arXiv:2111.09259v1 [cs.AI] 17 Nov 2021

Acquisition of Chess Knowledge in AlphaZero

Thomas McGrath[1,+], **Andrei Kapishnikov**[2,+], **Nenad Tomašev**[1], **Adam Pearce**[2], **Demis Hassabis**[1], **Been Kim**[2], **Ulrich Paquet**[1], **and Vladimir Kramnik**[3]

[1]DeepMind
[2]Google Brain
[3]World Chess Champion, 2000–2007[*]
[+]these authors contributed equally to this work

ABSTRACT

What is being learned by superhuman neural network agents such as AlphaZero? This question is of both scientific and practical interest. If the representations of strong neural networks bear no resemblance to human concepts, our ability to understand faithful explanations of their decisions will be restricted, ultimately limiting what we can achieve with neural network interpretability. In this work we provide evidence that human knowledge is acquired by the AlphaZero neural network as it trains on the game of chess. By probing for a broad range of human chess concepts we show when and where these concepts are represented in the AlphaZero network. We also provide a behavioural analysis focusing on opening play, including qualitative analysis from chess Grandmaster Vladimir Kramnik. Finally, we carry out a preliminary investigation looking at the low-level details of AlphaZero's representations, and make the resulting behavioural and representational analyses available online.

1 Introduction

Machine learning systems are commonly believed to learn opaque, uninterpretable representations that have little in common with human understanding of the domain they are trained on. Recently, however, empirical evidence has suggested that at least in some cases neural networks learn human-understandable representations. Notable examples include single neurons in a classifier corresponding to mountain tops or snow [1], syntactic information in language models [2], surprisingly sophisticated conceptual representations in a multimodal network [3] emerging from the alignment of visual and textual data, and the wide range of findings in the the growing field of "BERTology"[4]. Given these findings, it is natural to ask: do they generalise? Should we expect other capable deep learning systems to have similarly meaningful representations? If they do not, then our ability to provide explanations which reflect the computational processes of the model, rather than some alternative rationalisation, will be severely limited.

Although these examples of human-understandable activations are compelling evidence, there are reasons they may not apply more widely. Perhaps these networks only learn human-understandable representations because they are exposed solely to human-generated data, and (at least in the case of classifiers) have human concepts imposed on them via the choice of classification categories. Alternatively, their relative simplicity may make them easier to interpret – none of the examples given earlier exceed human performance. In order to further test the emergence of human-understandable concepts we need to find a domain where superhuman-level performance emerges without the use of human-labelled data. AlphaZero [5] meets both of these requirements: it is trained via self-play, and so has never been exposed to human data, and it performs at a superhuman level in all three domains it was trained on (Chess, Go, and Shogi) when the network is paired with Monte Carlo tree search. In this paper we investigate AlphaZero's representations, and their relation to human concepts in chess. Studying the AlphaZero network an important frontier in our understanding of strong neural networks: if we can find human-understandable concepts here, it is likely we will find them elsewhere.

The AlphaZero network provides a rare opportunity for us to study a system that performs at a superhuman level in a complex domain. The ability to interpret AI systems is particularly valuable when these systems exceed human performance [6] on the tasks they were optimized for, as the systems could uncover previously unknown patterns that could prove to be useful beyond the systems themselves. Understanding such complex systems is likely to involve a substantial amount of analytical effort, so it is desirable to first understand whether the system's representations have any correspondence with human concepts. If no such correspondence exists, then further investigations are unlikely to pay off, whereas if they can be found then a deeper investigation may well uncover more. Although the results we show in this paper are far from a complete understanding of the AlphaZero system, we believe that they show strong evidence for the existence of human-understandable concepts of surprising complexity within AlphaZero's neural network. This lays a solid foundation for further investigation of the AlphaZero network.

---

[*] Classical World Chess Champion (2000–2006); FIDE and Undisputed World Chess Champion (2006–2007).

1.1 Our approach

Our approach to investigating the acquisition of chess knowledge in AlphaZero is a three-pronged one: we probe whether human chess concepts are linearly decodable from the neural network's internal layers; we examine changing behaviours over the course of full training runs; we investigate layers and activations directly.

Probe for concepts Our first priority is to discover whether AlphaZero's internal representations (the activations of its neural network) can be related to human chess concepts. If human concepts can be easily predicted from the network's internal representation, then we should expect deeper investigations to reveal even more information, whereas if there is no relation to human concepts then it is likely that AlphaZero's internal computations will remain opaque after further study. Much of our work therefore focuses on concept-based methods [7, 8, 9, 10, 11, 12]. Concept-based methods involve the detection of human concepts from network activations on a large dataset of inputs. Because chess has been so extensively theorised – and much of this theory has been instantiated in chess engine evaluation functions – we have a broad range of human-defined concepts covering a wide range of complexity, from the existence of a passed pawn, through higher-level concepts such as mobility, all the way to a full position evaluation. The probing process is automated, so we can probe for every concept, at every block and over many checkpoints during self-play training. This allows us to build up a picture of what is learned, when it was learned during training, and where in the network it is computed. Concept-based methods are far from the only approach to understanding neural network computations, and we review other approaches in Section 2.

Study behavioural changes After studying how internal representations change over time, it is natural to investigate how these changing representations give rise to changing behaviours. During the course of training, some moves are preferred over others in the same position, and this preference evolves with training steps. When AlphaZero operates without Monte Carlo Tree Search (MCTS), behavioural changes are simply changes in its prior move selection probability (a complete description of the AlphaZero network and training protocol is given in Section 3). We study these behavioural changes by measuring changes in move probability on a curated set of chess positions and compare the evolution of play during self-play training to the evolution of move choices in top-level human play. Our analysis primarily focuses on openings as these are most heavily theorised, and have made a dataset of these positions with both human and AlphaZero play data available online.

Investigate activations directly Finally, having established that many human concepts can be predicted from AlphaZero's activations after training, we begin to investigate these activations directly. We use the established technique of non-negative matrix factorisation (NMF) [13, 14, 15] to decompose AlphaZero's representations into multiple factors. This approach provides information that is not tied to pre-existing human concepts, providing a complementary view on what is being computed by the AlphaZero network. We make a dataset of matrix factorisations available online. We also investigate a new approach: measuring covariance directly between single-neuron activations and inputs. This method is essentially providing a combination of input features whose presence is most correlated with the activation of a given neuron.

1.2 Summary of results

In this paper we consider the progression of AlphaZero's neural network from initialization until the end of training. We examine chess concepts as they emerge in the network, determine when they were learned in training and where they are computed in the network, and assess their impact on the aggregate value assessment. Our contributions are an improved understanding of: 1) Encoding of human knowledge; 2) Acquisition of knowledge during training; 3) Reinterpreting the value function via the encoded chess concepts; 4) Comparison of AlphaZero's evolution to that of human history; 5) Evolution of AlphaZero's candidate move preferences; 6) Proof of concept towards unsupervised concept discovery.

Many human concepts can be found in the AlphaZero network In Section 4 we demonstrate that the AlphaZero network's internal learned representation of the chess board can be used to reliably reconstruct many human chess concepts. We adopt the approach of using concept activation vectors (CAV) [8] by training sparse linear probes for a wide range of concepts. Our definition of concepts is given in Section 4.1 and our probing methodology is described in Section 4.2. We discuss the strengths and weaknesses of our approach in Section 4.5. We show that many concepts can be regressed more accurately from internal representations than from the network input, indicating that relevant information is being computed by the AlphaZero network. We also show that, while AlphaZero's knowledge of chess seems to be tightly correlated with our concept probes, there is indeed a difference between them, as the reconstruction tends to be incomplete. An analysis of prediction errors in Section 4.4 shows features in common between positions with high prediction error.

A detailed picture of knowledge acquisition during training By using our concept probing methodology we can measure the emergence of relevant information over the course of training, and at every layer in the network. This allows us to produce what we refer to as **what-when-where plots**: *what* concept is learned *when* in training time *where* in the network. What-when-where plots are plots of concept regression accuracy across training time and network depth. We show results for a range of concepts in Section 4, with the remainder in the Supplementary Material. We repeat this analysis for a second training run, demonstrating remarkable consistency in the development of concepts. We also find that many concepts emerge at approximately the same point early in training, at which point AlphaZero's move selection also changes rapidly and dramatically. We discuss this rapid change in Section 6.

Use of concepts; Relative concept value In Section 4.6 we focus on the evolution of AlphaZero's value function over time. We again use a concept-based approach: we attempt to predict the output of the value function from sets of human concepts. Studying the evolution of concept weights over the course of training gives us a picture of how AlphaZero's behaviour is related to high-level human chess concepts, which we view as a surrogate for its 'style'. We find that early in training AlphaZero focuses primarily on material, with more complex and subtle concepts such as king safety and mobility emerging as important predictors of the value function only relatively late in training.

Comparison to historical human play When looking at the evolution of chess understanding within a system like AlphaZero, one has to wonder whether there is such a thing as a natural progression of knowledge, a way of developing an understanding of the game that is specific to the game itself, rather than being purely arbitrary and left to chance – or is it a mix of both? We investigate this question in Sections 5 and 6 by comparing AlphaZero training to human history and across multiple training runs respectively. Our analysis shows that there are both similarities as well as differences, and that AlphaZero doesn't recapitulate human history: there are notable differences in how human play has developed, but striking similarities in self-play policy. We also present a qualitative assessment of differences in playstyle over the course of training.

Unsupervised knowledge discovery In Section 7 we present preliminary findings from using unsupervised methods to inspect AlphaZero's representations directly. Section 7.1 uses non-negative matrix factorisation to decompose activations at each layer into multiple factors, allowing us to identify factors relating to move selection. In Section 7.2 we compute the covariance between activations and inputs, providing a new view on the relationship between activations in early layers and network representations.

2 Past and related work

Our work relates two areas: machine learning explainability/interpretability (which we refer to simply as interpretability for conciseness), and chess as a testing ground for AI research. We use interpretability tools as experimental instruments that help us learn about the way a given model (in this case AlphaZero) operates on a specific domain. In this section we review prior work on neural network interpretability (Section 2.1) and the role of chess as a proving ground for AI (Section 2.2).

2.1 Interpretability

Neural network interpretability is a broad research area [16], with different notions of interpretability covering a range of use cases [17, 18]. There are two distinct approaches to explaining neural networks: (1) build an inherently interpretable model or (2) generate post-hoc explanations given an already trained model. While (1) is an important approach for high stakes domains, it may not be always feasible: sometimes (as in this case) we already have a model we wish to understand, and the highest-performance models may often not be the ones with the most inherently interpretable architectures. In this case we must rely on post-hoc interpretability methods. Because of our interest in relating network activations to human concepts our work primarily uses concept-based explanations (Section 4), although we also explore approaches that focus directly on network activations in Section 7.

Concept-based interpretability Concept-based interpretability methods build on the idea of network probing [19], and use human understandable concepts to explain neural network decisions in terms that users can understand [7, 8, 9, 10, 11]. If these users are domain experts, they can specify complex domain-specific concepts, allowing them to probe network operation and understand network decisions in greater detail. Concept-based explanations have been successfully used in complex scientific and medical domains, as seen in [20, 21, 22, 23, 24, 25, 26, 27]. Chess poses similar challenges and opportunities for interpretability research: complex concepts developed over centuries often cannot be simply expressed using a set of board positions/features and encode deep domain knowledge. However, because of the rich history of chess as a domain for AI research, many of these concepts are already expressed (at least in a simplified, easy-to-compute form) inside modern chess engines, giving a natural starting point for concept-based investigations.

Other post-hoc interpretability methods A widely-used method for generating post-hoc explanations is to learn weights for each input feature (e.g., pixel for images) indicating how 'important' each feature is to the prediction. The definition of importance varies in papers depending on the goal of explanations: some use game theoretic credit assignment approaches [28], while others derive methods from a set of axioms [29] or use simpler approaches of perturbation [30, 31] or optimization of desiderata [32]. While many of these methods are shown to be useful for end-tasks, more recent studies called some of these methods into question in terms of their validity [33, 34], robustness [35] or their formulation [36]. For example, [33] showed that explanations from a trained and an untrained model are visually indistinguishable and [37] also hinted that the explanations do not seem to be useful for debugging typical ML model problems.

An alternative approach to post-hoc interpretability focuses on obtaining a low-level mechanistic understanding of a given neural network, understanding the precise algorithms implemented at each layer and the representations they give rise to. Early work in this approach focused on understanding the representations at intermediate layers of vision networks using matrix factorisation and feature visualisation approaches [15, 38]. More recent work has focused on representations in multimodal networks [3], as well as circuit-level understanding of neural network algorithms [39, 40]. The fact that these approaches are successful indicates some degree of alignment between learned representations and human-understandable concepts. This remarkable fact (which is also implicitly used by concept-based approaches) is probed in further detail by 'network dissection' approaches [1, 7]. Network dissection searches for interpretable hidden units in intermediate layers of neural networks, and has been used to quantify the conditions under which they emerge.

Challenges for post-hoc interpretability methods It is worth noting that post-hoc interpretation (feature or concept-based) may lack fidelity relative to inherently interpretable model: the alignment between the model's reasoning and the explanation. For concept-based explanation, the alignment between the representation and a human's mental model of the concept remains an open problem [12, 41, 42]. Our research is no exception: we conduct probing-based approach that can measure only a proximate correlation (between human concepts and the representation), and not causation in any forms. Understanding the causal relationship between concepts and behaviour is a challenging problem which has received relatively little attention. The current best solution requires a generative model for inputs which can be conditioned on specific concept values (simulating the do-operator) [43].

Explainability in reinforcement learning Recently, there has been also increasing interest in improving explainability for RL methods [44]. These types of systems pose some unique challenges, due to the complexity of both the environments in which they operate, as well as the complexity of many agent architectures. Some approaches aim to help design transparent and explainable RL models, while others aim to help improve the post-hoc explainability of existing systems. Representation learning in RL agents [45, 46, 47, 48, 49, 50] can be beneficial when used to learn low-dimensional representations for states, policies and actions, hoping to meaningfully capture the variations in agent's environment and disentangle the contributing factors. Symbolic approaches have been proposed for representing objects and relations and better utilisation of either background or acquired knowledge in RL agents [51, 52, 53, 54].

Learning the explanations alongside the agent policy is another promising option when it is possible to define useful auxiliary goals and/or codify the knowledge of the environment. Structural causal models have been used in learning the action influence models [55], to answer counterfactual questions about the actions taken. Reward difference explanations [56] have shown to be helpful in understanding why certain actions are being taken instead of others. Hierarchical reinforcement learning [57] and sub-task decomposition [58] can be seen as more interpretable, by introducing structure in the action space, where high-level goals are divided into sub-goals, and the corresponding sets of actions can therefore be more easily interpreted. Yet, this involves a fairly specialized approach to agent building, and is not as applicable to understanding pre-existing systems. In contrast, our own work presented in this paper is of a post-hoc nature, given that we were trying to better understand AlphaZero as a fixed, pre-existing RL system that has demonstrated a high level of performance across multiple domains. In terms of prior approaches for post-hoc explainability of RL systems, most work to date has involved relying on saliency maps in the input space [59, 60, 61]. An interesting approach to understanding temporally-extended behaviour in agents with recurrent neural networks is to extract finite-state models of an agent's recurrent state [62]. Agent behavior can also be analysed by trying to identify interesting points in behavioral trajectories [63, 64] and highlight the key decisions.

2.2 Chess and AI research

The game of chess was part of the narrative of AI research since the early days of computing, and it is growing into a testing ground for AI interpretability.

Chess as a model system for AI research Chess has for a long time been a "Drosophila" of AI research [65]; a model game used to prototype novel approaches and models of cognition. Although many of the chess engines that play at superhuman levels do so by 'brute force' — leveraging an enormous number of position evaluations to compensate for a lack of intuition — more recent neural network engines such as AlphaZero [5] and Lc0 [66] achieve superhuman strength while evaluating far fewer positions. Although neural network chess engines still evaluate many more positions than a human player would consider, their evaluation functions are learned rather than hardcoded. This raises the possibility that we could learn more about the game of chess by studying neural networks that play it well. The combination of extensive human knowledge, an ultimately verifiable 'truth' in the position, and involving play that is partly intuition and partly move-by-move calculation mean that the role of chess in AI research could be reinvigorated by recasting it at the forefront of AI explainability research. This sentiment has been voiced by former world champion Gary Kasparov [65, 67], while recalling the match with DeepBlue, and looking forward towards AlphaZero and the importance of understanding what makes each version play better. AlphaZero has been used to help prototype new variants of chess [68], under a number of atomic rule alterations proposed by grandmaster Vladimir Kramnik, a former chess world champion.

Chess as a testing ground for interpretability A number of approaches have been proposed recently that aim to explain the play of chess engines in human-understandable terms. Chess explanations patterns that are organized in a tree based on their complexity have been introduced in [69] for an educational chess system and further considered in [70]. Focused feature saliency [71] that balances specificity and relevance has been applied to chess-playing agents to produce saliency maps that highlight the pieces of highest importance for playing the selected move. The authors performed a user study involving chess players that have an ELO rating between 1600 and 2000 These players were randomly shown chess tactics puzzles with saliency maps derived via different methods, and the authors report a significant difference in accuracy and time invested depending on the method. It is worth noting that interest in saliency maps goes beyond the understanding of machines and that human attention over the board during chess games has been analysed to predict saliency with respect to chess players' pattern recognition and line calculation [72]. Even AI systems themselves have been used to model human behavior in chess, and better capture the playing style of different players [73] and players of different strength [74]. In [75], the authors show activation maps for a deep neural network trained to play the Crazyhouse chess variant at a superhuman level.

Natural language processing has been used for automatically generating move-by-move commentary of chess games based on social media forum data [76]. Another recent application of NLP focused instead on training the evaluation function based on the sentiment of free-text chess comments [77]. A question answering dataset for chess has been proposed in [78], as a benchmark for improving specific types of deep learning approaches, requiring the models to demonstrate knowledge of basic positional concepts on the board. In terms of conceptual chess understanding, DecodeChess [79] is maybe the most well-known example of an application of using chess concepts to provide human-understandable explanations of critical factors in each position. Concepts are suggested to be relevant, in this context, only if they provably affect the course of the game, based on an extensive search tree generated by a chess engine. In our work, we have focused on understanding the AlphaZero neural network that produces the value assessment and the candidate moves, rather than the computations generated via Monte Carlo tree search (MCTS). In doing so, we are trying to capture the "intuitive" aspect of chess play, rather than determine the truth in the position. When it comes to understanding the deep calculations performed by AlphaZero via MCTS, it would be interesting to consider an approach like the one in DecodeChess in the future, although this falls outside of the scope of our current study.

3 AlphaZero: Network structure and training

AlphaZero [5] comprises of two components, a deep neural network that computes a policy and value estimate from a state, and Monte Carlo tree search (MCTS) that uses the neural network to repeatedly evaluate states and update its action selection rule. A 'state' is a position and possibly a history of preceding positions, along with ancillary information such as castling rights, and is represented as a real-valued vector $\mathbf{z}^0 \in \mathbb{R}^{d_0}$. The purpose of superscript zero is to indicate that $\mathbf{z}^0$ is an input to the network, or the network's representation after layer *zero*. The neural network

$$\mathbf{p}, v = f_{\boldsymbol{\theta}}(\mathbf{z}^0) \tag{1}$$

predicts two quantities that are learned from training data games: It predicts the expected outcome of the game v from the current position, as well as a probability distribution $\mathbf{p}$ on the next move. Both are used in MCTS, and are referred to as the 'value head' and the 'policy head' in the AlphaZero network in Figure 1.

Starting with a neural network with randomly initialized parameters $\boldsymbol{\theta}$, the AlphaZero network is trained from data that is generated as the system repeatedly plays against itself. A buffered queue of self-play games is generated, which serves as training data for the neural network. Because self-play involves search with MCTS, the games are of slightly better quality than those in the training data buffer. Better self-play games are added to the queue while earlier self-play games are dropped. Concurrently, gradient descent is used to minimize the difference between the network's current predictions and a position's played move and game outcome, with positions taken from the self-play game queue.

3.1 AlphaZero neural network

This report is concerned with the evolution of AlphaZero's network; how chess knowledge is progressively acquired and represented. Figure 1 illustrates the AlphaZero network.

The network takes input $\mathbf{z}^0 \in \mathbb{R}^{d_0}$. In Figure 1, the input is $\mathbf{z}^0 \in \mathbb{R}^{8\times 8\times (14h+7)}$ for a history length of h plies. If $h = 1$ and only the current position is represented, $\mathbf{z}^0 \in \mathbb{R}^{8\times 8\times 21}$. The first twelve 8×8 channels in $\mathbf{z}^0$ are binary, encoding the positions of the playing side and opposing side's king, queen(s), rooks, bishops, knights and pawns respectively. It is followed by 8×8 binary channels representing the number of repetitions (for three-fold repetition draws), the side to play, and four binary channels for whether the player and opponent can still castle king and queenside. Finally, the last two channels are an irreversible move counter (for 50 move rule) and total move counter, both scaled down. The input representation is always oriented toward the playing side, so that the board position with black to play is first flipped horizontally and vertically before being represented in the stack of 8×8 channels $\mathbf{z}^0$. Even though the state is fully captured with $h = 1$ when only the current position is encoded, there is a marginal empirical increase in performance when a few preceding positions are also incorporated into $\mathbf{z}^0$, and $\mathbf{z}^0 \in \mathbb{R}^{8\times 8\times 119}$ if the board positions of the last eight plies are stacked. Unless otherwise stated, $h = 8$ is used in this report, following [5].

If we have a large set of inputs we denote them by $\{\mathbf{z}_n^0\}_{n=1}^N$. Typically, N is few million, and the inputs would be, for example, all positions from a large collection of grandmaster games.

3.1.1 Layers

The network in Figure 1 has a residual neural network (ResNet) backbone [80], and every ResNet block will form a layer indexed by $l = 1, \ldots, L$. Each ResNet block contains internal layers, and in this paper we index layers at the points where the skip-connections meet. We denote the activations at layer l with $\mathbf{z}^l \in \mathbb{R}^{d_l}$, with $\mathbf{z}^0$ being the input. In the AlphaZero network, as illustrated in Figure 1, $\mathbf{z}^l \in \mathbb{R}^{8\times 8\times 256}$ for each $l = 1, \ldots, 20$. There are therefore 16384 activations at the end of each layer, and we will use notation z_i^l to refer to activation i in layer l's activations $\mathbf{z}^l$.

The network progressively transforms input $\mathbf{z}^0$ to $\mathbf{z}^1$, then $\mathbf{z}^2$, and so on through a series of residual blocks and final policy/value heads, as shown in Figure 1. The activations of layer l is given by the function $\mathbf{z}^l = f_{\boldsymbol{\theta}}^l(\mathbf{z}^{l-1})$, and hence $f_{\boldsymbol{\theta}}^l : \mathbb{R}^{d_{l-1}} \to \mathbb{R}^{d_l}$, where d_l is dimensionality of layer l. We are going to omit the dependence of the layer on its parameters where it is clear from the context. For layers $l \geq 2$ the ResNet backbone in Figure 1 has the form

$$\mathbf{z}^l = f^l(\mathbf{z}^{l-1}) = \mathrm{ReLU}(\mathbf{z}^{l-1} + g^l(\mathbf{z}^{l-1})) \; , \tag{2}$$

which directly copies activations $\mathbf{z}^{l-1}$, adds an additional nonlinear function $g^l(\mathbf{z}^{l-1})$ composed of two more convolution layers to it, and clips the result to be nonnegative through a rectified nonlinar unit (ReLU). For stability, activations $\mathbf{z}^l$ are additionally clipped to a maximum value of 15. The form of Equation 2 means that z_i^l is equal to a clipped version of z_i^{l-1} and information from layers of additional local convolutions on $\mathbf{z}^{l-1}$. We revisit this link in Section 7.2 where we show that z_i^l and z_i^{l-1} would typically be activated for similar input patterns.

We introduce additional notation. Considering parts of the network in Figure 1, the neural network function between layers k and l with $k \leq l$ takes $\mathbf{z}^{k-1}$ as input and maps it to outputs $\mathbf{z}^l$. That is $f^{k:l}(\mathbf{z}^{k-1}) = f^l \circ \cdots \circ f^{k+1} \circ f^k(\mathbf{z}^{k-1})$, where $\circ$ denotes function composition, and therefore

$$\mathbf{z}^l = f^{1:l}(\mathbf{z}^0) = f^l \circ \cdots \circ f^2 \circ f^1(\mathbf{z}^0) \; . \tag{3}$$

Layers 1 to 20 in Figure 1 form the 'torso' of the network. We restrict our analysis in this paper to the layers in the torso. Two 'heads' complete the neural network by performing a computation on $\mathbf{z}^{20}$, the activations of the last layer in the torso. The 'value head' computes v in (1), while the 'policy head' computes $\mathbf{p}$, a distribution over all moves. The policy head, before flattening, produces a $8 \times 8 \times 73$ tensor. For every square, it encodes 73 possible moves to a next square: 7 horizontally left and right; 7 vertically up and down; 7 diagonal moves north west, north east, south west and south east; 8 knight moves; 3 promotion options to ♗, ♘, ♖ (a ♕ is default when a pawn reaches the eight rank) for the three single-square forward moves.

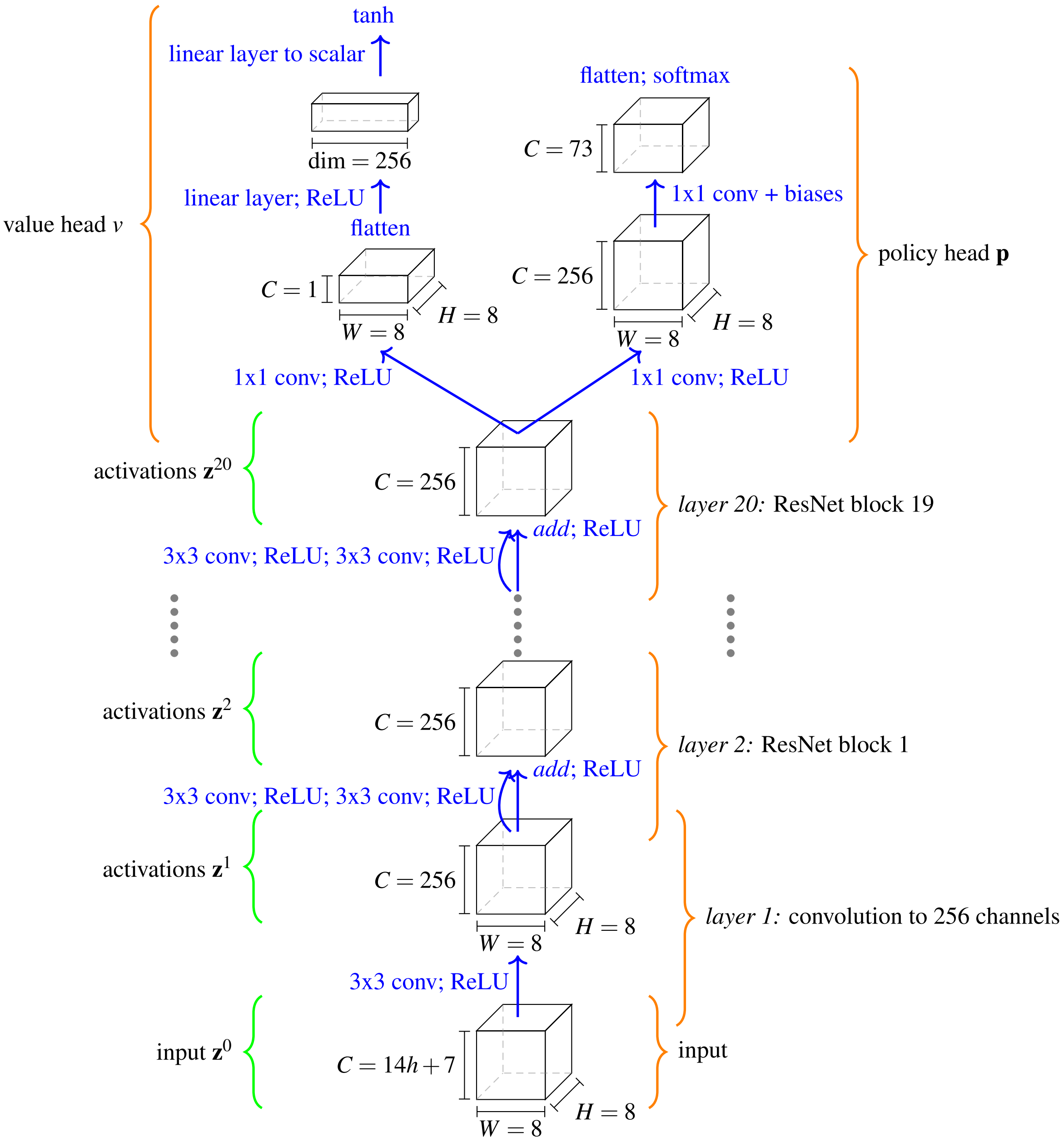


Figure 1. The AlphaZero network. Each 3×3 convolution indicates the application of 256 filters of kernel size 3×3 with stride 1. A ResNet block contains two rectified batch-normalized convolutional layers with a skip connection. In the input $\mathbf{z}^0$, a history length of $h = 8$ plies is used, encoding the current board position and those of the seven preceding plies. The input is a $8 \times 8 \times 119$-dimensional tensor.

3.2 AlphaZero training iterations

Our experimental setup updates the parameters $\boldsymbol{\theta}$ of the AlphaZero network over 1,000,000 gradient descent training steps. A million steps is an arbitrary training time slightly longer than that of AlphaZero in [5]. We will use t to index the training step, and $\boldsymbol{\theta}_t$ the network parameters after gradient descent step t.

The network is trained through positions with their associated MCTS move probability vectors that are sampled from self-play buffer containing the previous 1 million positions. At most 30 positions are sampled from a game on average, as positions on subsequent moves are strongly correlated, and including all of them may lead to increased overfitting. Stochastic gradient descent steps are taken with a batch size of 4096 in training. A synchronous decaying optimizer is used with initial learning rate 0.2, which is multiplied by 0.1 after 100k, 300k, 500k and 700k iterations.

After every 1000 training steps, the networks that are used to generate self-play games are refreshed, so that MCTS search uses a newer network. We refer to networks and their parameters $\boldsymbol{\theta}_t$ that are saved to disk at these points as 'checkpoints'. Self-play moves are executed upon reaching 800 MCTS simulations. Diversity in self-play is increased in two ways: through stochastic move sampling and through adding noise to the prior. The first thirty plies are sampled according to the softmax probability of the visit counts, and only after the thirtieth move are the moves with most visits in the MCTS simulations played deterministically. To further increase diversity in the self-play games, Dirichlet(0.3) noise is added to 25% of all priors $\mathbf{p}$ from Equation (1) and renormalized. Of all self-play games, 20% are played out until the end, whereas in the remaining 80%, an early termination condition is introduced where a game is resigned if the value gives an expected score of 5% or less. The maximum game length is capped at 512 plies.

4 Encoding of human conceptual knowledge

In this section we use a sparse linear probing methodology to determine the extent to which the AlphaZero network represents a wide range of human chess concepts. We describe our notion of concepts in Section 4.1, describe the probing methodolgy in Section 4.2 and present our results in Sections 4.3 and 4.4. In Section 4.5 we discuss limitations of our approach and future avenues of research suggested by these limitations. Finally, in Section 4.6 we investigate how concepts predict AlphaZero's value function.

4.1 Concepts

We adopt a simple definition of concepts as user-defined functions on network input $\mathbf{z}^0$. Each concept is a mapping of the input space onto the real line,

$$c : \mathbb{R}^{d_0} \to \mathbb{R} \ . \tag{4}$$

A concept could be any function of the position $\mathbf{z}^0$. A simple example concept could detect whether the playing side has a bishop pair, i.e. both a dark-squared and a light-squared bishop,

$$c(\mathbf{z}^0) = \begin{cases} 1 & \text{if } \mathbf{z}^0 \text{ contains a ♗-pair for the playing side} \\ 0 & \text{otherwise} \end{cases} \ . \tag{5}$$

Most concepts are more intricate than merely looking for bishop pairs, and can take a range of integer values (for instance difference in number of pawns) or continuous values (such as total score as measured by the Stockfish 8 chess engine). An example of a more intricate concept is mobility, where a chess engine designer can write a function that gives a score for how mobile pieces are in $\mathbf{z}^0$ for the playing side compared to that of the opposing side. What concepts have in common is that they are pre-specified functions which encapsulate a particular piece of domain-specific knowledge.

Concepts from Stockfish's evaluation function Stockfish's position evaluation function is comprised of sub-functions that give a score for different features of $\mathbf{z}^0$. We use the publicly exposed sub-functions from Stockfish 8's evaluation function as concepts. They are enumerated and explained in Table 2 in Appendix A. The use of Stockfish 8 is intentional, as it allows insights from this report to refer to observations in [81]. The root concepts in Table 2 are for `material`, `imbalance`, `pawns`, `knights`, `bishops`, `rooks`, `queens`, `mobility`, `king_safety`, `threats`, `passed_pawns`, `space` and `total`. The root concepts are further enumerated according to whether it is White or Black to play, and is further enumerated by the phase of the game, so that a concept like `threats_w_mg` would quantify white's threats during the middle game. Many Stockfish concepts are computed for the current player as well as the opponent: a given concept c (for instance `threats`) will have values for 'white', 'black' and 'total'. Assuming we are in the middle game (`mg`) with White (`w`) to play, the total (`t`) value is simply

$$c_{\texttt{threats\_t\_mg}}(\mathbf{z}^0) = c_{\texttt{threats\_w\_mg}}(\mathbf{z}^0) - c_{\texttt{threats\_b\_mg}}(\mathbf{z}^0) \ .$$

We focus on the total values, adjusted for the phase of the game, as these are the highest-level versions of each concept, and the ones that are used in Stockfish's position evaluation.

