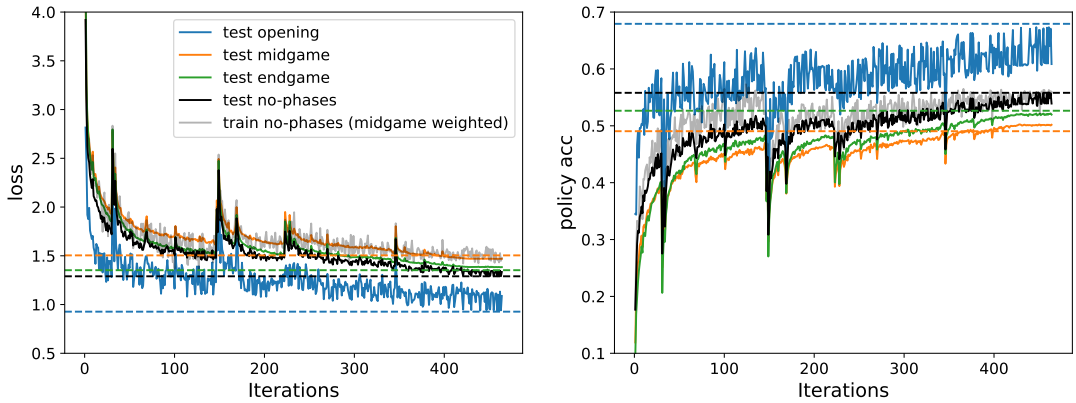
(b) Policy Accuracy ($a = 4$)



(c) Loss ($a = 10$)

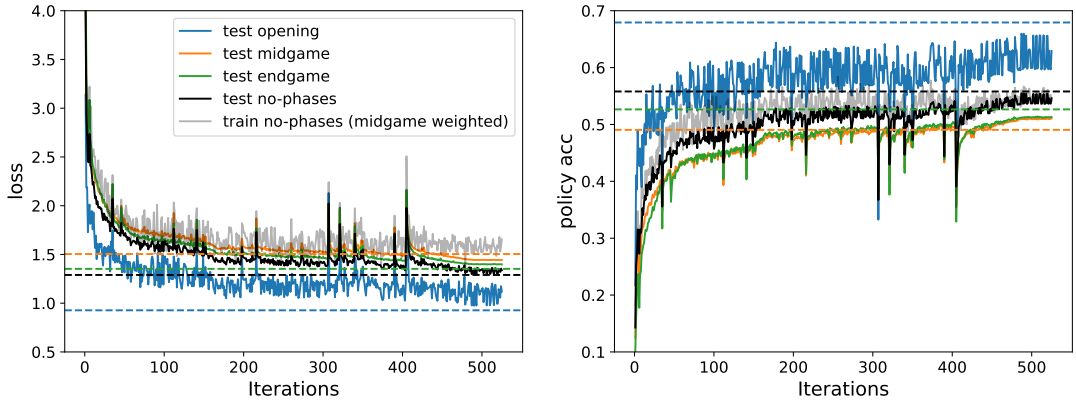
(d) Policy Accuracy ($a = 10$)

Figure 4.18: The **opening expert's** loss (Fig. 4.18a and Fig. 4.18c) and policy accuracy (Fig. 4.18b and Fig. 4.18d) over the course of the training process for the **Weighted Learning** approach with different a values. The metrics were evaluated on the unweighted opening (blue), midgame (orange), endgame (green) and no-phases (black) test set. The evaluation on the train set (weighted) is shown in the background. The values on the x-axis represent iterations, where one iteration is defined as doing backpropagation on one batch of training data. The metric values of the final model checkpoint of the Regular Learning approach are added as dashed reference lines (same colors represent the same test set).



(a) Loss ($a = 4$)

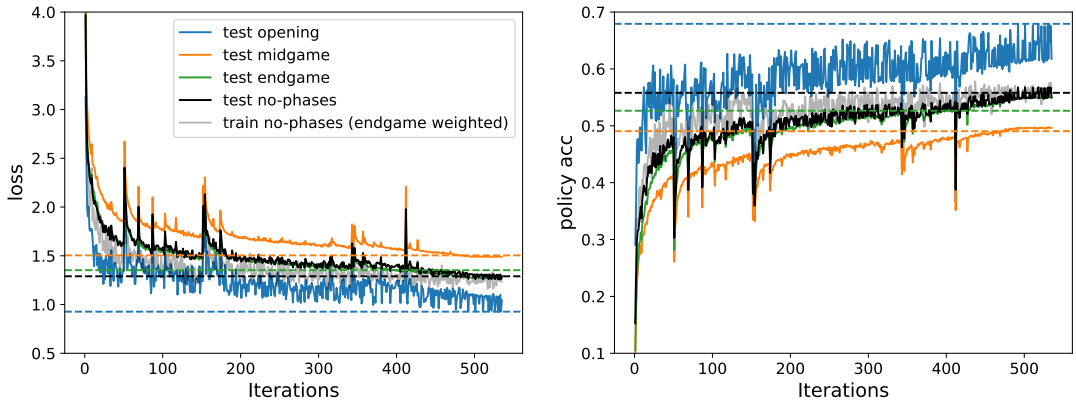
(b) Policy Accuracy ($a = 4$)



(c) Loss ($a = 10$)

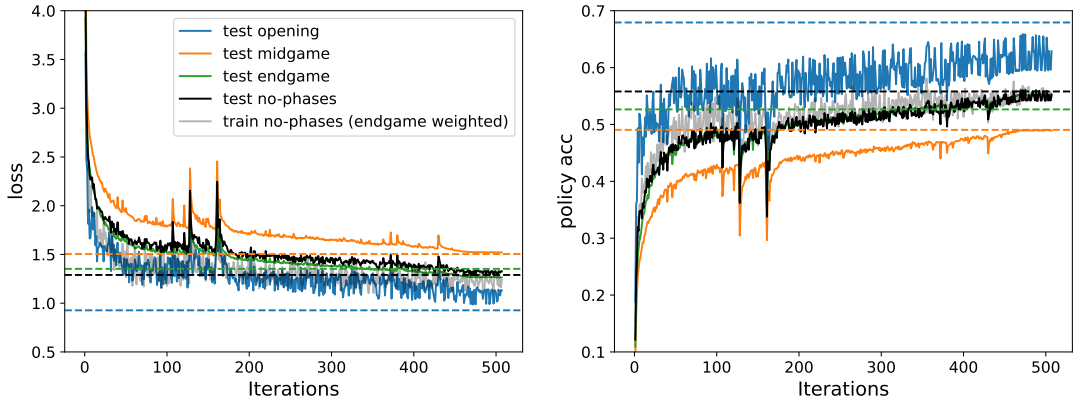
(d) Policy Accuracy ($a = 10$)

Figure 4.19: The **midgame expert's** loss (Fig. 4.19a and Fig. 4.19c) and policy accuracy (Fig. 4.19b and Fig. 4.19d) over the course of the training process for the **Weighted Learning** approach with different a values. The metrics were evaluated on the unweighted opening (blue), midgame (orange), endgame (green) and no-phases (black) test set. The evaluation on the train set (weighted) is shown in the background. The values on the x-axis represent iterations, where one iteration is defined as doing backpropagation on one batch of training data. The metric values of the final model checkpoint of the Regular Learning approach are added as dashed reference lines (same colors represent the same test set).



