Representation Matters for Mastering Chess: Improved Feature Representation in AlphaZero Outperforms Switching to Transformers

Johannes Czech[1] Jannis Blüml[1,2] Kristian Kersting[1,2,3,4] Hedinn Steingrimsson[5,6]

[1] Artificial Intelligence and Machine Learning Lab, TU Darmstadt, Germany
[2] Hessian Center for Artificial Intelligence (hessian.AI), Darmstadt, Germany
[3] Centre for Cognitive Science, TU Darmstadt, Germany
[4] German Research Center for Artificial Intelligence (DFKI), Darmstadt, Germany
[5] Department of Electrical and Computer Engineering, Rice University, Houston, TX, USA
[6] The Steingrimsson Foundation (safesystem2.org), Houston, TX, USA

E-mail: {johannes.czech, blueml, kersting}@cs.tu-darmstadt.de, hedinn.steingrimsson@rice.edu

Abstract

While transformers have gained recognition as a versatile tool for artificial intelligence (AI), an unexplored challenge arises in the context of chess — a classical AI benchmark. Here, incorporating Vision Transformers (ViTs) into AlphaZero is insufficient for chess mastery, mainly due to ViTs' computational limitations. The attempt to optimize their efficiency by combining MobileNet and NextViT outperformed AlphaZero by about 30 Elo. However, we propose a practical improvement that involves a simple change in the input representation and value loss functions. As a result, we achieve a significant performance boost of up to 180 Elo points beyond what is currently achievable with AlphaZero in chess. In addition to these improvements, our experimental results using the Integrated Gradient technique confirm the effectiveness of the newly introduced features.

Keywords: Transformer, Input Representation, Loss Formulation, Chess, Monte-Carlo Tree Search, AlphaZero

1 Introduction

Transformers, a neural network architecture introduced in 2017 by Vaswani et al. [30], have become one of the most dominant paradigms in modern Artificial Intelligence (AI). Their range of applications has rapidly expanded, making them prevalent in tasks related to many areas of AI, including natural language processing, computer vision, and multimodal context learning. By employing self-attention mechanisms, they differentiate themselves from traditional Convolutional Neural Networks (CNNs). This distinctive attribute allows the network to dynamically assess the significance of individual elements within the input sequence, providing an alternative to the limitations of sequential processing or fixed-size context windows. The rise in popularity of transformers can be attributed to their proficiency in handling long-range dependencies, a crucial characteristic of computer vision. For this reason, transformer models are now favored over classical CNN approaches in various domains, including computer vision [5].

In the field of Reinforcement Learning (RL), transformers hold great promise for creating robust models that can solve complex decision problems [2, 13]. These models can depict the connections and correlations between sequences of observations, actions, and rewards in the context of RL. They can be used to model the state representation, policy, and value function objectively [11, 17]. In addition, they showcase superior performance as general world models [18].

The Transformer architecture is known for its impressive performance and versatility and has been compared to the "Swiss Army Knife" of AI. However, the question remains: does it really live up to this analogy? Relying on transformers solely due to their growing popularity across various domains does not necessarily lead to improvements. As demonstrated by Siebenborn et al. [23], the effect of transformer architecture on specific applications, including continuous control tasks, can result in differing outcomes. Their research found superior results by replacing the trans-

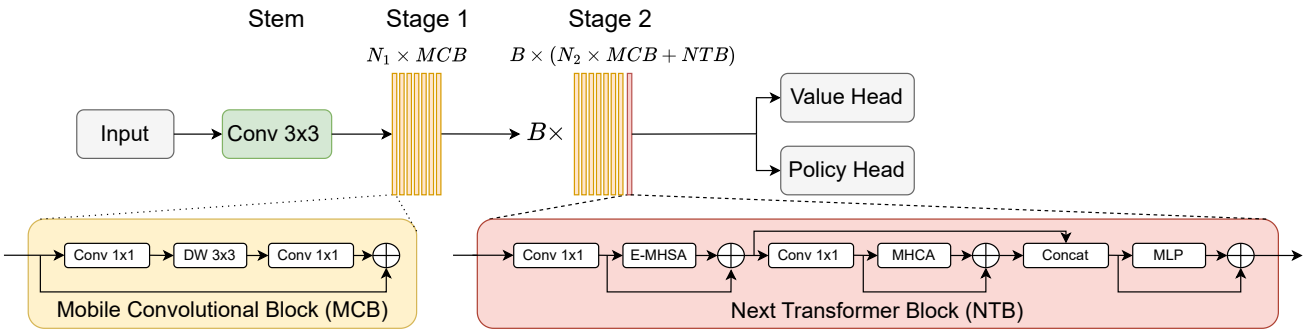


Figure 1. Architectural Overview of the Predictor Network in AlphaVile. The Mobile Convolutional Block (MCB) is inspired by Sandler et al.'s work [22], while the Next Transformer Block (NTB) is integrated from Li et al.'s research [16]. The parameter B denotes the number of hybrid blocks within the architecture, offering scalability to the model. Our standard AlphaVile model employs ten MCBs in Stage 1 ($N_1 = 10$) and two Stage 2 Blocks ($B = 2$). Each Stage 2 Block consists of seven MCBs ($N_2 = 7$) and one NTB.

former with a Long Short-Term Memory network (LSTM). This emphasizes the nuanced nature of the transformer's suitability. While their capabilities are undeniable, the substantial scale of transformers, demanding billions of parameters for peak performance, imposes additional constraints. Substantial computational resources and memory are required, resulting in high latency and efficiency maintenance challenges [20, 32]. These limitations become particularly significant in scenarios such as chess, where minimizing latency is crucial for computational efficiency. This paper assesses the capabilities of transformers in playing chess, a benchmark game in the field of AI. A popular SOTA chess AI architecture is AlphaZero, as introduced by Silver et al. [25, 24]. The methodology seamlessly integrates neural networks with Monte-Carlo tree search (MCTS) [14]. AlphaZero is appealing because it can learn from scratch, adapt to new challenges, and consistently perform well. In this paper, we present AlphaVile[1], a novel convolutional transformer hybrid network. Incorporating a ViT within the AlphaZero system enables testing of the potential of utilizing transformers and CNNs together to enhance chess performance.

Although it is commonly believed that "deep learning removes the need for feature engineering", as argued by Francois Chollet in his book [3], we believe that modifications to feature representation can generate improvements for methods such as AlphaZero. It can help us in achieving similar goals as "exploring the agent state space and having diverse agents with a heterogeneous skill set", which can lead to "creative", broader and more diverse agent behavior [26]. Further, adding useful information to the features, i.e., the moves left in a chess game, can play a crucial role in making progress on an advanced chess task, outperforming much larger neural networks [26].

In addition to making architectural changes, we are investigating potential enhancements regarding the feature representation. We begin by introducing AlphaVile and providing the necessary context. We then discuss the improved input representation and value loss, followed by presenting our empirical evaluation, which highlights the significant impact of our extended representation. Finally, we go over related work and draw a conclusion.

2 AlphaVile: Integrating transformers into AlphaZero

AlphaZero, as introduced by Silver et al. [25, 24], represents a well-established model-based RL approach. Its primary strength is rooted in its ability to make predictions about the likely course of a given situation through the use of Monte-Carlo Tree Search (MCTS) [14] with an innovative integration of neural networks. Specifically, we make use of Prediction Upper Confidence bounds for Trees algorithm (PUCT) that was later refined in AlphaZero. For a detailed description of the PUCT algorithm, see [26].

Our approach centers on the substitution of the residual network architecture (ResNet) [9], a framework heavily reliant on convolutional layers, with a transformer-based architecture. Notably, elements such as the Monte-Carlo Tree Search (MCTS) algorithm and the loss function:

$$\ell = \alpha \left[z - v\right]^2 - \boldsymbol{\pi}^\top \log \boldsymbol{p} + c \cdot \|\theta\|_2^2 \,, \tag{1}$$

remain consistent with the original AlphaZero design. Here, $[z - v]^2$ quantifies the mean squared error between the ac-

[1]For more exhaustive information regarding our network architectures, input representations, and value loss formulations, please check our supplementary material and code on GitHub: `https://github.com/QueensGambit/CrazyAra/releases/tag/1.0.4`, accessed on 2023-10-26.

tual game outcome, denoted as z, and the predicted value v. Similarly, $\boldsymbol{\pi}^\top \log \boldsymbol{p}$ represents the cross-entropy between the target policy vector $\boldsymbol{\pi}$ and our predicted vector $\boldsymbol{p}$, a configuration adopted from Silver et al. [25]. To further refine our model, we use the scalar parameter α, serving as a weighting factor for the value loss. In our experiments, we set α to 0.01, a choice made to mitigate the risk of overfitting.