Additional custom concepts In addition to the concepts from Stockfish's public API, we implemented 116 custom concepts, which are enumerated in Tables 3 and 4 in Appendix A. These concepts encapsulate more specific lower-level features, such as the existence of forks, pins, or contested files, as well as a range of features regarding pawn structure.

Concepts and activations dataset We randomly selected 10^5 games from the full ChessBase archive and computed concept values and AlphaZero activations for every position in this set. This set was then deduplicated by removing any duplicate positions using the position's Forsyth-Edwards Notation (FEN) string. We then randomly sampled training, validation, and test sets from the deduplicated data. Continuous-valued concepts used training sets of 10^5 unique positions (not games), with validation and test sets consisting of a further 3×10^4 unique positions each. Binary-valued concepts were balanced to give equal numbers of positive and negative examples, which restricted the data available for training as some concepts only occur rarely. The minimum training dataset size for any binary concept was 50,363, and the maximum size was 10^5.

4.2 Probing concept learning with sparse linear regression

To detect the emergence of the human concepts of Tables 2, 3 and 4 within the AlphaZero network, we employ a simple probing methodology related to concept activation vectors [8]. We train a sparse regression model from activations $\mathbf{z}^{l_t}$ at a given layer l and training step t to a human concept j, using a linear predictor for continuous concepts and a logistic predictor for binary ones. More precisely, for each layer l, training step t and concept $c_j(\mathbf{z}^0)$ we train a parameterised regression function

$$g^j_{lt}(\mathbf{z}^{l_t}) = \mathbf{w}^T_{jlt}\mathbf{z}^{l_t} + b_{jlt} \quad \text{(continuous concepts)} \tag{6}$$

$$g^j_{lt}(\mathbf{z}^{l_t}) = \sigma(\mathbf{w}^T_{jlt}\mathbf{z}^{l_t} + b_{jlt}) \quad \text{(binary concepts)} \tag{7}$$

from the output of the l^{th} layer of the t^{th} network in training, $\mathbf{z}^{l_t} = f^{1:l}_{\boldsymbol{\theta}_t}(\mathbf{z}^0)$, to approximate the concept value $c_j(\mathbf{z}^0)$. Function σ is the sigmoid function $\sigma(x) = 1/(1+e^{-x})$. As an example, concept j might be a `mobility_t_mg` score for the playing side in position $\mathbf{z}^0$, and consequently the trained function g^j_{lt} would indicate how linearly predictable `mobility_t_mg` is from the representation $\mathbf{z}^l$ after layer l. *The accuracy with* $g^j_{lt}(\mathbf{z}^{l_t})$ *predicts the concept value* $c_j(\mathbf{z}^0)$ *on a held-out test set indicates how much information the activations at layer l carry regarding that concept at a given point in training.*

Learning the probes We define the data matrix of activations $\mathbf{Z}^l_t \in \mathbb{R}^{d_l \times N}$ to contain layer l's activations $\mathbf{z}^l_n = f^{1:l}_{\boldsymbol{\theta}}(\mathbf{z}^0_n)$ for training set inputs $\mathbf{z}^0_1, \dots \mathbf{z}^0_N$ at training step t. Furthermore, let $\mathbf{c}_j \in \mathbb{R}^N$ be a vector that contains the value of concept j for each of the inputs of the training set, hence each $c_j(\mathbf{z}^0_n)$. Vector $\mathbf{c}_j$ has no superscript l or subscript t; the concepts are purely functions of the input and is independent of the neural network layer or training step.

The parameters $\mathbf{w}_{jlt}$ and b_{jlt} for Equation 6 are found by minimizing the empirical mean squared error between $g^j_{lt}(\mathbf{z}^{l_t})$ and $c_j(\mathbf{z}^0)$. From Figure 1, each block's activations are $8 \times 8 \times 256$ dimensional, and hence $\mathbf{z}^l \in \mathbb{R}^{16{,}384}$. Because of the high dimensionality of $\mathbf{z}^l$, we regularize $\mathbf{w}_{jlt}$ to avoid any overfitting, as well as testing on a held-out test set. Hence

$$\mathbf{w}_{jlt}, b_{jlt} = \min_{\mathbf{w},b} \frac{1}{N}\left\|\mathbf{w}^T\mathbf{Z}^l_t + b\mathbf{1} - \mathbf{c}_j\right\|_2^2 + \lambda\|\mathbf{w}\|_1 + \lambda|b| \quad \text{(continuous concepts),} \tag{8}$$

$$\mathbf{w}_{jlt}, b_{jlt} = \min_{\mathbf{w},b} \frac{1}{N}\left\|\sigma(\mathbf{w}^T\mathbf{Z}^l_t + b\mathbf{1}) - \mathbf{c}_j\right\|_2^2 + \lambda\|\mathbf{w}\|_1 + \lambda|b| \quad \text{(binary concepts),} \tag{9}$$

where $\|\cdot\|_p$ denotes the L_p norm, $\sigma(\cdot)$ is applied elementwise, and $\mathbf{1}$ is simply the all-one vector in $\mathbb{R}^N$. The parameter λ was determined individually for each regression using cross-validation on a held-out validation set, with $\lambda \in \{0.003, 0.006, 0.01\}$ for continuous concepts and $\lambda \in \{0.01, 0.1\}$ for binary-valued concepts.

Controls: regression from inputs and random concepts We provide two controls for comparison: regression from network input, and random concept regression. Regression from network input trains the probe from $\mathbf{z}^0$, and random concept regression uses a normally-distributed random target $c_{\text{random}}(\mathbf{z}^0) \sim \mathcal{N}(0,1)$. By comparing to the network input we can see what is being added by network computation, as even an untrained network substantially increases the dimensionality of the regression targets compared to the network input. The random concept control ensures that what we are learning relates to specific concepts, rather than simply separability of arbitrary points.

Information-theoretic concerns From a purely information-theoretic perspective no neural network can be said to be adding any information that is not present in the input; the data processing inequality $I(X,Z_1) \geq I(X;Z_2)$ for random variables X, Z_1, and Z_2 in a Markov chain $X \rightarrow Z_1 \rightarrow Z_2$ still applies. Even though f_θ is deterministic, it can still lose information if it is not bijective (consider a function mapping its entire range to zero as a simple example), so it is entirely possible that information relevant to a concept may be lost during network computation, although it can never be created. The data processing inequality therefore presents conceptual challenges to probing, as well as to our intuitive understanding of information. For example, the information content of an image is the same whether it is represented unaltered, with the pixels scrambled, or even encrypted (so long as the encryption function is bijective), but we find it much easier to understand the unaltered image. One way to understand the function of neural network layers is to see them not as creating information in the sense of Shannon information, but making that information available to a computationally-bounded agent (in our case, the probe network), as is suggested by [82].

4.3 Visualising concept learning with what-when-where plots

We visualise the acquisition of conceptual knowledge by illustrating *what* concept is learned *when* in training time *where* in the network. This is done by scoring concept regression over the layer (l) and temporal (t) axes as AlphaZero is trained.

4.3.1 Scoring concept regression

The regression models trained in Section 4.2 are probing for the existence of information on each individual concept at every layer and checkpoint[1]. This means that the accuracy of these models can be used as a proxy to measure the presence or absence of (linearly-decodable) information. By comparing accuracy scores across layers and checkpoints we aim to understand what concepts are encoded, when in training they emerge, and where in the network they are strongly encoded. Because the training process for each sparse regression model is identical, we know that any changes in accuracy between different layers and/or checkpoints must be due to changes in AlphaZero's internal representations. Using a fixed dataset of human play rather than games from AlphaZero self-play means that we also avoid changes in the distribution of positions due to different policies affecting either the distribution of concepts or activations.

To generate what-when-where plots, we require an accuracy measure. For continuous concepts we use the coefficient of determination r^2, and for binary concepts we use the increase in accuracy over random label prediction. For concept j, checkpoint t and layer l, the coefficient of determination r^2_{jlt} using the test set of positions $\mathbf{z}^0_1, \ldots, \mathbf{z}^0_M$ is given by the equation:

$$r^2_{jlt} = 1 - \frac{\sum_m (c_j(\mathbf{z}^0_m) - g^j_{lt}(\mathbf{z}^{l_t}_m))^2}{\sum_m (c_j(\mathbf{z}^0_m) - \overline{c_j})^2} \tag{10}$$

where $\mathbf{z}^{l_t}_m$ is the activation vector of input $\mathbf{z}^0_m$ at layer l at training iteration t for position m (see Equation 3), $g^j_{lt}(\mathbf{z}^{l_t}_m)$ is the best (sparse) linear predictor (as found by Equation 8), and $\overline{c_j}$ is the mean concept value

$$\overline{c_j} = \frac{1}{M} \sum_m c_j(\mathbf{z}^0_m). \tag{11}$$

A score of $r^2 = 1$ means that a concept is perfectly linearly predictable from a layer's activations, whereas a predictor that always returned $\overline{c_j}$ would have $r^2 = 0$. For binary concepts we report the accuracy increase over random guessing (recall that for all binary concepts we used balanced datasets for training, validation, and testing). For binary concepts the accuracy score s_{jlt} is given by

$$s_{jlt} = 2 \sum_m \delta(c_j(z^0_m) - \tilde{g}^j_{lt}(z^{l_t}_m)) - 1, \tag{12}$$

where $\tilde{g}^j_{lt}(z^{l_t}_m)$ is the maximum-probability class output by the logistic regression g. This is normalised such that predicting either 0 or 1 for all m gives an accuracy $s = 0$, whereas predicting the correct class every time gives $s = 1$. Normalising in this way allows for direct comparison with continuous-valued concepts.

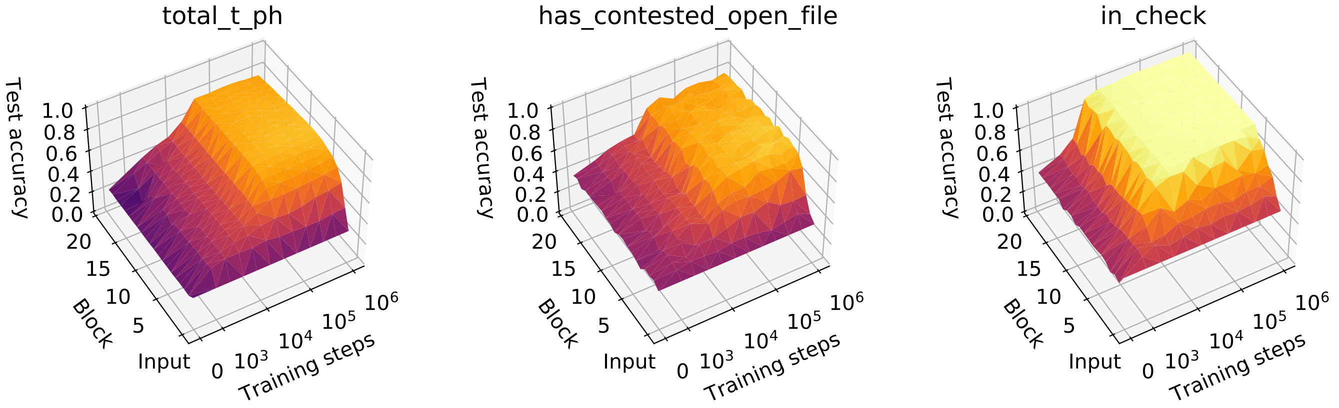
For every concept j, there is a grid of r^2_{jlt} or s_{jlt} values (depending on whether the concept is binary), with one axis representing the neural network layer l and the other axis representing the training step t. Plotting this grid gives a what-when-where plot, allowing us to visualise changes in concept regression accuracy with network depth, as well as the evolution of concepts over time. We show a selection of what-when-where plots in Figure 2, and the full set of concepts are visualised in Appendix B. Results for a second AlphaZero training run are visualised in Appendix E, demonstrating stability of these results under network retraining.

4.3.2 The evolution of human concepts in AlphaZero

At a high level, the data shown in Figure 2 and Appendix E show that information related to human-defined concepts of multiple levels of complexity is being learned over the course of training, and that many of these concepts are computed over the course of multiple blocks. Many surprisingly complex concepts can be regressed with high accuracy, and this accuracy increases substantially over the course of training. Many concepts begin to increase in accuracy around 32,000 steps, which appears to be a period of rapid development in both AlphaZero's representations and opening play (as we demonstrate in later sections).

---

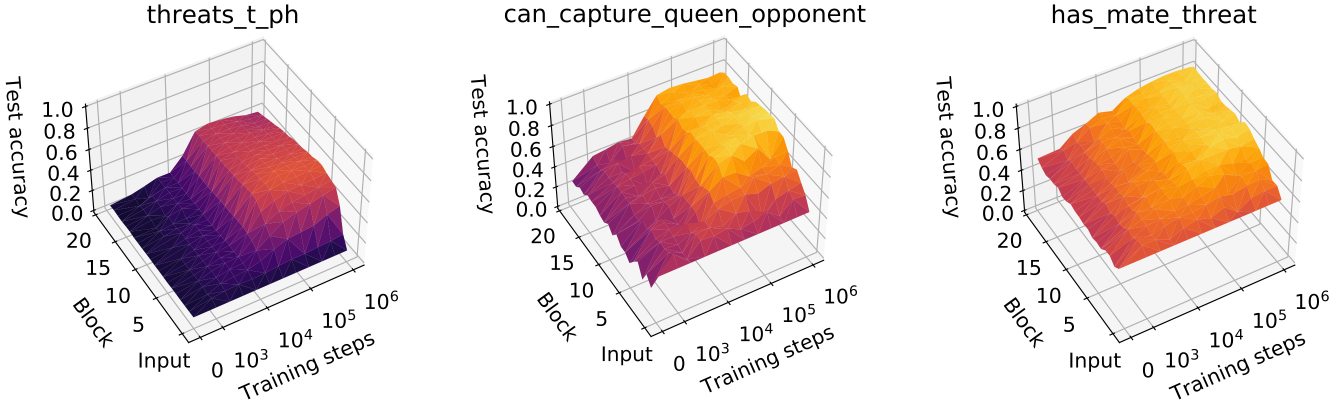
[1] A 'checkpoint' refers to a network and its parameters $\boldsymbol{\theta}_t$ as saved to disk at training step t; see Section 3.2.



(a) Stockfish 8's total score

(b) A contested open file is occupied by rooks and/or queens of opposite colours

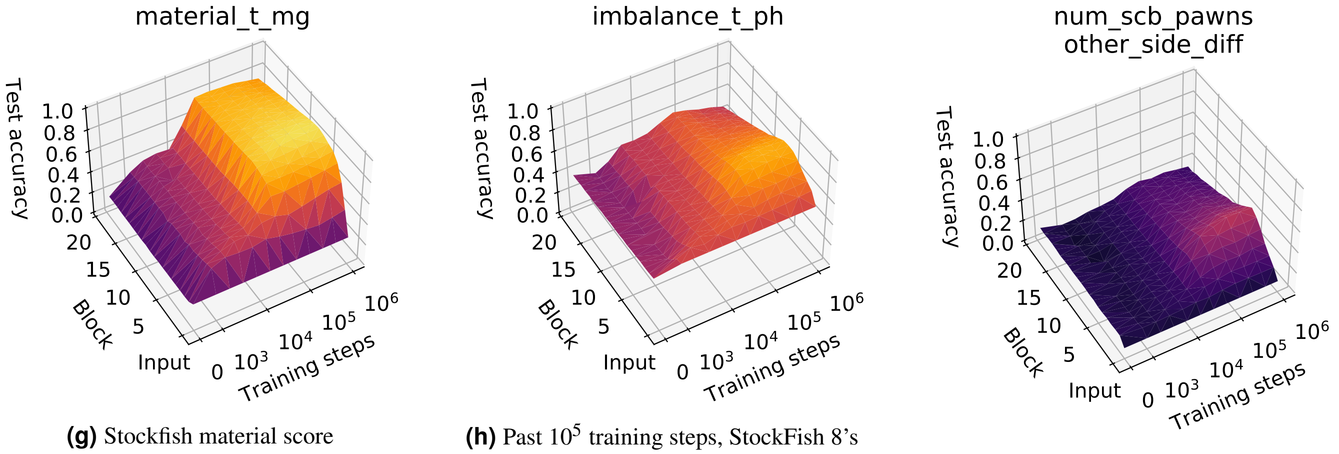
(c) Is the playing side in check?



(d) Stockfish 8's evaluation of threats.

(e) Can the playing side capture their opponent's queen?

(f) Could the opposing side checkmate the playing side in one move?



(g) Stockfish material score

(h) Past 10^5 training steps, StockFish 8's material imbalance score becomes *less* predictable from AlphaZero's later layers.

(i) When each side has one bishop only, the difference in number of pawns on squares of the *opposite* colour than the opponent's bishop.

Figure 2. What-when-where plots for a selection of Stockfish 8 and custom concepts. Following Figure 1, we count a ResNet 'block' as a layer.

Stockfish 8 score We find a surprising level of accuracy on Stockfish 8 total score `total_t_ph`, with $r^2 > 0.75$ after 10 layers at 64,000 training steps and beyond; see Figure 2a. We explore Stockfish 8 score further in Section 4.4 below. Regression accuracy for Stockfish 8 score is very low early in training, reflecting the complexity of this concept, and only begins to increase substantially after 16,000 steps before plateauing at 128,000 training steps. This pattern occurs repeatedly across a wide range of concepts, and we also observe a rapid change in the policy prior during this window, which we investigate further in Section 6.

Threat-related concepts The progressive increase in regression accuracy for check, queen capture and threats over the initial layers of the trained network indicates that computation of potential moves takes place over the course of early network layers, an observation which is corroborated by studying network activations directly in Section 7. Relatively low regression accuracy for threats may be due to a difference in evaluating the value of different threats, or due to highly-distributed representations of threats being penalised by sparsity. High accuracy for `can_capture_queen_opponent` and `has_mate_threat` in Figures 2e and 2f suggest that distributed representations may be at least partly responsible, as we can accurately predict more specific threat-related concepts.

Accuracy for `has_mate_threat` rises throughout the network, suggesting that further processing of threats occurs in later layers, and that AlphaZero is predicting the consequences of its opponent's potential moves. The ability to predict `has_mate_threat` from AlphaZero's activations indicates that AlphaZero is not simply modelling its potential moves, but also its opponent's potential moves and their consequences during position evaluation. This is interesting in light of AlphaZero's training loss $\mathscr{L}$:

$$\mathscr{L} = -\boldsymbol{\pi}^T \log \mathbf{p} + \text{value and regularisation terms} \tag{13}$$

for policy head output $\mathbf{p}$ and returned MCTS probability vector $\boldsymbol{\pi}$ [5]. This loss indirectly pushes the policy network to predict the consequences of its actions. Our analysis suggests that training using Equation 13 has led to at least a single-move lookahead being implemented in the AlphaZero network, rather than this being deferred to MCTS rollouts.

Material Earlier analysis of AlphaZero's play suggested that [81] that AlphaZero views material imbalance differently from Stockfish 8. The what-when-where plot of Figure 2h gives *empirical* evidence that this is the case at the representational level; the test r^2 drops for later layers after 10^5 training steps. Note that Stockfish's material evaluation is not simply a linear combination of piece counts, but also includes position-specific evaluation terms. We also include a simpler material concept measuring only piece count in Appendix B. This simpler material evaluation can be trivially regressed for either player at all network layers (including the input) and at all points during training. This is unsurprising because piece counts are easily available from the input planes. Material difference only becomes predictable after training, and increases with network depth, even for this simpler concept of material. This indicates that our level of sparsity is such that the regression probe must use only a limited part of the input or internal activations, and that material difference is relatively compactly encoded in later layers of the network, i.e. the regression probe is making use of a specific evaluation of material, rather than simply using representations of piece locations.

Drop in linearly-available information What-when-where plots for threats (`threats_t_ph`; Figure 2d), piece count difference (`material_t_mg`; Figure 2g) and imbalance (`imbalance_t_ph`; Figure 2h) reveal a surprising pattern: for well-trained networks, regression accuracy on these concepts peaks at an early layer before dropping dramatically in later layers. The drop in accuracy indicates that information on these concepts is either being used in further computations and then discarded, or encoded in a more complex nonlinear way, indicating that these concepts are likely to be used as intermediate computations which are useful for later layers but do not feed directly into policy or value calculations. This suggests that the common practice of visualising only last-layer activations (for instance via t-SNE) may miss the presence of important information in complex domains, and underlines the importance of considering intermediate computations for understanding neural networks on complex domains.

Some concepts cannot be regressed, or are trivial to predict Not all concepts can be regressed from the AlphaZero network, however. Some, such as the number of same colour bishop pawns, have low regression accuracy even at their peak, and a few have constant near-zero accuracy (see Appendix B, Figures 14 to 25 for more examples). Concepts with low regression accuracy are disproportionately associated with pawns, and these may be more heavily penalised by sparsity than other concepts, as there are typically more pawns on the board than other pieces. Other concepts (shown in Supplementary Information) have constant high regression accuracy, indicating either a low variety in the number of possible values for the concept, or that they can be trivially regressed from the input.

Implications Under the assumption that human concepts are represented when they are linearly predictable from a layer, these results suggest that AlphaZero is developing representations which are closely related to a number of human concepts over

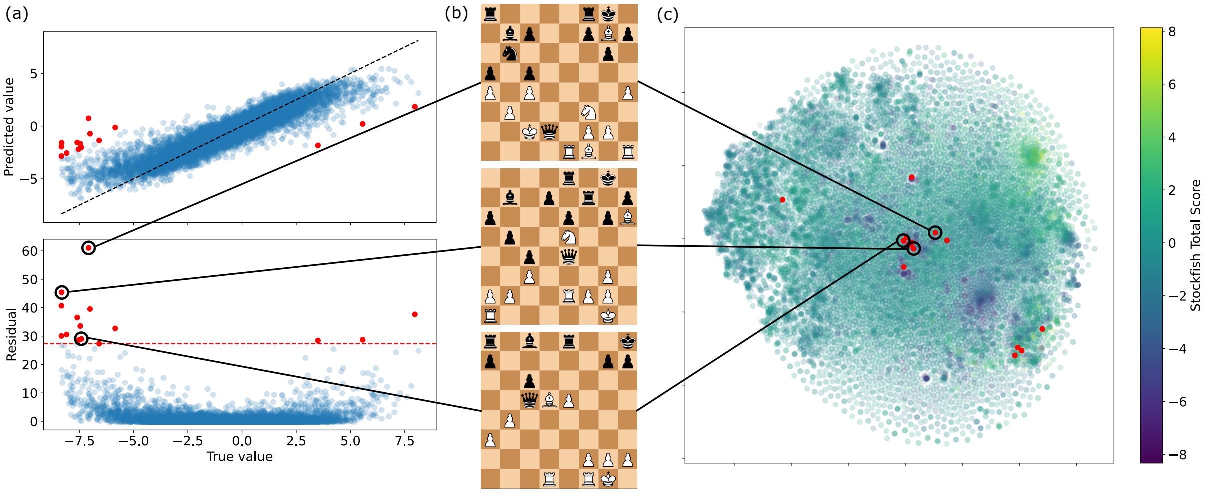


Figure 3. Evidence of patterns in regression residuals. **(a)** Upper panel: true and predicted values for Stockfish 8 score `total_t_ph`, regressed from block 10 at 1,000,000 training steps, evaluated on the test set. Dashed line indicates perfect fit. Red markers indicate predictions in the 99.95th percentile of residuals. Lower panel: True value and residual for Stockfish 8 score, as in top panel. Red dashed line indicates 99.95th percentile cutoff. **(b)** High-residual positions, corresponding to the data points marked in (a) and (c). Note that Black's queen can be taken in all positions. This is true for all 12 high-residual pieces where the regressed score is more favourable to White than the Stockfish score. **(c)** t-SNE of block activations at 1,000,000 steps. Red markers indicate high-residual positions shown in (a), showing substantial clustering.

the course of training, including high-level evaluation of position, potential moves and consequences, and specific positional features. The fact that accurate regression can be attained with only a small fraction of neurons at each layer is striking, and indicates that channels learn to specialise in performing distinct functions, and that these functions align to some degree with human concepts. When the network is well-trained many of these these concepts are computed progressively over multiple layers, as shown by the increase in accuracy as network depth increases. Interestingly, some of these concepts cannot be as accurately predicted from later layers (as indicated by a decrease in test r^2), showing that information is either lost in network computations or becomes non-linearly encoded and that not all of the information relevant to network computation is linearly available in the last layer.

4.4 Systematic semantic differences in Stockfish score outliers

There are instances when the AlphaZero network's value head and Stockfish 8's evaluation function take on fundamentally different roles, purely because of the way *search* differs between the engines. AlphaZero's MCTS runs simulations up to a fixed ply depth. The AlphaZero network therefore has to encode some minimal form of *look-ahead*; we've already seen evidence of this in Figure 2f which indicates that the network encodes whether the opposing side can checkmate the playing side in one move. This minimal form of look-ahead would also be required when an exchange sequence is simulated only halfway before the maximum MCTS simulation depth is reached. Stockfish, on the other hand, dynamically increases evaluation depth during exchange sequences. There is information that comes from looking ahead that Stockfish's evaluation function *doesn't have to encode*, simply because there is other code that ensures such information is incorporated in the final evaluation.

Remarkably, differences between the two styles of chess engines surface in the positions that are outliers in Stockfish 8 score `total_t_ph` concept regression. The prediction error outliers have semantic meaning, which we explore in this section.

Why can we learn from prediction errors? In the previous section (4.3), we showed that many complex concepts can be predicted with surprisingly high accuracy by probing the AlphaZero network. However, in almost all cases concept regression was not perfect. In this section we investigate whether we can learn anything from these prediction errors. It may seem surprising to expect this to be possible, as prediction errors could simply be noise with no interesting structure, the relevant data may not be present in AlphaZero's activations, or sparsity constraints may artificially limit regression accuracy (as we discuss in Section 4.5). In these cases we would expect to learn little or nothing from prediction errors. For concepts which are simple and objective (for example `has_contested_open_file`) one or more of these possibilities is very likely to be true. Where concepts are more complex or have an element of subjectivity, however (for example `total_t_ph`), there is another possibility: regression errors may point to a 'difference of opinion'. We alluded to the example of evaluating the Stockfish 8 score during an exchange sequences, where Stockfish can afford to differ from AlphaZero, simply because their search algorithms differ in fundamental ways. Prediction errors due to 'differences of opinion' should appear as consistent structure in the regression errors: inputs on which the network fails to predict well should have something in common.

Interpretable structure in Stockfish score regression residuals We investigated this possibility using the Stockfish score concept `total_t_ph`. For each position n in the test set we computed the Stockfish 8 score concept which we denote as $c_n = c_{\texttt{total\_t\_ph}}(\mathbf{z}_n^0)$ in this section. Following Equation 6, we additionally compute the predicted score $\hat{c}_n = g_{l_t}^{\texttt{total\_t\_ph}}(\mathbf{z}_n^{l_t})$ from the linear regression model trained on block $l = 10$ of the $t = 1,000,000$ steps checkpoint. The block and checkpoint were chosen because regression achieves high accuracy at this point. We then calculated the residuals

$$\varepsilon_n = (c_n - \hat{c}_n)^2. \tag{14}$$

These residuals are shown in the bottom panel of Figure 3a. We then visualised the positions corresponding to the residuals in the 99.95th percentile, representing the positions with the most extreme prediction errors. These outliers are marked in red in Figure 3a, and a selection are shown in Figure 3b. In all 12 outlier positions where the regressed score is more favourable to White than the Stockfish score, Black's queen can be taken, often without requiring an exchange: the 'difference of opinion' is maximal halfway through a sequence of moves exchanging queens. This suggests that the regression model is picking up on White's ability to take Black's queen, rendering these positions more favourable than the Stockfish score would suggest. Although this case is likely to be exceptional, and in many cases prediction errors are likely to indicate a simple failure of the regression model, this example shows that we can sometimes learn more from the regression probes trained when generating what-when-where plots. Furthermore, the outlier positions also cluster in activation space: Figure 3c shows a t-SNE projection [83] of AlphaZero activations from the same layer and checkpoint, coloured by Stockfish score. Outliers are again shown by red markers, and show a surprising level of clustering. Extreme positive and negative values of Stockfish score also cluster, and many of these clusters do not contain any outlier values, suggesting that the similarity in the outliers is due to representational similarity, rather than simply a common feature of low Stockfish score positions.

Although the degree of structure shown here is surprising, this example is not cherry-picked; it was the first concept/layer/checkpoint combination we tried, and the positions presented are simply those whose residuals are past a cutoff that we did not tune.

4.5 Challenges and limitations for concept probing

What's the right probing architecture? In this work we have used sparse linear probes because they are a simple to understand and low-capacity regression model. Keeping the capacity of the regression model low is important in order to ensure that the model captures structure in AlphaZero's representations, rather than learning complex relationships of its own, but we believe that better probing architectures are possible. Although sparsity is essential due to the high dimensionality of neural network activations, it can cause difficulties with spatially-represented concepts, which AlphaZero's architecture is naturally biased towards. For example, consider a channel representing the possibility of capturing the opponent's queen by a positive activation at the position of the queen and zero activations otherwise. There are locations on the board in which the queen is more likely to be captured than other locations, and therefore there are levels of sparsity that lead to positive regression weights only at the positions where the queen is most likely to be captured. In this (hypothetical) example, all of the information is there in the network, but a sparse linear regression model is unable to capture all of it and thus will have $r^2 < 1$. More sophisticated regression methods such as data-dependent sparsity using Gated Linear Networks [84], information-theoretic regularisation using the information bottleneck [85], minimum description length probing [86] or Bayesian probing [87] may provide ways to mitigate this problem.

How should we interpret complex or subjective concepts? Many concepts in our dataset have a subjective element as well as a more objective one. Some concepts, for instance `has_contested_open_file`, simply indicate the existence of a specific feature, whereas others such as `threats` combine both the presence of various threats as well as a subjective assessment of their value. Complex concepts like this can pose a challenge to interpreting regression results: when regression accuracy is low, is this because the components of the concept aren't present (i.e. the relevant threats aren't being computed by the model) or because our assessment of the value of those threats differs from that of the network? Our evidence in Sections 4.2 and 7.1 indicate that threats are being computed, suggesting that either our assessment of the value of different threats differs from AlphaZero's or much of the relevant information in the network remains distributed or nonlinearly encoded throughout the residual stack.

Complex concepts often incorporate a degree of judgement or subjectivity. In our experiments, we rely on the components of the public API of Stockfish 8, along with a number of independently implemented low-level features, to provide us with a notion of *ground truth*, which we then subsequently try to identify within the AlphaZero network This choice is in itself arbitrary, as there would be differences between different chess engines and their implementations, as well as across different versions of Stockfish. It should also be clear that the positional assessment of any individual chess grandmaster isn't equivalent to that of Stockfish's evaluation function.

When can we definitively say a concept is represented? There is no clear line between $r^2 = 0$ and $r^2 = 1$ at which we can say a concept is definitely present or absent. More sophisticated probing methodologies (as discussed above) could go some way to alleviating this problem by allowing better use of available data, which would either increase regression accuracy to a level where we are confident the concept is being represented or increase our confidence that the relevant concept is not present. The comparative approach introduced by what-when-where plots means that we are able to tell when linearly-decodable information is being generated by the network, and sharp jumps to high regression values intuitively seem (at least to us) to provide stronger evidence of a concept being computed by the network. Understanding prediction failures can also be useful, particularly in the case of complex composite concepts: if a concept is accurately predicted half of the time, and completely wrongly the other half, then we may be able to refine our concepts to better reflect network activations by understanding what the successes or failures have in common. We give an example of this in Section 4.4.

Spatial representations can also lead to $r^2 < 1$ for reasons unrelated to sparsity: consider the previous example of determining whether the opponent's queen can be captured. As we demonstrate in Section 7.1 the set of possible moves is developed over the course of multiple network layers. This is necessarily the case because convolutional blocks can capture only local information: a 3x3 convolutional kernel must take several layers to propagate a bishop's diagonal moves across a chessboard, for instance. Because of this gradual development, only potential captures where the capturing piece is near the queen can be regressed from the earliest layers, with longer-distance captures becoming predictable later in the network. This will lead to a steady increase in regression accuracy with network depth. We notice this pattern of gradual increase in a broad range of threat-related concepts.