(a) Loss ($a = 4$)

(b) Policy Accuracy ($a = 4$)



(c) Loss ($a = 10$)

(d) Policy Accuracy ($a = 10$)

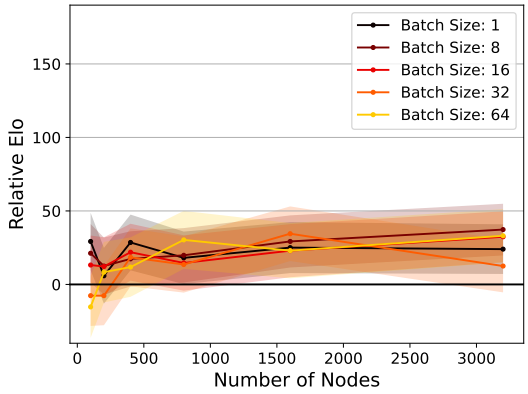
Figure 4.20: The **endgame expert's** loss (Fig. 4.20a and Fig. 4.20c) and policy accuracy (Fig. 4.20b and 4.20d) over the course of the training process for the **Weighted Learning** approach with different a values. The metrics were evaluated on the unweighted opening (blue), midgame (orange), endgame (green) and no-phases (black) test set. The evaluation on the train set (weighted) is shown in the background. The values on the x-axis represent iterations, where one iteration is defined as doing backpropagation on one batch of training data. The metric values of the final model checkpoint of the Regular Learning approach are added as dashed reference lines (same colors represent the same test set).

the move time raises the resulting relative Elo gain (for both a values) when averaging over the results of all batch sizes.
For $a = 4$, the overall best Elo gain of 42.95 ($\pm$16.08) was achieved with batch size 32 and move time 800. The match ended with 340 wins, 443 draws, and 217 losses for the Weighted Learning approach (score of 561.5).
The agent using the models trained with $a = 10$ achieved its most decisive result for batch size 32 and move time 400. The match ended in 434 wins, 403 draws, and 163 draws (score of 635.5), corresponding to an Elo gain of 96.57 ($\pm$16.8).

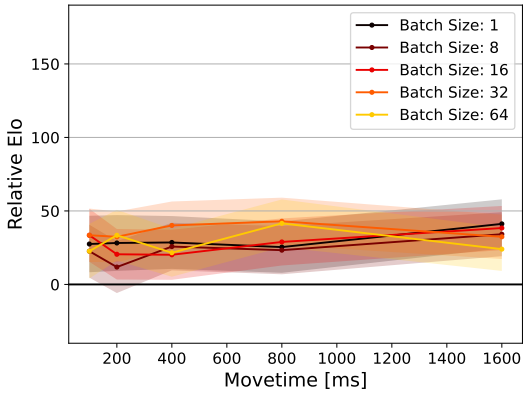
Overall, the Weighted Learning approach is about 23.18 (std of 11.95, $a = 4$), respectively 55.84 (std of 19.57, $a = 10$), Elo stronger than the Regular Learning approach when averaging over all 55 experimental outcomes.

4.2.5 Expert Impact

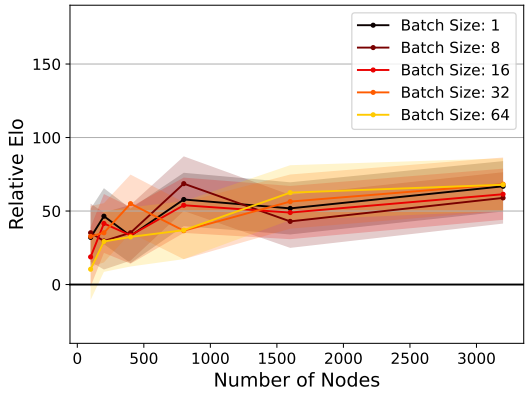
In order to estimate which phase expert has the most impact on the Elo gain, we perform 1000 game matches using ME models that only use one specific phase expert and use the baseline model for the other two phases (see Section 3.4.5 for a more detailed explanation). The single expert models are taken from the Separated Learning approach, and the batch size is set to 64 for all experiments.
Figure 4.22 shows the resulting relative Elo gains for different node and move time values. Using only the opening expert (blue) does not lead to any improvement over the baseline approach (average Elo gain of -3.82 with an std of 5.67724).
Looking at the performance gains for different node values in 4.22a, we see that only using the single midgame (orange) or the single endgame expert (green) already leads to notable and mostly similar Elo improvements. The skill gap to the full baseline approach increases with more nodes. For the endgame expert, the improvement is even more substantial when limiting the MCTS by move time instead of by nodes. The same can not be said for the midgame expert.
On average, the single midgame expert achieves a relative Elo gain of 65.71 with an std of 8.31. For the single endgame expert, the average improvement is 73.34 Elo (std of 22.95).



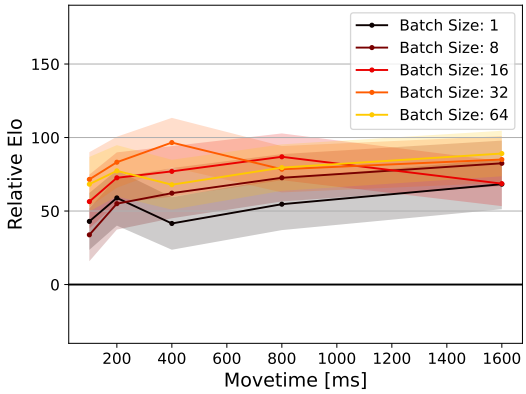
(a) Elo gain by Nodes ($a = 4$)



(b) Elo gain by move time ($a = 4$)