As can be seen in Figure 1, AlphaVile is the result of a synergistic fusion of components from AlphaZero [25], NextViT [16], and MobileNet [10]. Our approach is based on the Next Hybrid Strategy as explained by Li et al. [16]. In this strategy, a single transformer block is coupled with multiple convolutional blocks. The architectural configuration is further trimmed for optimal performance using TensorRT by combining different blocks and operations into a single block. This is motivated by the work of Dosovitskiy et al. [5] who conducted a comparative analysis that placed their vision ViT architecture in competition with SOTA ResNet [9] and EfficientNet [28] architectures, both reliant on CNNs. Their evaluation yields results that showcase ViT's superior performance in image classification tasks, particularly on well-established benchmark datasets such as ImageNet and CIFAR. Subsequently, Han et al. [8] extend this exploration with another comprehensive evaluation, once again comparing ViT architectures to contemporary CNN-based counterparts. However, their study also highlights the concern of efficiency. Transformer models, by design, tend to be extensive and computationally more demanding than their CNN-based counterparts, often requiring extensive datasets for training. Han et al. emphasizes the symbiotic relationship that emerges when CNN and transformer models are combined. Efforts are also being made to tackle the efficiency issues of ViTs, with a focus on improving performance. Innovations such as TensorRT as well as specially tailored architectures such as Trt-ViT [32] and NextViT [16] are contributing to the ongoing search for efficiency improvements.

In the context of CNNs, the MobileNet architecture was developed by Howard et al. [10]. MobileNet innovatively combines depthwise separable convolutions with pointwise convolutions to reduce the computational overhead and memory demands typically associated with traditional CNNs, all while preserving high accuracy. MobileNets have consistently demonstrated their capability to achieve accuracy across a spectrum of computer vision tasks, all the while exhibiting significantly enhanced speed and memory efficiency compared to conventional CNNs. These characteristics make MobileNet a particularly well-suited choice for integration within the AlphaZero framework. Various iterations of MobileNet have been introduced, including MobileNetV2 and MobileNetV3, each bringing additional optimizations and enhancements to the original architecture.

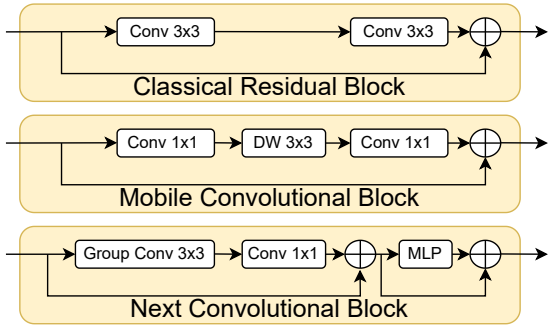


Figure 2. Comparing Architectural Components of Convolution-Based Blocks. This diagram utilises "DW" to denote Depthwise Convolution. Batchnorm and ReLU layers have been omitted for clarity. The conventional residual block, initiated by He et al. [9], is substituted in AlphaVile with the mobile convolution block, found in MobileNet, as stated by Sandler et al. [22]. Additionally, we make use of the next convolution block originally introduced in NextViT by Li et al. [16]

For our work, we leverage the mobile convolution block, as presented in MobileNetV2 and reutilized in MobileNetV3.

Further, we employ stochastic depth techniques [12]. This strategy serves a dual purpose, accelerating the training process while enhancing convergence. Additionally, we implement a scaling technique adapted from EfficientNet [28], which facilitates the generation of networks in varying sizes to suit our needs. More details on this can be found in the supplementary materials section.

To assess the performance of AlphaVile, our evaluation begins with a comparative analysis of the MobileNet block compared to ResNet [9] and the "Next Convolution Block", a component that outperformed the ConvNext Transformer, PoolFormer, and Uniformer blocks in the study conducted by Li et al. [16]. These three convolutional blocks are visually represented in Figure 2. Notably, the mobile convolutional block [22] demonstrates a slightly superior performance compared to the classical residual block [9], while the next convolutional block [16] exhibits notably inferior results under equivalent latency constraints. Consequently, based on experimental results in Table 1, we select the mobile convolutional block as the default convolutional base block for AlphaVile.

Our investigation regarding the combination of convolutional base blocks with transformer blocks covers different sets of transformer blocks according to the integration strategy presented in Table 2. Following the proposal by Li et al. [16], we place a single transformer block after a given number of convolutional blocks, rather than grouping all

Table 1. Training Results for Comparing Convolutional Blocks in the Core of the Model. The mobile convolutional block, with an expansion ratio of three, outperforms the classical residual block and notably surpasses the next convolutional block. Furthermore, all three configurations exhibit similar latency on the GPU. The best results are highlighted in **bold**.

Convolutional Block	Blocks	Channels	Combined Loss	Policy Acc. (%)	Latency (μs)
Classical residual block [9]	10	192	1.2350 ± 0.0031	57.50 ± 0.14	36.17
Mobile conv. block [22]	9	256	$\mathbf{1.2343 \pm 0.0023}$	$\mathbf{57.53 \pm 0.05}$	**34.78**
Next conv. block[16]	10	256	1.2411 ± 0.0009	57.33 ± 0.05	34.84

transformer blocks at the end of the network. For architectures with 15 convolutional blocks, our study shows that sparsity predominates. Smaller quantities of transformer blocks show better results compared to their more numerous counterparts. The configuration with precisely two transformer blocks is the optimal choice and yields the best results. After introducing the AlphaVile architecture, we now turn to the importance of representation.

3 AlphaVile-FX: The importance of representation

In AlphaZero, the traditional representation of the game state is a nuanced art. It manifests as a stack of planes, each of which encodes a particular facet of the complex state of the chessboard. These planes are structured as an 8×8 grid, with each cell serving as a single square on the chessboard. Within this framework, there are two distinct plane types: *bool* and *int*. The first type describes planes where the value of each square is restricted to binary limits, i. e. 0 or 1. As an illustrative example, the first plane indicates the areas occupied by the first player's pawns, where the value 1 indicates their presence and 0 their absence. In contrast, the *int* planes deal with integer numbers and provide a range of values instead of a binary contrast, e. g. the no-progress counter value is set on the entire 22nd plane. To improve computational robustness and numerical stability, these *int* features are thoughtfully scaled to cover the floating point range from -1 to 1, using the extreme feature values as the reference points. The original representation (Inputs V1.0) and can be found in Table 3. In total, Inputs V1.0 comprises 39 planes and gives our input a multidimensional structure: a tensor with dimensions $39 \times 8 \times 8$.

3.1 Expanding the input representation

It is a generally accepted principle that the role of representation is central to traditional machine learning, but there is an ongoing discussion about its importance in the field of deep learning. In this paper, we present a novel interpretation, as shown in both the top and bottom segments of Table 3, of the input by introducing additional features into the existing framework while also removing two features. We exclude the color information as this can distort the evaluation in favor of the White player regardless of the underlying position, as White has often an advantage in chess positions. The differentiation between the active player to move and the opponent player still persists. Moreover, we remove the current move number, but not the no-progress counter for the 50 move rule. Additionally, we add several features including two masks for all pieces of each player, a checkerboard pattern, the relative material difference, a boolean map signaling if there are opposite color bishops, all checking pieces and the overall material count of the current player. The new input definition brings significant improvements, particularly regarding policy and value loss functions. We argue that these supplementary features, though derivable from existing features, enhance the network's capacity by providing essential information in advance, thereby eliminating the need for in-network computations. Table 4 presents evidence that highlights the significance of feature engineering, leading to an advantage of about 100 Elo and demonstrating its continued relevance in the realm of deep neural networks.

3.2 Redefining the value loss representation

In order to boost the performance of chess engines, it is necessary to improve the quality of chess engines' ability to evaluate a given position. Taking inspiration from [26], where it was shown that the approach led to performance improvements on a dataset called chess fortresses, we explore an inventive method advocated by Henrik Forstén, embodied by the Win-Draw-Loss-Head (WDL) framework[2]. This framework accurately predicts the percentage distribution of winning, drawing, and losing scenarios, while also introducing the Moves Left Head[3] to forecast the remaining number of moves until the game's comple-

[2]Further information can be found at `https://github.com/LeelaChessZero/lc0/pull/635`, accessed on 2022-11-11.

[3]Further details can be accessed at `https://github.com/LeelaChessZero/lc0/pull/961`, accessed on 2022-11-11.

Table 2. Grid Search Results for Different Transformer Block Configurations within the AlphaVile Model, as Shown in Figure 1. Optimal performance, highlighted in **bold**, is achieved by including two level 2 blocks (B). The number of mobile blocks (N_1, N_2) is adjusted carefully to ensure the comparability of the models. The number of Transformer Blocks (NTBs) is set according to B.