When is a network 'really' representing or using a concept? High regression score is only a proxy of concept encoding. It is entirely possible that the network contains a concept that confounds with the human concept, rather than a representation of the concept itself. When we train a probe we cannot tell if we are getting a confounder or the concept itself. Our use of held-out validation and test sets ensure that any such confounders carry over accurately to novel positions, but cannot determine whether the network is 'really' computing a given concept (a philosophically tricky question in itself). Dealing with confounders is typically achieved by intervention, but the correct way to intervene is not obvious. We could intervene on the input to the network (as is done in [43]) or use instrumental variables (as is done in [88]), but even with such measure, many concepts are still entangled with one another. For example, consider a situation where a queen is in an absolute pin (it cannot be moved without putting the king in check). In this case, we cannot intervene to set the concept `can_capture_queen` to 0 without setting `in_check` to 1, in addition to altering a whole range of other concepts. Alternatively, we could intervene on the network's internals directly, altering the value of the activations from which we can regress `can_capture_queen` and seeing what other concepts change in later layers. This is not, strictly speaking, an intervention on the concept itself, but is rather an intervention on the parts of the network that we believe encode the relevant concept. As such it would whether the network is using the concept, rather than the effect of changing the external situation to alter the value of the concept. The challenge here is that such an intervention may push the network far off-distribution and result in incoherent downstream computation.

4.6 Relating human concepts to AlphaZero's value function

We have found that many human-defined concepts can be predicted from intermediate representations in AlphaZero when training has progressed sufficiently (Section 4.2), but this does not determine how these concepts relate to the outputs of the network. In this section we investigate the relation of concepts (piece count differences and high-level Stockfish concepts) to AlphaZero's value predictions. We use concept values to predict the output of AlphaZero's value function using a linear predictor. By analysing the regression coefficients we can investigate how AlphaZero's value function relates to human concepts over the course of training. A schematic of our approach is shown in Figure 4a, and full details are given below.

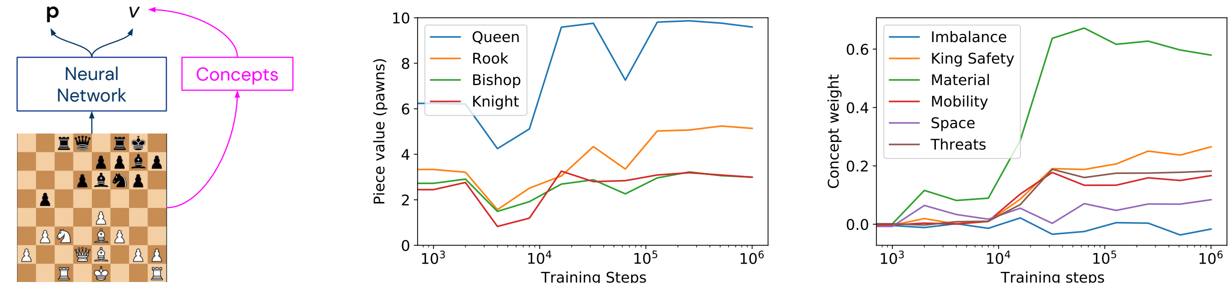
Equation 1 stated the AlphaZero neural network as $\mathbf{p}, v = f_{\boldsymbol{\theta}_t}(\mathbf{z}^0)$ at training step t. In this section we consider only the value head output, which we denote by $v_{\boldsymbol{\theta}_t}(\mathbf{z}^0)$. Given a vector of concept values $\mathbf{c}(\mathbf{z}^0) = [c_1(\mathbf{z}^0), c_2(\mathbf{z}^0), \ldots, c_J(\mathbf{z}^0)]$ as input vector, we train a generalized linear model to predict $v_{\boldsymbol{\theta}_t}(\mathbf{z}^0)$ from these concepts. The generalized linear model has weights $\mathbf{w}$ and bias b,

$$\hat{v}_{\mathbf{w},b}(\mathbf{z}^0) = \tanh\left(\mathbf{w}^T \mathbf{c}(\mathbf{z}^0) + b\right) . \tag{15}$$

The use of tanh nonlinearity is because $v_{\boldsymbol{\theta}_t}(\mathbf{z}^0) \in (-1, 1)$, and it also corresponds to the final non-linear function in the value head; see Figure 1. The weights $\mathbf{w}$ and biases b are trained using the L_1 loss

$$\mathbf{w}_t, b_t = \min_{\mathbf{w},b} \frac{1}{N} \sum_n \left| \hat{v}_{\mathbf{w},b}(\mathbf{z}_n^0) - v_{\boldsymbol{\theta}_t}(\mathbf{z}_n^0) \right| . \tag{16}$$

The weights $\mathbf{w}_t$ and biases b_t will differ between checkpoints t because AlphaZero's value network changes over the course of self-play, leading to different $v_{\boldsymbol{\theta}_t}$ outputs. We train on $N = 10^5$ training positions, deduplicated as in Section 4.2, before testing



(a) Value regression methodology: we train a generalized linear model on concepts to predict AlphaZero's value head for each neural network checkpoint.

(b) Piece value weights converge to values close to those predicted by conventional theory.

(c) Material predicts value early in training, with more subtle concepts such as mobility and king safety emerging later.

Figure 4. Value regression from human-defined concepts over time.

on 3×10^4 held-out test positions. We use the L_1 loss rather than L_2 in training as we found that the L_2 loss systematically underestimated piece weights.

Piece value Simple values for material are one of the first things a beginner chess player learns, and allow for basic assessment of a position. We began our investigation of the value function by using only piece count difference as the 'concept vector'. In this case, similar to [68], the vector $\mathbf{c}(\mathbf{z}^0) = [d_{♙}(\mathbf{z}^0), d_{♘}(\mathbf{z}^0), d_{♗}(\mathbf{z}^0), d_{♖}(\mathbf{z}^0), d_{♕}(\mathbf{z}^0)]$ contains the difference d in numbers of pawns, knights, bishops, rooks and queens between the current player and the opponent. For piece value regression we use only positions where at least one piece count differs between White and Black. The evolution of piece weights are shown in Figure 4b, showing that piece values converge towards commonly-accepted values after 128,000 training steps.

Higher-level concepts We furthermore explored the relative contributions of high-level Stockfish concepts imbalance, king safety, material, mobility, space and threats (see Table 2 for details) to predicting $v_{\boldsymbol{\theta}_t}(\mathbf{z}^0)$. We normalized the high-level Stockfish concepts by dividing by their respective standard deviations $c' = c/\text{std dev}$, and use the vector $\mathbf{c}(\mathbf{z}^0) = [c'_{\texttt{imbalance}}(\mathbf{z}^0), c'_{\texttt{king\_safety}}(\mathbf{z}^0), c'_{\texttt{material}}(\mathbf{z}^0), c'_{\texttt{mobility}}(\mathbf{z}^0), c'_{\texttt{space}}(\mathbf{z}^0), c'_{\texttt{threats}}(\mathbf{z}^0)]$ with the concepts' '`t_ph`' function calls[2] in the generalized linear model of (15). Figure 4c illustrates the progression of the weights $\mathbf{w}_t$ over training steps t. Material is the first concept to be learned, consistent with initial human learning, and begins to be used at 16,000 steps. More sophisticated concepts, such as king safety, threats, and mobility, are used at 32,000 steps and beyond. This coincides with the point at which many of these concepts begin to be accurately regressed from some layers of the neural network. The space concept has a comparatively low weight, and emerges late in training. The imbalance concept receives a small negative weight, and 'imbalance' makes a negligible contribution to linearly predicting $v_{\boldsymbol{\theta}_t}(\mathbf{z}^0)$. Interestingly, Figure 2h shows that regression accuracy for the imbalance concept declines in later layers at all points in training, suggesting that this information is not preserved in a linear form in the later layers of the network.

5 Progression through AlphaZero and human history

In this section we depart from the progression of human concepts in the AlphaZero network as the system trains, and compare AlphaZero training to the progression of human knowledge.

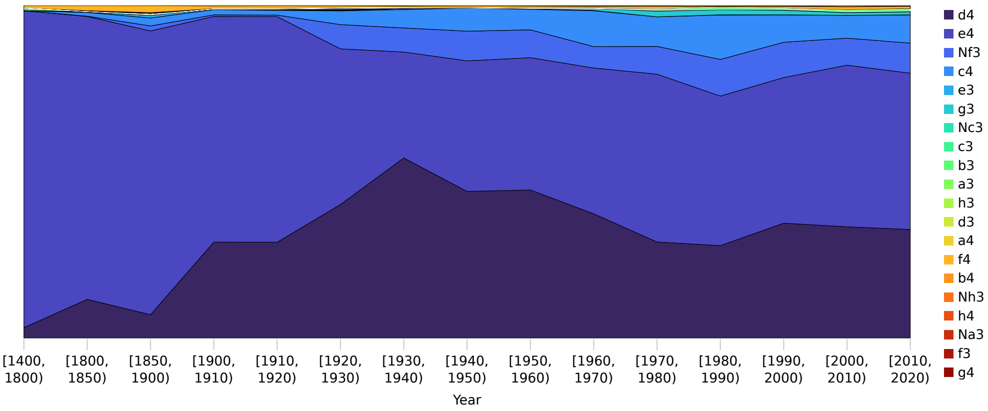
There is a marked difference between AlphaZero's progression of move preferences through its history of training steps, and what is known of the progression of human understanding of chess since the fifteenth century. AlphaZero starts with a uniform opening book, allowing it to explore all options equally, and largely narrows down plausible options over time. Recorded human games over the last five centuries point to an opposite pattern: an initial overwhelming preference for 1. e4, with an expansion of plausible options over time.

5.1 Five centuries of data

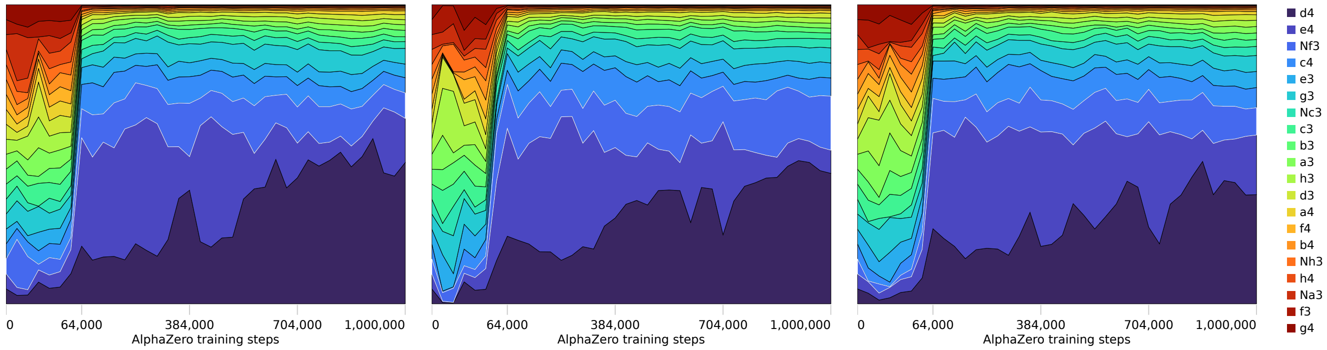
We base our analysis on a subsample of games from the ChessBase Mega Database [89]. Much of the early history of chess has not been recorded. The earliest recorded game in the data was played in Valencia in 1475, and we use all 135 available recorded games played between the years 1400 and 1800. It is followed by 1175 games played between 1800 and 1850, and

---

[2] See Table 2 for reference. '`t`' stands for the 'total' side (White-Black difference), while '`ph`' indicates a phased value, a composition that is made up of a middle game '`mg`', end game '`eg`' and the phase of the game evaluations.



(a) The evolution of the first move preference for White over the course of human history, spanning back to the earliest recorded games of modern chess in the Chessbase database. The early popularity of 1. e4 gives way to a more balanced exploration of different opening systems and an increasing adoption of more flexible systems in modern times.



(b) The AlphaZero policy head's preferences of opening move, as a function of training steps. Here AlphaZero was trained three times from three different random seeds. AlphaZero's opening evolution starts by weighing all moves equally, no matter how bad, and then narrows down options. It stands in contrast with the progression of human knowledge, which gradually expanded from 1. e4.

Figure 5. A comparison between AlphaZero's and human first-move preferences over training steps and time.

a further 9615 games played between 1850 and 1900. From the twentieth century onward we used roughly 10,000 games per decade, either by randomly sampling games before 1970 subject to a minimum length of 20 moves, or taking the games played by players with the highest ELO ratings after 1970. It is true that there is a difference in the average quality of games recorded across different periods, especially given the emergence of professional chess players and the benefits of computer chess engines analysis in the recent years. However, these are all part of the human progression of the understanding of chess.

5.2 First move progression

Figure 5a shows the opening move preference for White at move 1. Most of the earliest recorded games seem to feature 1. e4 as the opening move. As this is one of the best theoretical tries, it remains popular throughout human history, but its dominance in the early years gives way to a more balanced distribution of chess openings, with 1. d4 being slightly more popular in the early 20th century, and an increasing popularity of more flexible systems like 1. c4 and 1. ♘f3. This is different to AlphaZero's exploration, shown in Figure 5b. It is more simultaneous in comparison, rather than relying mainly on a single initial move before branching into alternatives, which seems to be the case in early human play.

By and large, AlphaZero decreases first-move entropy through training. What we know from data, human first-move entropy increases over the last five centuries. With a uniform prior, the AlphaZero neural network's initial move preference encodes $-\sum_{i=1}^{20}\frac{1}{20}\log_2\frac{1}{20} = 4.32$ bits per move. This entropy reduces to 2.76, 2.81 and 3.02 bits for the first move prior after 1 million training steps for each of the seeds in Figure 5b respectively. The prior reflects the data in training games, which were played with 800 MCTS simulations, injected noise and some stochastic move sampling (see Section 3.2). As a result, the first-move priors in Figure 5b places *some* nonzero mass on each move.

From recorded data, the first move preferences between years 1400 and 1800 empirically encode a mere 0.33 bits of information. At the end of the twentieth century, the first move in top level play empirically encodes 1.87 bits of information.

	seed *(left)*	seed *(centre)*	seed *(right)*	seed *(additional)*
3… ♘f6	5.5%	92.8%	88.9%	7.7%
3… a6	89.2%	2.0%	4.6%	85.8%
3… ♗c5	0.7%	0.8%	1.3%	1.3%

Table 1. The AlphaZero prior network preferences after 1. e4 e5 2. ♘f3 ♘c6 3. ♗b5, for four different training runs of the system. The prior is given after 1 million training steps, and the seeds *(left, centre, right)* correspond to those in Figure 5b.

5.3 The Ruy Lopez

We consider a specific example, the Ruy Lopez 1. e4 e5 2. ♘f3 ♘c6 3. ♗b5, where AlphaZero's preferences and human knowledge can take different paths. Table 1 shows the response of four different versions of AlphaZero, trained from four different seeds. Training either converges to the Berlin defense 3… ♘f6 as its main reply to the Ruy Lopez, or to the more traditional 3…a6. From a theoretical standpoint, there isn't really a major difference between the two, so the partial interchangeability is ultimately not really surprising.

Figure 6 illustrates a case when AlphaZero's preferences take a different path from human history, and where the Berlin defense system (3… ♘f6) is preferred. This system was championed by GM Vladimir Kramnik at top level, and has since become a highly fashionable modern equalizing attempt in the Ruy Lopez, leading to a deeply theoretical Berlin endgame. Yet, for a long period of time, it was thought of as being slightly worse for Black, compared to the more typical 3… a6. Looking back in time, it took a while for human chess opening theory to fully appreciate the benefits of Berlin defense and to establish effective ways of playing with Black in this position. On the other hand, AlphaZero develops a preference for this line of play quite rapidly, upon mastering the basic concepts of the game. This already highlights a notable difference in opening play evolution between the humans and the machine.

5.4 Remarks

The lens of comparison through which we view AlphaZero training and human history is narrow; we've only sketched a main difference by looking at the narrowing (AlphaZero) or expansion (human) of options at the start of the game. For initial opening moves, an empirical density could easily be estimated from human positions. For the same reason we have not taken account of middle game themes or a greater understanding of endgames in human play, which require a different approach of estimating and predicting human preferences from data.

6 Rapid increase of basic knowledge

During the course of AlphaZero's million training steps, there is an inflection point where the network understands "enough" about piece value that playing basic opening sequences (like playing 1. e4 and not 1. f3) lead to tangible advantages.

From the evidence we have in Section 4.6, notably Figure 4, Stockfish's evaluation sub-function of `material` is strongly indicative of the AlphaZero network's value assessment. Figure 4 suggests that the concept of `material` and its importance in the evaluation of a position is largely learned between training steps 10k and 30k, and then refined. Furthermore, the concept of piece `mobility` is progressively incorporated in AlphaZero's value head in the same period. It is plausible that a basic understanding of the material value of pieces should precede a rudimentary understanding that greater piece mobility is advantageous.

6.1 Discovery of standard openings

In this section we examine the discovery of standard openings. Evidence suggests recognizable opening theory develops between 25k and 60k training steps, after the period between 10k and 30k steps when knowledge of basic material value is acquired. Section 5.2 and notably Figure 5b illustrates a rapid transition from a largely uniform prior to commonly played moves. Figure 6 shows how the Ruy Lopez opening is refined over a million training steps, with the inflection point where it starts gaining significant pass being clearly visible. We investigate the transition further here.

To examine the transition, we consider the contribution of familiar opening moves as a fraction of the AlphaZero prior over all moves; see Figure 7. Figure 7a shows that after 25k training iterations, 1. d4 and 1. e4 are discovered to be good opening moves, and are **rapidly adopted** within a short period of around 30k training steps. Similarly, AlphaZero's preferred continuation after 1. e4 e5 is determined in the same short temporal window. Figure 7b illustrates how both 2. d4 and 2. ♘f3 are quickly learned as reasonable White moves, but 2. d4 is then dropped almost as quickly in favour of 2. ♘f3 as a standard reply. Looking beyond individual moves, we group AlphaZero's responses to 1. e4 into two sets, 1…c5, c6, e5 or e6 and then all 16 other moves. Figure 7c shows how 1…c5, c6, e5 or e6 together account for 20% of the prior mass when the network is initialized, as they should, as they are four out of twenty possible Black responses. However, after 25k training steps their contribution rapidly grows to account for 80% of the prior mass.

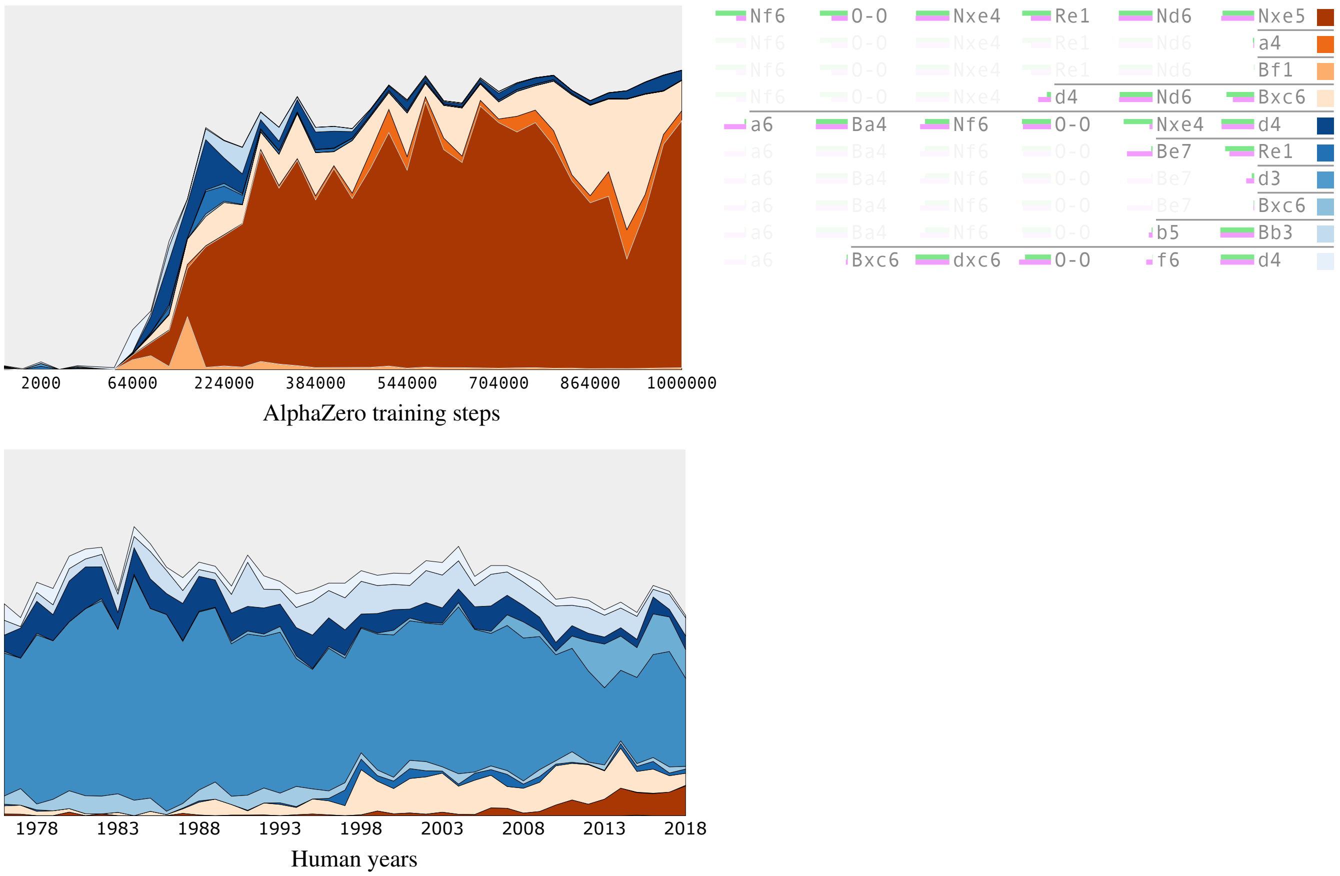
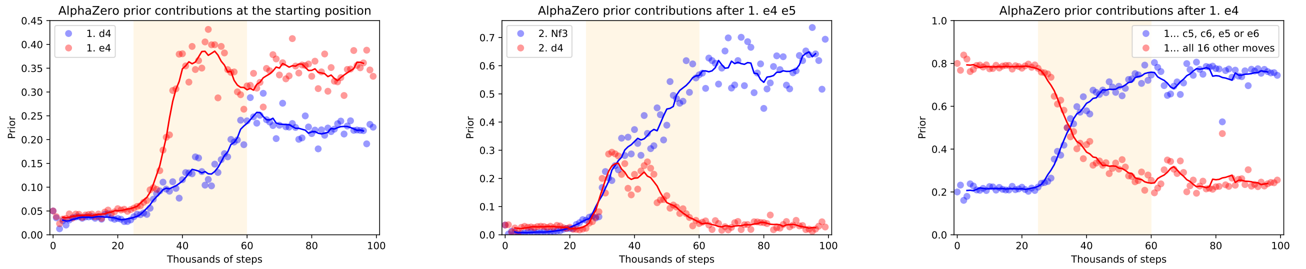


Figure 6. The prior continuation after 1. e4 e5 2. ♘f3 ♘c6 3. ♗b5. The figures show the proportional weight out of one of the top 11 continuations that are six plies deep. The 10 continuations are AlphaZero's top 4 and human grandmasters' top 6 over the last 30 years. Assuming 25 moves per ply, there should be another quarter of a billion (25^6) lines in the figure. The light green bars next to the moves show AlphaZero's prior's preference compared to grandmaster human frequency over the last 30 years in pink.

After the rapid adoption of basic opening moves in the window from 25k to 60k training steps, opening theory is progressively refined through repeatedly updating the training buffer of games with fresh self-play games. While these examples of the rapid discovery of basic openings are not exhaustive, further examples across multiple training seeds are given in Appendix C.

Training configuration dependence Our conclusions are dependent on the particular hyperparameter settings that empirically yielded the strongest version of the system over a hyperparameter search. We refer the reader to Section 3.2 for training details for a specific configuration: At the start of training, the self-play training buffer (queue) is filled with 1 million training positions from completely random games. At 30 positions per game, at least 30k games are there played using the randomly initialized network in MCTS. With a batch size of 4096, the queue is empty before 250 training steps are reached. The policy head $\mathbf{p}$ in Figure 1 is a $8 \times 8 \times 73 = 4672$ tensor of possible moves, and from these random games, the network starts assigning most weight to legal moves. After every 1000 training steps, the networks responsible for generating self-play games are refreshed with the latest copy of the training network. After 25k training steps, when familiar openings start gaining traction, the neural networks responsible for generating the self-play games would have been reloaded with fresh copies from the training job 25 times. The onset is therefore not a result of the initial random training buffer being depleted of examples. The onset is dependent on the training configuration settings; it would shift if the self-play networks were refreshed at a faster or slower rate, for example. We do not examine suboptimal training configuration settings leading to significantly weaker versions of AlphaZero in this work.

Sequential knowledge acquisition Figures 4, 7 and 6 suggest a sequence: that piece value is learned before basic opening knowledge; that once discovered, there is an explosion of basic opening knowledge in a short temporal window; that the



(a) After 25k training iterations, e4 and d4 are discovered to be good opening moves, and **rapidly adopted** within a short period of around 30k training steps.

(b) Rapid discovery of options given 1. e4 e5. Within a short space of time, ♘f3 is settled on as a standard reply, whereas d4 is considered and discarded.

(c) In the window between 25k and 60k training steps, AlphaZero learns to put 80% of its mass on four replies to e4, and 20% of its mass on all other 16 moves.

Figure 7. Rapid discovery of basic openings. The randomly initialized AlphaZero network gives a roughly uniform prior over all moves. The distribution stays roughly uniform for the first 25k training iterations, after which popular opening moves quickly gain prominence. In particular, 1. e4 is fully adopted as a sensible move in a window of 10k training steps, or in a window of 1% of AlphaZero's training time.

network's opening theory is slowly refined over hundreds of thousands of training steps.

6.2 Vladimir Kramnik's qualitative assessment

To further shed light on the early period in the evolution of chess knowledge within the AlphaZero network, we complemented our quantitative experiments by a qualitative assessment by GM Vladimir Kramnik, a former world chess champion. This assessment is an attempt to identify themes and differences in style of play between AlphaZero checkpoints in the early phase of training. By generating games played between versions of AlphaZero at different training steps, their strengths and weaknesses would potentially be exposed in the games. We generated games played between AlphaZero at 16k and 32k, 32k and 64k, and 64k and 128k training steps. For each, Vladimir Kramnik was presented with a random sample of 100 games from each pairing. These qualitative assessments were provided prior to seeing our other quantitative results in order to avoid introducing biases into the analysis.

Perhaps surprisingly, GM Kramnik was able to easily spot recurring themes in the shared game samples and to thereby formulate opinions as hypotheses for the differences between model performance at these points in time:

16k to 32k The comparison between 16k and 32k was the simplest. AlphaZero at 16k has a crude understanding of material value and fails to accurately assess material in complex positions. This leads to potentially undesirable exchange sequences, and ultimately losing games on material. On the other hand, AlphaZero at 32k seemed to have a solid grasp on material value, thereby being able to capitalize on the material assessment weakness of 16k.

32k to 64k The main difference between 32k and 64k, in GM Kramnik's view, lies in the understanding of king safety in imbalanced positions. This manifests in the 32k version potentially underestimating the attacks and long-term material sacrifices of the 64k version, as well as the 32k version overestimating its own attacks, resulting in losing positions. Despite the emphasis on king safety, GM Kramnik also comments that 64k appears to be somewhat stronger in all aspects than 32k, which is well-aligned with our quantitative observations for the changes in the period.

64k to 128k The theme of king safety in tactical positions resurfaces in the comparisons between 64k and 128k, where it seems that 128k has a much deeper understanding of which attacks will succeed and which would fail, allowing it to sometimes accept material sacrifices made by 64k for purposes of attack, yet then proceed to defend well, keep the material advantage, and ultimately convert to a win. There seems to be less of a difference in terms of positional and endgame play.

6.3 Are tactical skills acquired before positional skills?

From observing AlphaZero's self-play games over increasing training steps, Vladimir Kramnik remarked that tactical skills appear to precede positional skills as AlphaZero learns.

The GM's observation can be explained by the difference in the abilities that are required to master tactical and positional lines. Playing tactical lines requires finding short term moves that force the opponent to lose material, being checkmated, etc.; while playing the positional lines requires foreseeing the development of the game many plies ahead, i.e. being able to perform deeper search. The depth of the search, that AlphaZero can perform given a fixed amount of compute, is directly related to the amount of entropy in the AlphaZero policy priors. As demonstrated in Section 5, the entropy is maximal in the beginning of the training process when the priors are distributed almost uniformly. Over time, AlphaZero learns to discard least promising moves; that makes deeper search possible. Since AlphaZero cannot properly evaluate positional lines before deeper search is possible, the emergence of the positional skills is delayed.

To collect more supporting evidence, GM Kramnik selected two sets of 10 opening lines based on their known theoretical properties. The first set contained starting positions corresponding to quiet and positional opening systems like the Berlin defense, Queen's gambit declined, Queen's gambit accepted and the English opening. The second set contained starting positions corresponding to aggressive and tactical opening systems like the Najdorf Sicilian, Nimzowitsch defense, the French Winawer and the King's Indian defense Both the tactical and the positional lines were selected such that they are considered equal, not conferring an advantage to either player. In the self-play games, each version of AlphaZero had a chance to play as White and Black from the same position.

We generated self-play games played between versions of AlphaZero after 32k, 64k and 128k training steps. For each of the 10 opening lines in each of the two sets, we generated 100 games played at approximately 1 second per move, resulting in 1000 games per set. The outcome of these games were:

32k to 64k; tactical openings In the tactical lines, AlphaZero at 64k won 82.3% of the games, lost 4.1% of the games, and drew 13.6% of the games, resulting in an estimated ELO difference of 365 points.

32k to 64k; positional openings In the positional lines, AlphaZero at 64k won 76.6% of the games, lost 1.2% of the games, and drew 22.2% of the games, resulting in the estimated ELO difference of 343 points.

64k to 128k; tactical openings In the tactical lines, AlphaZero at 128k won 53.7% of the games, lost 12.5% of the games, and drew 33.8% of the games, resulting in an estimated ELO difference of 152 points.

64k to 128k; positional openings In the positional lines, AlphaZero at 128k won 37.8% of the games, lost 3.8% of the games, and drew 58.4% of the games, resulting in the estimated ELO difference of 123 points.

These experiments show that there is a marginal difference between game outcomes from tactical positions and game outcomes from more positional openings. The slight difference in favour of tactical positions persists both in the comparison between the 128k and 64k model, as well as the 64k and 32k model. The observed difference in ELO between the checkpoints playing positional and tactical lines is circumstantial evidence that tactical skill acquisition might happen before or faster than positional skill acquisition. However, this is only one potential explanation of the differences. GM Kramnik's remark points to further work to understand the order in which skills are acquired.

7 Exploring activations with unsupervised methods

In Section 4.2 we investigated when and where the AlphaZero network encodes human conceptual knowledge. The approach was confined to labeled data of different human concepts, and was made possible through supervised methods. The public sub-functions of Stockfish's evaluation function and a large collection of custom chess concept functions allow the automation of large supervised data sets. Our approach so far has relied on supervised learning, either to regress predefined concepts from the network or to predict the value function from those concepts. This runs the risk of overlooking anything we have not included in our concept set, as well as biases our understanding of the AlphaZero network towards those concepts. In this section we take an alternative approach, using simple unsupervised approaches involving matrix decomposition (Section 7.1) and correlation analysis (Section 7.2). These methods generate a wealth of data to analyse at each layer of the network. We present some highlights here and make the full dataset available in the Supplementary Materials.

7.1 Non-negative matrix factorisation

Non-negative matrix factorisation (NMF) [14] is an approach to discover features in an unsupervised way. It has previously been used to interpret vision [15] and simple RL models [90]. In this section we explore NMF as a complementary unsupervised approach to the probes trained in Section 4: rather than probe activations for specific concepts, we instead simplify the activations in a concept-agnostic way. This allows the structure of the activations to reveal itself, instead of imposing our assumptions on it.

7.1.1 Methodology

Layer l's activations $\mathbf{z}^l \in \mathbb{R}^{H\times W\times C} = \mathbb{R}^{8\times 8\times 256}$ are non-negative. Each of the 256 channels is an 8×8 plane. We treat each plane as a vector and reshape $\mathbf{z}^l$ into a matrix $\hat{\mathbf{z}}^l \in \mathbb{R}^{HW\times C}$. This is a square-by-channel matrix, where every row corresponds to a square and is a non-negative C-dimensional vector. We compress the activations $\mathbf{z}^l$ by representing each square as a non-negative K-dimensional weight vector, with $K < C$. It means the compression would be into a matrix $\mathbf{\Omega} \in \mathbb{R}^{HW \times K}$. In the reduced representation, the rows of $\mathbf{\Omega}$ correspond to squares, and its entries are K weights that sum together non-negative global factors $\mathbf{f}_k \in \mathbb{R}^C$ for $k = 1, \ldots, K$ so that the original row in $\hat{\mathbf{z}}^l$ is closely approximated. If the factor matrix $\mathbf{F} \in \mathbb{R}^{K \times C}$ contains the global factors as columns, the activations are approximated with $\hat{\mathbf{z}}^l \approx \mathbf{\Omega F}$. For brevity we omit the dependence of the factor and weight matrices on the layer index l.