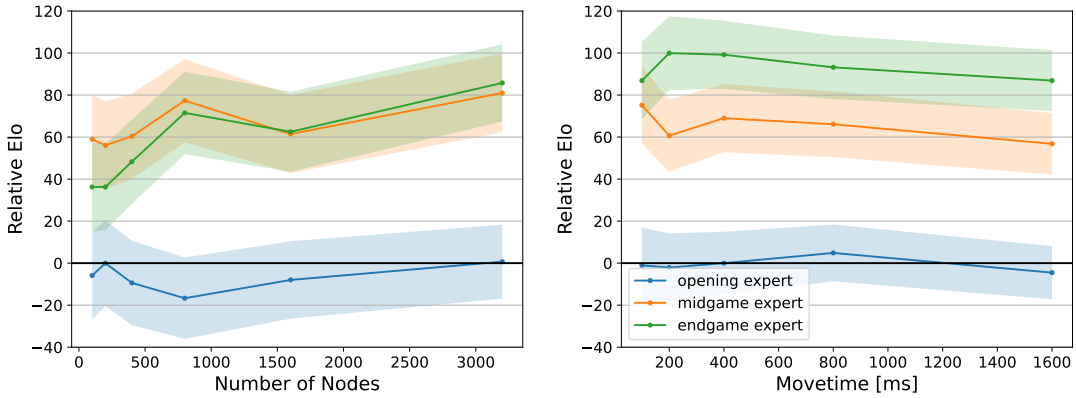


(c) Elo gain by Nodes ($a = 10$)



(d) Elo gain by move time ($a = 10$)

Figure 4.21: Relative Elo gain of the **Weighted Learning** approach for different batch sizes and a values of 4 and 10. Node values of 100, 200, 400, 800, 1600 and 3200 in Fig. 4.21a and Fig. 4.21c. Move time values of 100, 200, 400, 800 and 1600 in Fig. 4.21b and 4.21d



(a) Elo gain by Nodes

(b) Elo gain by move time

Figure 4.22: Relative Elo gain of using only one specific phase expert in the MCTS and using the baseline network for all remaining predictions. The used experts are taken from the Separated Learning approach for **chess** and the batch size was set to 64 for all matches. Node values of 100, 200, 400, 800, 1600 and 3200 in Fig. 4.22a. Move time values of 100, 200, 400, 800 and 1600 in Fig. 4.22b

metric	test set	seed					
		1	2	3	4	5	avg
loss	opening	0.9269	0.8905	0.9091	0.9080	0.8859	**0.9041**
	midgame	1.5043	1.4643	1.5074	1.4857	1.4656	**1.4855**
	endgame	1.3523	1.3234	1.3556	1.3432	1.3101	**1.3369**
	no-phases	1.2897	1.2547	1.2873	1.2731	1.2490	**1.2707**
policy acc	opening	0.6793	0.6853	0.6806	0.6812	0.6855	**0.6824**
	midgame	0.4905	0.5013	0.4930	0.4982	0.5035	**0.4973**
	endgame	0.5265	0.5361	0.5267	0.5310	0.5376	**0.5316**
	no-phases	0.5580	0.5670	0.5593	0.5629	0.5683	**0.5631**

Table 4.3: Evaluation metric values on all tests for different random seeds of the Regular Learning approach. Seed 1 was used as the baseline model in all chess experiments in this thesis.

4.2.6 Baseline Stability

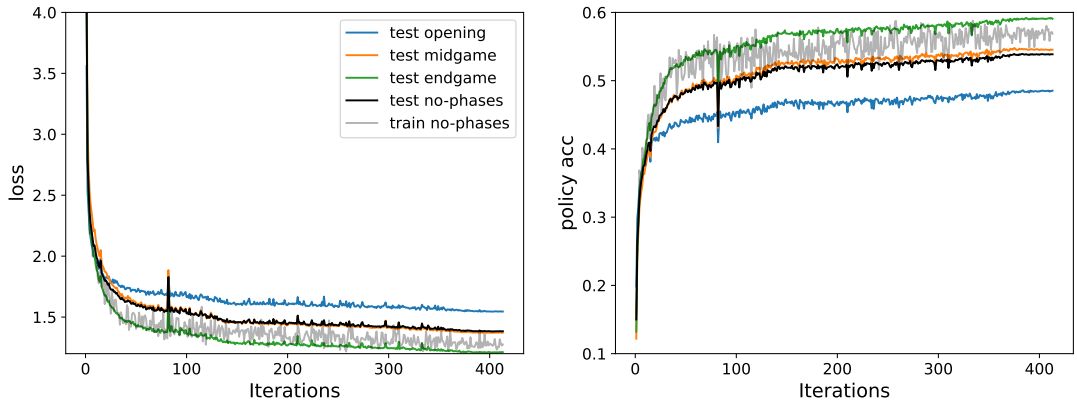
In order to test the stability of the baseline model performance, we trained it five times in an identical fashion, only changing the random seed. The resulting values of losses and policy accuracies on all test sets are shown in Table 4.3. The model we used for all experiments in the previous sections was trained using seed 1 (first column). The average loss over all five random seeds on the no-phases dataset is 1.2707 ($\pm$0.0185), which is slightly lower than that of our baseline model but still considerably higher than the losses of the ME models of our evaluated approaches.
The average policy accuracy for the five random seeds does not change significantly.

4.3 Chess960 Results

This section serves as an exemplary showcase of the results of our Chess960 experiments. We show the evaluation metrics during the training and give an overview of the final losses of each approach for all test sets. The Elo gains are derived based on the results of 1000 game matches against the baseline approach. We use three different batch sizes and eleven values to control the search length, leading to 33 matches in total. Additionally, we analyze each expert's impact during the MCTS for the Separated Learning approach.

4.3.1 Training Performance

Figure 4.23 shows the evaluation metrics for our baseline approach during training. Contrary to regular chess, for Chess960, the opening phase leads to the biggest losses and lowest policy accuracies. This change can be attributed to the fact that Chess960 has 960 different starting positions, which is why nearly no opening theory exists. Consequently, the variety of positions and moves in the opening is substantially higher than for regular chess.



(a) Loss (b) Policy Accuracy

Figure 4.23: The loss (Fig. 4.23a) and policy accuracy (Fig. 4.23b) over the course of the training process for the **Regular Learning** approach for **Chess960**. The metrics were evaluated on the opening (blue), midgame (orange), endgame (green) and no-phases (black) test set. The evaluation on the train set is shown in the background in grey. The values on the x-axis represent iterations, where one iteration is defined as doing backpropagation on one batch of training data.