#Stage 2 Blocks (B)	N_1	N_2	Combined Loss	Policy Acc. (%)	Latency (μs)
0	18	0	1.1901 ± 0.0049	58.67 ± 0.17	53.54
1	8	8	1.1920 ± 0.0021	58.57 ± 0.17	54.40
2	5	5	$\mathbf{1.1887 \pm 0.0065}$	$\mathbf{58.67 \pm 0.12}$	54.86
3	5	4	1.2061 ± 0.0140	58.30 ± 0.43	54.67
4	4	3	1.2327 ± 0.0045	57.57 ± 0.47	**52.68**

tion. To achieve this, we have incorporated an additional output into the value head. This enhanced model, dubbed the WDLP Value Head, accurately predicts the number of half moves left until the conclusive end of the game. Originally developed for finishing off won endgames, Steingrimsson [26] discovered that the WDL approach is suitable more generally for complex chess tasks.

The following equation derives the classic value output (v). The parameter is bounded within the interval $[-1, +1]$ and derived from the interplay of L_{output} and W_{output}:

$$v = -\mathrm{L}_{output} + \mathrm{W}_{output} \quad . \tag{2}$$

In this formulation, L_{output} indicates the likelihood of experiencing a defeat in the game, while W_{output} depicts the probability of achieving victory. To validate our findings empirically, we refer to the information extracted from Table 5. The results clearly demonstrate the advantages of this novel weight loss approach, leading to a 33 Elo improvement.

In this redefined framework, we employ a consistent value policy loss, which is accompanied by an auxiliary goal of the remaining number of plies. This results in a significantly transformed loss function

$$\ell = -\alpha(\mathbf{WDL}_{\mathrm{t}}^{\top} \log \mathbf{WDL}_{\mathrm{p}}) - \boldsymbol{\pi}^{\top} \log \boldsymbol{p} + \beta(ply_{\mathrm{t}} - ply_{\mathrm{p}})^2 + c\|\theta\|^2 \tag{3}$$

employing $\mathbf{WDL}_{\mathrm{p}}$, a probability distribution that predicts the probabilities of win, draw, or loss, while $\mathbf{WDL}_{\mathrm{t}}$ defines the target distribution. Within this framework, the scalar parameters α and β allow weighting each loss component.

Our experiments reveal that the incorporation of input features and optimization of the loss function enhance performance, leading to an improvement in both loss and accuracy. Therefore, we introduce the "FX" suffix to indicate our **F**eature e**X**tension — a combination of the expanded input representation and the WDLP value head.

4 Investigating the significance of representation in chess mastery

This section presents an empirical study of the AlphaZero, AlphaVile models and their "-FX" variants. We test these across the dimensions of accuracy, latency, and overall playing strength while providing comprehensive context through comparative assessments against other baseline models. Each experimental investigation adheres to a carefully defined set of training hyperparameters, as detailed in the supplementary materials. Our research efforts are strengthened by using three different seeds in our different configurations. We utilize the KingBase Lite 2019 dataset[4] as for training chess networks. This extensive collection of chess data comprises of more than one million games played by expert human players since 2000, each with an Elo rating of over 2200. Furthermore, our scientific exploration includes an ablation study in the field of alternative chess variations, in particular atomic and crazyhouse. Utilizing the lichess.org variant database[5], we extract and examine data from the top decile of players. This approach provides a valuable perspective on model performance in diverse and challenging chess scenarios. The latency evaluation is performed on a NVIDIA GeForce RTX 2070 OC, using a batch size of 64 and benefiting from the advanced TensorRT-8.0.1.6 backend for accelerating throughput.

4.1 Trade-off between efficiency and accuracy

As Han et al. [8] demonstrate that the integration of transformers is widely praised for its versatility, but also presents inherent latency concerns. This is particularly noticeable in competitive contexts where precision and swift processing of large datasets are of importance. To decrease latency without sacrificing accuracy, we suggest combining

[4] https://archive.org/details/KingBaseLite2019, accessed on 2022-11-02

[5] https://database.lichess.org/#variant_games, accessed on 2023-10-23

Table 3. Plane-based Feature Representation for Chess (Inputs V1.0). Features are encoded as binary maps, and specific features are indicated with $*$ as single values applied across the entire 8×8 plane. The historical context is captured as a trajectory spanning the last eight moves. The table begins with traditional input features (listed above the horizontal line). The extended input representation (Inputs V.2.0) incorporates additional features below the horizontal line, while omitting two features marked with ~~strike-through~~.

Feature	Planes	Type	Comment
P1 pieces	6	bool	order: {PAWN, KNIGHT, BISHOP, ROOK, QUEEN, KING}
P2 pieces	6	bool	order: {PAWN, KNIGHT, BISHOP, ROOK, QUEEN, KING}
Repetitions*	2	bool	how often the board positions has occurred
En-passant square	1	bool	the square where en-passant capture is possible
~~Color~~*	1	bool	all zeros for black and all ones for white
~~Total move count~~*	1	int	integer value setting the move count (UCI notation)
P1 castling*	2	bool	binary plane, order: {KING_SIDE, QUEEN_SIDE}
P2 castling*	2	bool	binary plane, order: {KING_SIDE, QUEEN_SIDE}
No-progress count*	1	int	sets the no progress counter (FEN halfmove clock)
Last Moves	16	bool	origin and target squares of the last eight moves
is960*	1	bool	if the 960 variant is active
P1 pieces	1	bool	grouped mask of all P1 pieces
P2 pieces	1	bool	grouped mask of all P2 pieces
Checkerboard	1	bool	chess board pattern
P1 Material difference*	5	int	order: {PAWN, KNIGHT, BISHOP, ROOK, QUEEN}
Opposite color bishops*	1	bool	if they are only two bishops of opposite color
Checkers	1	bool	all pieces giving check
P1 material count*	5	int	order: {PAWN, KNIGHT, BISHOP, ROOK, QUEEN}
Total	39 / 52		

convolutional base blocks and transformer blocks in our architecture. We present four different configurations of our AlphaVile architecture to illustrate the impact of network size on latency, as detailed in Table 6.

We begin our investigation by examining the performance of a fully transformer-based neural network, the ViT [5], when integrated with AlphaZero. In order to ensure a fair comparison of latency, we trimmed these ViT models to match the latency of our AlphaVile architecture, as employing a ViT model with an equivalent number of blocks would considerably increase latency. A comparison of these networks is and depicted in Figure 3. These networks display a relatively high loss and decreased accuracy when compared to AlphaZero. Consequently, we start incorporating convolutional and transformer blocks. This assessment incorporates LeViT [7], NextViT [16], and our proposed AlphaVile design. These strategies offer a better solution than the ViT-based approach and deliver better outcomes than AlphaZero.

4.2 Comparative assessment of playing strength

In order to evaluate the playing strength of our ViT models, we conducted a comprehensive round-robin tournament, in which we pitted AlphaVile, ViT, and AlphaZero* against each other. AlphaZero* here relates to a reimplementation of AlphaZero in the form of ClassicAra using the same model architecture as AlphaZero. The results of this tournament are graphically illustrated in Figure 4. The ViT model falls short of matching the playing strength achieved by AlphaZero* and AlphaVile. This outcome is consistent with the results presented in Figure 3, which highlights the computational disparities between ViTs and our other models. AlphaVile-FX slightly outperform the AlphaZero*-FX version by about 30 Elo. Elo is a metric that measures the relative playing strength difference. We set our baseline Elo rating to 0 Elo, which refers to the weakest participant, here ViT, in our tournament. We refrain from using a baseline Elo rating from ClassicAra from engine rating lists, because we were testing on a different hardware than used for creating the engine list. ClassicAra-1.0.1 participated at the Top Chess Engine Championship (TCEC) season 23 and achieved an Elo rating of 3279, compared to Stockfish-

Table 4. Understanding the Impact of Input Representations on Performance Metrics. This table presents experimental results from different input representations, highlighting their impact on value and policy loss. The adapted Inputs V.2.0 is shown to be the most sophisticated, signaling the ongoing quest for optimization.

Input Representation	Combined Loss	Policy Acc. (%)	Value Loss	Latency (μs)	Elo Difference
Inputs V.1.0	1.1918 ± 0.0028	58.63 ± 0.05	0.4448 ± 0.0007	**52.08**	-
Inputs V.2.0	$\mathbf{1.1901 \pm 0.0049}$	$\mathbf{58.67 \pm 0.17}$	$\mathbf{0.4371 \pm 0.0002}$	53.54	96.7 ± 30.4

Table 5. Finding the Optimal Value Head for Chess Engines. This table reveals the results of two value head types. The Win-Draw-Loss-Ply (WDLP) value head emerges as the winner.