To determine the global factors $\mathbf{F}$, we stack the network activations of N randomly selected inputs $\hat{\mathbf{z}}^l_1, \ldots, \hat{\mathbf{z}}^l_N$ into a matrix $\hat{\mathbf{Z}}^l \in \mathbb{R}^{NHW \times C}$ (we use $N = 10^4$, and randomly select 50 positions to visualise). The factors and their weights $\mathbf{\Omega}_{\text{all}} \in \mathbb{R}^{NHW \times K}$ are found by minimizing

$$\begin{aligned} &\mathbf{F}^*, \mathbf{\Omega}^*_{\text{all}} = \min_{\mathbf{F}, \mathbf{\Omega}_{\text{all}}} \left\| \hat{\mathbf{Z}}^l - \mathbf{\Omega}_{\text{all}} \mathbf{F} \right\|_2^2 \\ &\mathbf{F}, \mathbf{\Omega}_{\text{all}} \geq \mathbf{0} \, . \end{aligned} \tag{17}$$

The stacked submatrices of $\mathbf{\Omega}^*_{\text{all}}$ correspond to the NMF weights for each input's activations. Alternatively, given $\mathbf{F}^*$, the NMF weights for any activations $\mathbf{z}^l$ could be retrieved by

$$\begin{aligned} &\mathbf{\Omega}^* = \min_{\mathbf{\Omega}} \left\| \hat{\mathbf{z}}^l - \mathbf{\Omega} \mathbf{F}^* \right\|_2^2 \\ &\mathbf{\Omega} \geq \mathbf{0} \, . \end{aligned} \tag{18}$$

To visualize the NMF factors for activations $\mathbf{z}^l$, we overlay the K columns of $\mathbf{\Omega}^*$ onto the input $\mathbf{z}^0$. The visualization of factor k's contributions to $\mathbf{z}^l$ is done by reshaping the column k of $\mathbf{\Omega}^*$ into a $H \times W$ or 8×8 matrix. The visualization shows how much NMF factor k contributes to each neuron's representation. This visualisation makes a strong assumption that representations in the residual block are spatially-aligned, i.e. an activation at a given spatial position can be interpreted in light of the pieces around that position. Although the network architecture biases towards this correspondence (all hidden layers are identical in shape to a chess board, and the residual network structure biases activations towards spatial correspondence) this is not strictly enforced by the architecture.

Using 36 factors per block, the full NMF dataset consists of 720 block/factor pairs. We report selected factors below, and the full dataset is available online. Most factors in later layers remain unexplained, however - we view explaining these factors (and developing the methods necessary to do so) as an important avenue for future work.

7.1.2 Results

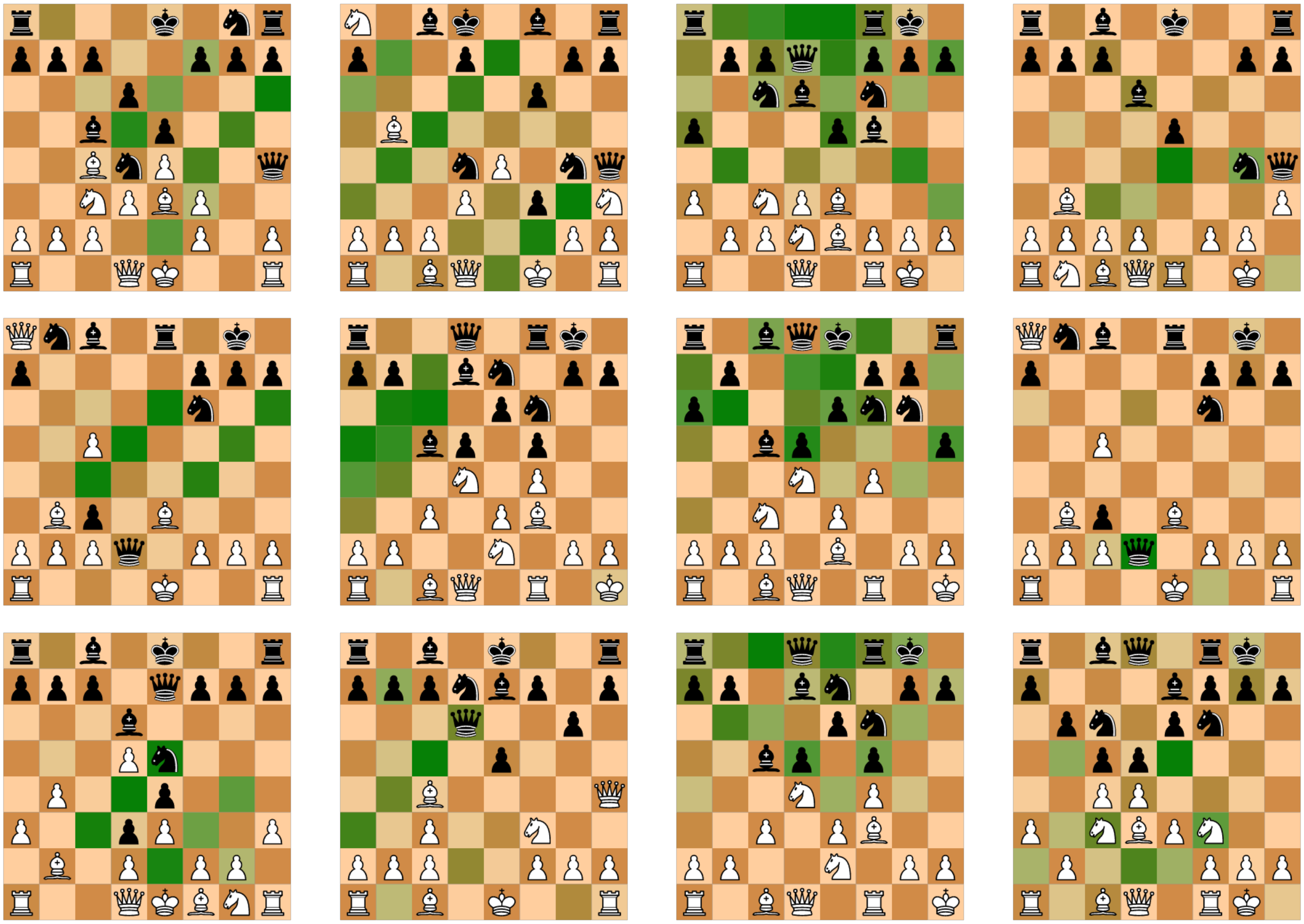
This section highlights some illustrative examples of interpretable factors in AlphaZero's activations. The factors shown in Figure 8a and Figure 8b show the development of potential move computations for the player's and opponent's diagonal moves respectively. In the first layer moves of only three squares or fewer are shown (and only those towards the upper right of the board), demonstrating that move calculations take multiple blocks to complete. The convolutional structure of the AlphaZero network means that all computations from one layer to the next involve only spatially adjacent neurons. Because of this, move computations must occur over the course of multiple layers. This partially explains the gradual increase in predictive power over the initial layers for threat-related concepts shown in Figure 2b.

Figure 8c shows a more complex factor in layer 3: a count of the number of the opponent's pieces that can move to a given square (darker weights indicates more pieces can move to that square). This factor is likely to be useful in computing potential exchanges, and indicates that AlphaZero is also considering potential opponent moves even early in the network. Figure 8d appears to show potential moves for the current player - darker squares indicate better moves (for instance the opponent's hanging queen on d2).

The NMF factor data contains many factors we have not yet interpreted, especially in later layers of the network. Where factors are uninterpretable, this could be because the number of factors used is incorrect, spatial correspondence is broken, or the factor represents something too complex to understand without reference to earlier layers. Development of methods for relating complex later-layer factors to well-understood early layer factors is an important priority for further interpretability work in complex domains. Finally, we note that although these interpretations of the above factors bear out in the majority of the randomly selected positions shown in the online database of factors, an interpretation can only be considered definitive once it has been quantitatively validated [91], ideally by intervening on the input.

7.2 Covariance between inputs and activations

Which input features in $\mathbf{z}^0$ covary most with a particular activation z^l_i? If the activation z^l_i increases, which input features in $\mathbf{z}^0$ are most correlated with the increase, and which input features have no bearing on a change in z^l_i? This section departs from earlier sections to look at individual activations. While our goal is to use complex concepts to understand AlphaZero, the concepts in layer l are composed of a distributed representation through the individual activations in $\mathbf{z}^l$. A simpler unsupervised



(a) Development of diagonal moves for player (block 1, factor 26 of 36).

(b) Fully developed diagonal moves for opponent (block 3, factor 22 of 36).

(c) Count of opponent's potential piece moves (block 3, factor 11 of 36).

(d) Potential good squares to move to? (block 18, factor 22 of 36).

Figure 8. Visualisation of NMF factors in a fully-trained AlphaZero network, showing development of threats and anticipation of possible moves by the opponent, as well as a factor that may be involved in move selection. Following Figure 1, we count a ResNet 'block' as a layer.

test is to determine how specific positions in a board $\mathbf{z}^0$ are "mapped" in each layer of the neural network, and here we measure how related inputs are to individual activations z_i^l.

Activation-input covariance There are many gradient-based methods to investigate how changes in $\mathbf{z}^0$ affect z_i^l by using some form of $\partial z_i^l / \partial \mathbf{z}^0$ [92]. However, with the exception of a move-counter plane, $\mathbf{z}^0$ is binary and $f_{\boldsymbol{\theta}}(\mathbf{z}^0)$ has never seen examples that are not on the $\{0,1\}^{8\times 8\times(14h+7)}$ hypercube; see Figure 1. We take a different approach. As $\mathbf{z}^0$ is much smaller than a typical image, we compute the covariance between $\mathbf{z}^0$ and each activation in the network. We use only the fully trained network, and determine which input features $\mathbf{z}^0$ are most correlated with each activation z_i^l in each layer l, $\mathbf{z}^l = f_{\boldsymbol{\theta}}^{1:l}(\mathbf{z}^0)$. The **activation-input covariance**

$$\mathrm{cov}(z_i^l, \mathbf{z}^0) = \mathbb{E}\left[z_i^l \mathbf{z}^0\right] - \mathbb{E}\left[z_i^l\right] \mathbb{E}\left[\mathbf{z}^0\right] \tag{19}$$

gives an indication of how correlated each component of $\mathbf{z}^0$ is with an activation output at layer l. Because $z_i^l \in [0,15]$ due to gradient clipping and $\mathbf{z}^0 \in [0,1]^{8\times 8\times(14h+7)}$ (due to input scaling, with piece location inputs being either 1 for a piece or 0 for an absence), the covariance is between -15 and 15. We estimate $\mathrm{cov}(z_i^l, \mathbf{z}^0)$ in (19) empirically over 1.4 million randomly selected positions from grandmaster games.

Visualizing the activation-input covariance We visualize the first twelve 8×8 planes of $\mathrm{cov}(z_i^l, \mathbf{z}^0)$ in this analysis – in particular in Figures 10, 11, 12 and 13 – as these correspond to the planes in $\mathbf{z}^0$ that specify the board position. The $8 \times 8 \times 12$

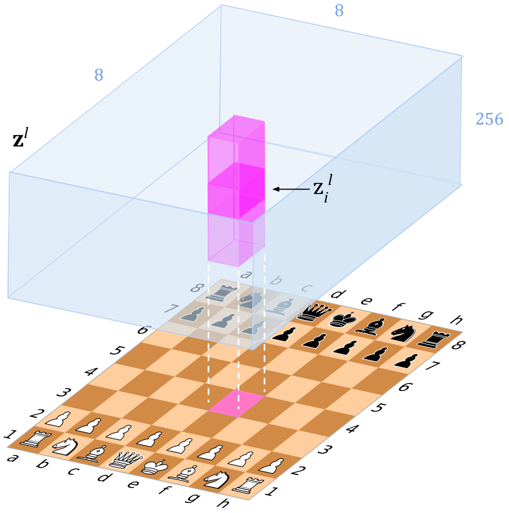


Figure 9. Because of the ResNet backbone of the AlphaZero network in Figure 1, activations z_i^l for $i = (5, 4, \cdot)$ *align spatially* with the e4 square. There are activations z_i^l in the same position in each $\mathbf{z}^l$ for $l = 1, \ldots 20$ that simply propagate local features like piece placement forward from the first layers to the last in the backbone (Section 7.2.2; Figure 12). Other activations z_i^l in the same position in each $\mathbf{z}^l$ for $l = 1, \ldots 20$ are active on an evolving pattern of global features (Section 7.2.3; Figure 13).

submatrix corresponding to the piece planes is clipped at zero and scaled so that its maximum value is one, so that only positive correlations – the presence and not the absence of pieces – are visualized. Recall that each of the first twelve planes of $\mathrm{cov}(z_i^l, \mathbf{z}^0)$ correspond to different piece types. All twelve planes are rendered on the same board. The *opacity* of each piece in a plane corresponds to the covariance of activation with that piece on a given square. The input encoding of the pieces in $\mathbf{z}^0$ is from the point of view of the playing side. The input encoding for Black to play would simply flip the standard chess board horizontally and vertically so that it is aligned from the point of view of the playing side. The visualizations are from the point of view of the playing side, and colour the playing side as White. In our illustrations of the activation-input covariances, we used a lighter board colour than other visualizations in this paper. This is solely to make the semi-translucent pieces more visible.

The sequence of convolutions in the ResNet backbone of $f_{\boldsymbol{\theta}}(\mathbf{z}^0)$ has a spatial structure and each $\mathbf{z}^l$'s $8 \times 8 \times 256$ tensor aligns with the input board encoding. This is illustrated in Figure 9. We let i index the width, height and channel in $\mathbf{z}^l$. As there are 16384 activations at the end of each layer, we use indexes $i = (5, 4, \cdot)$ in Figures 10, 11, 12 and 13. These indexes align with the central e4 square on a chess board.

We illustrate different roles of individual activations as feature detectors below, including where they detect basic patterns on the input board position, seem to copy the presence of features over from one layer to the next, or seem to combine lower level features.

7.2.1 Feature detectors

There is evidence that lower layers are likely to encode less abstract and therefore more location specific concepts [7]. To illustrate the role of lower layers as feature detectors, we pick square e4 and visualize the activations from layer 1 that align with it spatially. In Figure 10 we show the activation-input covariances of channels $c = 1, \ldots, 15$ of $i = (5, 4, c)$ in $\mathbf{z}^1$. The choice of the first fifteen channels over any of the other 256 channels is arbitrary. Layer 1 in Figure 1 follows after applying a bank of 3×3 convolution filters, and we expect only local 3×3 patterns around e4 to be correlated with the activations in all channels. The correlations are indeed localized; index $(5, 4)$ corresponds to the e4 square on the board, and after a 3×3 convolution (see Figure 1) piece patterns around the e4 square are detected.

There are also a number of global positions that appear in the covariance visualizations in Figure 10. It is because of *correlations in the data*: the features they detect are globally correlated to other piece positions on the board in the sample of grandmaster chess positions. As an example, activation $i = (5, 4, 6)$ detects the pattern (♙d3, ♝e3, ♘f3) with both rows (d4, e4, f4) and (d5, e5, f5) in front of it being empty. This piece pattern predominantly appears in certain opening positions and almost never elsewhere in middle- and endgames, and hence rooks on a1 and a8 are also visible.

The features detected by the activations in $\mathbf{z}^1$ go beyond detecting specific localized patterns of pieces. There are a small number of activations that clearly detect *move types* from a square. In Figure 11 we identified the first five such activations for the channels corresponding to the e4 square. There, the activation-input correlations are large when there is a long-range

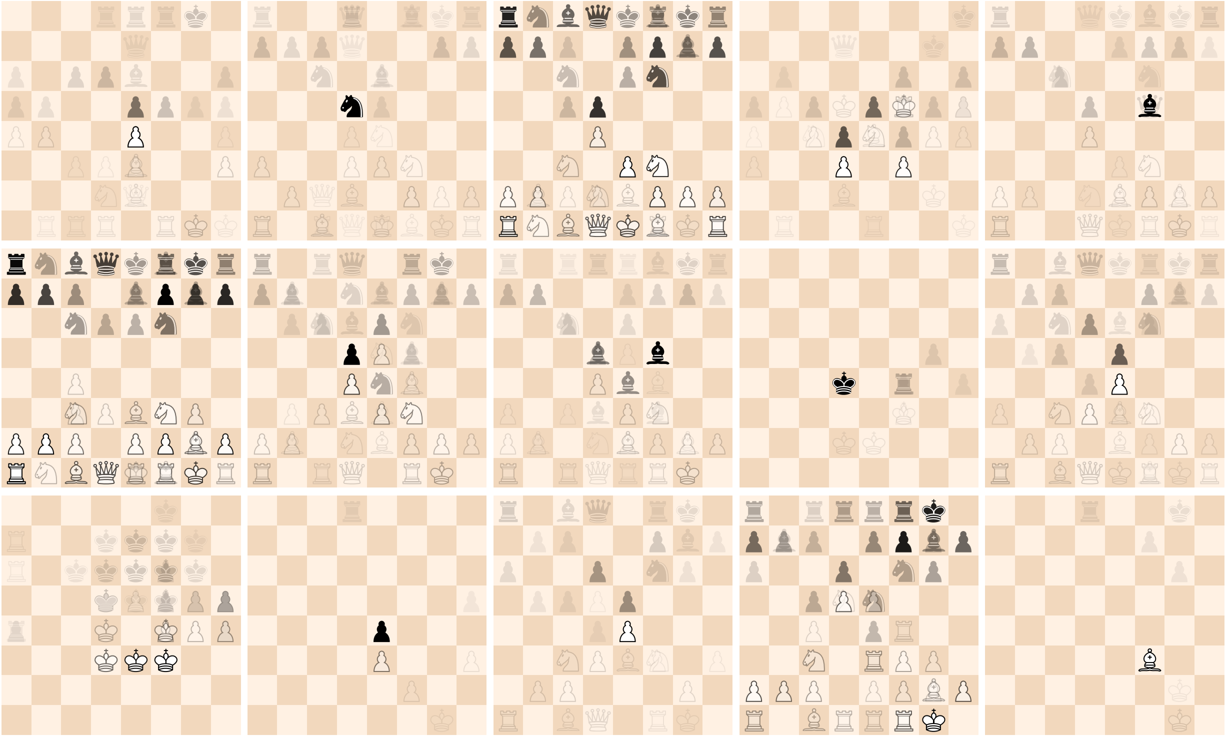


Figure 10. The activation input covariance $\mathrm{cov}(z_i^1, \mathbf{z}^0)$ for the **first layer**, with activations $i = (5,4,c)$, iterating from channels $c = 1,\ldots,15$ for layer $l = 1$. The top row shows channels 1 to 5, the middle row channels 6 to 10, and the bottom row channels 11 to 15. Because $(5,4,c)$ is centered at square e4, this is a feature detector for groupings of pieces around e4.

diagonal-moving piece on a square (either a Bishop or a Queen) or a long-range horizontal or vertical-moving piece (either a Rook or a Queen).

7.2.2 Persisted feature detectors

The structure of the ResNet links activations z_i^l and z_i^{l-1} with

$$z_i^l = \mathrm{ReLU}(z_i^{l-1} + g^l(\mathbf{z}^{l-1})_i) \;, \tag{20}$$

as given in (2). Depending on the purpose of the activation z_i^l in the distributed encoding $\mathbf{z}^l$, it combines

1. propagating information forward through z_i^{l-1} and
2. evolving by incorporating additional local context through $g^l(\mathbf{z}^{l-1})_i$.

In the first case, information is persisted. Some feature might be useful in a distributed representation at any layer in the network, and the feature propagated forward in each layer of the network. As an example, it is important that information about the position of pieces are propagated through the network, as the prior $\mathbf{p}$ in (1) has to put mass on legal moves. Equivalently, other local features might be universally useful in each layer of representation, and the structure of (20) allows activations to be propagated forward.

We illustrate how a feature detector z_i^l is persistent from layer 1 to 20. Activation $i = (5,4,10)$ in layer 1 in Figure 10 detects opposing central pawns, and we follow that activation through all layers of the network in Figure 12. The activation-input covariance is largely similar, suggesting that the feature is persisted and reused through layers of the network, being propagated up from the first layer to the last.

7.2.3 Evolving feature detectors

Some information is useful for the distributed representation of $\mathbf{z}^0$ in layer l as $\mathbf{z}^l$, but possibly not at higher layers in the network architecture. The activations in layer 1 detect localized patterns, but these could become increasingly global as activations are propagated through the network. We consider activation $i = (5,4,8)$ from Figure 10, which detects a central light-squared bishop for the opposing side. Figure 13 shows how that activation evolves through the layers of the trained AlphaZero network.

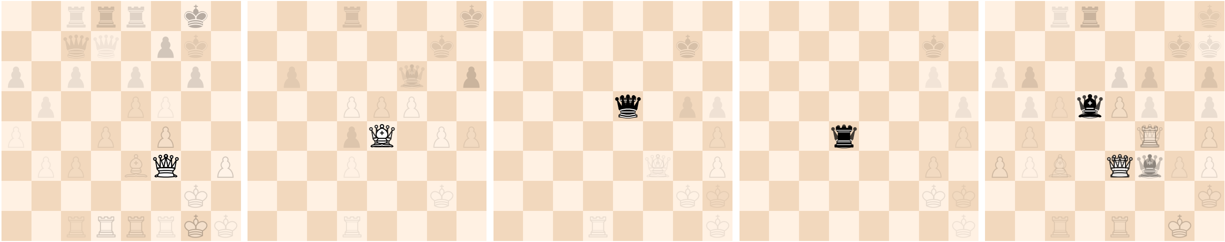
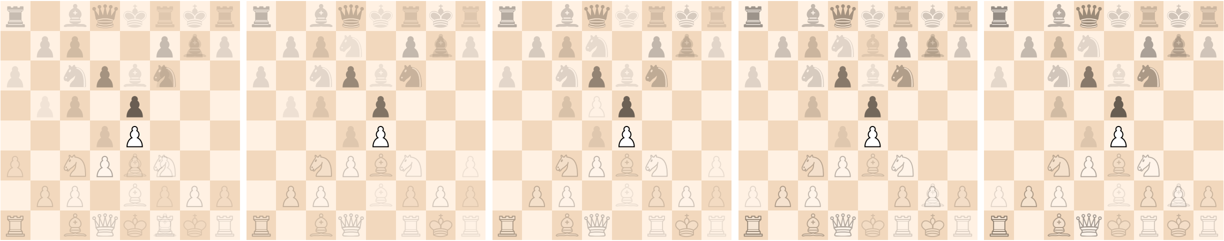


Figure 11. Activations detecting **move types from a square**, illustrated with the activation input covariance $\text{cov}(z_i^1, \mathbf{z}^0)$ for the **first layer**. From left to right: Activations $i = (5,4,23)$ and $(5,4,90)$ show a ♗ and ♕ overlaid, and detect possible long-range diagonal piece movement from f3 and e4 (compare $i = (5,4,15)$ in the bottom right of Figure 10, which only detects the bishop). Activations $i = (5,4,33)$ and $(5,4,92)$ show a ♖ and ♕ overlaid, and detect when the (opponent's) piece on e5 can move horizontally or vertically across the board. Activation $i = (5,4,91)$ looks for two patterns: a diagonal moving piece (♗ or ♕) on d5 and a horizontal-vertical moving piece (♖ or ♕) on e3.

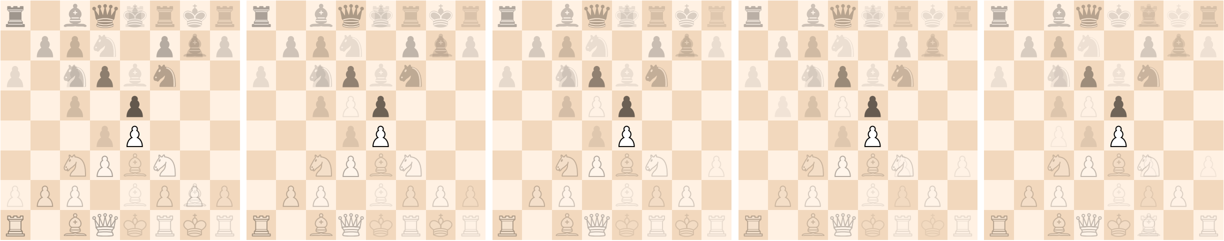
Moving up the network, with White to play, features revolve around a central light-squared bishop. These evolve to a central opposing pawn on e4, to finally a knight on c3 attacking e4.

7.2.4 Open questions: distributed encoding

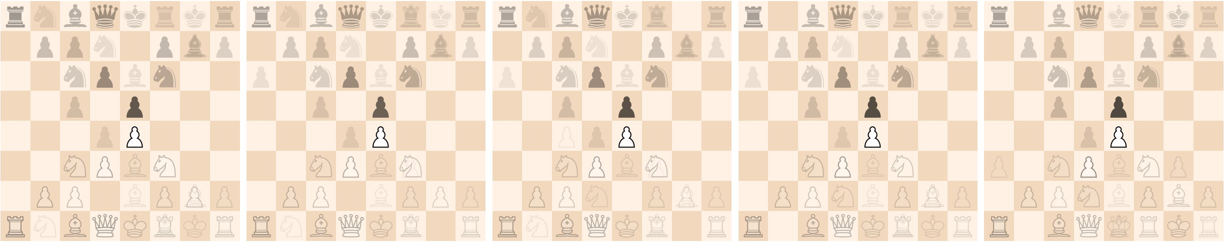
On a micro scale, the activation-input covariances in this section show how each of the individual activations z_i^l in $\mathbf{z}^l$ is correlated with input positions. On a macro scale, Section 4 showed how common human recognizable concepts could be regressed (linearly) from the distributed encoding. On a macro scale, the 16384 activations in $\mathbf{z}^l$ together form a distributed encoding of many concepts. An interesting open direction is the relation between the micro and the macro scale: for a given chess position, decomposing a concept (approximated as a linear function of the activations; see Section 4) into principal features in the position.



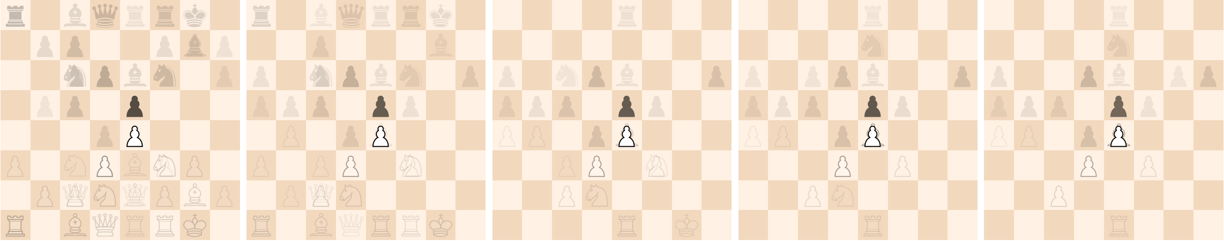
activation-input covariance for activation $i = (5,4,10)$; layers 1 to 5



activation-input covariance for activation $i = (5,4,10)$; layers 6 to 10

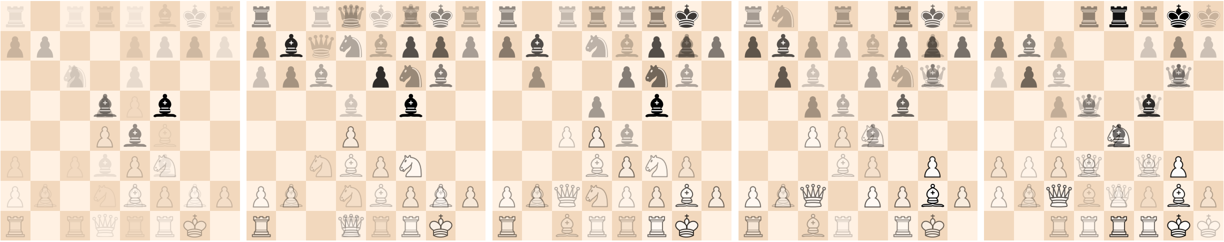


activation-input covariance for activation $i = (5,4,10)$; layers 11 to 15

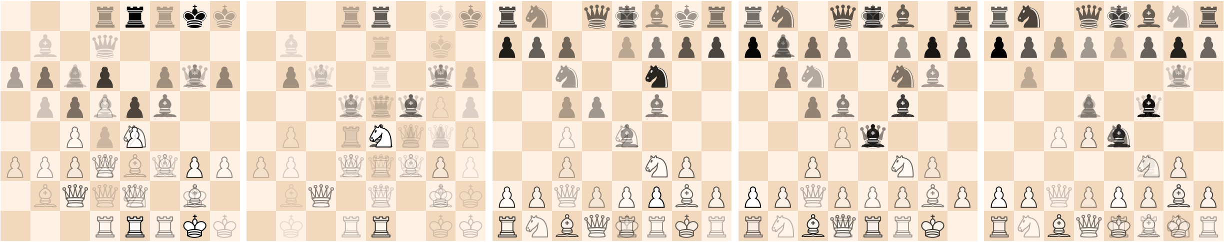


activation-input covariance for activation $i = (5,4,10)$; layers 16 to 20

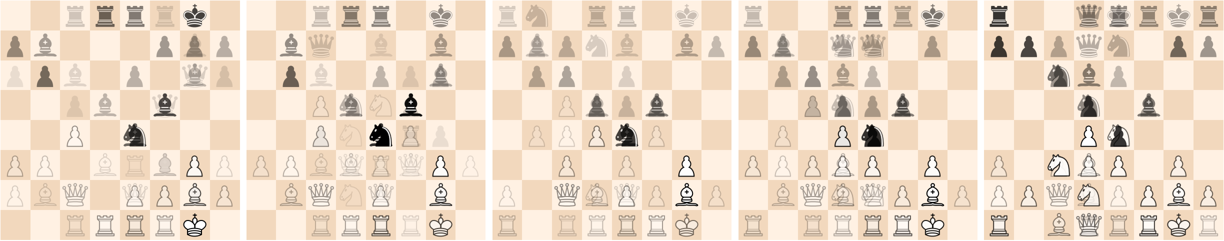
Figure 12. The activation-input covariance $\mathrm{cov}(z_i^l, \mathbf{z}^0)$ for **layers 1 to 20** for activations $i = (5,4,10)$ in each layer l. Unlike Figure 13, the unit is persistently activated when there are opposing central pawn; the detection of two opposing central pawns are propagated through all layers of the network.



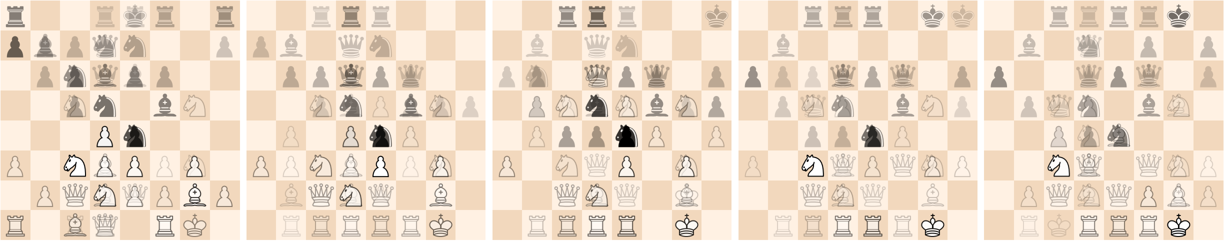
activation-input covariance for activation $i = (5,4,8)$; layers 1 to 5



activation-input covariance for activation $i = (5,4,8)$; layers 6 to 10



activation-input covariance for activation $i = (5,4,8)$; layers 11 to 15



activation-input covariance for activation $i = (5,4,8)$; layers 16 to 20

Figure 13. The activation-input covariance $\mathrm{cov}(z_i^l, \mathbf{z}^0)$ for **layers 1 to 20** for activations $i = (5,4,8)$ in each layer l. Moving up the network, with White to play, features revolve around a central light-squared bishop. These evolve to a central opposing pawn on e4, to finally a ♘c3 (attacking e4).

8 Conclusions

In this work we studied the evolution of AlphaZero's representations and play through a combination of concept probing, behavioural analysis, and examination of AlphaZero's activations. Examining the evolution of human concepts using probing showed that many human concepts can be accurately regressed from the AZ network after training, even though AlphaZero has never seen a human game of chess, and there is no objective function promoting human-like play or activations. Following on from our concept probing we reflected on the challenges associated with this methodology and suggested future directions of research. Our concept regression findings are supported by finding multiple human-interpretable factors in AlphaZero's activations. We have released these online and are excited to see what other researchers can find in this data. We have also analysed the evolution of AlphaZero's play style through a study of opening theory and an expert qualitative analysis of AlphaZero's self-play during training. Our analysis suggests that opening knowledge undergoes a period of rapid development around the same time that many human concepts become predictable from network activations, suggesting a critical period of rapid knowledge acquisition. The fact that human concepts can be located even in a superhuman system trained by self-play broadens the range of systems in which we should expect to find human-understandable concepts. We believe that the ability to find human-understandable concepts in the AZ network indicates that a closer examination will reveal more. The next question is: can we go beyond finding human knowledge and learn something new?