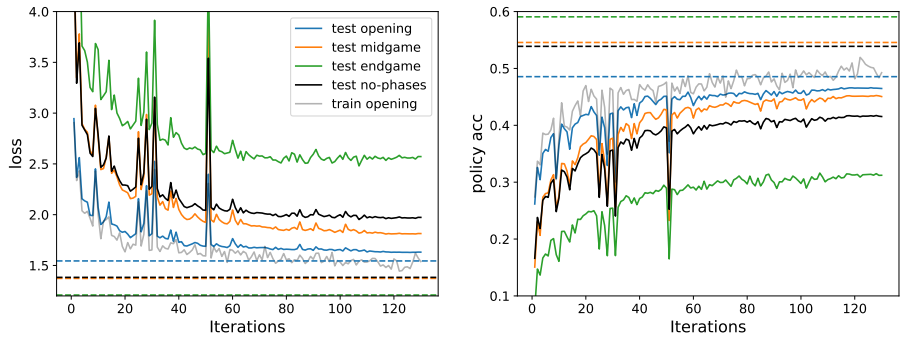
As an exemplary insight into the metrics of our ME approaches for Chess960, Figure 4.24 shows the losses and policy accuracies of all phase experts for the Separated Learning approach. We immediately see that the experts perform considerably worse than the baseline model (final metric values added as dashed lines), even on the phase they were trained on. Even though the specialization of each expert is easily recognizable by comparing the test losses for the different phases, the experts overall have higher losses than the Regular Learning approach. Note that each expert was only trained on

approach	expert	test set			
		opening	midgame	endgame	no-phases
Regular Learning	-	**1.544521**	**1.37247**	**1.208817**	**1.382707**
Separated Learning	opening	1.639244	1.818238	2.547496	1.970087
	midgame	1.694583	1.460292	1.536295	1.561291
	endgame	2.367885	1.720483	1.285305	1.808829
	mixture	**1.63924**	**1.46029**	**1.28530**	**1.46888**
Continual Learning	opening	1.592808	1.617218	2.104227	1.750602
	midgame	1.686303	1.423646	1.438175	1.517366
	endgame	2.155602	1.685057	1.293216	1.729787
	mixture	**1.59281**	**1.42365**	**1.29322**	**1.44182**
Weighted Learning ($a = 10$)	opening	1.566103	1.447515	1.298218	1.444093
	midgame	1.581337	1.393878	1.267132	1.420311
	endgame	1.607898	1.444763	1.223375	1.435811
	mixture	**1.56610**	**1.39388**	**1.22338**	**1.40154**

Table 4.4: Overview of **all losses** of each expert's final model checkpoint for all approaches for **Chess960**. The losses are shown for the opening, midgame, endgame, and no-phases test sets. In addition to the specific expert's losses, we added a row for the loss of the resulting mixture model that uses all three experts. The mixture models choose which expert to use based on the phase of the current position in the test set (i.e., the opening expert evaluates all opening positions, the midgame expert evaluates all midgame positions, etc.). We color-coded each column individually from high loss (dark red) to low loss (dark green).

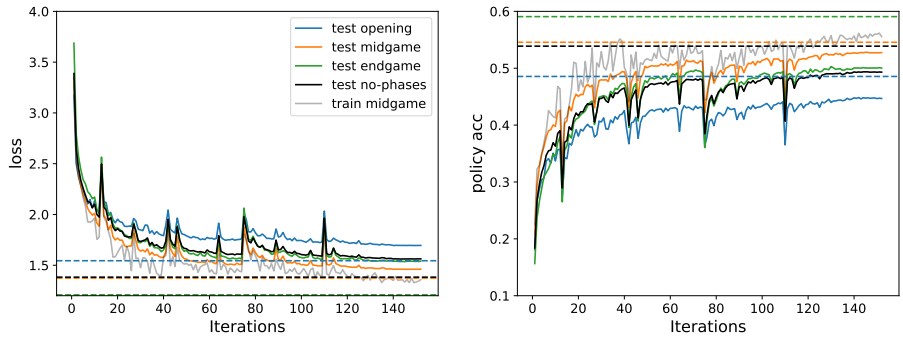
about a third of all available Chess960 positions, which might not have been enough to learn general chess concepts sufficiently, especially given that the Chess960 datasets are substantially smaller and of lower quality than the chess datasets.

Table 4.4 depicts the final losses of all approaches on all test sets for Chess960. The "mixture" expert corresponds to the ME model that uses all three experts. We see that all approaches perform significantly worse than the baseline, which has a final loss of 1.382 compared to 1.4688 (Separated Learning), 1.44182 (Continual Learning), and 1.40154 (Weighted Learning with $a = 10$). The Weighted Learning approach comes closest in performance but also uses the complete dataset (in weighted form) for all its experts, increasing the training time.



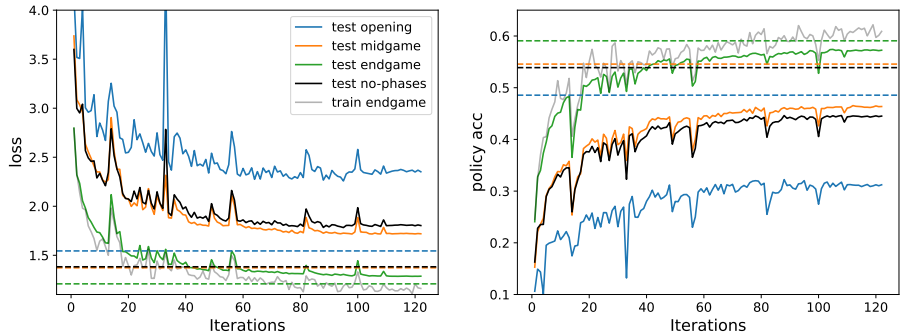
(a) Opening Expert Loss

(b) Opening Expert Policy Acc



(c) Midgame Expert Loss

(d) Midgame Expert Policy Acc



(e) Endgame Expert Loss

(f) Endgame Expert Policy Acc

Figure 4.24: The losses and policy accuracies of all phase experts over the course of the training process for the **Separated Learning** approach for **Chess960**. The values on the x-axis represent iterations, where one iteration is defined as doing backpropagation on one batch of training data. The metric values of the final model checkpoint of the Regular Learning approach are added as dashed reference lines (same colors represent the same test set).

approach	batch size			
	1	16	64	avg
Separated Learning	-175.67	-184.05	-190.51	**-183.41**
Continual Learning	-130.25	-132.84	-134.96	**-132.68**
Weighted Learning ($a = 10$)	-152.65	-151.76	-158.51	**-154.31**

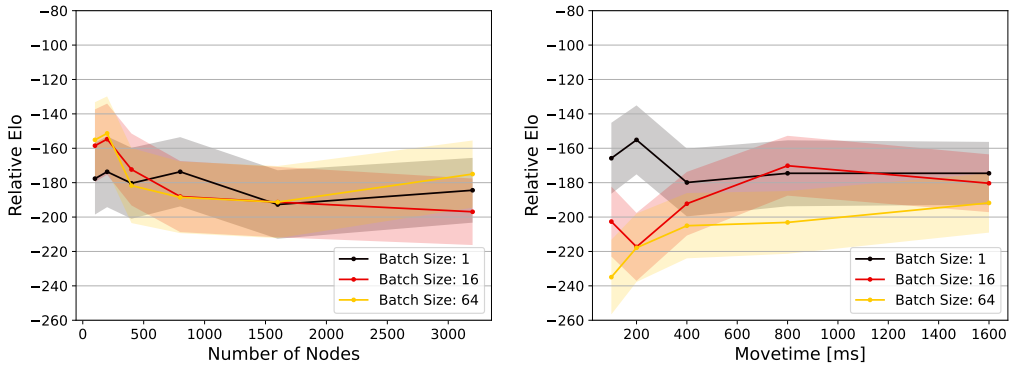
Table 4.5: Overview about the **relative Elo gains of the different approaches** (compared to the baseline approach) for different batch sizes for **Chess960**. The values resemble the average Elo gain of all node and move time configurations for the particular batch size. The cells are color-coded from best (dark green) to worst (dark red). The last column displays the overall average Elo gain for all 33 performed experiments of each approach.