Value Head Type	Combined Loss	Policy Acc. (%)	Value Loss	Latency (μs)	Elo Difference
MSE	1.1933 ± 0.0021	58.50 ± 0.08	0.4406 ± 0.0002	**53.35**	-
WDLP	$\mathbf{1.1901 \pm 0.0051}$	$\mathbf{58.73 \pm 0.12}$	$\mathbf{0.4356 \pm 0.0006}$	53.38	33.2 ± 19.0

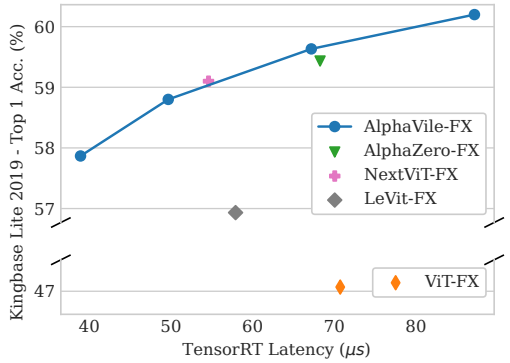


Figure 3. A comparison between AlphaVile and other efficient neural network architectures, with a focus on achieving an optimal balance between accuracy and latency. The results were obtained from three independent seed runs.

dev16 (3625), LCZero-0.30 (3599). We also add Stockfish 16.1-NNUE (15k nodes per move) and FairyStockfish 14-NNUE (50k nodes and 40k per move) as horizontal lines to Figure 4 to make the results more comparable. The modifications to the input and loss representations in our study are substantial, leading to a significant increase in playing ability. In particular, the modification improves the performance of AlphaZero* by 180 Elo points in chess. A lessened increase is evident in chess variants, such as crazyhouse (Figure 4b) and atomic chess (Figure 4c). This emphasizes the significance of these changes, which are apparent in the enhanced playing strength across chess variants. Opening suites were incorporated into the gameplay to introduce a range of game scenarios.

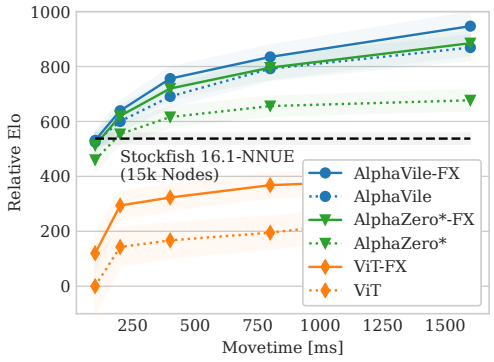
Table 6. Architectural Configurations of AlphaVile in Different Sizes. Note: All versions feature a channel expansion ratio of 2 and use a combination of 50 % 3×3 and 50 % 5×5 convolutions. We use base channel counts that are dividable by 32 for faster inference.

Size	B	N_1	N_2	# Blocks	Base Channels
AlphaVile (tiny)	1	8	6	15	192
AlphaVile (small)	1	11	10	22	192
AlphaVile (normal)	2	10	7	26	224
AlphaVile (large)	2	13	11	37	224

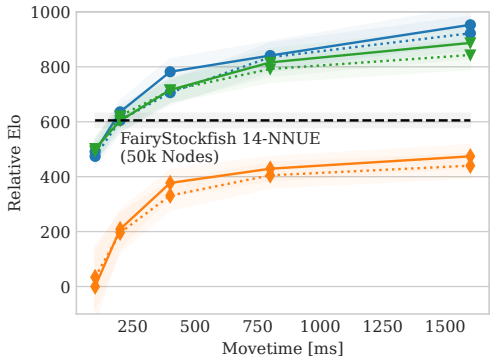
4.3 Interpretability of FX-features

Next, we investigate the interpretability of the FX features and their influence on the model. To determine the importance of each feature channel, we use the Integrated Gradients (IG) method, a widely accepted technique in the field of neural networks' interpretability [27]. We calculate the average attribution of each channel to the model's output value. It is crucial to establish an appropriate baseline for computing gradients. In Figure 5, we utilize the mean of all validation input features.

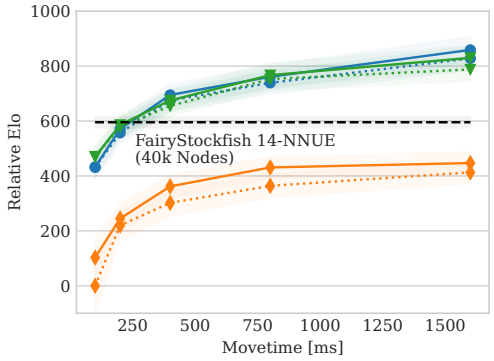
Our analysis shows that the newly introduced feature channels within the FX representation have significant utility. Some feature channels display positive attribution, while others exert a negative influence. This observation supports the logical assumption that a greater number of the opponent's pieces corresponds to a lower value loss. Our analysis suggests, interestingly, that the color channel and move history information are of limited importance, as shown in the second graph. Additionally, it seems that the king's position is of low relevance. This may be misleading, as the



(a) Performance evaluation in chess.



(b) Assessment of playing strength in crazyhouse.



(c) Comparative analysis of playing strength in atomic chess.

Figure 4. The AlphaZero-FX network showcases excellent performance in chess (4a), crazyhouse (4b), and atomic chess (4c), surpassing the vanilla version using Input Representation Version 1 without the WDLP head. The performance increase in chess is noteworthy, with an increase of 180 Elo point. The performance level of the AlphaVile network is comparable to that of the AlphaZero network, especially at longer move times.

king is always present, and a weak or strong king's position has either a positive or negative influence on the value target. The IG method highlights the significance of the player 1 and player 2 masks as the most influential feature channels of the FX-Features. Their integration, in conjunction with the refined value loss representation, significantly improves the evaluation of endgames with opposite-colored bishops, as further elaborated in the appendix.

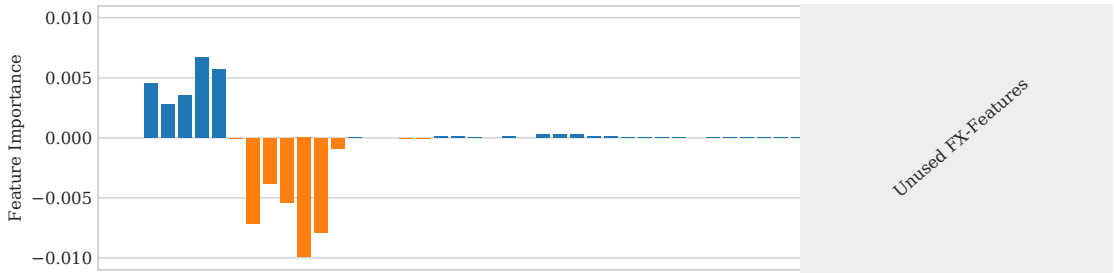
5 Related Work

Originally created for supervised learning tasks, transformers have been widely adopted across multiple domains, such as natural language processing and computer vision. In the realm of RL, where sequential decision-making is crucial, the usage of transformers has become an active area of research [13]. The primary goal of combining transformers and RL is to model and, in several instances, improve decision-making by utilizing attention mechanisms. The sequential nature of RL tasks makes transformers a flexible framework for addressing them. Several approaches have been suggested to overcome the gap between transformers and RL, including enhancements in architecture and trajectory optimization strategies [11].

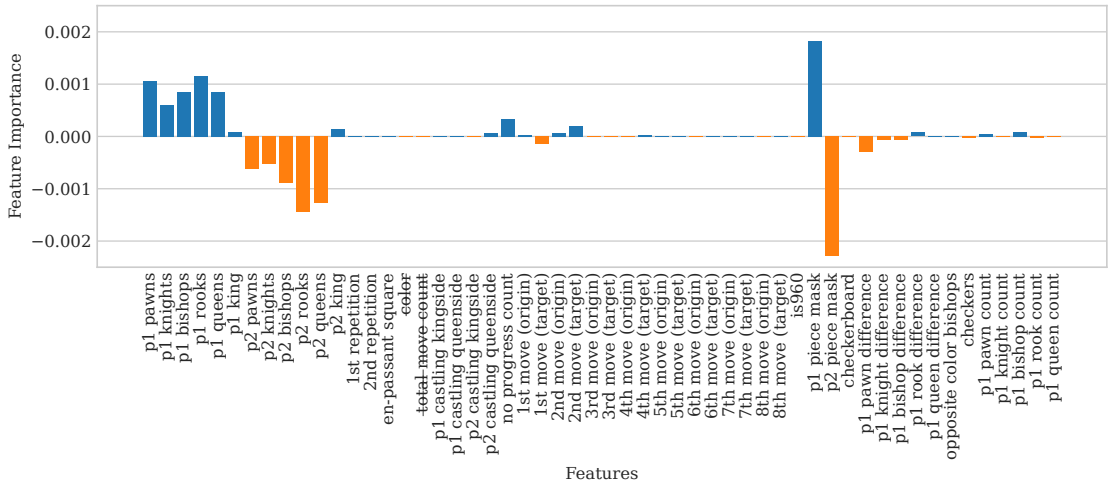
Currently, the Trajectory Transformer [13] and Decision Transformer [2] are among the prominent paradigms. For a thorough comprehension of the combination of transformers and RL, we suggest studying the insightful surveys by Li et al. [17] and Hu et al. [11].