Acknowledgements

We would like to thank ChessBase for providing us with an extensive database of historical human games [89]. This was used in our experiments to evaluate the explainability of AlphaZero, as well as contrast it to the historical developments in human chess theory. We are grateful to Frederic Friedel at ChessBase for his enduring support, and to Shane Legg, Neil Rabinowitz, Aleksandra Faust, Lucas Gill Dixon, Martin Wattenberg, and Nithum Thain for insightful comments and discussions.

References

1. Bau, D. *et al.* Understanding the role of individual units in a deep neural network. *Proc. Natl. Acad. Sci.* **117**, 30071–30078 (2020).
2. Manning, C. D., Clark, K., Hewitt, J., Khandelwal, U. & Levy, O. Emergent linguistic structure in artificial neural networks trained by self-supervision. *Proc. Natl. Acad. Sci.* **117**, 30046–30054 (2020).
3. Goh, G. *et al.* Multimodal neurons in artificial neural networks. *Distill* **6**, e30 (2021).
4. Rogers, A., Kovaleva, O. & Rumshisky, A. A primer in bertology: What we know about how bert works. *Transactions Assoc. for Comput. Linguist.* **8**, 842–866 (2020).
5. Silver, D. *et al.* A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science* **362**, 1140–1144 (2018).
6. Andrulis, J., Meyer, O., Schott, G., Weinbach, S. & Gruhn, V. Domain-level explainability – a challenge for creating trust in superhuman ai strategies (2020). 2011.06665.
7. Bau, D., Zhou, B., Khosla, A., Oliva, A. & Torralba, A. Network dissection: Quantifying interpretability of deep visual representations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 6541–6549 (2017).
8. Kim, B. *et al.* Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (tcav) (2018). 1711.11279.
9. Alvarez-Melis, D. & Jaakkola, T. S. Towards robust interpretability with self-explaining neural networks. In *Advances in Neural Information Processing Systems*, 7786–7795 (2018).
10. Ghorbani, A., Wexler, J., Zou, J. Y. & Kim, B. Towards automatic concept-based explanations. In *Advances in Neural Information Processing Systems*, 9273–9282 (2019).
11. Koh, P. W. *et al.* Concept bottleneck models. *arXiv preprint arXiv:2007.04612* (2020).
12. Chen, Z., Bei, Y. & Rudin, C. Concept whitening for interpretable image recognition. *Nat. Mach. Intell.* **2**, 772–782 (2020).
13. Paatero, P. & Tapper, U. Positive matrix factorization: A non-negative factor model with optimal utilization of error estimates of data values. *Environmetrics* **5**, 111–126 (1994).

14. Lee, D. D. & Seung, H. S. Learning the parts of objects by non-negative matrix factorization. *Nature* **401**, 788–791 (1999).

15. Olah, C. *et al.* The building blocks of interpretability. *Distill* **3**, e10 (2018).

16. Arrieta, A. B. *et al.* Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai. *Inf. Fusion* **58**, 82–115 (2020).

17. Doshi-Velez, B., Finale; Kim. Towards a rigorous science of interpretable machine learning. In *eprint arXiv:1702.08608* (2017).

18. Lipton, Z. C. The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery. *Queue* **16**, 31–57 (2018).

19. Alain, G. & Bengio, Y. Understanding intermediate layers using linear classifier probes. *arXiv preprint arXiv:1610.01644* (2016).

20. Clough, J. R. *et al.* Global and local interpretability for cardiac mri classification. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, 656–664 (Springer, 2019).

21. Graziani, M., Andrearczyk, V. & Müller, H. Regression concept vectors for bidirectional explanations in histopathology. In *Understanding and Interpreting Machine Learning in Medical Image Computing Applications*, 124–132 (Springer, 2018).

22. Sprague, C., Wendoloski, E. B. & Guch, I. Interpretable ai for deep learning- based meteorological applications. In *American Meteorological Society Annual Meeting* (2019).

23. Mincu, D. *et al.* Concept-based model explanations for electronic health records. In *Proceedings of the Conference on Health, Inference, and Learning*, 36–46 (2021).

24. Bouchacourt, D. & Denoyer, L. Educe: Explaining model decisions through unsupervised concepts extraction. *arXiv preprint arXiv:1905.11852* (2019).

25. Yeche, H., Harrison, J. & Berthier, T. Ubs: A dimension-agnostic metric for concept vector interpretability applied to radiomics. In *Interpretability of Machine Intelligence in Medical Image Computing and Multimodal Learning for Clinical Decision Support*, 12–20 (Springer, 2019).

26. Sreedharan, S., Soni, U., Verma, M., Srivastava, S. & Kambhampati, S. Bridging the gap: Providing post-hoc symbolic explanations for sequential decision-making problems with black box simulators. *arXiv preprint arXiv:2002.01080* (2020).

27. Schwalbe, G. & Schels, M. Concept enforcement and modularization as methods for the iso 26262 safety argumentation of neural networks (2020).

28. Lundberg, S. M. & Lee, S.-I. A unified approach to interpreting model predictions. In *NeurIPS* (2017).

29. Sundararajan, M., Taly, A. & Yan, Q. Axiomatic attribution for deep networks. In *International Conference on Machine Learning* (2017).

30. Fong, R. & Vedaldi, A. Interpretable explanations of black boxes by meaningful perturbation. *CoRR* **abs/1704.03296** (2017). 1704.03296.

31. Selvaraju, R. R. *et al.* Grad-cam: Visual explanations from deep networks via gradient-based localization. *Int. conference on computer vision* (2017).

32. Dabkowski, P. & Gal, Y. Real Time Image Saliency for Black Box Classifiers. In *Advances in Neural Information Processing Systems 30 (NIPS)* (2017).

33. Adebayo, J. *et al.* Sanity checks for saliency maps. In *Advances in Neural Information Processing Systems*, 9525–9536 (2018).

34. Kindermans, P.-J. *et al.* The (un) reliability of saliency methods. *arXiv preprint arXiv:1711.00867* (2017).

35. Alvarez-Melis, D. & Jaakkola, T. S. On the robustness of interpretability methods. *arXiv preprint arXiv:1806.08049* (2018).

36. Srinivas, S. & Fleuret, F. Gradient alignment in deep neural networks. *CoRR* **abs/2006.09128** (2020). 2006.09128.

37. Adebayo, J., Muelly, M., Liccardi, I. & Kim, B. Debugging tests for model explanations. *CoRR* **abs/2011.05429** (2020). 2011.05429.

38. Carter, S., Armstrong, Z., Schubert, L., Johnson, I. & Olah, C. Activation atlas. *Distill* DOI: 10.23915/distill.00015 (2019). Https://distill.pub/2019/activation-atlas.

39. Olah, C. *et al.* Zoom in: An introduction to circuits. *Distill* DOI: 10.23915/distill.00024.001 (2020). Https://distill.pub/2020/circuits/zoom-in.

40. Cammarata, N. *et al.* Thread: Circuits. *Distill* DOI: 10.23915/distill.00024 (2020). Https://distill.pub/2020/circuits.

41. Cai, C. J. *et al.* Human-centered tools for coping with imperfect algorithms during medical decision-making. *CoRR* **abs/1902.02960** (2019). 1902.02960.

42. Mahinpei, A., Clark, J., Lage, I., Doshi-Velez, F. & Pan, W. Promises and pitfalls of black-box concept learning models. *CoRR* **abs/2106.13314** (2021). 2106.13314.

43. Goyal, Y., Feder, A., Shalit, U. & Kim, B. Explaining classifiers with causal concept effect (CaCE). *arXiv preprint arXiv:1907.07165* (2019).

44. Heuillet, A., Couthouis, F. & Díaz-Rodríguez, N. Explainability in deep reinforcement learning (2020). 2008.06693.

45. Raffin, A. *et al.* Decoupling feature extraction from policy learning: assessing benefits of state representation learning in goal based robotics (2019). 1901.08651.

46. Raffin, A. *et al.* S-rl toolbox: Environments, datasets and evaluation metrics for state representation learning (2018). 1809.09369.

47. Lesort, T., Seurin, M., Li, X., Díaz-Rodríguez, N. & Filliat, D. Deep unsupervised state representation learning with robotic priors: a robustness analysis. In *2019 International Joint Conference on Neural Networks (IJCNN)*, 1–8, DOI: 10.1109/IJCNN.2019.8852042 (2019).

48. Traoré, R. *et al.* Discorl: Continual reinforcement learning via policy distillation (2019). 1907.05855.

49. Doncieux, S. *et al.* Dream architecture: a developmental approach to open-ended learning in robotics (2020). 2005.06223.

50. Doncieux, S. *et al.* Open-ended learning: A conceptual framework based on representational redescription. *Front. Neurorobotics* **12**, 59, DOI: 10.3389/fnbot.2018.00059 (2018).

51. Sreedharan, S., Srivastava, S. & Kambhampati, S. Tldr: Policy summarization for factored ssp problems using temporal abstractions. In *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 30, 272–280 (2020).

52. Garnelo, M., Arulkumaran, K. & Shanahan, M. Towards deep symbolic reinforcement learning (2016). 1609.05518.

53. d'Avila Garcez, A., Dutra, A. R. R. & Alonso, E. Towards symbolic reinforcement learning with common sense (2018). 1804.08597.

54. Zambaldi, V. *et al.* Relational deep reinforcement learning (2018). 1806.01830.

55. Madumal, P., Miller, T., Sonenberg, L. & Vetere, F. Explainable reinforcement learning through a causal lens (2019). 1905.10958.

56. Juozapaitis, Z., Koul, A., Fern, A., Erwig, M. & Doshi-Velez, F. Explainable reinforcement learning via reward decomposition. In *in proceedings at the International Joint Conference on Artificial Intelligence. A Workshop on Explainable Artificial Intelligence.* (2019).

57. van Seijen, H. *et al.* Hybrid reward architecture for reinforcement learning (2017). 1706.04208.

58. Kawano, H. Hierarchical sub-task decomposition for reinforcement learning of multi-robot delivery mission. In *2013 IEEE International Conference on Robotics and Automation*, 828–835, DOI: 10.1109/ICRA.2013.6630669 (2013).

59. Selvaraju, R. R. *et al.* Grad-cam: Visual explanations from deep networks via gradient-based localization. *Int. J. Comput. Vis.* **128**, 336–359, DOI: 10.1007/s11263-019-01228-7 (2019).

60. Greydanus, S., Koul, A., Dodge, J. & Fern, A. Visualizing and understanding atari agents. In *International Conference on Machine Learning*, 1792–1801 (PMLR, 2018).

61. Mundhenk, T. N., Chen, B. Y. & Friedland, G. Efficient saliency maps for explainable ai (2020). 1911.11293.

62. Koul, A., Greydanus, S. & Fern, A. Learning finite state representations of recurrent policy networks. *arXiv preprint arXiv:1811.12530* (2018).

63. Zahavy, T., Ben-Zrihem, N. & Mannor, S. Graying the black box: Understanding DQNs. In *International Conference on Machine Learning*, 1899–1908 (PMLR, 2016).

64. Sequeira, P. & Gervasio, M. Interestingness elements for explainable reinforcement learning: Understanding agents' capabilities and limitations. *Artif. Intell.* **288**, 103367, DOI: 10.1016/j.artint.2020.103367 (2020).

65. Kasparov, G. Chess, a drosophila of reasoning. *Science* **362**, 1087–1087, DOI: 10.1126/science.aaw2221 (2018). https://science.sciencemag.org/content/362/6419/1087.full.pdf.

66. LCZero Development Community. Leela Chess Zero. lczero.org (2018). Accessed: 20 November 2019.

67. Scott Zoldi. Kasparov on AI: Why Is Explainable AI Important? https://www.fico.com/blogs/index.php/kasparov-ai-why-explainable-ai-important-video (2018).

68. Tomašev, N., Paquet, U., Hassabis, D. & Kramnik, V. Assessing game balance with alphazero: Exploring alternative rule sets in chess (2020). 2009.04374.

69. Kerner, Y. Case-based evaluation in computer chess. In Haton, J.-P., Keane, M. & Manago, M. (eds.) *Advances in Case-Based Reasoning*, 240–254 (Springer Berlin Heidelberg, Berlin, Heidelberg, 1995).

70. HaCohen-Kerner, Y. Learning strategies for explanation patterns: Basic game patterns with application to chess. In *ICCBR* (1995).

71. Puri, N. *et al.* Explain your move: Understanding agent actions using specific and relevant feature attribution (2020). 1912.12191.

72. Louedec, J. L., Guntz, T., Crowley, J. L. & Vaufreydaz, D. Deep learning investigation for chess player attention prediction using eye-tracking and game data. In *Proceedings of the 11th ACM Symposium on Eye Tracking Research & Applications*, ETRA '19, DOI: 10.1145/3314111.3319827 (Association for Computing Machinery, New York, NY, USA, 2019).

73. McIlroy-Young, R., Wang, R., Sen, S., Kleinberg, J. & Anderson, A. Learning personalized models of human behavior in chess (2021). 2008.10086.

74. McIlroy-Young, R., Sen, S., Kleinberg, J. & Anderson, A. Aligning superhuman ai with human behavior: Chess as a model system. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, 1677–1687, DOI: 10.1145/3394486.3403219 (Association for Computing Machinery, New York, NY, USA, 2020).

75. Czech, J., Willig, M., Beyer, A., Kersting, K. & Fürnkranz, J. Learning to play the chess variant Crazyhouse above world champion level with deep neural networks and human data. *Front. Artif. Intell.* **3**, DOI: 10.3389/frai.2020.00024 (2020).

76. Jhamtani, H., Gangal, V., Hovy, E., Neubig, G. & Berg-Kirkpatrick, T. Learning to generate move-by-move commentary for chess games from large-scale social forum data. In *The 56th Annual Meeting of the Association for Computational Linguistics (ACL)* (Melbourne, Australia, 2018).

77. Kamlish, I., Chocron, I. B. & McCarthy, N. Sentimate: Learning to play chess through natural language processing (2019). 1907.08321.

78. Cirik, V., Morency, L.-P. & Hovy, E. Chess q & a : Question answering on chess games (2015).

79. DecodeChess. Understanding Chess with Explainable AI. https://decodechess.com/about/, (2017).

80. He, K., Zhang, X., Ren, S. & Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 770–778 (2016).

81. Sadler, M. & Regan, N. *Game Changer: AlphaZero's Groundbreaking Chess Strategies and the Promise of AI* (New In Chess, 2019).

82. Xu, Y., Zhao, S., Song, J., Stewart, R. & Ermon, S. A theory of usable information under computational constraints. In *International Conference on Learning Representations* (2019).

83. Van der Maaten, L. & Hinton, G. Visualizing data using t-sne. *J. machine learning research* **9** (2008).

84. Veness, J. *et al.* Gated linear networks. *arXiv preprint arXiv:1910.01526* (2019).

85. Schulz, K., Sixt, L., Tombari, F. & Landgraf, T. Restricting the flow: Information bottlenecks for attribution. *arXiv preprint arXiv:2001.00396* (2020).

86. Voita, E. & Titov, I. Information-theoretic probing with minimum description length. *arXiv preprint arXiv:2003.12298* (2020).

87. Pimentel, T. & Cotterell, R. A bayesian framework for information-theoretic probing. *arXiv preprint arXiv:2109.03853* (2021).

88. Bahadori, M. T. & Heckerman, D. Debiasing concept-based explanations with causal analysis. In *International Conference on Learning Representations* (2020).

89. ChessBase. Chessbase Mega Database. database.chessbase.com (2021).

90. Hilton, J., Cammarata, N., Carter, S., Goh, G. & Olah, C. Understanding rl vision. *Distill* **5**, e29 (2020).

91. Leavitt, M. L. & Morcos, A. Towards falsifiable interpretability research. *arXiv preprint arXiv:2010.12016* (2020).

92. Sundararajan, M., Taly, A. & Yan, Q. Axiomatic attribution for deep networks. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, 3319–3328 (2017).

Concept names	**Description**
`material [t] [mg|eg|ph]`	Material score, where each piece on the board has a predefined value that changes depending on the phase of the game.
`imbalance [t] [mg|eg|ph]`	Imbalance score that compares the piece count of each piece type for both colours. E.g., it awards having a pair of bishops vs a bishop and a knight.
`pawns [t] [mg|eg|ph]`	Evaluation of the pawn structure. E.g., the evaluation considers isolated, double, connected, backward, blocked, weak, etc. pawns.
`knights [t] [mg|eg|ph]`	Evaluation of knights. E.g., extra points are given to knights that occupy outposts protected by pawns.
`bishops [t] [mg|eg|ph]`	Evaluation of bishops. E.g., bishops that occupy the same color squares as pawns are penalised.
`rooks [t] [mg|eg|ph]`	Evaluation of rooks. E.g., rooks that occupy open or semi-open files have higher valuation.
`queens [t] [mg|eg|ph]`	Evaluation of queens. E.g., queens that have relative pin or discovered attack against them are penalized.
`mobility [t] [mg|eg|ph]`	Evaluation of piece mobility score. It depends on the number of squares attacked by the pieces.
`king_safety [t] [mg|eg|ph]`	A complex concept related to king safety. It depends on the number and type of pieces that attack squares around the king, shelter strength, number of pawns around the king, penalties for being on pawnless flank, etc.
`threats [t] [mg|eg|ph]`	Evaluation of threats to pieces, such as whether a pawn can safely advance and attack an opponent's higher value piece, hanging pieces, possible xray attacks by rooks, etc.
`passed_pawns [t] [mg|eg|ph]`	Evaluates bonuses for passed pawns. The closer a pawn is to the promotion rank, the higher is the bonus.
`space [t] [mg|eg|ph]`	Evaluation of the space. It depends on the number of safe squares available for minor pieces on the central four files on ranks 2 to 4.
`total [t] [mg|eg|ph]`	The total evaluation of a given position. It encapsulates all the above concepts.

Table 2. A summary of 93 concepts from Stockfish 8's public API. The concepts are enumerated as `<concept_name>_<side>_<game_phase>`, where the side is `[w|b|t]` for White, Black, total (difference) respectively. The game phase abbreviations `[mg|eg|ph]` stands for middle game, end game and phased value respectively. The phased value is a weighted sum of the middle and end game values based on the actual phase of a given position. As AlphaZero represents positions from the playing side's view, "side" is orientated and represented as `mine` and `opponent` instead of `w` and `b`.

A Full concept list

Table 2 summarizes 93 concepts from Stockfish 8's public API. The concepts are taken from Stockfish 8 and not a later version, as there is a wealth of observations on the difference between AlphaZero and Stockfish 8 [81]. We implemented a long list of additional concepts programatically, and these are presented in Tables 3 and 4.

Concept names	**Description**
`pawn_fork [m|o]`	True if a pawn is attacking two pieces of higher value (knight, bishop, rook, queen, or king) and is not pinned.
`knight_fork [m|o]`	True if a knight is attacking two pieces of higher value (rook, queen, or king) and is not pinned.
`bishop_fork [m|o]`	True if a bishop is attacking two pieces of higher value (rook, queen, or king) and is not pinned.
`rook_fork [m|o]`	True if a rook is attacking two pieces of higher value (queen, or king) and is not pinned.
`has_pinned_pawn [m|o]`	True if the side has a pawn that is pinned to the king of the same colour.
`has_pinned_knight [m|o]`	True if the side has a knight that is pinned to the king of the same colour.
`has_pinned_bishop [m|o]`	True if the side has a bishop that is pinned to the king of the same colour.
`has_pinned_rook [m|o]`	True if the side has a rook that is pinned to the king of the same colour.
`has_pinned_queen [m|o]`	True if the side has a queen that is pinned to the king of the same colour.
`material [m|o|diff]`	Material calculated as (#♙) + 3 * (#♘) + 3 * (#♗) + 5 * (#♖) + 9 * (#♕)
`num_pieces [m|o|diff]`	Number of pieces that a side has.
`in_check`	True if the side that makes a turn is in check.
`has_bishop_pair [m|o]`	True if the side has a pair of bishops.
`has_connected_rooks [m|o]`	True if the side has connected rooks.
`has_control_of_open_file [m|o]`	True if the side controls an open file (with the rooks, queen)
`has_mate_threat`	True if the opponent could mate the current side in a single move if the turn was passed to the opponent.
`has_check_move [m|o]`	True if the side can check the opponent's King.
`can_capture_queen [m|o]`	True if the side can capture the opponent's queen.
`num_king_attacked_squares [m|o|diff]`	The number of squares around the opponent's king that the playing side attacks. Can include occupied squares.
`has_contested_open_file`	True if an open file is occupied simultaneously by a rook and/or queen of both colours.
`has_right_bc_ha_promotion [m|o]`	True if 1) the side has a passed pawn on a or h files and 2) the side has a bishop that is of the colour of the promotion square of that pawn.
`num_scb_pawns_same_side [m|o|diff]`	The number of own pawns that occupy squares of the same colour as the colour of own bishop. Applicable only when the side has a single bishop.
`num_ocb_pawns_same_side [m|o|diff]`	The number of own pawns that occupy squares of the opposite colour to that of own bishop. Applicable only when the side has a single bishop.
`num_scb_pawns_other_side [m|o|diff]`	The number of opponent's pawns that occupy the squares of the same colour as the colour of own bishop. Applicable only when the side has a single bishop.
`num_ocb_pawns_other_side [m|o|diff]`	The number of opponent's pawns that occupy the squares of the opposite colour to the colour of own bishop. Applicable only when the side has a single bishop.

`capture_possible_on_{sq} [m|o]`
`sq=[d1|d2|d3|e1|e2|e3|g5|b5]`
	True is the side can capture a piece on the given square. The squares are named as if the side were playing White.

`capture_happens_next_move_...`
`...on_{sq}`
`sq=[d1|d2|d3|e1|e2|e3|g5|b5]`
	True if the capture of a piece on the given square had happened according to the game data. The squares are named as if the side were playing White.

Table 3. Custom chess concepts (self implemented; i.e. not from Stockfish 8's API) used in this paper. We use `m` as shorthand for `mine` and `o` as shorthand for `opponent`. `diff` stands for the `difference` between the mine and opponent values of the same concept.

Concept names	**Description**
`num_double_pawn_files [t] [m|o|diff]`	Number of files that contain one or more pawns of a given colour.
`has_double_pawn [m|o]`	True if a file contains one or more pawns of a given colour.
`num_isolated_pawns [m|o|diff]`	Number of pawns that have no friendly pawns in the files to their left and right.
`has_isolated_pawn [m|o]`	True if there is a pawn that has no pawns in the file to their left or right.
`has_pawn_on_7th_rank [m|o]`	True if the side has a pawn that reached the 7th rank.
`pawns_on_7th_rank [m|o|diff]`	Number of pawns that reached the 7th rank.
`has_passed_pawn [m|o]`	True if the side has a pawn with no opposing pawns to prevent it from advancing to the eighth rank.
`num_passed_pawns [m|o|diff]`	The number of passed pawns.
`has_protected_passed_pawn [m|o]`	True if the side has a passed pawn that is protected by its own pawn.
`num_protected_passed_pawns [m|o|diff]`	The number of protected passed pawns.
`num_pawn_islands [m|o|diff]`	The number of pawn islands.
`has_iqp [m|o]`	True if the side has an isolated queen's pawn (d file).
`has_connected_passed_pawns [m|o]`	True if the side has two or more passed pawns on adjacent files.
`num_connected_passed_pawns [m|o|diff]`	The number of connected passed pawns that the side has.

Table 4. Custom chess concepts related to pawns (self implemented; i.e. not from Stockfish 8's API) used in this paper. We use `m` as shorthand for `mine` and `o` as shorthand for `opponent`. `diff` stands for the `difference` between the mine and opponent values of the same concept.

B Regression results for all concepts

In our experiments, we have considered a large number of potential human chess concepts within a set of concepts we have tried to identify, localize, and explore the acquisition of within the AlphaZero chess model. Even this extended list is far from being able to explicitly capture the vast chess knowledge that has accumulated over centuries and the multitude of patterns that can appear on the board. It is merely a starting point for further exploration.

While we focused our discussion in the main text on a smaller number of relevant human concepts and their regression from different layers in the AlphaZero network over time, we present the results for the extended list of concepts we've used in our experiments here in Figures 14 to 25.

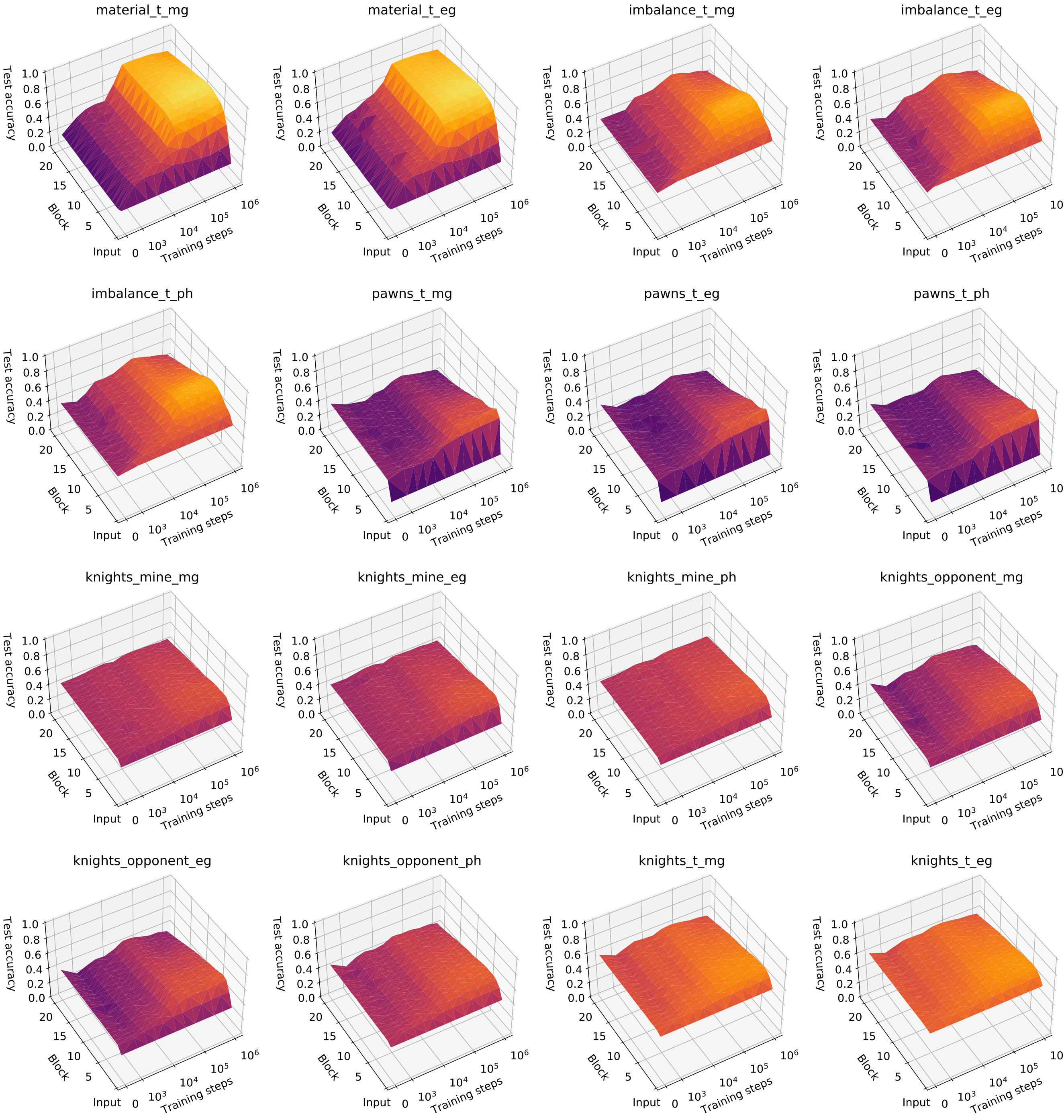
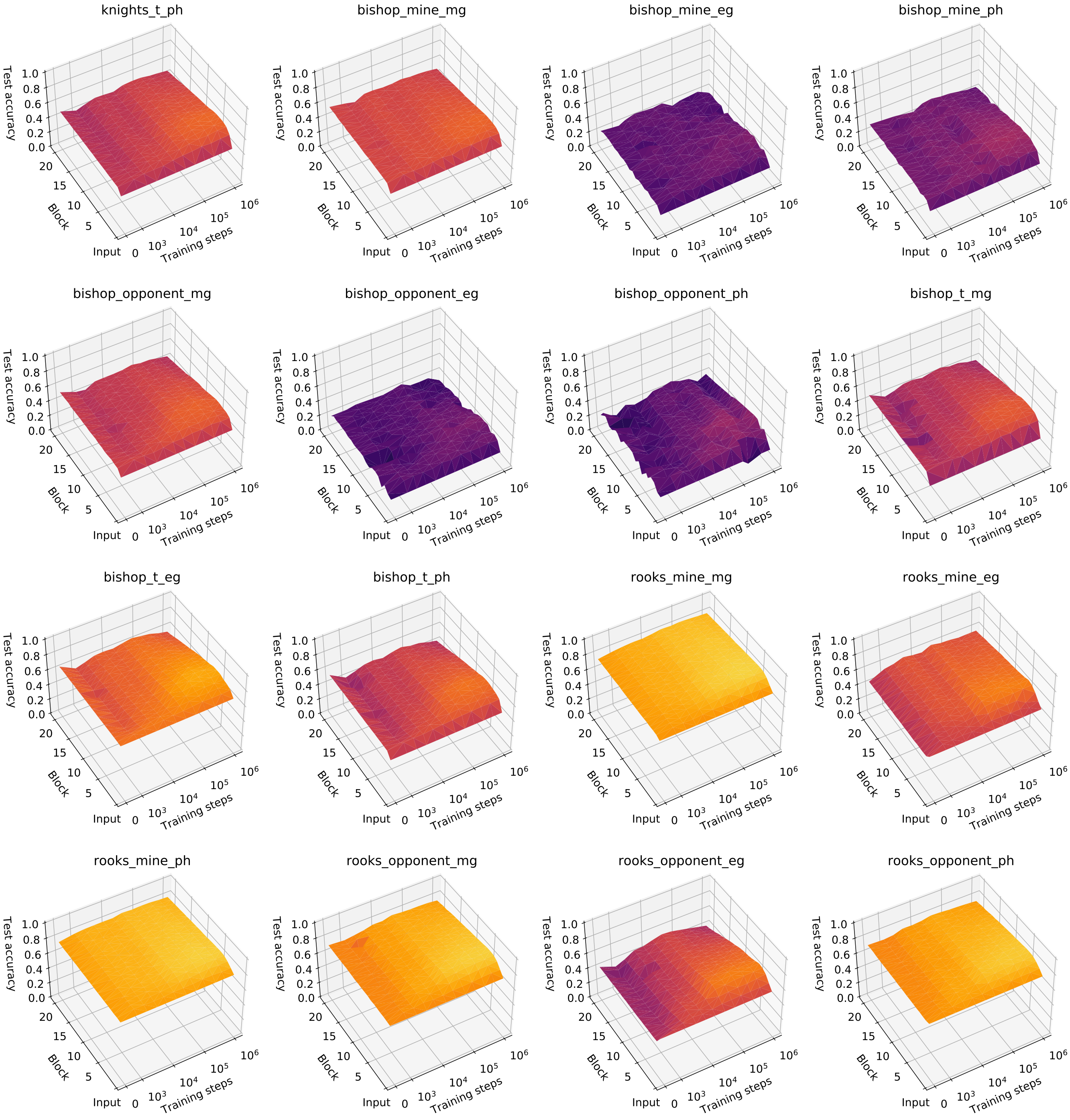


Figure 14. Regression results for Stockfish concepts from Table 2.


knights_t_ph

bishop_mine_mg

bishop_mine_eg

bishop_mine_ph

bishop_opponent_mg

bishop_opponent_eg

bishop_opponent_ph

bishop_t_mg

bishop_t_eg

bishop_t_ph

rooks_mine_mg

rooks_mine_eg

rooks_mine_ph

rooks_opponent_mg

rooks_opponent_eg

rooks_opponent_ph

Figure 15. Regression results for Stockfish concepts from Table 2, continued.