4.3.2 Elo Gain

The relative Elo gains for Chess960 can be seen in Figure 4.25. All approaches perform significantly worse than the baseline, as expected, given the higher losses. Table 4.5 summarizes the results by averaging over all node and move time values. Bigger batch sizes lead to more decisive results.

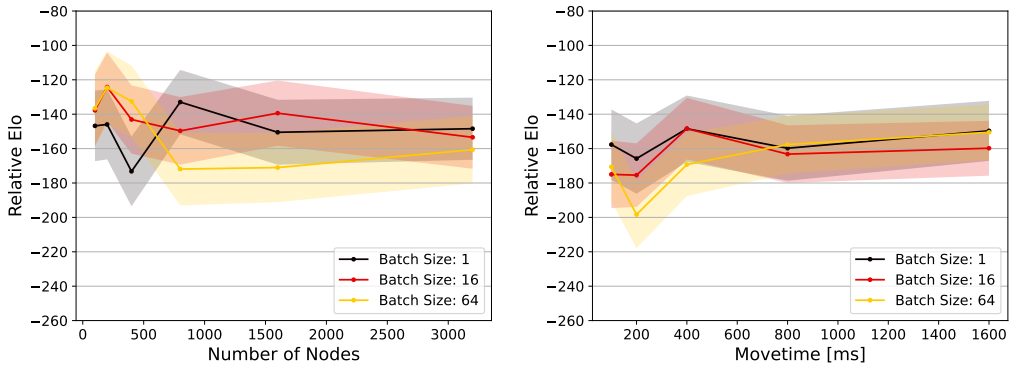
4.3.3 Expert Impact

Figure 4.26 shows the impact of the three experts for the Separated Learning approach with batch size 64. Similar to regular chess, the opening expert has the smallest influence on overall Elo, indicating that the inaccuracies and mistakes of the ME approaches happen in the later game phases of the matches. Especially the midgame expert, leads to immense performance drops. Changing the number of nodes or the move time does not influence the results significantly.



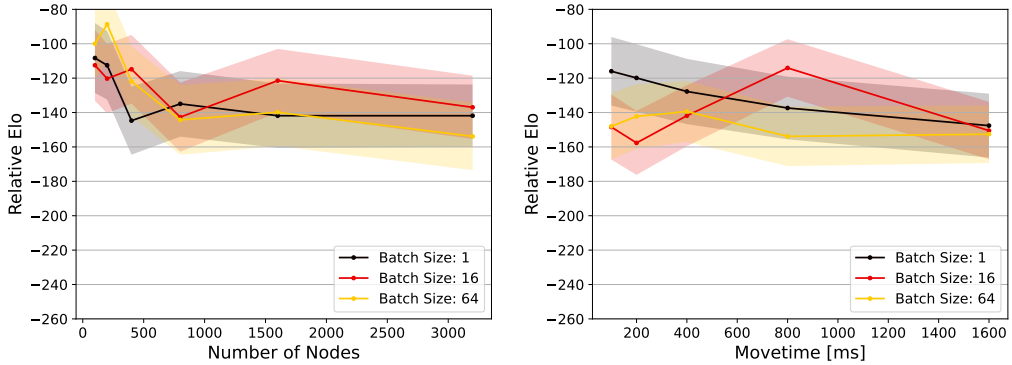
(a) Nodes (Separated Learning)

(b) Move time (Separated Learning)



(c) Nodes (Continual Learning)

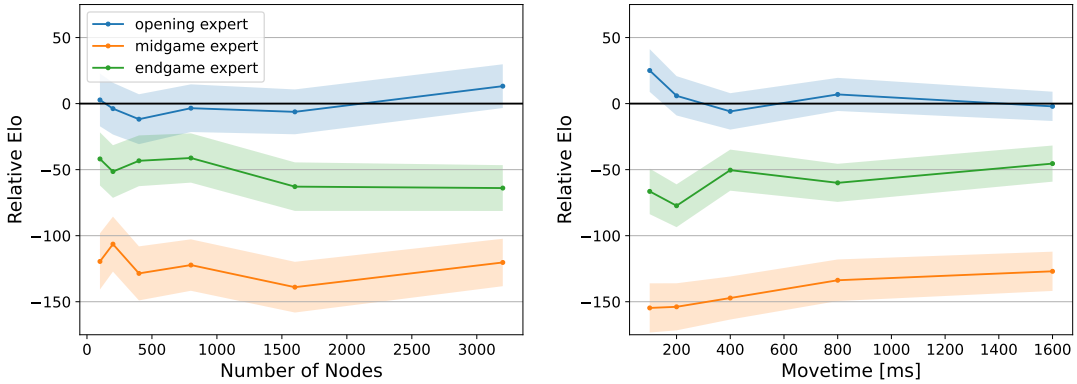
(d) Move time (Continual Learning)



(e) Nodes (Weighted Learning)

(f) Move time (Weighted Learning)

Figure 4.25: Relative Elo gain of all approaches for different batch sizes for Chess960. Node values of 100, 200, 400, 800, 1600 and 3200. Move time values of 100, 200, 400, 800 and 1600.



(a) Elo gain by Nodes

(b) Elo gain by move time

Figure 4.26: Relative Elo gain of using only one specific phase expert in the MCTS and using the baseline network for all remaining predictions. The used experts are taken from the Separated Learning approach for **Chess960** and the batch size was set to 64 for all matches. Node values of 100, 200, 400, 800, 1600 and 3200 in Fig. 4.26a. Move time values of 100, 200, 400, 800 and 1600 in Fig. 4.26b

5 Discussion

In this chapter, we evaluate and interpret the most relevant results of the previous chapter, assess their reliability, and point out their consequences. In particular, we discuss the validity of our chosen game phase definition and point out possible improvements. Furthermore, we compare our selected approaches and point out their advantages and disadvantages with respect to their performance. We also examine potential reasons why the approaches failed to deliver similar results for Chess960.