Recent work [6] has shown that utilizing classification-based approaches is generally superior to alternative regression-based approaches. Game-specific features have also been found to be beneficial for the game of Go [31].

The utilization of transformers in chess-related tasks has previously been explored in the literature [4, 19, 29], albeit these investigations differ significantly from our approach. Previous studies mainly utilized Large Language Models (LLMs) and analyzed chess problems from a linguistic perspective. They particularly relied on techniques like Portable Game Notation (PGN) and area-specific terminology to represent chess positions textually. A recent paper [21] explores the utilization of transformer without search and achieves grandmaster-level performance. Although these attempts succeeded in teaching the regulations of chess, they did not achieve the playing skills demonstrated by AlphaZero. In another line of research, the use of transformers in chess has also been envisioned to produce annotations on chess positions, with the objective of enhancing the comprehensibility and traceability of chess analysis [15]. Steingrimsson [26] demonstrated through rigorous experiments on an advanced chess benchmark which SOTA chess architectures still struggle with, the crucial role of neural architecture improvements. This consisted



(a) Analysis of average feature importance for the default input representation.



(b) Analysis of the average importance of FX features.

Figure 5. The newly introduced FX-features demonstrate significant usage, highlighted by the Integrated Gradients (IG) method for feature importance analysis. In the conventional input representation (5a), both positive and negative feature attributions are predominantly related to piece maps. In the enhanced input representation presented in (5b), supplementary features are incorporated, while two features marked with ~~strike-through~~ are omitted. The IG method uses the average of all inputs as a baseline for the attribution calculation.

of broader output variables. They also emphasized the importance of behavioral diversity among agents, which led to creative and varied chess strategies and experiments with additional heads. In this work, we take the next step, exploring input features and their representation.

6 Conclusion

Our study has shown that an optimized input representation and value loss definition significantly enhance the playing strength of chess AIs. Despite the prevailing belief that feature engineering has decreased in relevance with the emergence of deep learning networks, our findings challenge this assumption. Our new input representation includes novel characteristics that arise from the combination of existing features, including material difference and material count. Additionally, there are implicit features derived from the basic rules of chess, such as pieces giving check and the identification of bishops of opposite colors. Transformers are a versatile tool for AI recognized for their ability to process global features and effectively handle extended input sequences, thanks to their use of attention mechanisms. However, their applicability in specific domains such as timed competitive games, like chess, leads to unique challenges beyond accuracy. In such contexts, efficiency is paramount. Efforts to improve the performance of ViTs in chess AI by fusing them with CNNs aimed to exploit the latter's efficient pattern recognition capabilities. Addressing the latency issues typically associated with transformers, these hybrid models generated slightly superior results compared to the pure convolutional network baseline, AlphaZero. Furthermore, custom-made transformers that cater to the specific requirements of chess may improve performance. Ongoing experiments carried out by the Lc0 developer team in

this area demonstrate potential, although further study is required and is outside the scope of this paper. We maintain that transformers hold substantial promise for advancing the field of computer chess. Particularly, their potential applications in areas such as multimodal inputs [33] and retrieval-based approaches [1] may open new avenues for enhancing the capabilities of computer chess engines. Our findings underscore the enduring importance of feature engineering, negating any suggestion of its becoming obsolete and proposing that it remains "forever young".

Acknowledgements

The authors wish to express their gratitude to all those who have contributed to this study. The valuable insights, discussions, and constructive feedback provided by Hung-Zhe Lin and Felix Friedrich have significantly improved the quality of this work. The authors also extend their sincere thanks to Ofek Shochat, Lukas Helff and Cedric Derstroff for their thoughtful comments and suggestions, which have been instrumental in shaping the direction of this research. We are grateful for Daniel Monroe's insightful discussions. We acknowledge the usage of ChatGPT, DeepL and Grammarly to enhance the language style of the paper.

Funding

Funding for this research was partially provided by the Hessian Ministry of Science and the Arts (HMWK) through the cluster project "The Third Wave of Artificial Intelligence - 3AI".

References

[1] S. Borgeaud, A. Mensch, J. Hoffmann, T. Cai, E. Rutherford, K. Millican, G. B. Van Den Driessche, J.-B. Lespiau, B. Damoc, A. Clark, et al. Improving language models by retrieving from trillions of tokens. In *International conference on machine learning*, 2022.

[2] L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch. Decision transformer: Reinforcement learning via sequence modeling. In *Advances in Neural Information Processing Systems*, 2021.

[3] F. Chollet. *Deep learning with Python*. 2021.

[4] M. DeLeo and E. Guven. Learning chess with language models and transformers. In *Data Science and Machine Learning*, 2022. doi: 10.5121/csit.2022.121515.

[5] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. 2020.

[6] J. Farebrother, J. Orbay, Q. Vuong, A. A. Taïga, Y. Chebotar, T. Xiao, A. Irpan, S. Levine, P. S. Castro, A. Faust, et al. Stop regressing: Training value functions via classification for scalable deep rl. 2024.

[7] B. Graham, A. El-Nouby, H. Touvron, P. Stock, A. Joulin, H. Jégou, and M. Douze. Levit: a vision transformer in convnet's clothing for faster inference. In *Proceedings of the IEEE/CVF international conference on computer vision*, 2021.

[8] K. Han, Y. Wang, H. Chen, X. Chen, J. Guo, Z. Liu, Y. Tang, A. Xiao, C. Xu, Y. Xu, Z. Yang, Y. Zhang, and D. Tao. A survey on vision transformer. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023. doi: 10.1109/tpami.2022.3152247.

[9] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.

[10] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF international conference on computer vision*, 2019.

[11] S. Hu, L. Shen, Y. Zhang, Y. Chen, and D. Tao. On transforming reinforcement learning by transformer: The development trajectory, 2023.

[12] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger. Deep networks with stochastic depth. In *European conference on computer vision*, 2016.

[13] M. Janner, Q. Li, and S. Levine. Offline reinforcement learning as one big sequence modeling problem. In *Advances in Neural Information Processing Systems*, 2021.

[14] L. Kocsis and C. Szepesvári. Bandit based monte-carlo planning. In *European conference on machine learning*, 2006.

[15] A. Lee, D. Wu, E. Dinan, and M. Lewis. Improving chess commentaries by combining language models with symbolic reasoning engines, 2022.

[16] J. Li, X. Xia, W. Li, H. Li, X. Wang, X. Xiao, R. Wang, M. Zheng, and X. Pan. Next-vit: Next generation vision transformer for efficient deployment in realistic industrial scenarios. 2022.

[17] W. Li, H. Luo, Z. Lin, C. Zhang, Z. Lu, and D. Ye. A survey on transformers in reinforcement learning. 2023.

[18] V. Micheli, E. Alonso, and F. Fleuret. Transformers are sample-efficient world models, 2023.

[19] D. Noever, M. Ciolino, and J. Kalin. The chess transformer: Mastering play using generative language models. 2020.

[20] R. Pope, S. Douglas, A. Chowdhery, J. Devlin, J. Bradbury, A. Levskaya, J. Heek, K. Xiao, S. Agrawal, and J. Dean. Efficiently scaling transformer inference. 2022.

[21] A. Ruoss, G. Delétang, S. Medapati, J. Grau-Moya, L. K. Wenliang, E. Catt, J. Reid, and T. Genewein. Grandmaster-level chess without search. 2024.

[22] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018.

[23] M. Siebenborn, B. Belousov, J. Huang, and J. Peters. How crucial is transformer in decision transformer? 2022.

[24] D. e. a. Silver. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362, . ISSN 0036-8075, 1095-9203. doi: 10.1126/science.aar6404.

[25] D. e. a. Silver. Mastering the game of Go without human knowledge. *Nature*, 550, . ISSN 1476-4687. doi: 10.1038/nature24270.

[26] H. Steingrimsson. Chess fortresses, a causal test for state of the art symbolic [neuro] architectures. In *2021 IEEE Conference on Games (CoG)*, 2021.