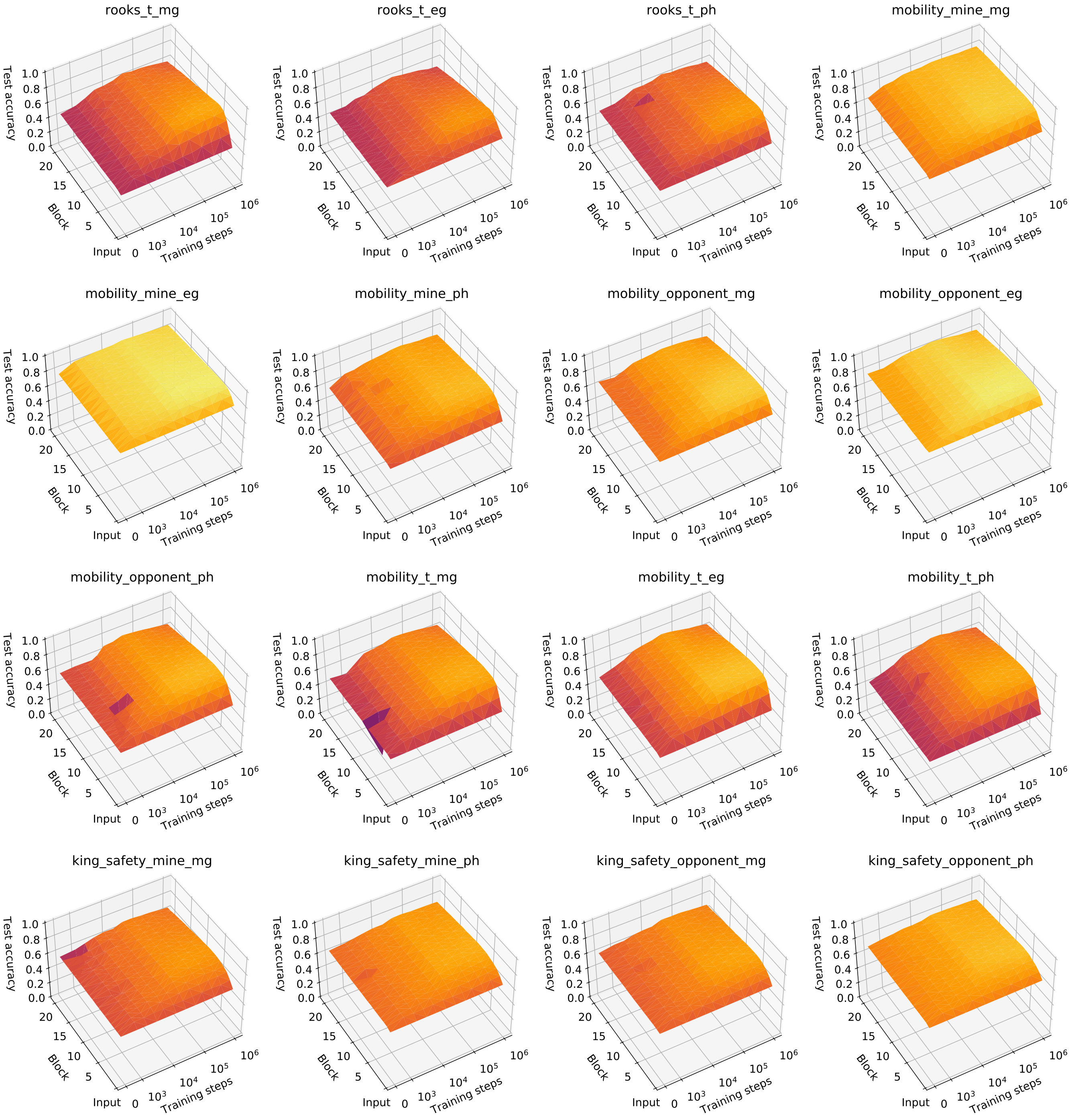


Figure 16. Regression results for Stockfish concepts from Table 2, continued.

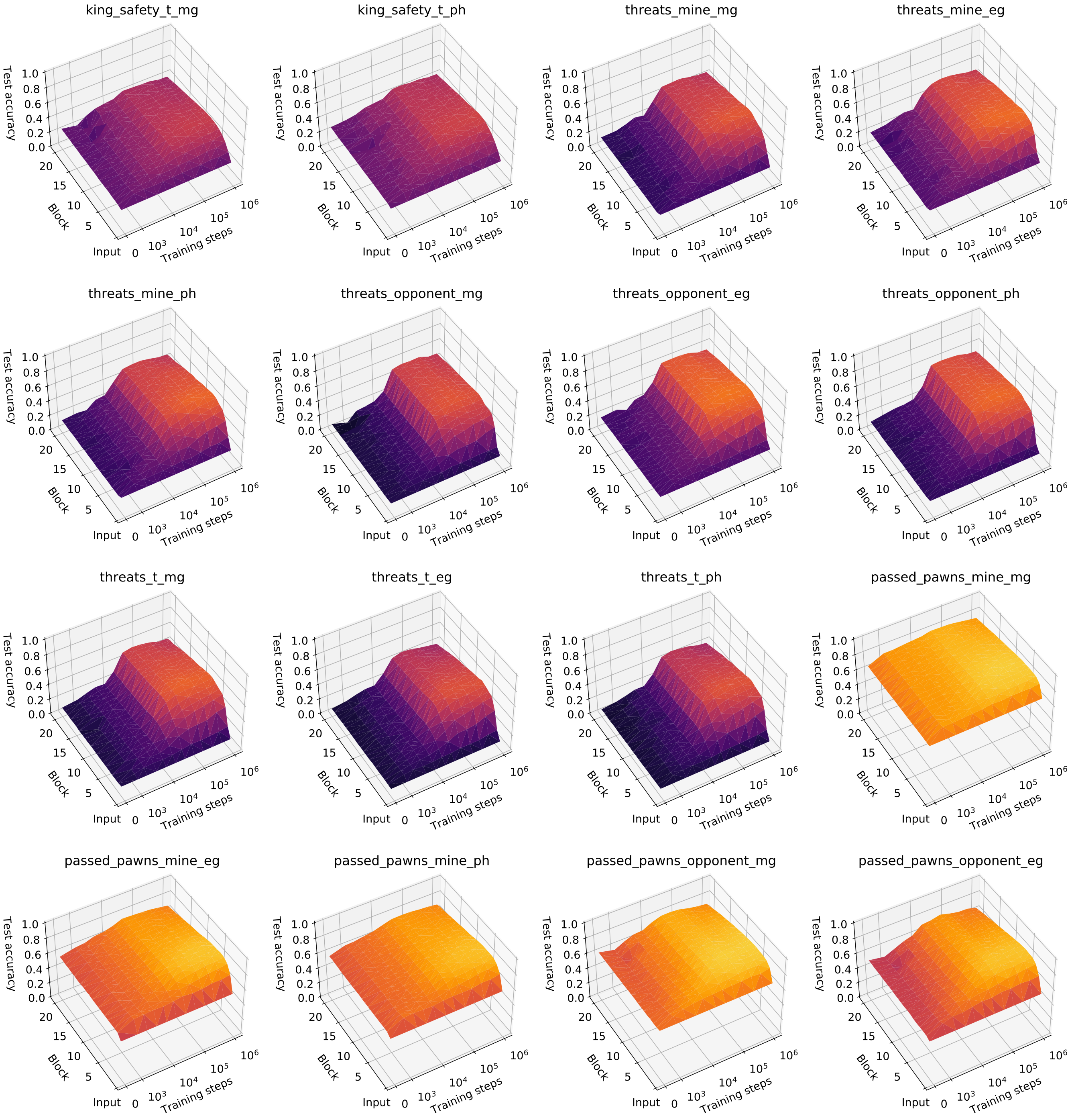


Figure 17. Regression results for Stockfish concepts from Table 2, continued.

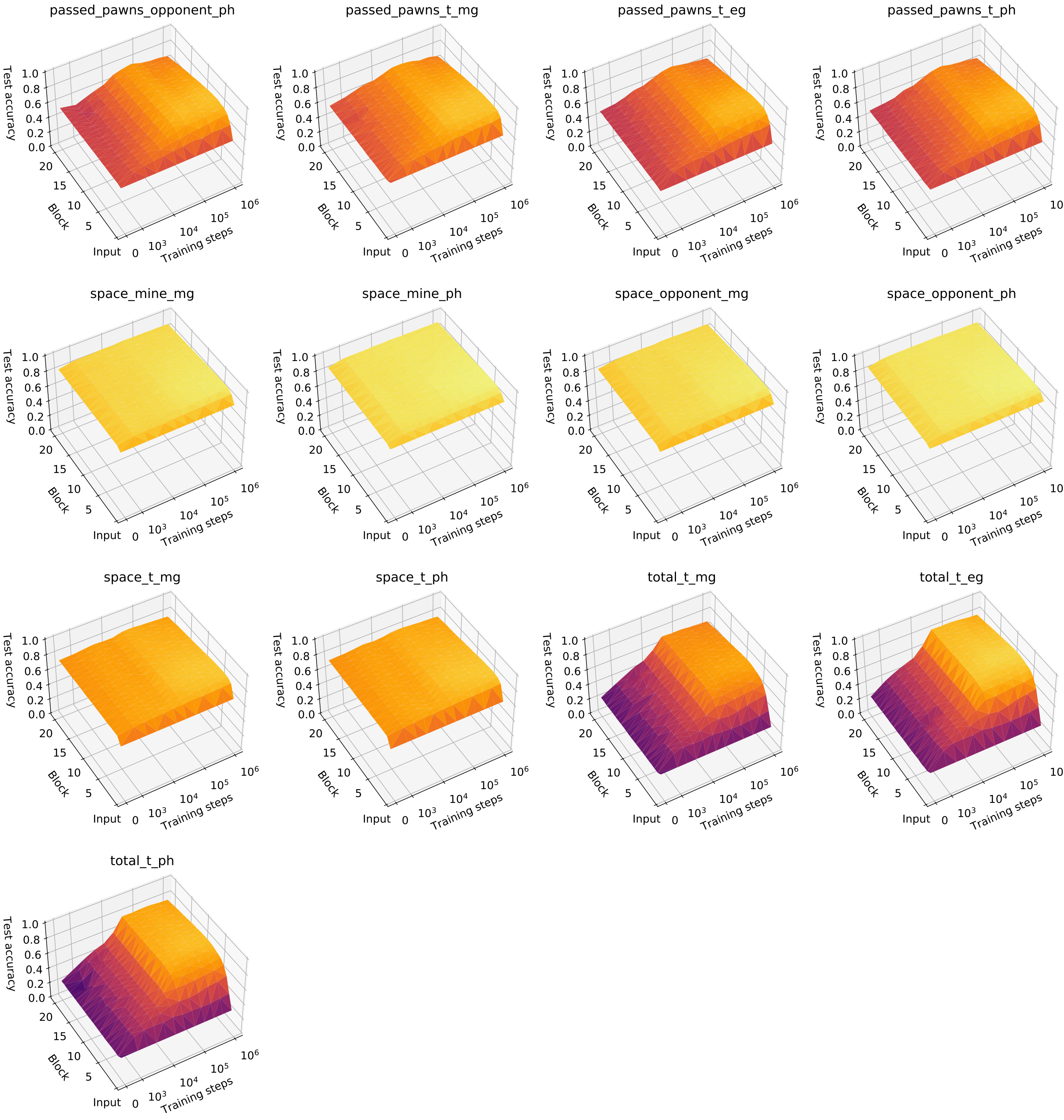


Figure 18. Regression results for Stockfish concepts from Table 2, continued.

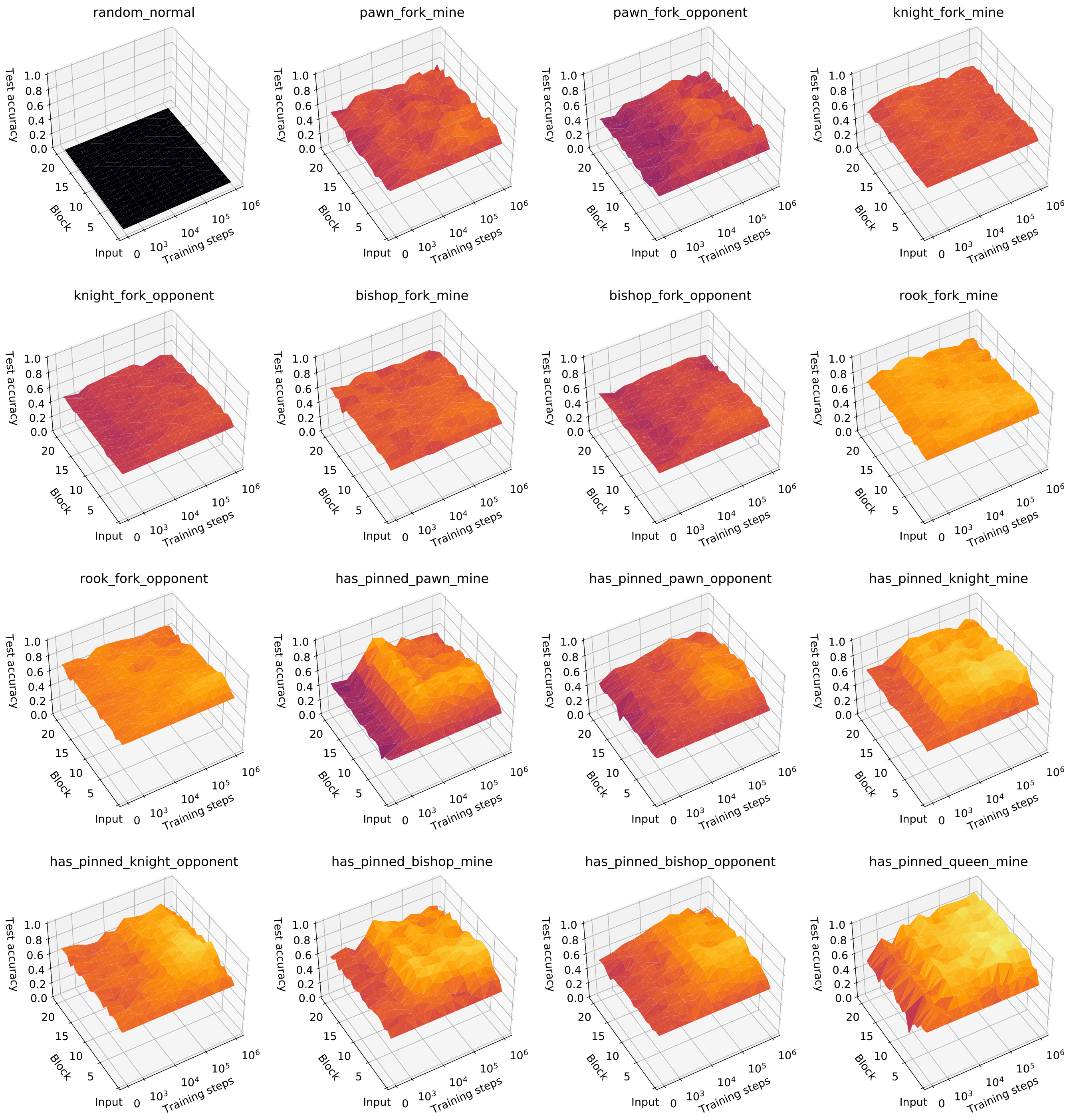


Figure 19. Regression results for custom concepts from Table 3, excluding capture-related concepts.

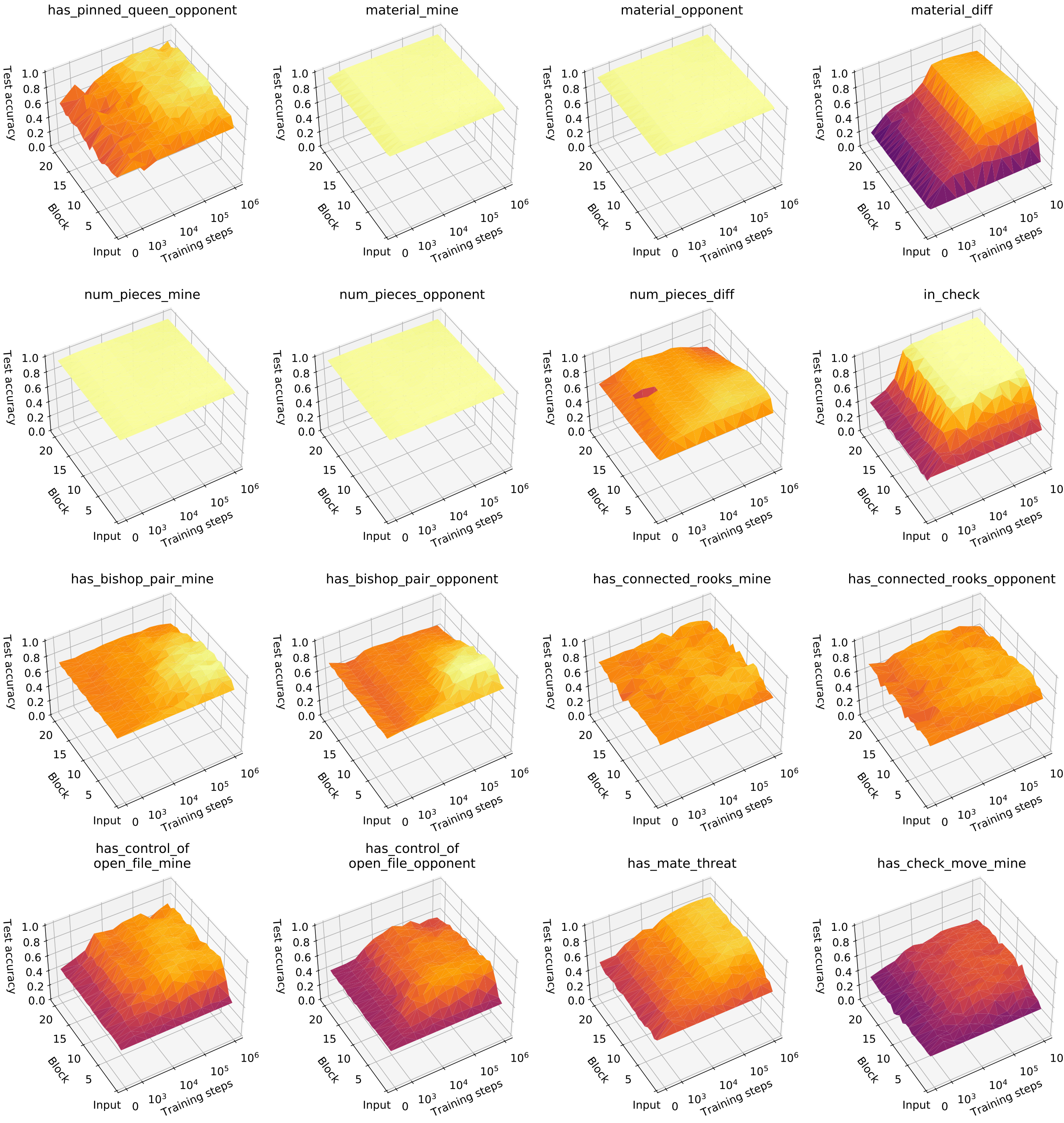
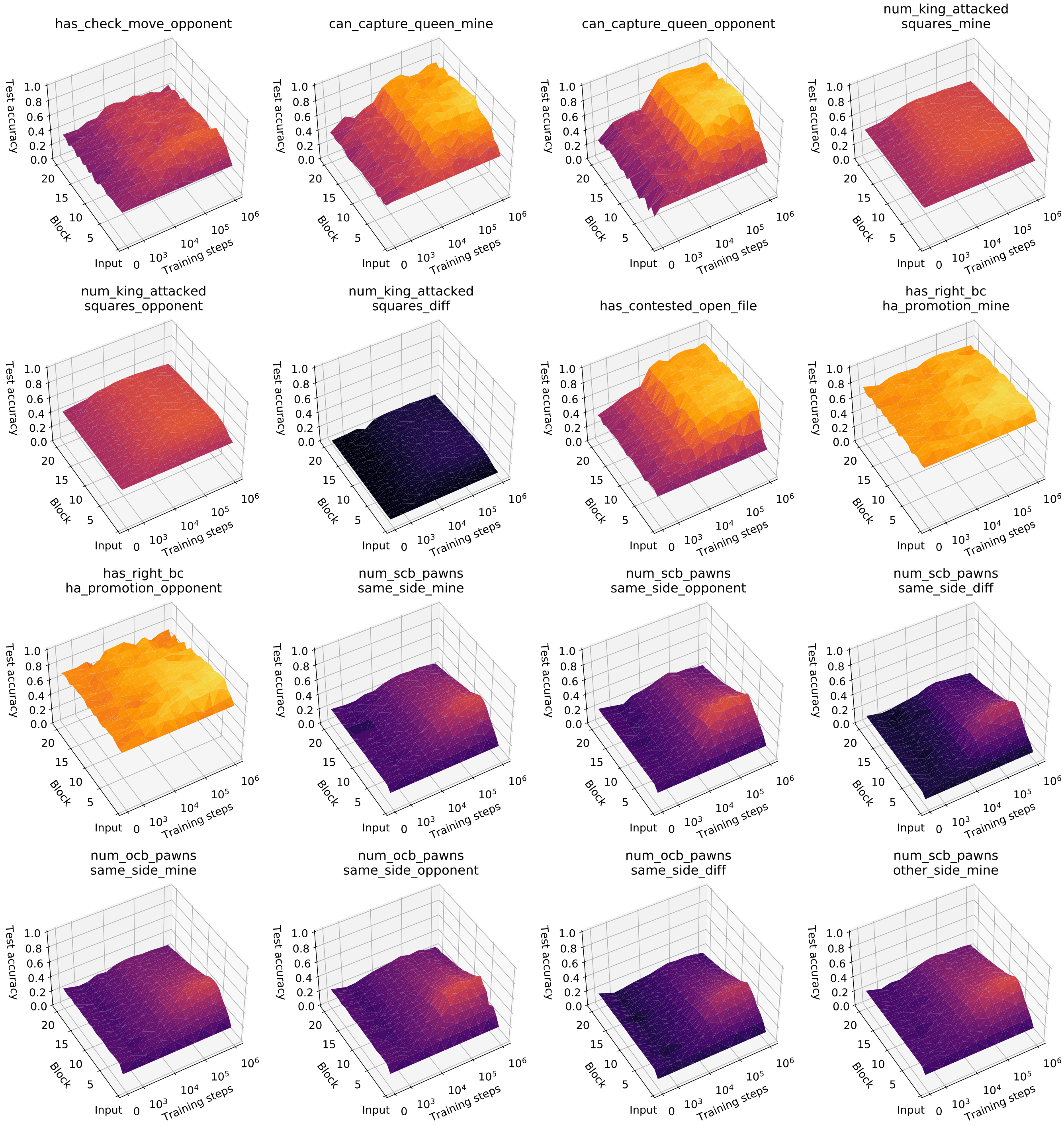


Figure 20. Regression results for custom concepts from Table 3, excluding capture-related concepts, continued.


has_check_move_opponent

can_capture_queen_mine

can_capture_queen_opponent

num_king_attacked squares_mine

num_king_attacked squares_opponent

num_king_attacked squares_diff

has_contested_open_file

has_right_bc ha_promotion_mine

has_right_bc ha_promotion_opponent

num_scb_pawns same_side_mine

num_scb_pawns same_side_opponent

num_scb_pawns same_side_diff

num_ocb_pawns same_side_mine

num_ocb_pawns same_side_opponent

num_ocb_pawns same_side_diff

num_scb_pawns other_side_mine

Figure 21. Regression results for custom concepts from Table 3, excluding capture-related concepts, continued.

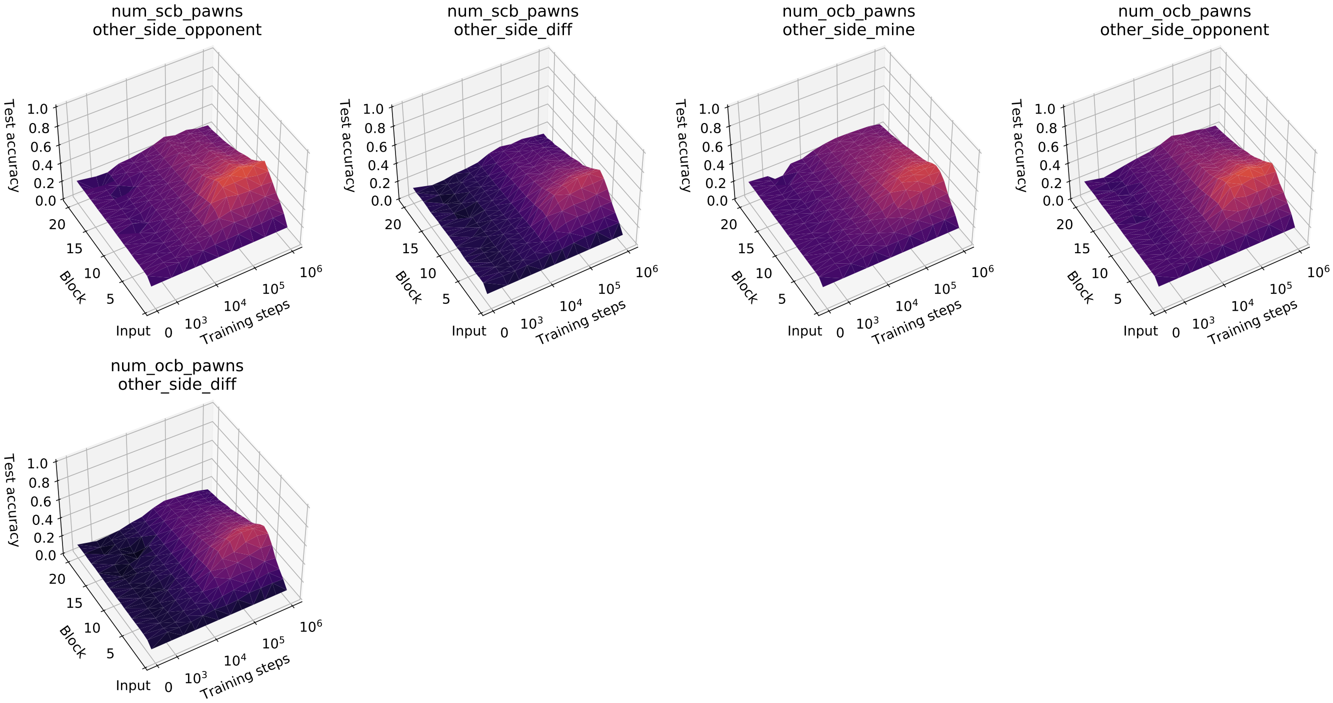


Figure 22. Regression results for custom concepts from Table 3, excluding capture-related concepts, continued.

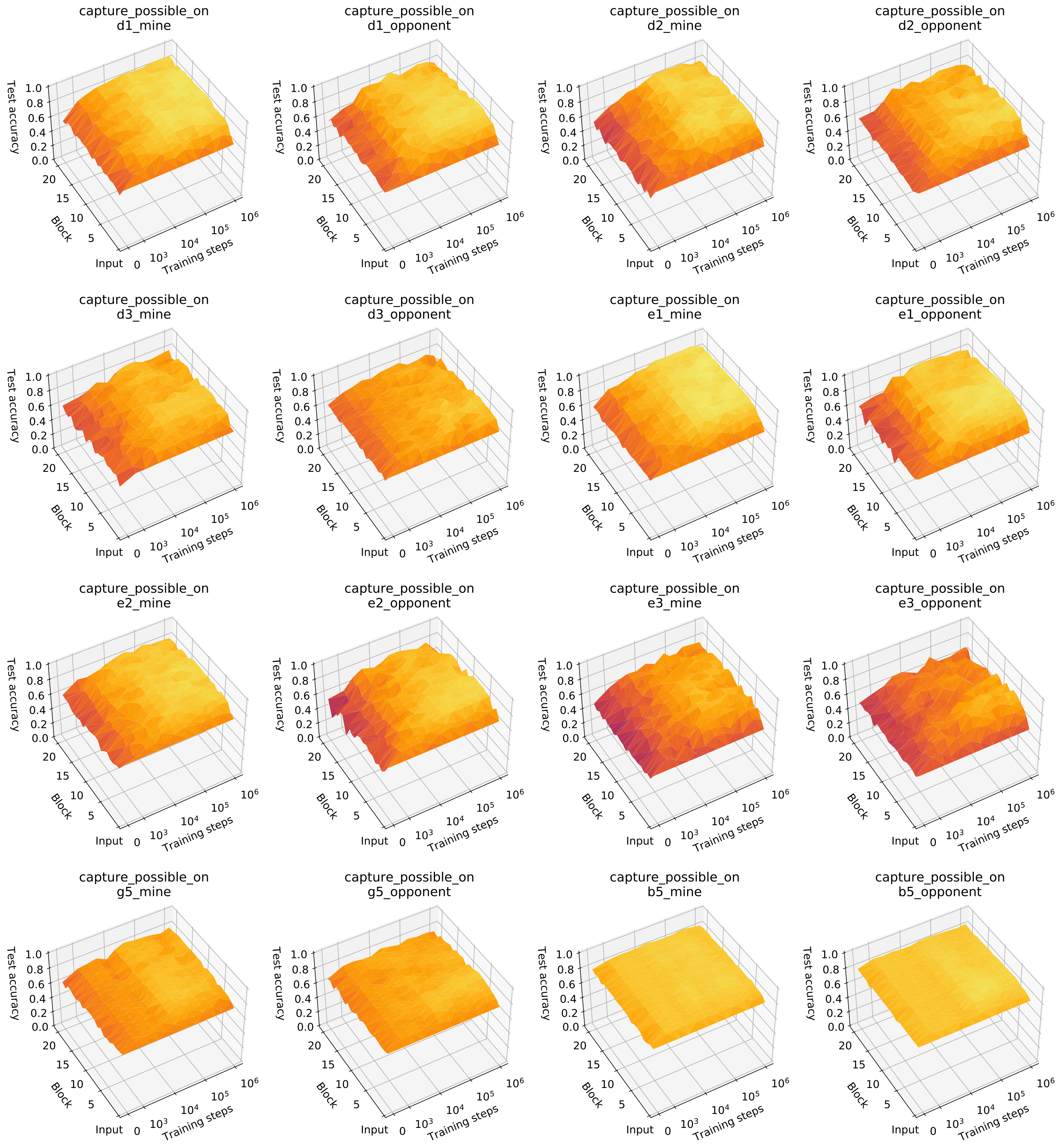


Figure 23. Regression results for custom concepts from Table 3 related to captures.

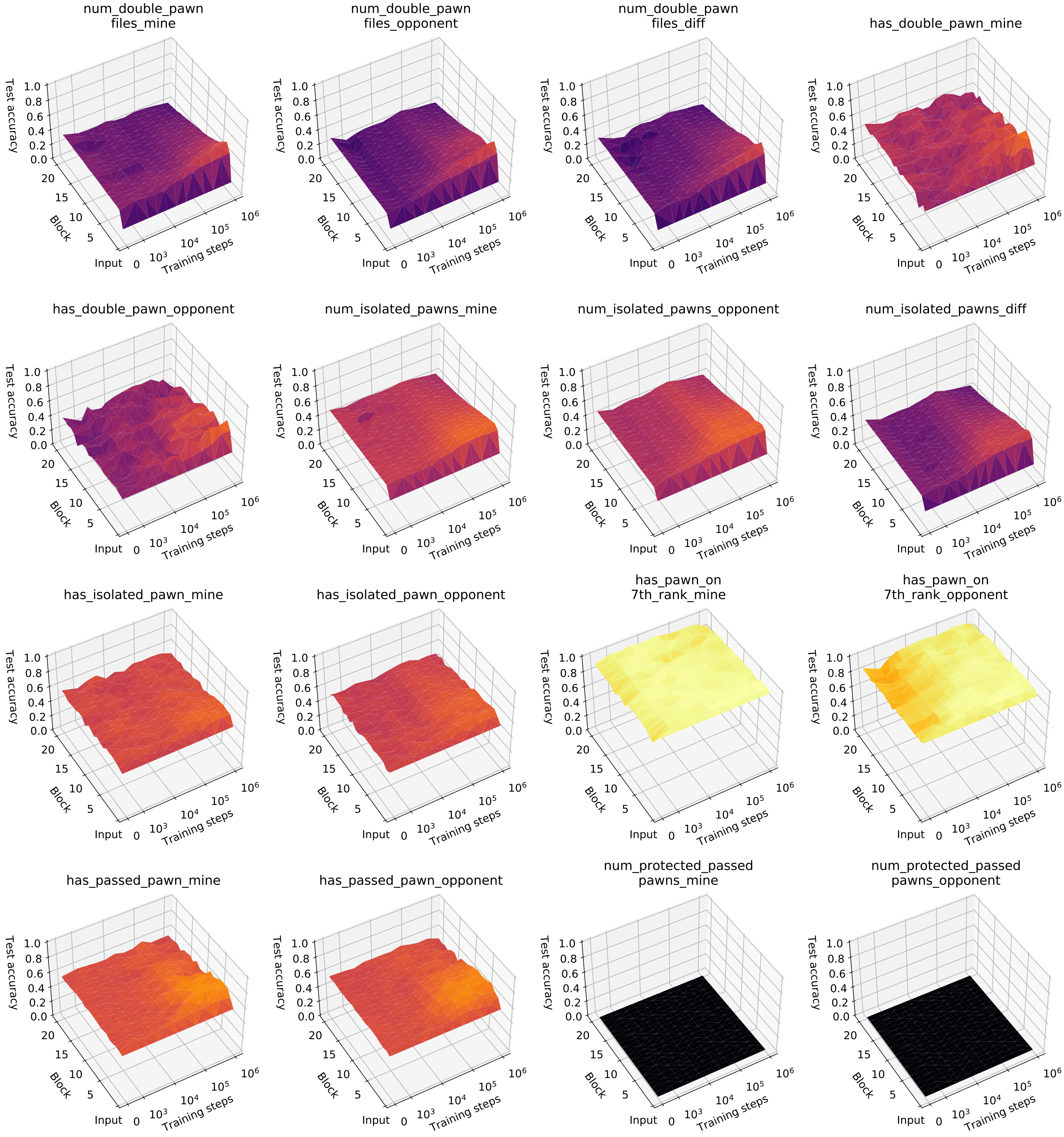


Figure 24. Regression results for custom pawn-related concepts from Table 4.