5.1 Game Phase Definitions

Section 4.1 clarifies that finding an appropriate definition for chess game phase transitions is not trivial. We see that the mixedness score has a very high variance and, on its own, is not suited for games with many piece trades in the early game because those never get to a position where the board is sufficiently mixed for the mixedness criterion to reach a value of 150. Similar things can be said about the backrank sparseness criterion, which also will not detect the transition to the midgame as early as would be needed for variations like the Berlin endgame. Nevertheless, it is essential to include these criteria as they, in return, are relevant to capture midgame transitions for slow maneuvering games with closed positions and nearly no piece trades. In these strategic cases, the piece count criterion fails, showing that it is also unsuited as a sole indicator for game phases.
In combination, however, the chosen criteria form a good baseline for phase separation, as shown through Figures 4.7 and 4.8b. Not only do they lead to a relatively balanced position count distribution for the three phases, but they also offer a nearly uniform game outcome distribution.
In order to balance the number of positions for each phase even more, it would be possible to adjust the criteria in a way that leads to later transitions to the midgame so that a portion of the positions are shifted from the midgame dataset to the opening dataset.

This change could be executed by investigating ways of combining the current criteria in conjunctive or other more sophisticated forms instead of just using a strict disjunction. An example would be requiring both a particular piece count and a minimum mixedness score simultaneously.
An adjustment like this would also be in line with our findings for the losses of the different phases. The test set losses of the Regular Learning approach in Figure 4.9a clearly show that midgames form by far the most challenging phase of the game. Their great variety and high piece counts lead to difficult policy predictions and, consequently, to a substantially higher loss than the other phases. Limiting the number of positions that count toward the midgame would mitigate this issue and delegate more responsibility to the opening expert.
A change like this could also counteract the fact that the opening expert currently provides no relevant improvement compared to the baseline network, as shown when analyzing the expert importance in Section 4.2.5. A criterion that is currently not included but would help in this regard is whether the players have already castled their king to safety. This criterion is also often intuitively used to judge chess positions but should probably not be added in isolation but instead be tied to fulfilling other criteria simultaneously in a weaker form (e.g., both players having castled + mixedness score of more than 130). The same can be said about a move counter criterion, which also is not reliable on its own but might be worth considering when tied to additional constraints.
The drawback of such adjustments to the definition is that they would require much manual work to find a fitting configuration without guaranteeing improvement for the resulting experts.
Another thing worth pointing out is that the number of jumps back to previous phases during a game would most likely increase the more sophisticated the transition criteria become. Figure 4.6 shows that more than 20% of our games had at least one such transition, which is already higher than we expected but should not lead to major problems for the proposed approaches.
The fact that the opening expert is not relevant in terms of Elo gain can also be attributed to the fact that we used an opening suite for our experiments to ensure game variety. This inclusion leads to a shorter opening phase and, therefore, fewer decisions for the opening experts as several moves have already been made. It also shifts the distribution of opening positions compared to the training data. In the training set, the positions are provided by games of high-level players, which leads to little variety in the opening as these players usually follow known theoretical lines. In contrast, the chosen opening suite consists of automatically generated, randomized positions that are only restricted to not favor the black or the white side too heavily. These positions, therefore, form out-of-distribution samples for the opening expert. It can even be argued that these positions are more similar

to typical midgame positions than to regular opening positions, as they often include early-game tactics and major structural imbalances. Consequently, it might be beneficial to have some midgame knowledge in these positions, which the opening expert does not have. Using a different opening suite with theoretical lines similar to those played by humans could, therefore, increase the impact of the opening expert.
In addition to that, it is generally harder for experts to excel in openings than in other phases, as openings are more trivial to learn and also often include many possible moves that are interchangeably playable, so different policy decisions are less impactful than in other phases. One could argue that decisions in the opening carry through to the other phases and, therefore, impact the whole game rather than only their phase. While this statement is generally true, Figure 4.22 clearly shows that this does not lead to the earlier phases being more impactful, with the endgame expert providing the highest relative Elo gain on average.

5.2 Approach Validity

Table 5.1 contains an overview of the test set losses for all approaches and all experts used for our experiments. The values are color-coded from high loss (red) to low loss (dark green). The resulting ME model of combining all three phase experts is denoted as ”mixture”. The ME model detects the current game phase and only queries the correct expert when making a prediction. The resulting loss of each approach on the complete no-phases test set is shown in the last column (underlined). As mentioned before, all approaches achieve significantly better combined losses than the baseline model of the Regular Learning approach. Most improvements in terms of loss were made on the midgame and endgame test sets, which is in line with our results about expert impact 3.4.5. Our unweighted approaches 1 and 2 achieve lower losses than those with weighted datasets. The difference is not as easily noticeable for the final no-phases test set policy accuracies, with all approaches hovering around 0.57 accuracy as shown in Table 5.2.

The Separated Learning and Continual Learning approaches significantly outperforming the Weighted Learning approach becomes even more apparent when looking at the relative Elo gain overview in Table 5.3. The values are averaged over all node and move time values and shown for all used batch sizes. No chosen batch size performs clearly better than the others, but a batch size of 1 leads to significantly worse gains in terms of Elo for approaches 1 and 2.

approach	expert	test set			
		opening	midgame	endgame	no-phases
Regular Learning	-	**0.9269**	**1.5043**	**1.3523**	**1.2897**
Separated Learning	opening	0.9169	2.4030	3.2433	2.2587
	midgame	1.6064	1.4371	1.5494	1.5260
	endgame	2.0097	1.6605	1.2594	1.6293
	mixture	**0.9169**	**1.4371**	**1.2594**	**1.2245**
Continual Learning	opening	0.8786	1.8775	2.5501	1.8153
	midgame	1.5966	1.4437	1.5293	1.5203
	endgame	2.0281	1.6748	1.2774	1.6458
	mixture	**0.8786**	**1.4437**	**1.2774**	**1.2218**
Weighted Learning ($a = 4$)	opening	0.8636	1.5125	1.3780	1.2820
	midgame	0.9661	1.5055	1.4207	1.3248
	endgame	0.9339	1.5041	1.2864	1.2695
	mixture	**0.8636**	**1.5055**	**1.2864**	**1.2433**
Weighted Learning ($a = 10$)	opening	0.8887	1.6600	1.5389	1.3995
	midgame	0.9955	1.4834	1.4470	1.3346
	endgame	1.0334	1.5547	1.3036	1.3241
	mixture	**0.8886**	**1.4834**	**1.3036**	**1.2483**

Table 5.1: Overview of **all losses** of each expert's final model checkpoint for all approaches. The losses are shown for the opening, midgame, endgame, and no-phases test sets. In addition to the specific expert's losses, we added a row for the loss of the resulting mixture model that uses all three experts. The mixture models choose which expert to use based on the phase of the current position in the test set (i.e., the opening expert evaluates all opening positions, the midgame expert evaluates all midgame positions, etc.). We color-coded each column individually from high loss (dark red) to low loss (dark green).