[27] M. Sundararajan, A. Taly, and Q. Yan. Axiomatic attribution for deep networks. In *International conference on machine learning*, 2017.

[28] M. Tan and Q. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, 2019.

[29] S. Toshniwal, S. Wiseman, K. Livescu, and K. Gimpel. Chess as a testbed for language model state tracking. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2022.

[30] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 2017.

[31] D. J. Wu. Accelerating self-play learning in go. 2019.

[32] X. Xia, J. Li, J. Wu, X. Wang, M. Wang, X. Xiao, M. Zheng, and R. Wang. Trt-vit: Tensorrt-oriented vision transformer. 2022.

[33] P. Xu, X. Zhu, and D. A. Clifton. Multimodal learning with transformers: A survey. 2022.

[34] S. Zagoruyko and N. Komodakis. Wide residual networks. 2016.

A Supplementary Materials

A.1 Final Performance Overview of AlphaVile

Table 7 provides a detailed report on the performance metrics of the AlphaVile architecture compared to other efficient neural network architectures.

A.2 Preliminary Experiments for Building AlphaVile

The AlphaVile architecture was developed in stages, commencing with preliminary experiments to evaluate the effectiveness of utilizing pure transformer-based neural networks for standard chess. Initial results, outlined in Table 3, indicated that these networks exhibited sluggish performance and relatively high rates of loss. As part of our study, we endeavored to create a hybrid architecture that integrates both convolutional and transformer components to boost both efficiency and performance.

The process involved several stages. First, we commenced by determining the appropriate convolutional base block. Afterwards, we investigated the best scaling factors for the network's depth and width. Finally, the convolutional stem and transformer block were cleverly combined to create the hybrid CNN-transformer architecture that powers AlphaVile.

In the study by Li et al. [16], various convolution blocks, such as the "Next Convolution Block", ConvNext, Transformer, PoolFormer and Uniformer block, are comprehensively compared. However, this comparison omitted two critical components: the classical residual block that utilizes two 3×3 convolutions, observed in the conventional AlphaZero network, and the mobile convolution block that uses group depthwise convolution. Thus, our primary aim is to establish which of these blocks, portrayed in Figure 2, functions optimally under equal latency constraints. The objective is to select this block as our standard convolutional block after conducting an initial experiment.

The data presented in Table 1 shows that the mobile convolutional block [22] outperforms the classical residual block [9] to a slight extent. However, the subsequent convolutional block [16] demonstrates noticeably lower performance under equivalent latency constraints.

A.3 Optimizing Scaling Ratios for Network

After identifying the most suitable convolutional base block, we explore the optimal scaling ratios for our network. Our approach is in alignment with the compound scaling methodology described in the EfficientNet framework [28]. The compound scaling method, introduced in the EfficientNet framework, utilizes a compound coefficient, denoted as ϕ, to adjust the network width, depth, and resolution in a systematic manner based on the following principles:

$$\begin{aligned} \text{depth: } d &= \alpha^\phi \\ \text{width: } w &= \beta^\phi \\ \text{resolution: } r &= \gamma^\phi \\ \text{s.t.} \quad \alpha \cdot \beta^2 \cdot \gamma^2 &\approx 2 \\ \alpha \geq 1, \beta \geq 1, \gamma &\geq 1 \end{aligned} \tag{4}$$

In our experiment, we maintain a constant input resolution of an 8×8 grid, thus we set γ to 1. However, we have noticed that increasing the width (β) does not entirely meet our criteria:

$$\alpha \cdot \beta^2 \approx 2 \tag{5}$$

This is evident in the "Latency (μs)" column of Table 8, where we observe increased latency as we raise β in comparison to α. Our results are consistent with previous studies, including Zagoruyko and colleagues' work [34], which suggests that widening the network is more effective on GPU architectures than on CPU and is not aligned with the number of floating-point operations.

To achieve an optimal balance in response to network latency variations, we have formulated an adjusted criterion. This refined criterion is defined as follows:

$$\alpha \cdot \beta^{1.6} \approx 2 \tag{6}$$

This adjustment aligns more accurately with our latency considerations, as shown in Table 9. Our exhaustive grid search indicates that the optimal configuration involves an expansion ratio of $\alpha = 1.8$ and $\beta = (10/9)^{5/8}$.

It should be noted that alterations to the network structure have a direct effect on latency, which can result in either an increase or decrease in latency. Therefore, this refined scaling ratio has been adopted for subsequent experiments to maintain consistent latency levels across different configurations.

A.4 GPU Utilization

As high GPU utilization is generally desired, it is noteworthy that the pure convolutional ResNet architecture employed by AlphaZero shows the maximum GPU utilization, peaking at 100 %. On the other hand, a range of ViT architectures demonstrate GPU utilization rates between 89 % and 97 %, as shown in Figure 6.

Table 7. A Comparison of Neural Network Architectures Using Extended Input Representation and WDLP Value Loss Formulation. The largest AlphaVile model provides the most favorable results, but with higher latency. The normal-sized AlphaVile model achieves comparable accuracy and latency to the ResNet architecture.

Network Architecture	Combined Loss	Policy Acc. (%)	Latency (μs)	Flops
AlphaZero-FX [25]	1.1673 ± 0.0040	59.43 ± 0.09	68.25	1.494 G
LeViT-FX [7]	1.2596 ± 0.0040	56.93 ± 0.09	57.90	0.413 G
NextViT-FX [16]	1.1725 ± 0.0040	59.10 ± 0.08	54.59	0.364 G
ViT-FX [5]	1.6866 ± 0.0014	47.40 ± 0.16	70.72	20.82 M
AlphaVile-FX (large)	1.1323 ± 0.0053	60.20 ± 0.16	87.15	0.508 G
AlphaVile-FX (normal)	1.1531 ± 0.0037	59.63 ± 0.13	67.18	0.374 G
AlphaVile-FX (small)	1.1861 ± 0.0082	58.80 ± 0.22	49.66	0.232 G
AlphaVile-FX (tiny)	1.2193 ± 0.0101	57.87 ± 0.25	38.92	0.171 G

Table 8. Results from a grid search experiment analyzing various scaling parameters to adhere to the principles of efficient net design [28]. The experiment systematically varied the depth scaling factor α and channel size scaling factor β. The best performance was attained with a depth scaling factor of α=1.8 and a channel size scaling factor of $\beta = \sqrt{10/9}$. It is noteworthy that scaling along β yielded a more favorable latency-to-FLOPS ratio than anticipated.

α	β	Depth	Channels	Combined Loss	Policy Accuracy (%)	Latency (μs)
1.0	$\sqrt{2}$	10	272	1.21497	58.1	45.40
1.2	$\sqrt{5/3}$	12	248	1.21381	58.0	**44.55**
1.4	$\sqrt{10/7}$	14	229	1.19888	58.5	47.88
1.6	$\sqrt{5/4}$	16	215	1.20036	58.5	50.64
1.8	$\sqrt{10/9}$	18	202	**1.19084**	**58.6**	51.15
2.0	1	20	192	1.19559	58.4	50.20

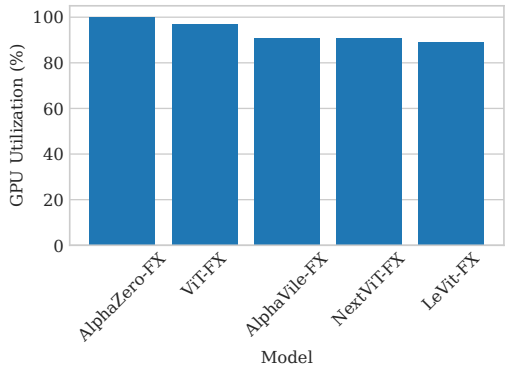
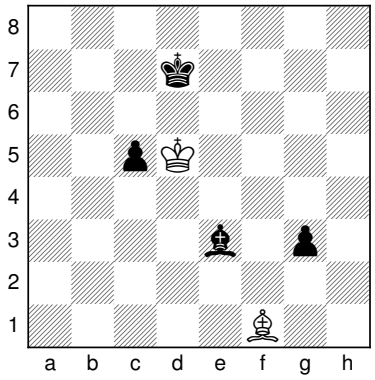


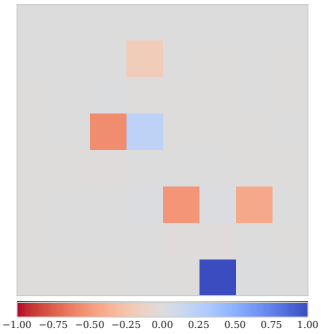
Figure 6. An in-depth analysis of GPU usage across different model structures, highlighting the exceptional performance of the standard AlphaZero-FX network compared to its competitors.