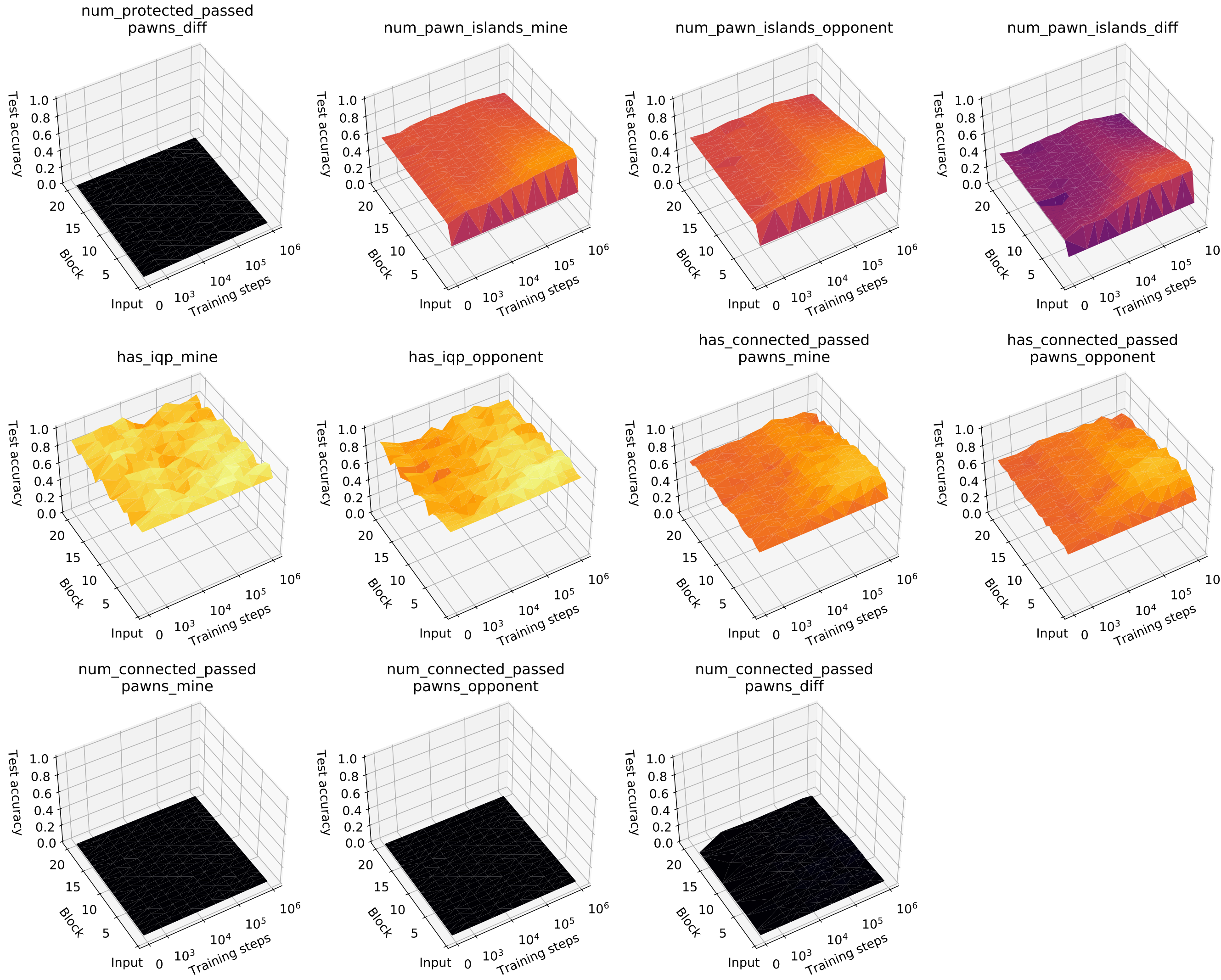
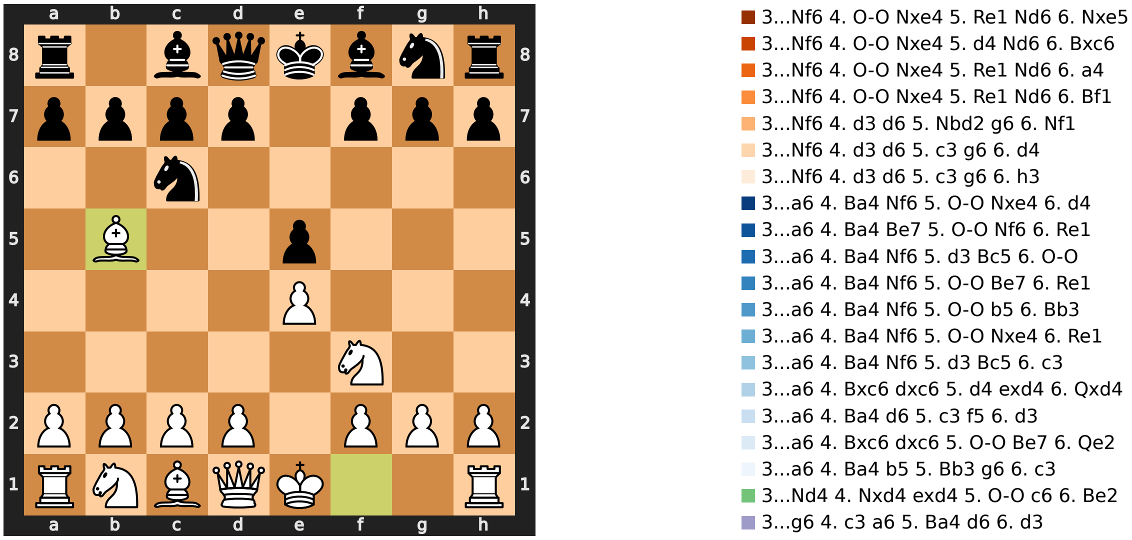


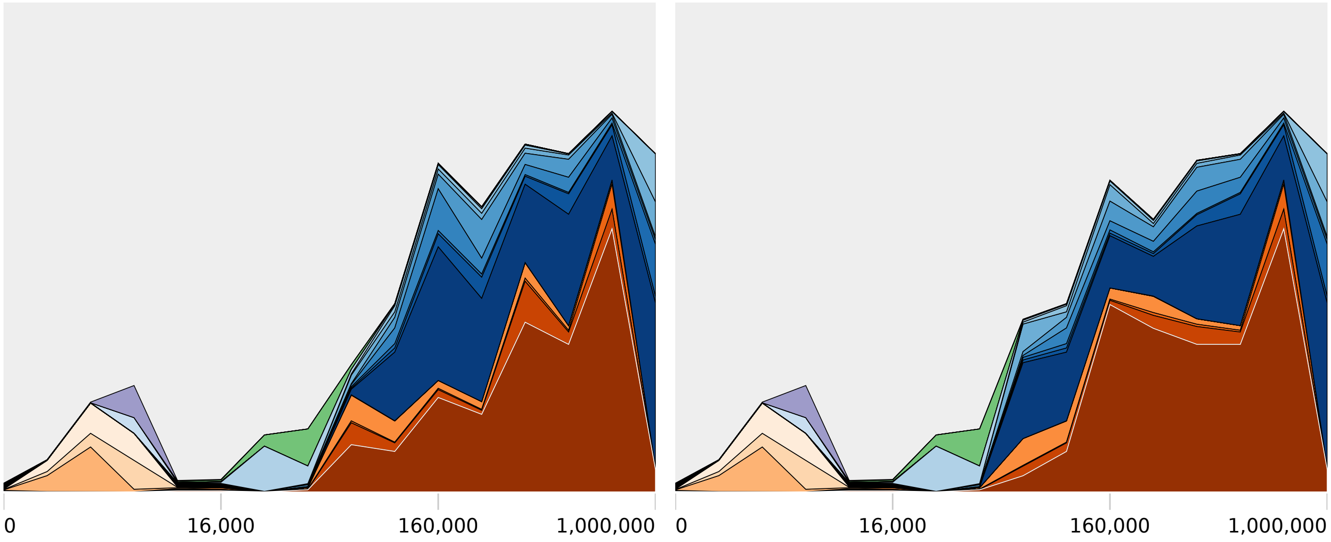
Figure 25. Regression results for custom pawn-related concepts from Table 4, continued.

C AlphaZero policy progression through training for different training seeds

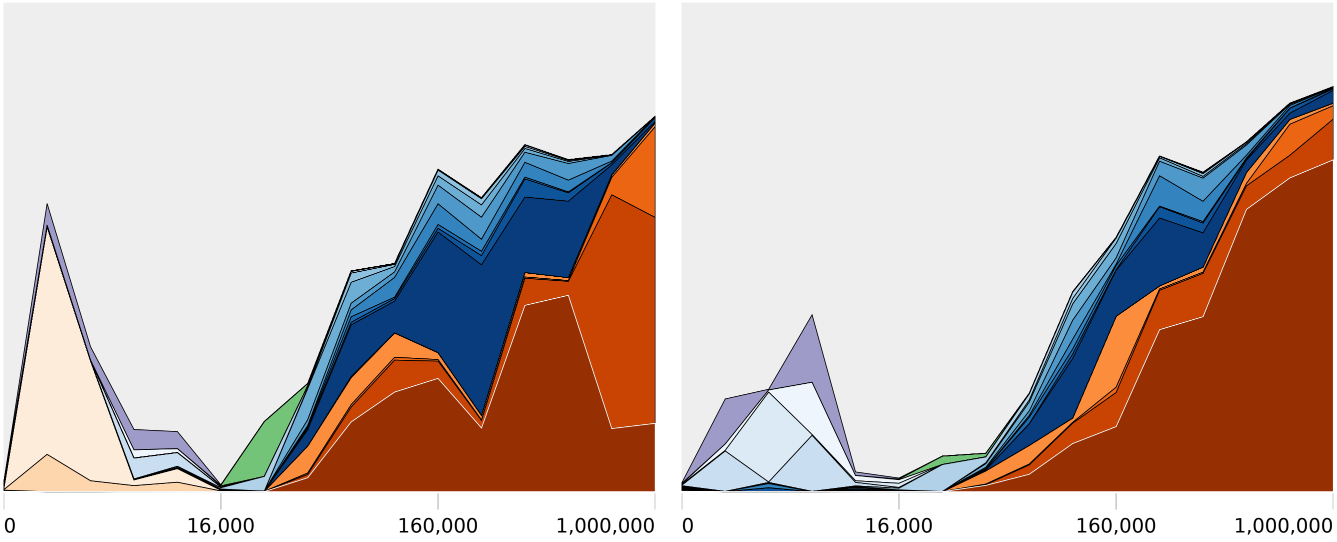
Figures 26 to 31 show the distribution of move sequences with the highest probabilities as predicted by the policy network. Every figure provides the sequence start position (a) and the probability distribution for different initial training seeds (c)-(f). The *x* axis shows the training iteration for the given seed. The white area on the plots show the cumulative probability of all move sequences that are not included in (b). Only moves that were made at least once by human players were considered.


a b c d e f g h
8 7 6 5 4 3 2 1
■ 3...Nf6 4. O-O Nxe4 5. Re1 Nd6 6. Nxe5
■ 3...Nf6 4. O-O Nxe4 5. d4 Nd6 6. Bxc6
■ 3...Nf6 4. O-O Nxe4 5. Re1 Nd6 6. a4
■ 3...Nf6 4. O-O Nxe4 5. Re1 Nd6 6. Bf1
■ 3...Nf6 4. d3 d6 5. Nbd2 g6 6. Nf1
■ 3...Nf6 4. d3 d6 5. c3 g6 6. d4
■ 3...Nf6 4. d3 d6 5. c3 g6 6. h3
■ 3...a6 4. Ba4 Nf6 5. O-O Nxe4 6. d4
■ 3...a6 4. Ba4 Be7 5. O-O Nf6 6. Re1
■ 3...a6 4. Ba4 Nf6 5. d3 Bc5 6. O-O
■ 3...a6 4. Ba4 Nf6 5. O-O Be7 6. Re1
■ 3...a6 4. Ba4 Nf6 5. O-O b5 6. Bb3
■ 3...a6 4. Ba4 Nf6 5. O-O Nxe4 6. Re1
■ 3...a6 4. Ba4 Nf6 5. d3 Bc5 6. c3
■ 3...a6 4. Bxc6 dxc6 5. d4 exd4 6. Qxd4
■ 3...a6 4. Ba4 d6 5. c3 f5 6. d3
■ 3...a6 4. Bxc6 dxc6 5. O-O Be7 6. Qe2
■ 3...a6 4. Ba4 b5 5. Bb3 g6 6. c3
■ 3...Nd4 4. Nxd4 exd4 5. O-O c6 6. Be2
■ 3...g6 4. c3 a6 5. Ba4 d6 6. d3

(a) Sequence start position. **(b)** Legend.

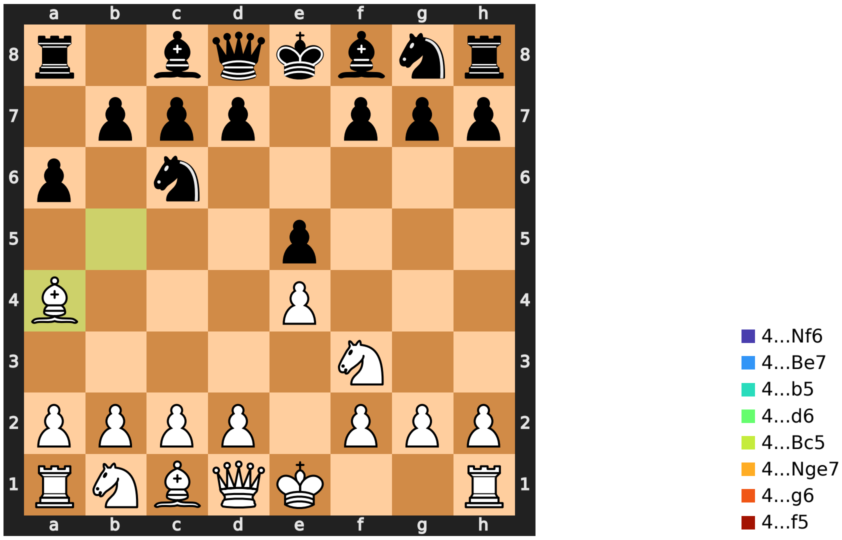


(c) Training seed 1. **(d)** Training seed 2.



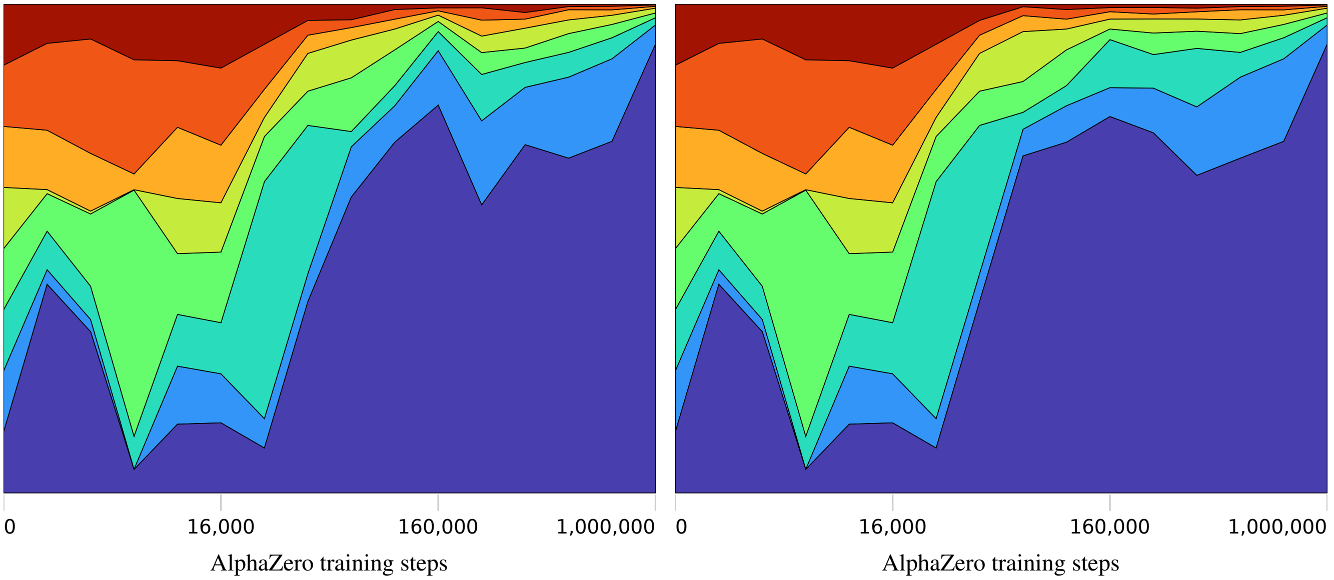
(e) Training seed 3. **(f)** Training seed 4.

Figure 26. Top 20 move sequences with the highest joint probability after 1. e4 e5 2. ♘f3 ♘c6 3. ♗b5.



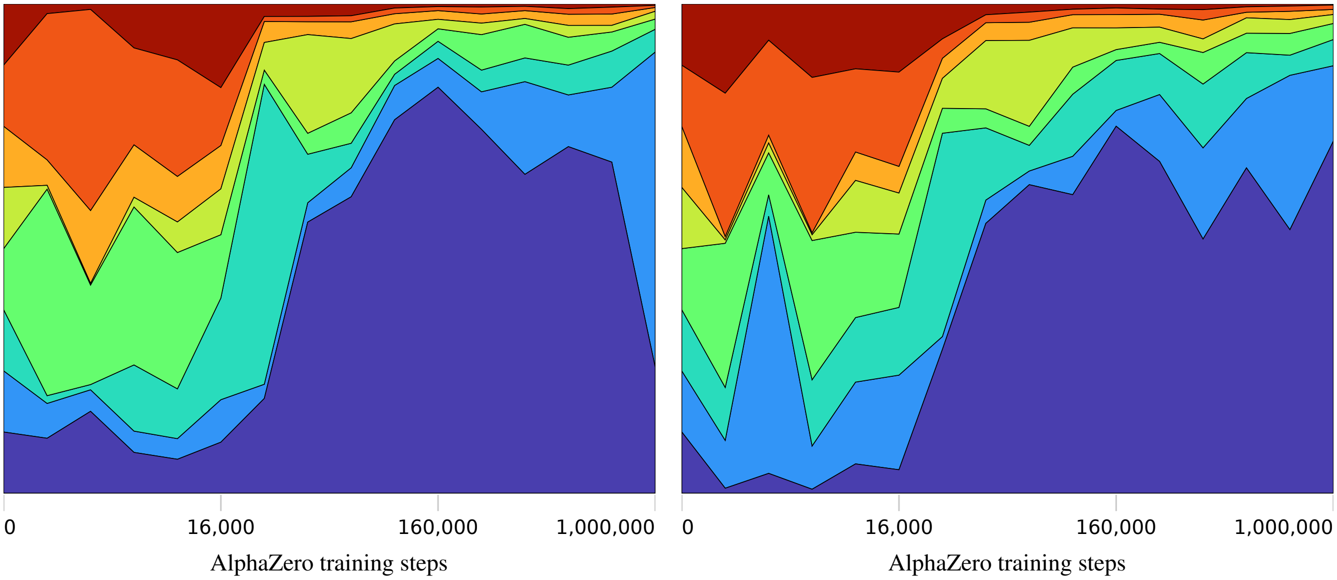
(a) Sequence start position.

(b) Legend.



(c) Training seed 1.

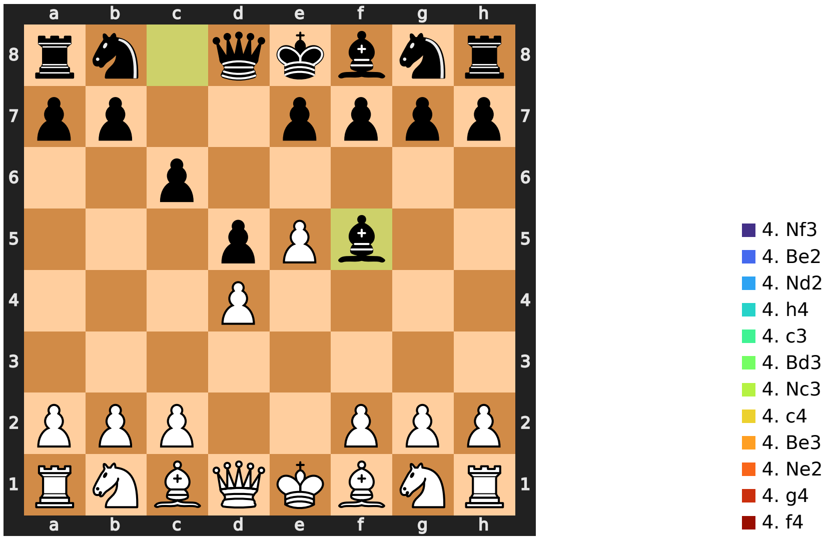
(d) Training seed 2.



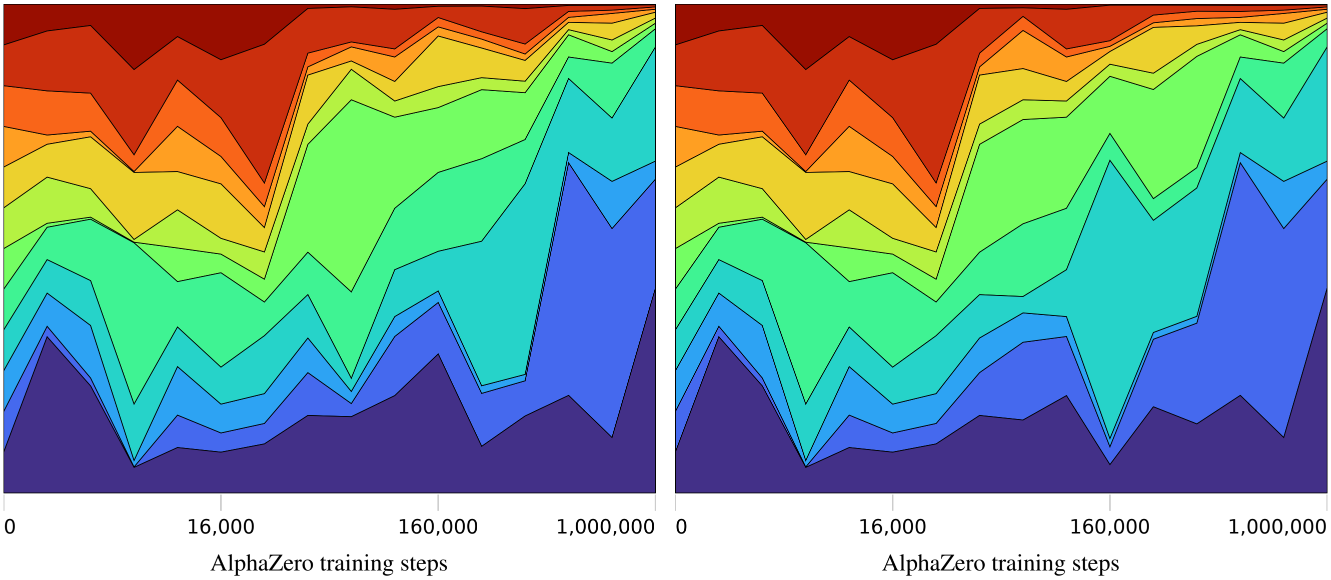
(e) Training seed 3.

(f) Training seed 4.

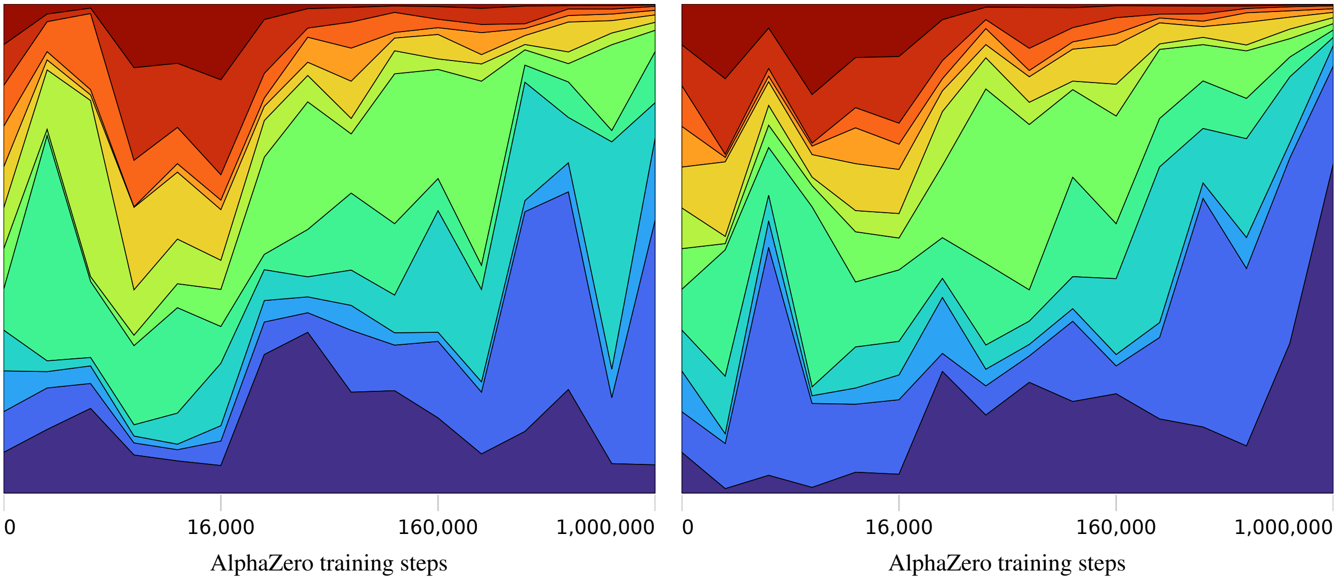
Figure 27. Top 8 moves with the highest probability after 1. e4 e5 2. ♘f3 ♘c6 3. ♗b5 a6 4. ♗a4.



(a) Sequence start position. **(b)** Legend.

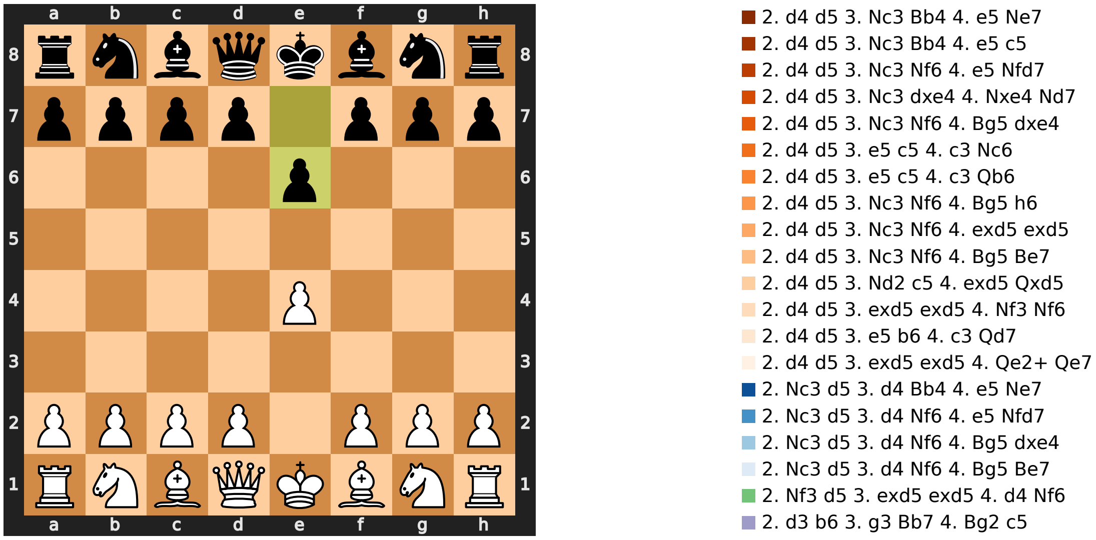


(c) Training seed 1. **(d)** Training seed 2.

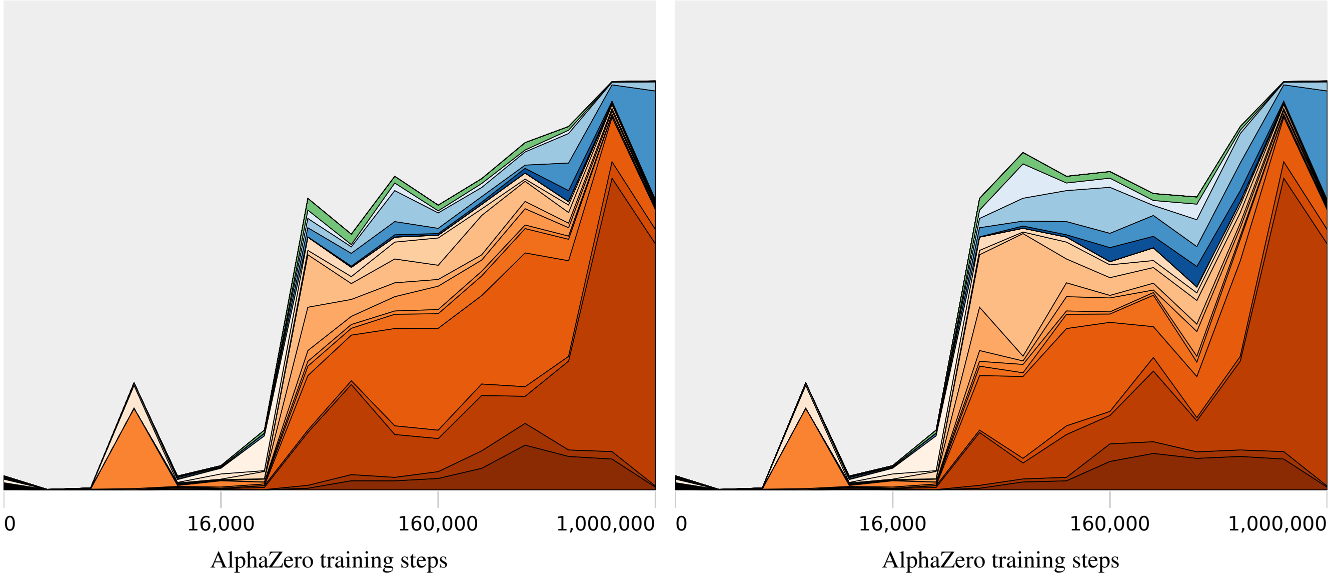


(e) Training seed 3. **(f)** Training seed 4.

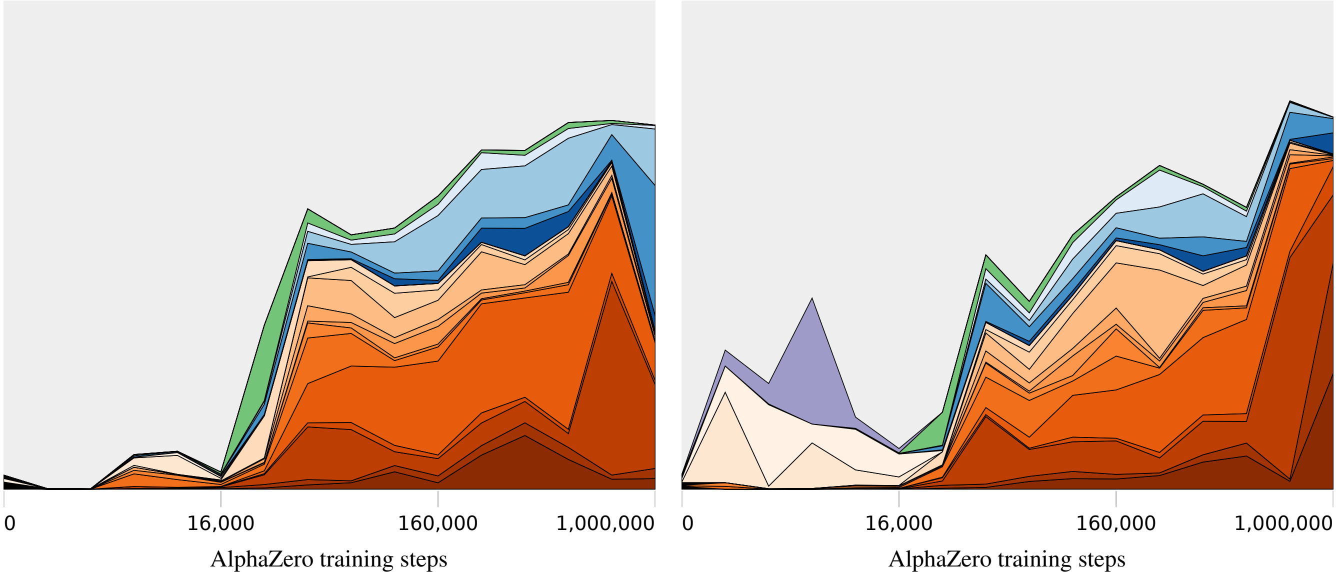
Figure 28. Top 12 moves with the highest probability after 1. e4 c6 2. d4 d5 3. e5 ♗f5.


a b c d e f g h
8 7 6 5 4 3 2 1
2. d4 d5 3. Nc3 Bb4 4. e5 Ne7
2. d4 d5 3. Nc3 Bb4 4. e5 c5
2. d4 d5 3. Nc3 Nf6 4. e5 Nfd7
2. d4 d5 3. Nc3 dxe4 4. Nxe4 Nd7
2. d4 d5 3. Nc3 Nf6 4. Bg5 dxe4
2. d4 d5 3. e5 c5 4. c3 Nc6
2. d4 d5 3. e5 c5 4. c3 Qb6
2. d4 d5 3. Nc3 Nf6 4. Bg5 h6
2. d4 d5 3. Nc3 Nf6 4. exd5 exd5
2. d4 d5 3. Nc3 Nf6 4. Bg5 Be7
2. d4 d5 3. Nd2 c5 4. exd5 Qxd5
2. d4 d5 3. exd5 exd5 4. Nf3 Nf6
2. d4 d5 3. e5 b6 4. c3 Qd7
2. d4 d5 3. exd5 exd5 4. Qe2+ Qe7
2. Nc3 d5 3. d4 Bb4 4. e5 Ne7
2. Nc3 d5 3. d4 Nf6 4. e5 Nfd7
2. Nc3 d5 3. d4 Nf6 4. Bg5 dxe4
2. Nc3 d5 3. d4 Nf6 4. Bg5 Be7
2. Nf3 d5 3. exd5 exd5 4. d4 Nf6
2. d3 b6 3. g3 Bb7 4. Bg2 c5

(a) Sequence start position. **(b)** Legend.