approach	expert	test set			
		opening	midgame	endgame	no-phases
Regular Learning	-	**0.6793**	**0.4905**	0.5265	**0.5580**
Separated Learning	opening	0.6803	0.3329	0.1923	0.3871
	midgame	0.4210	0.5106	0.4713	0.4711
	endgame	0.3337	0.4544	0.5557	0.4533
	mixture	**0.6803**	**0.5106**	**0.5557**	**0.5756**
Continual Learning	opening	0.6914	0.4204	0.2904	0.4558
	midgame	0.4580	0.5088	0.4725	0.4816
	endgame	0.3312	0.4509	0.5502	0.4495
	mixture	**0.6914**	**0.5088**	**0.5502**	**0.5763**
Weighted Learning ($a = 4$)	opening	0.6955	0.4915	0.5245	0.5624
	midgame	0.6665	0.4938	0.5044	0.5480
	endgame	0.6764	0.4927	0.5484	0.5654
	mixture	**0.6955**	**0.4938**	**0.5484**	**0.5714**
Weighted Learning ($a = 10$)	opening	0.6890	0.4554	0.4771	0.5312
	midgame	0.6558	0.5002	0.4979	0.5450
	endgame	0.6446	0.4800	0.5423	0.5493
	mixture	**0.6890**	**0.5002**	**0.5423**	**0.5698**

Table 5.2: Overview of **all policy accuracies** of each expert's final model checkpoint for all approaches. The values are shown for the opening, midgame, endgame, and no-phases test sets. In addition to the specific expert's policy accuracies, we added a row for the loss of the resulting mixture model that uses all three experts. The mixture models choose which expert to use based on the phase of the current position in the test set (i.e., the opening expert evaluates all opening positions, the midgame expert evaluates all midgame positions, etc.). We color-coded each column individually from low policy accuracy (dark red) to high policy accuracy (dark green).

Surprisingly, using $a = 10$ for our Weighted Learning approach leads to substantially higher Elo gains than using $a = 4$, even though the overall loss was the same for both a values. We conclude that the actual expert impacts in some scenarios will only be visible after performing benchmark matches and that only comparing the achieved losses is insufficient. The fact that the higher a value performs better shows that more specialization is beneficial as the samples of the other phases are weighted less the higher a gets. Therefore, the Weighted Learning approach may achieve similar results as the other approaches when increasing a even further. There might even be an a value that corresponds to a better tradeoff than not using the other phases at all (like in approaches 1 and 2). However, we have to keep in mind that using all samples for training, as in approach 3, triples the training time compared to approach 1, where each expert is only trained on about a third of the entire dataset.

In theory, the Continual Learning approach should be strictly superior to the Separated Learning approach. Its final training stage is exactly equivalent to the overall training of a Separated Learning expert, the only difference being the network initialization. Consequently, we expected the approach to perform better as it had access to the samples of the other phases during the first two training stages. However, due to the catastrophic forgetting behavior, nearly no previously learned knowledge was still present at the start of the final stage, and the learned starting weights achieved about the same final losses as random starting weights. We still think the approach can be beneficial when carefully adjusting the learning rate and momentum schedule for the different stages in order to limit the amount of forgetting. However, restricting the parameter updates in this manner during the final stage may lead to fewer specialization opportunities for the expert so that the final network will be too similar to the baseline network.

Overall, we are confident that an ME approach for MCTS can be beneficial for chess as the shown experimental improvements are significant, not only because of their magnitude but also because of the high amount of games used per match, leading to reliable results. However, the Elo gains might not be as drastic as our results show because the baseline model corresponds to the weakest of all seeds tested in Section 4.2.6.

approach	batch size					
	1	8	16	32	64	avg
Separated Learning	106.89	123.45	126.81	124.36	129.49	**122.20**
Continual Learning	111.68	120.81	122.49	125.00	125.55	**121.11**
Weighted Learning ($a = 4$)	25.61	23.20	23.52	22.28	21.31	**23.18**
Weighted Learning ($a = 10$)	50.41	52.42	56.33	63.55	56.51	**55.84**

Table 5.3: Overview about the **relative Elo gains of the different approaches** (compared to the baseline approach) for different batch sizes. The values resemble the average Elo gain of all node and move time configurations for the particular batch size. The cells are color-coded from best (dark green) to worst (dark red). The last column displays the overall average Elo gain for all 55 performed experiments of each approach.

5.3 Chess960 Problems

All our tested approaches lead to a significant performance decrease for Chess960. The increased loss on the opening test set for Chess960 does not indicate that the opening is the deciding factor for the poor performance of our approaches, as the opening loss also increased for the baseline model compared to regular chess. The comparison of expert impacts for Chess960 in Figure 4.26 supports that claim, showing that the opening expert was not particularly relevant for the Elo differences.
Instead, possible reasons for the poor results might be that the game phases are not suitable for the new environment (Chess960), the increased complexity of the game, or the used dataset. We assume that the chosen game phase definitions are still appropriate for Chess960 as the nature of the game does not change dramatically apart from the first few moves. Additionally, Table 4.1 shows that the chosen definitions still split the input space into three approximately equally sized subspaces for Chess960.

We suspect the main reason for the performance decrease is the dataset's nature. It being significantly smaller than the chess dataset is exacerbated by the fact that each expert is only trained on about one-third of the available samples. The resulting datasets for each expert might not contain enough information to adequately learn the intricacies of Chess960. Additionally, the quality of the samples was worse as the average player skill was considerably lower, and low time control games were included.

As a result, using an ME approach for MCTS is not always beneficial, and the success depends on the environment, the chosen game phases, and especially the quality and size of the underlying training sets.

6 Conclusion

In this thesis, we analyzed how chess can be split into three phases and whether the resulting subspaces are appropriate for an ME approach to MCTS.
We conclude that the rules for transitioning between game phases should be sophisticated and carefully chosen instead of simply using a criterion such as the current move number. The Lichess game phase definitions form a solid baseline for further adjustments in this area.
Recapping the results of our experiments, we conclude that using ME approaches can lead to significant performance increases for chess. Strictly splitting the input space, as in the Separated and Continual Learning approaches, lead to the best results.
Our Chess960 experiments show that only choosing appropriate game phase criteria does not necessarily lead to improvements. The quality and size of the underlying dataset have a substantial impact on the applicability of ME approaches and should be carefully chosen.