A.5 Case Study: Opposite Color Bishop Positions

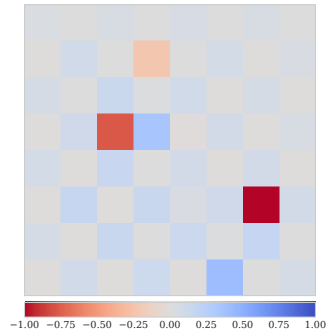
Opposite color bishop positions are widely recognized as complex endgame scenarios in chess, often leading to draws even when one player has a material advantage. This section aims to determine whether the use of FX representation improves the evaluation of these challenging positions. To conduct this analysis, fully trained neural networks were subjected to a series of tests covering 20 different opposing color bishop positions. Details are shown in Table 11. For Figure 8, an all-zero input is used as the baseline. The neural network integrated with the FX-representation, shown in the representative position in Figure 7a, produces value estimates that tend to approach zero for drawn positions. On the other hand, the same network assigns significantly higher value scores for positions that represent decisive victories, as demonstrated in Figure 8a.



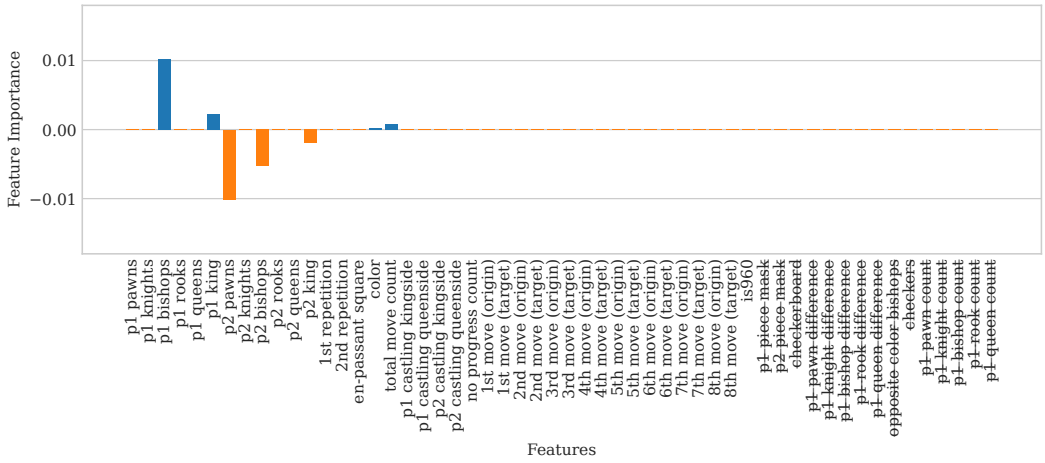
(a) N. Miller vs. A. Saidy. White to move.



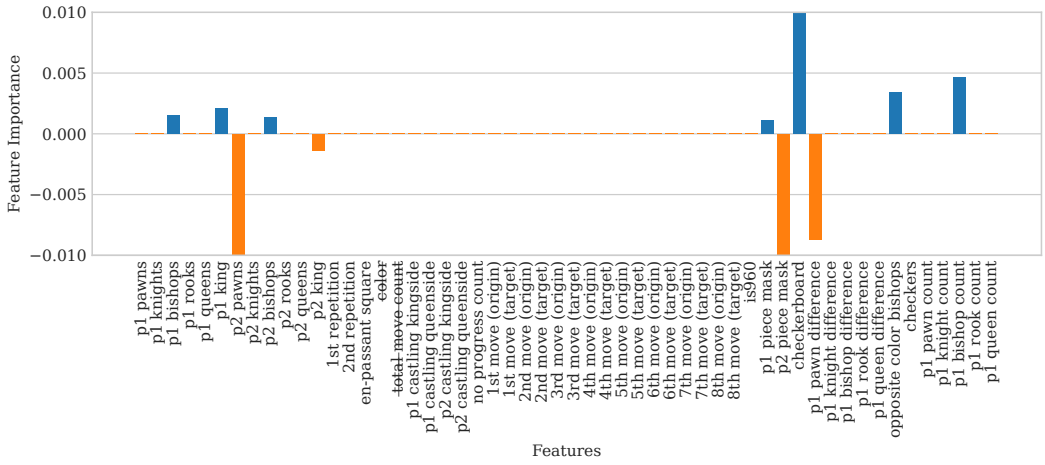
(b) Spatial average feature importance for the default input representation.



(c) Spatial average feature importance for the FX representation.

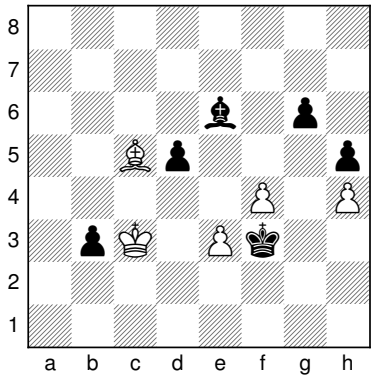


(d) Average feature importance for the default input representation.

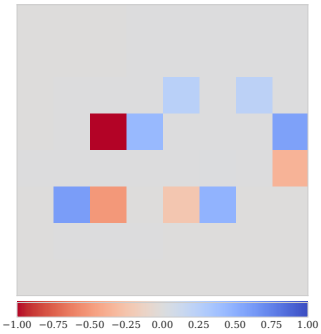


(e) Average feature importance for the FX-features.

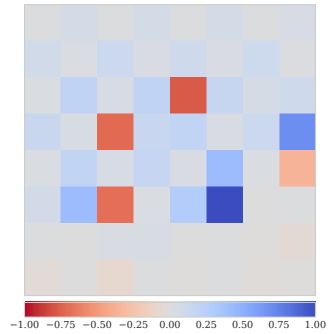
Figure 7. Chess endgame position from the historic game N. Miller vs. A. Saidy, 1971, along with the corresponding feature importance analyses for both the standard AlphaZero-Resnet and the AlphaZero-FX Resnet. White resigned in this drawn position (Forsyth–Edwards Notation (FEN)): `8/3k4/8/2pK4/8/4b1p1/8/5B2 w`). Single value evaluation by the default network: -0.4940, while the FX-Network evaluates it as -0.2304. Values are presented with respect to the current player to move. Integrated gradient analysis employed all-zero inputs as the baseline.



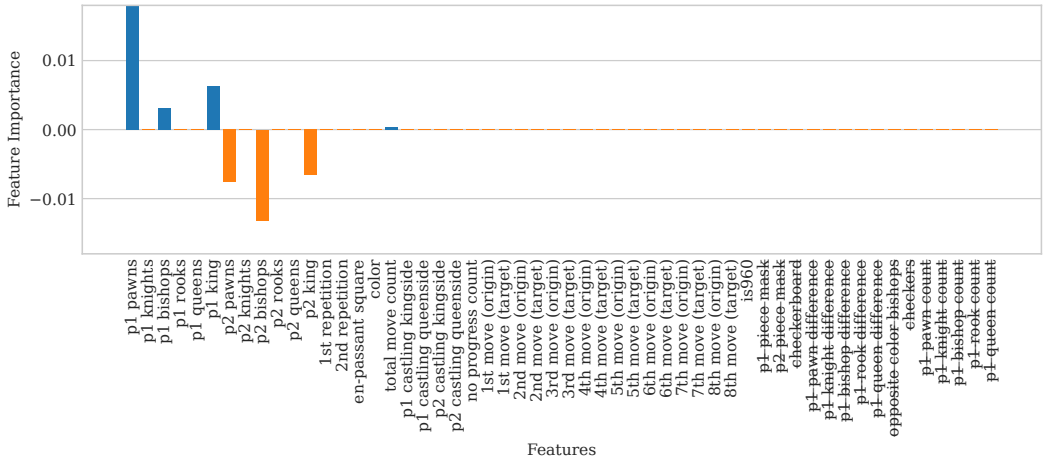
(a) Kotov vs. Botvinnik. Black to move.



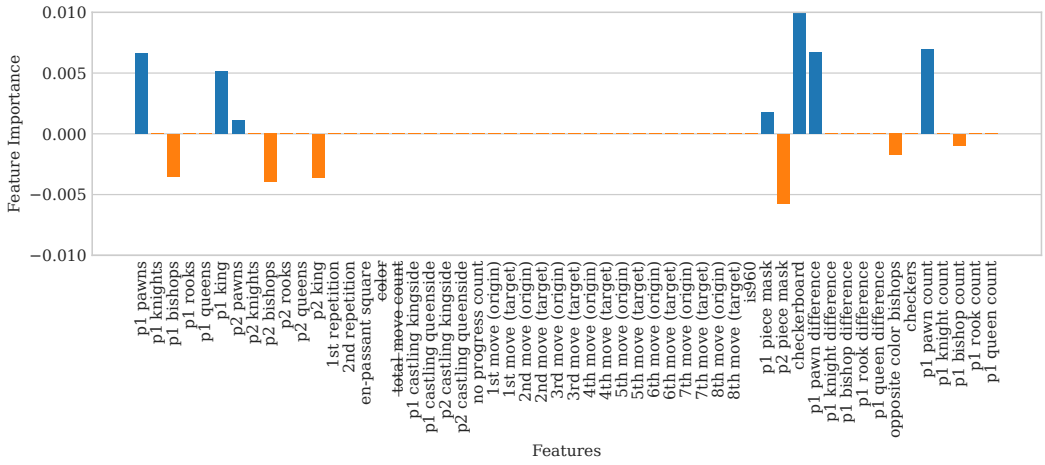
(b) Spatial average feature importance for default input representation.



(c) Spatial average feature importance for FX-representation.



(d) Average feature importance for default input representation.



(e) Average feature importance for FX-features.

Figure 8. Visualization of a chess endgame position from the historic Kotov vs. Botvinnik game in 1955. The associated feature importance of both the standard AlphaZero-Resnet and the AlphaZero-FX Resnet models is displayed. Black secures victory by sacrificing both the g- and d-pawns, creating a new passed-pawn in the process. Forsyth–Edwards Notation (FEN): `8/8/4b1p1/2Bp3p/5P1P/1pK1Pk2/8/8 b`. Single-value evaluations by the default network: 0.4053 vs. FX-Network: 0.4225. All-zero inputs serve as the baseline for the Integrated Gradient method.

Table 9. Results of a grid search experiment with scaling parameters adapted according to the principles of efficient net design [28]. The best performance was achieved with a depth scaling factor of $\alpha = 1.8$ and a channel size scaling factor of $\beta = (10/9)^{5/8}$. It is worth noting that latency measurements remain consistent across all configurations.

α	β	**Depth**	**Channels**	**Combined Loss**	**Policy Accuracy (%)**	**Latency (μs)**
1.0	$2^{5/8}$	10	296	1.21062 ± 0.00117	58.133 ± 0.047	49.19
1.2	$(5/3)^{5/8}$	12	264	1.19977 ± 0.00263	58.333 ± 0.124	51.26
1.4	$(10/7)^{5/8}$	14	240	1.19820 ± 0.00791	58.500 ± 0.216	**49.01**
1.6	$(5/4)^{5/8}$	16	221	1.20031 ± 0.00924	58.400 ± 0.216	51.67
1.8	$(10/9)^{5/8}$	18	205	**1.19009 ± 0.00488**	**58.667 ± 0.169**	53.54
2.0	1	20	192	1.19039 ± 0.00740	58.600 ± 0.308	50.20

Table 10. Results of a grid search exploring different ratios of 5×5 kernel filters. The best outcome is achieved by combining 50 % 5×5 convolutions with 50 % 3×3 convolutions, and using an adjusted expansion ratio of 2.04 instead of the traditional 3.0.

5×5 Kernel Filter Ratio (%)	**Depth**	**Channels**	**Combined Loss**	**Policy Accuracy (%)**	**Latency (μs)**
0	16	221	1.20031 ± 0.00924	58.400 ± 0.216	51.67
25	16	221	1.18016 ± 0.00594	59.000 ± 0.216	**50.66**
50	16	221	**1.17336 ± 0.00472**	**59.199 ± 0.141**	51.57
75	16	221	1.18521 ± 0.00758	58.766 ± 0.205	52.41
100	16	221	1.18435 ± 0.00514	58.833 ± 0.124	52.32

Table 11. Comparison of evaluations between the FX-network and AlphaZero for opposite color bishop endgames. The evaluations are presented based on the current player's perspective. The chess positions are sourced from `https://en.wikipedia.org/wiki/Opposite-colored_bishops_endgame` as of October 19, 2023. Ground truth evaluations indicate whether the position should result in a draw (DRAW) or a win for White (WHITE WIN) or Black (BLACK WIN). Net Eval and FX-Net Eval are the evaluations produced by AlphaZero's default and FX-network, respectively. Abs. Net Eval and Abs. FX-Net Eval indicate the respective absolute values. Values close to 0 are better for drawn positions, while values close to -1 or 1 are better for winning positions. The best evaluation in comparison to the absolute ground truth evaluation are denoted in **bold**.

FEN	Ground Truth	Net Eval	Abs. Net Eval	FX-Net Eval	Abs. FX-Net Eval
8/2k1b3/2P5/3KP2B/8/8/8/8 w - - 0 56	DRAW	0.3332	0.3332	**0.2025**	**0.2025**
8/3k4/8/2pK4/8/4b1p1/8/5B2 w - - 0 56	DRAW	−0.4940	0.4940	**-0.2304**	**0.2304**
5k2/8/8/7p/1b1p4/8/B7/5K2 b - - 0 56	DRAW	0.4385	0.4385	**0.2500**	**0.2500**
8/2b1k3/8/1B1PP3/3K4/8/8/8 w - - 0 56	DRAW	**0.4313**	**0.4313**	0.4561	0.4561
8/2k5/4Bp2/2b1p1p1/4K2p/7P/8/8 b - - 0 56	DRAW	**0.1648**	**0.1648**	0.2121	0.2121
8/8/8/5B2/1p3b2/2k1p3/8/5K2 w - - 0 56	DRAW	−0.6412	0.6412	**-0.4157**	**0.4157**
8/3k4/p2P4/2P4p/2bB4/P6P/5K2/8 w - - 0 56	DRAW	**0.3874**	**0.3874**	0.4465	0.4465
7b/4k2P/6K1/2p2P2/7P/1B6/8/8 b - - 0 56	DRAW	−0.6286	0.6286	**-0.6233**	**0.6233**
4k2b/7P/5PK1/7P/8/1B6/8/8 w - - 0 56	DRAW	**0.7649**	**0.7649**	0.8562	0.8562
8/5pK1/4k3/6B1/5PbP/6P1/8/8 b - - 0 56	DRAW	−0.4810	0.4810	**-0.4294**	**0.4294**
2r3k1/5ppp/p7/5q2/3P4/b2B2P1/P1R2P1P/5QK1 b - - 0 56	DRAW	**-0.5099**	**0.5099**	−0.5594	0.5594
5k2/5pp1/p6p/5B2/3P4/6P1/P3KP1P/2b5 w - - 0 56	DRAW	**0.3222**	**0.3222**	0.4718	0.4718
3b4/p4B1p/8/6k1/6P1/8/1P3PK1/8 w - - 0 56	DRAW	0.2920	0.2920	**0.0783**	**0.0783**
6B1/4b3/7p/3Pk2P/6PP/7K/8/8 w - - 0 56	DRAW	**0.4867**	**0.4867**	0.5590	0.5590
8/8/8/7p/2p5/5K1k/2Bb4/8 w - - 0 56	DRAW	−0.3318	0.3318	**-0.1916**	**0.1916**
3R4/4BK1k/r5p1/2P2bP1/8/8/8/8 w - - 0 56	WHITE WIN	0.8304	0.8304	**0.8714**	**0.8714**
8/2k1b3/2P5/3K1P1B/8/8/8/8 w - - 0 56	WHITE WIN	**0.4072**	**0.4072**	0.3321	0.3321
3k1b2/8/3PP3/1B1K4/8/8/8/8 w - - 0 56	WHITE WIN	0.7121	0.7121	**0.8229**	**0.8229**
8/2k5/2P1K3/6p1/5p2/2b2B1P/6P1/8 b - - 0 56	WHITE WIN	**-0.2412**	**0.2412**	−0.1441	0.1441
8/8/4b1p1/2Bp3p/5P1P/1pK1Pk2/8/8 b - - 0 56	BLACK WIN	0.4053	0.4053	**0.4225**	**0.4225**
Mean values for draws ↓	0		0.4472		**0.3988**
Mean values for wins ↑	1		**0.5192**		0.5186

Table 12. Hyperparameter Configuration for Experimental Settings. This table provides a comprehensive overview of the essential hyperparameters utilised in our experimental design.

Hyperparameter	Value	Hyperparameter	Value
max learning rate	0.07	value loss factor	0.01
min learning rate	0.00001	policy loss factor	0.988
batch size	1024	wdl loss factor	0.01
max momentum	0.95	plys to end loss factor	0.002
min momentum	0.8	stochastic depth probability	0.05
epochs	7	pytorch version	1.12.0
optimizer	NAG		