6.1 Future Work

In order to test our hypothesis that the dataset quality was the reason for the performance decreases in Chess960, the experiments should be repeated using a bigger and maybe also more robust dataset without low time control games. Additionally, slight adjustments to the phase definitions could be made that increase the opening length to raise the impact of the opening expert. In return, the subspace for the midgame expert is decreased, reducing the complexity of this phase.
Because the amount of overfitting was relatively low for all experts, increasing the number of epochs might be beneficial and might lead to the experts specializing even more. Similarly, the model architecture could be specially chosen for different experts. For example, the endgame typically features fewer pieces, so different input representations

and customized convolutional blocks could prove beneficial. We also did not tune any hyperparameters specifically for certain experts, which can be investigated in future work as well. Especially for the Continual Learning approach, changes in this regard could lead to significant improvements if the amount of catastrophic forgetting can be limited.
A different approach to finding suitable separations of the game would be clustering. Even though we lose interpretability, clustering might find better alternatives for grouping certain positions, leading to more localized experts. The resulting clusters could either be used as fixed phase criteria or adapted during the learning process using expectation-maximization techniques. In addition to that, the impact of using more than three experts can easily be investigated by using more clusters.
There are also alternative approaches for integrating ME in MCTS that could be analyzed. It is possible to use one common network stem sharing the parameters for each expert and only finetune specific expert heads. Furthermore, instead of using the majority phase in a batch during MCTS, we could query multiple networks and weigh the outcome based on the batch phase distribution. Another approach would be to fill separate clean batches for each expert and only query the expert once its batch is full so that no positions are passed through the wrong expert. It is also possible to adjust our Weighted Learning approach so that each sample is not given a fixed weight according to its phase. Instead, a sample would be weighted according to some heuristic that describes how close it is to the phase of the currently trained expert.
Future work should also examine the impact of using reinforcement learning. Through self-play, the experts could get even more specialized, which might lead to better results. Finally, it has to be examined whether the proposed approaches can also work for environments that can not easily be divided into natural phases.

Bibliography

[1] Aleksandar Botev, Guy Lever, and David Barber. *Nesterov's Accelerated Gradient and Momentum as approximations to Regularised Update Descent*. 2016. arXiv: 1607.01981 [stat.ML].

[2] *Crazyara - A Deep Learning UCI-Chess Variant Engine written in C++ & Python*. Nov. 6, 2023. URL: https://github.com/QueensGambit/CrazyAra (visited on 11/06/2023).

[3] *Cute Chess is a graphical user interface, command-line interface and a library for playing chess.* Nov. 6, 2023. URL: https://github.com/cutechess/cutechess (visited on 11/06/2023).

[4] Johannes Czech, Jannis Blüml, and Kristian Kersting. *Representation Matters: The Game of Chess Poses a Challenge to Vision Transformers*. 2023. arXiv: 2304.14918 [cs.AI].

[5] Johannes Czech, Patrick Korus, and Kristian Kersting. *Monte-Carlo Graph Search for AlphaZero*. 2020. arXiv: 2012.11045 [cs.AI].

[6] Johannes Czech et al. "Learning to play the Chess Variant Crazyhouse above World Champion Level with Deep Neural Networks and Human Data". In: *CoRR* abs/1908.06660 (2019). arXiv: 1908.06660. URL: http://arxiv.org/abs/1908.06660.

[7] Omid E. David, Nathan S. Netanyahu, and Lior Wolf. "DeepChess: End-to-End Deep Neural Network for Automatic Learning in Chess". In: *Artificial Neural Networks and Machine Learning – ICANN 2016*. Springer International Publishing, 2016, pp. 88–96. DOI: 10.1007/978-3-319-44781-0_11. URL: https://doi.org/10.1007%2F978-3-319-44781-0_11.

[8] Arthur P Dempster, Nan M Laird, and Donald B Rubin. "Maximum likelihood from incomplete data via the EM algorithm". In: *Journal of the royal statistical society: series B (methodological)* 39.1 (1977), pp. 1–22.

[9] *Ender is a neural network trained specifically on late game chess positions*. Nov. 15, 2023. URL: https://github.com/dkappe/leela-chess-weights/wiki/Endgame-Net (visited on 11/15/2023).

[10] *Game phase separation implementation of Lichess*. Nov. 15, 2023. URL: https://github.com/lichess-org/scalachess/blob/master/src/main/scala/Divider.scala (visited on 11/15/2023).

[11] J.B. Hampshire and A. Waibel. "The Meta-Pi network: building distributed knowledge representations for robust multisource pattern recognition". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 14.7 (1992), pp. 751–769. DOI: 10.1109/34.142911.

[12] *Implementation of Elo error margins in the cutechess repository*. Nov. 6, 2023. URL: https://github.com/cutechess/cutechess/blob/master/projects/lib/src/elo.cpp (visited on 11/06/2023).

[13] Robert A. Jacobs et al. "Adaptive Mixtures of Local Experts". In: *Neural Computation* 3.1 (1991), pp. 79–87. DOI: 10.1162/neco.1991.3.1.79.

[14] Xin Jin and Jiawei Han. "K-Means Clustering". In: *Encyclopedia of Machine Learning*. Ed. by Claude Sammut and Geoffrey I. Webb. Boston, MA: Springer US, 2010, pp. 563–564. ISBN: 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8_425. URL: https://doi.org/10.1007/978-0-387-30164-8_425.

[15] Matthew Lai. *Giraffe: Using Deep Reinforcement Learning to Play Chess*. 2015. arXiv: 1509.01549 [cs.AI].

[16] Li-Cheng Lan et al. "Multiple Policy Value Monte Carlo Tree Search". In: *International Joint Conference on Artificial Intelligence*. 2019. URL: https://api.semanticscholar.org/CorpusID:173188160.

[17] *Leela Chess Zero - Open source neural network based chess engine*. Nov. 15, 2023. URL: https://github.com/LeelaChessZero/lc0 (visited on 11/15/2023).

[18] *lichess.org is a free/libre, open-source chess server powered by volunteers and donations.* Nov. 6, 2023. URL: https://lichess.org/ (visited on 11/06/2023).

[19] Yu Nasu. "Efficiently updatable neural-network-based evaluation functions for computer shogi". In: *The 28th World Computer Shogi Championship Appeal Document* 185 (2018).

[20] David Silver et al. “A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play”. In: *Science* 362.6419 (2018), pp. 1140–1144. DOI: 10.1126/science.aar6404. eprint: https://www.science.org/doi/pdf/10.1126/science.aar6404. URL: https://www.science.org/doi/abs/10.1126/science.aar6404.

[21] David Silver et al. “Mastering the game of Go with deep neural networks and tree search”. In: *Nature* 529 (Jan. 2016), pp. 484–489. DOI: 10.1038/nature16961.

[22] David Silver et al. “Mastering the game of go without human knowledge”. In: *nature* 550.7676 (2017), pp. 354–359.

[23] *The RISEv3.3 model architecture description in the CrazyAra wiki*. Nov. 16, 2023. URL: https://github.com/QueensGambit/CrazyAra/wiki/Model-architecture (visited on 11/16/2023).

[24] *The Scorpio chess engine repository*. Nov. 16, 2023. URL: https://github.com/dshawul/Scorpio (visited on 11/16/2023).

[25] Seniha Esen Yuksel, Joseph N. Wilson, and Paul D. Gader. “Twenty Years of Mixture of Experts”. In: *IEEE Transactions on Neural Networks and Learning Systems* 23.8 (2012), pp. 1177–1193. DOI: 10.1109/TNNLS.2012.2200299.