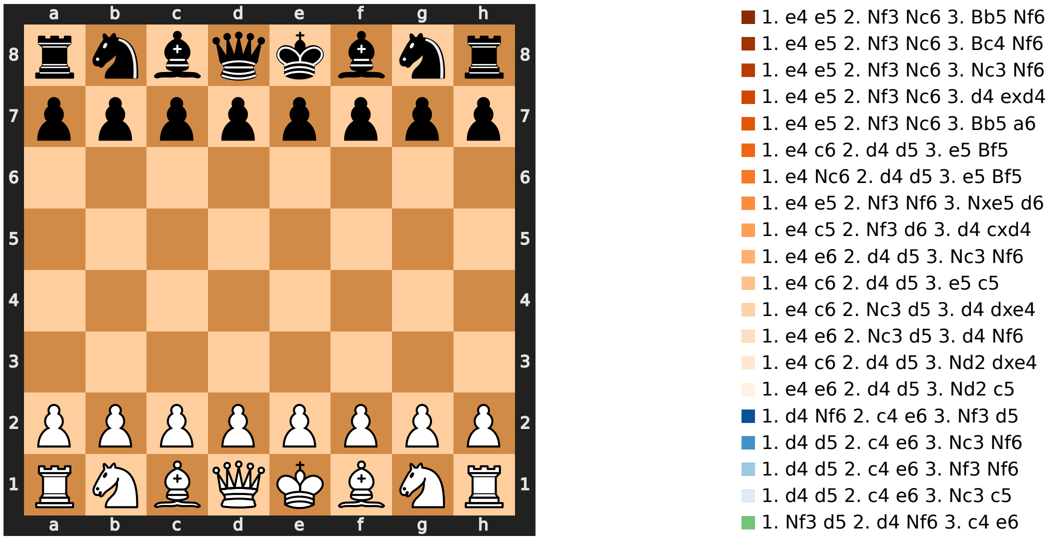


(c) Training seed 1. **(d)** Training seed 2.



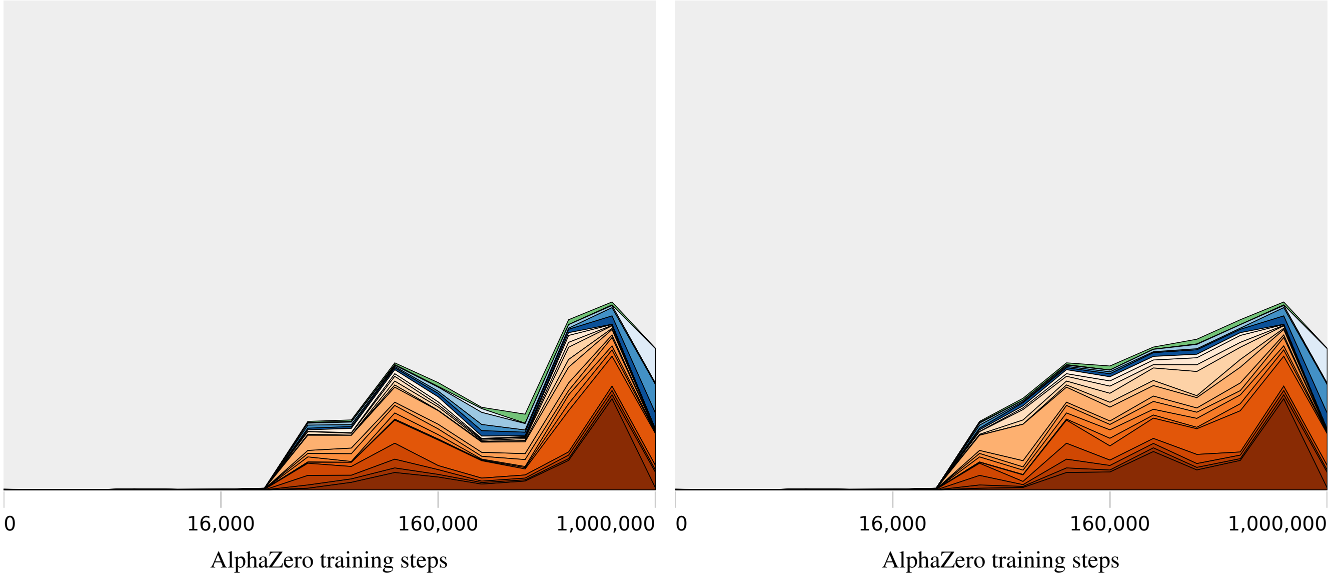
(e) Training seed 3. **(f)** Training seed 4.

Figure 29. Top 20 move sequences with the highest joint probability after 1. e4 e6.


a b c d e f g h
8 7 6 5 4 3 2 1
1. e4 e5 2. Nf3 Nc6 3. Bb5 Nf6
1. e4 e5 2. Nf3 Nc6 3. Bc4 Nf6
1. e4 e5 2. Nf3 Nc6 3. Nc3 Nf6
1. e4 e5 2. Nf3 Nc6 3. d4 exd4
1. e4 e5 2. Nf3 Nc6 3. Bb5 a6
1. e4 c6 2. d4 d5 3. e5 Bf5
1. e4 Nc6 2. d4 d5 3. e5 Bf5
1. e4 e5 2. Nf3 Nf6 3. Nxe5 d6
1. e4 c5 2. Nf3 d6 3. d4 cxd4
1. e4 e6 2. d4 d5 3. Nc3 Nf6
1. e4 c6 2. d4 d5 3. e5 c5
1. e4 c6 2. Nc3 d5 3. d4 dxe4
1. e4 e6 2. Nc3 d5 3. d4 Nf6
1. e4 c6 2. d4 d5 3. Nd2 dxe4
1. e4 e6 2. d4 d5 3. Nd2 c5
1. d4 Nf6 2. c4 e6 3. Nf3 d5
1. d4 d5 2. c4 e6 3. Nc3 Nf6
1. d4 d5 2. c4 e6 3. Nf3 Nf6
1. d4 d5 2. c4 e6 3. Nc3 c5
1. Nf3 d5 2. d4 Nf6 3. c4 e6

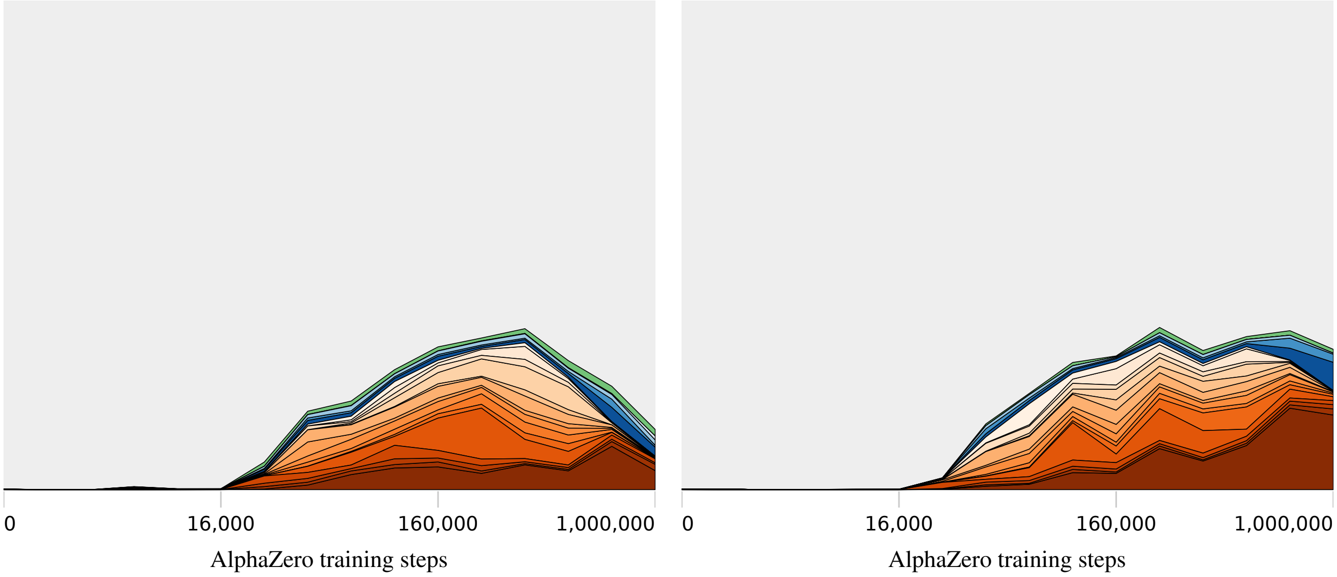
(a) Sequence start position.

(b) Legend.



(c) Training seed 1.

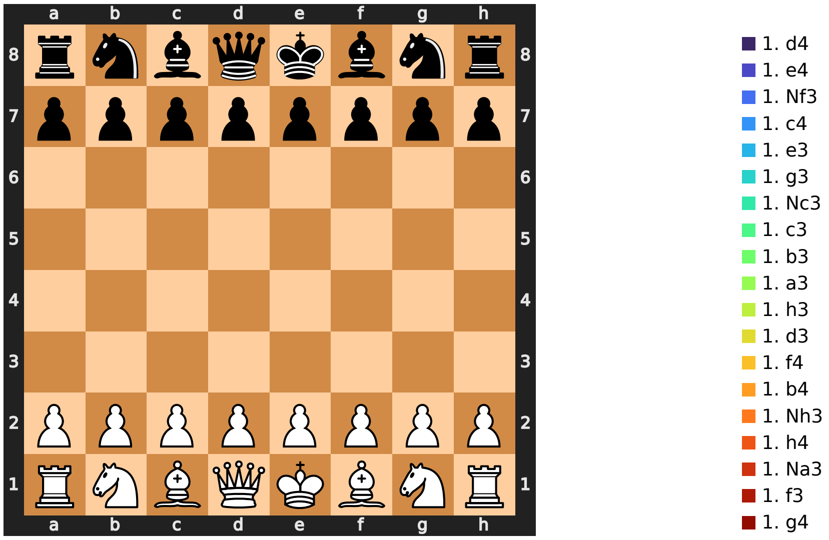
(d) Training seed 2.



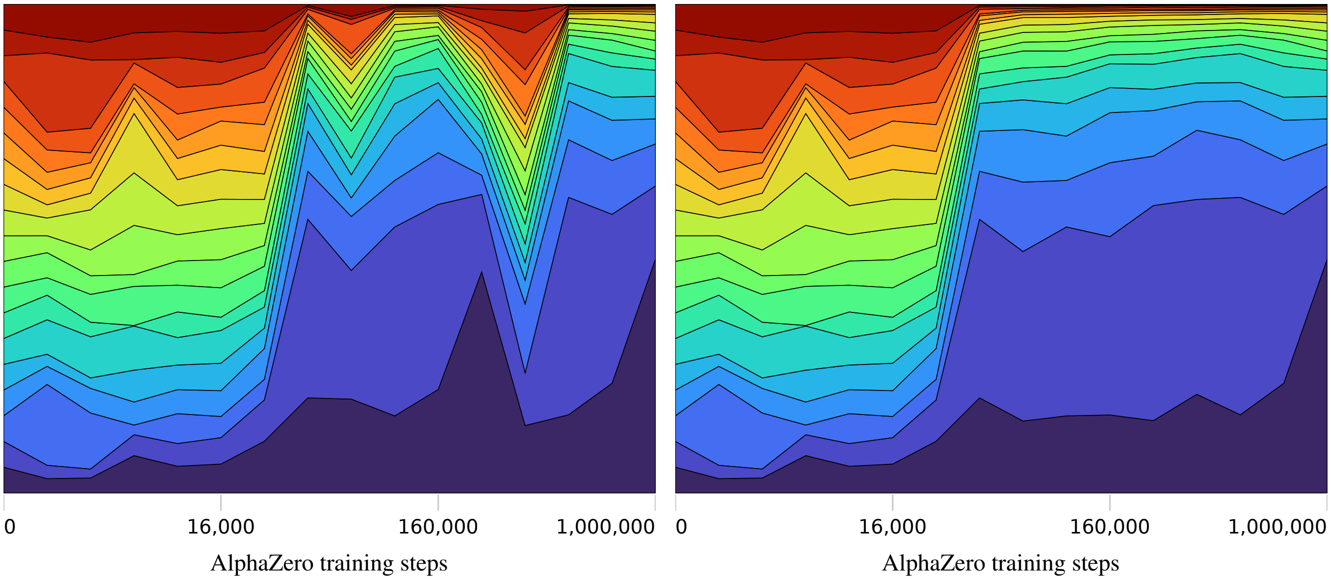
(e) Training seed 3.

(f) Training seed 4.

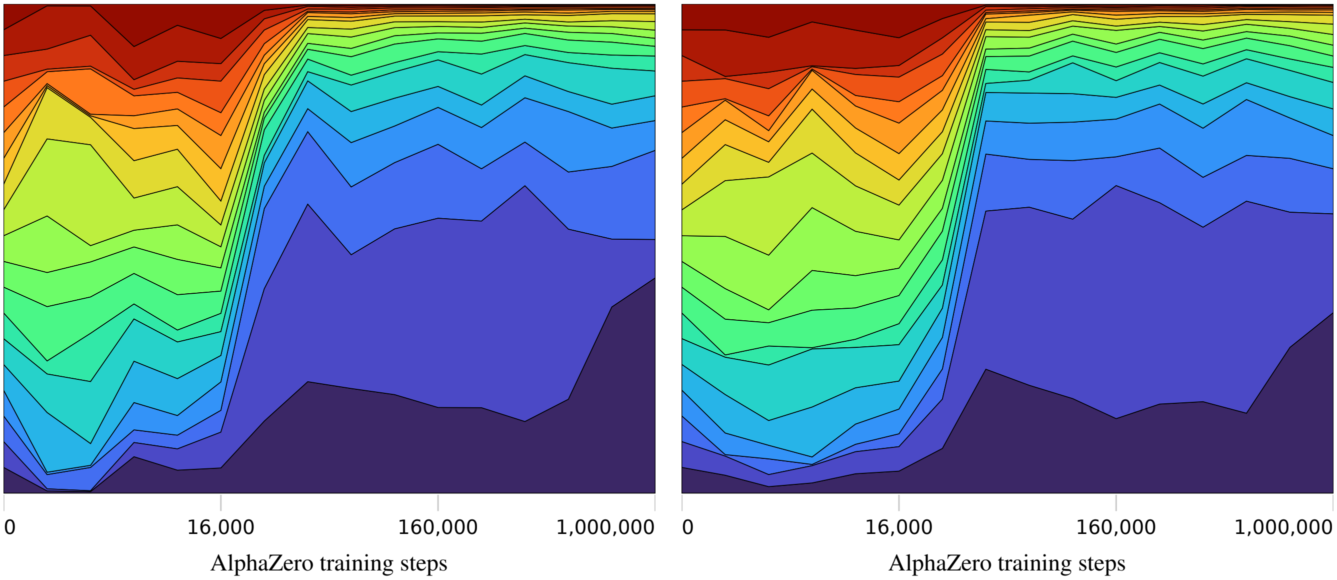
Figure 30. Top 20 move sequences with the highest joint probability at the start position.



(a) Sequence start position. **(b)** Legend.



(c) Training seed 1. **(d)** Training seed 2.



(e) Training seed 3. **(f)** Training seed 4.

Figure 31. Top 19 moves with the highest probability at the start position.

D Outlier positions in score regression

This section shows all the negative outlier positions shown in Figure 3 (those highlighted in red with true value less than zero). Black’s queen can be captured in every position.

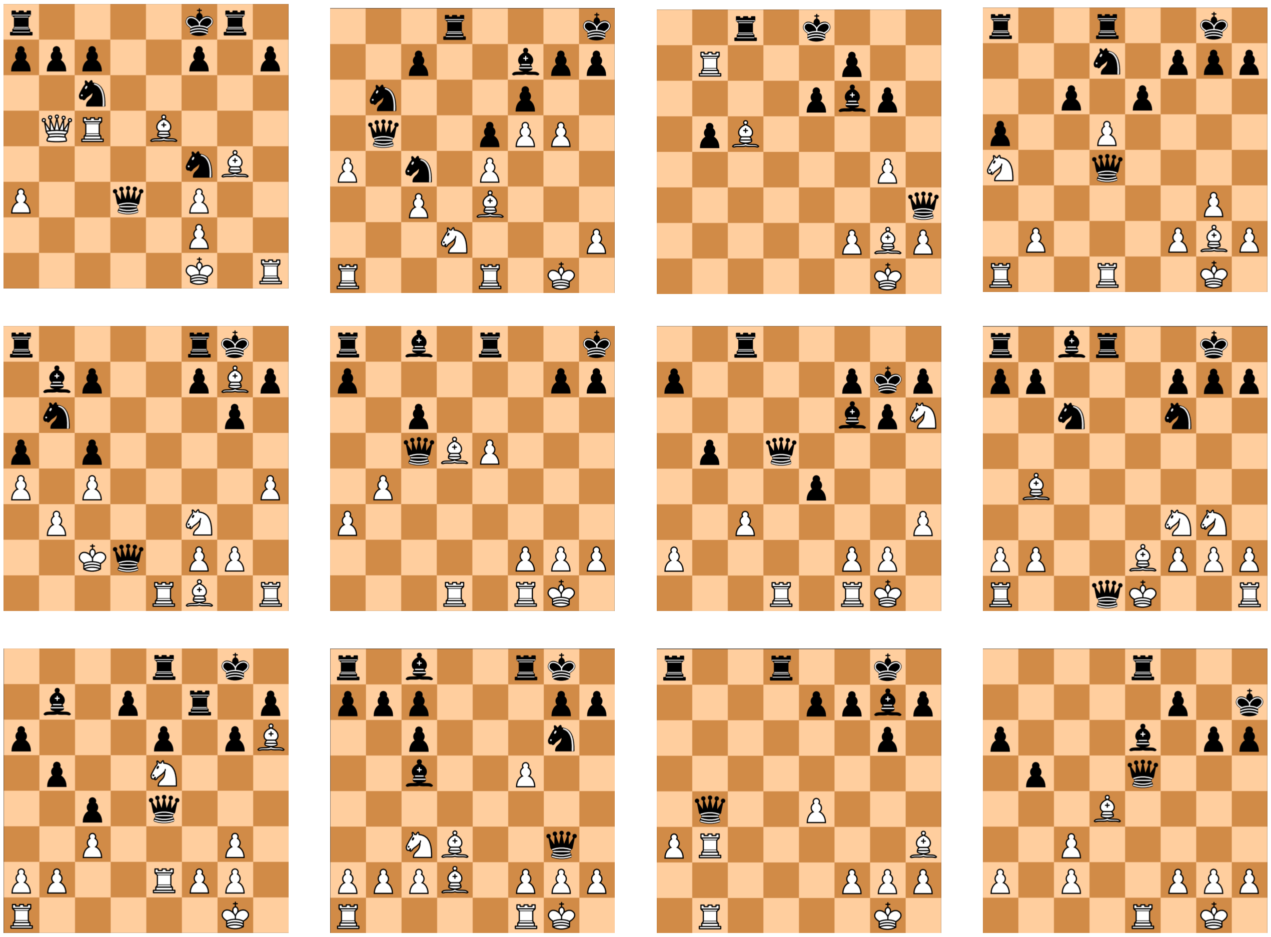
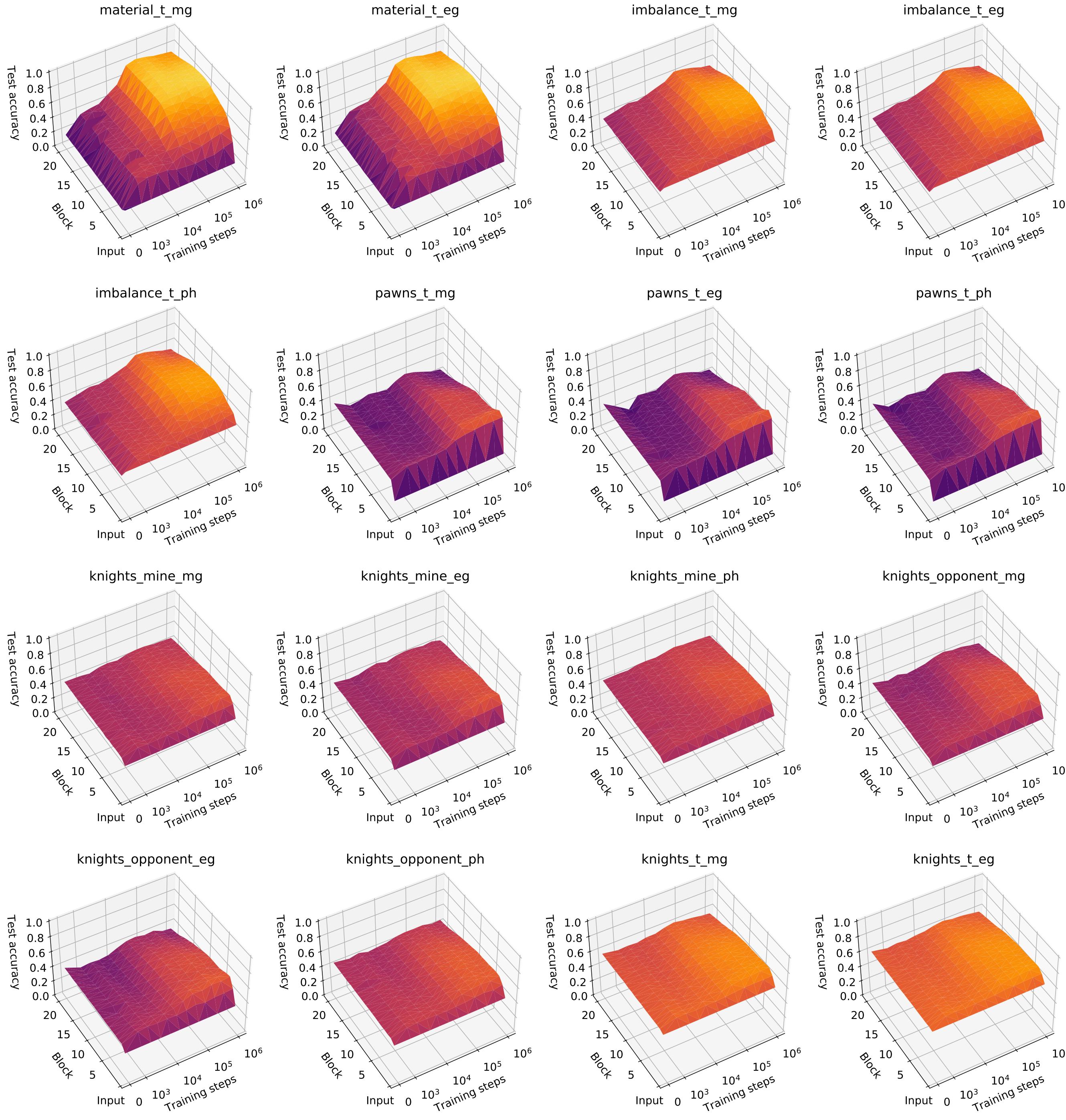


Figure 32. All negative outliers shown in Figure 3. Black’s queen can be taken in every position.

E Concept regression results for second seed

In order to ensure that our results for concept regression generalise beyond a single training run, we applied our concept regression methodology to a second trainingrun of AlphaZero. Our initial analysis was conducted on seed 3, and the following regression plots are obtained from seed 1. They plots match those in Figures 14 to 25 in Appendix B.



material_t_mg

material_t_eg

imbalance_t_mg

imbalance_t_eg

imbalance_t_ph

pawns_t_mg

pawns_t_eg

pawns_t_ph

knights_mine_mg

knights_mine_eg

knights_mine_ph

knights_opponent_mg

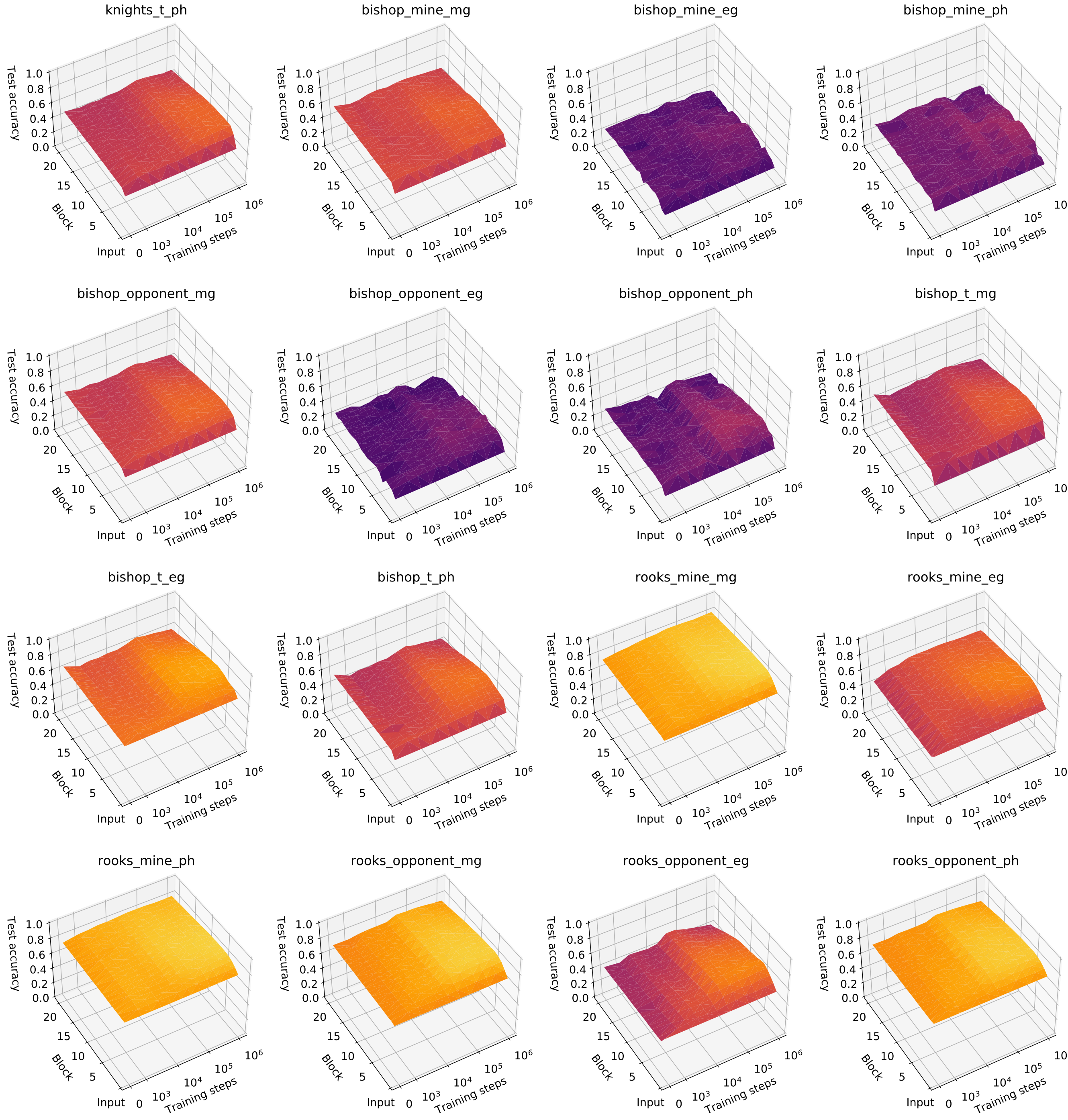
knights_opponent_eg

knights_opponent_ph

knights_t_mg

knights_t_eg

Figure 33. Regression results for Stockfish concepts from Table 2.



knights_t_ph

bishop_mine_mg

bishop_mine_eg

bishop_mine_ph

bishop_opponent_mg

bishop_opponent_eg

bishop_opponent_ph

bishop_t_mg

bishop_t_eg

bishop_t_ph

rooks_mine_mg

rooks_mine_eg

rooks_mine_ph

rooks_opponent_mg

rooks_opponent_eg

rooks_opponent_ph

Figure 34. Regression results for Stockfish concepts from Table 2, continued.

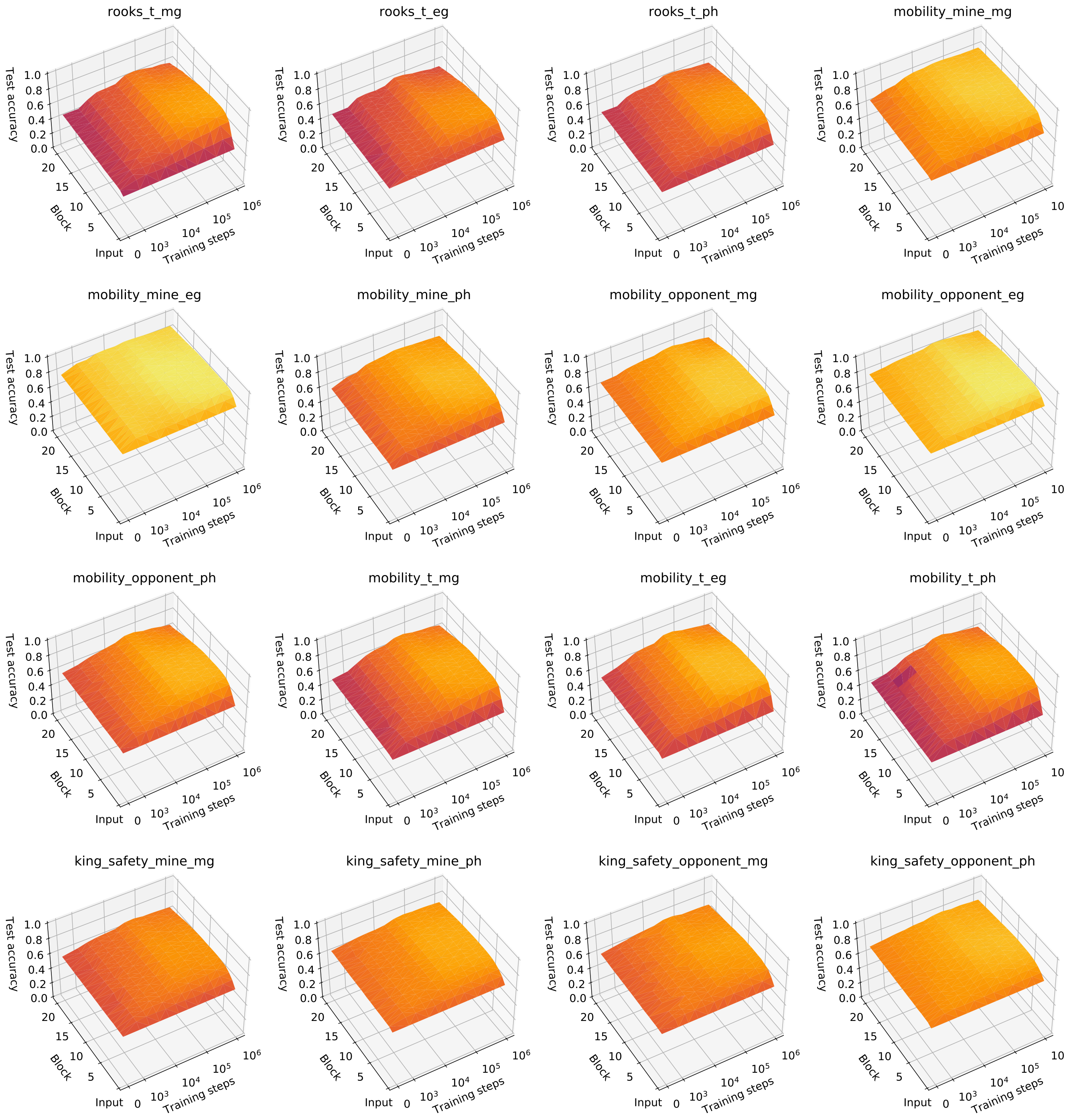


Figure 35. Regression results for Stockfish concepts from Table 2, continued.

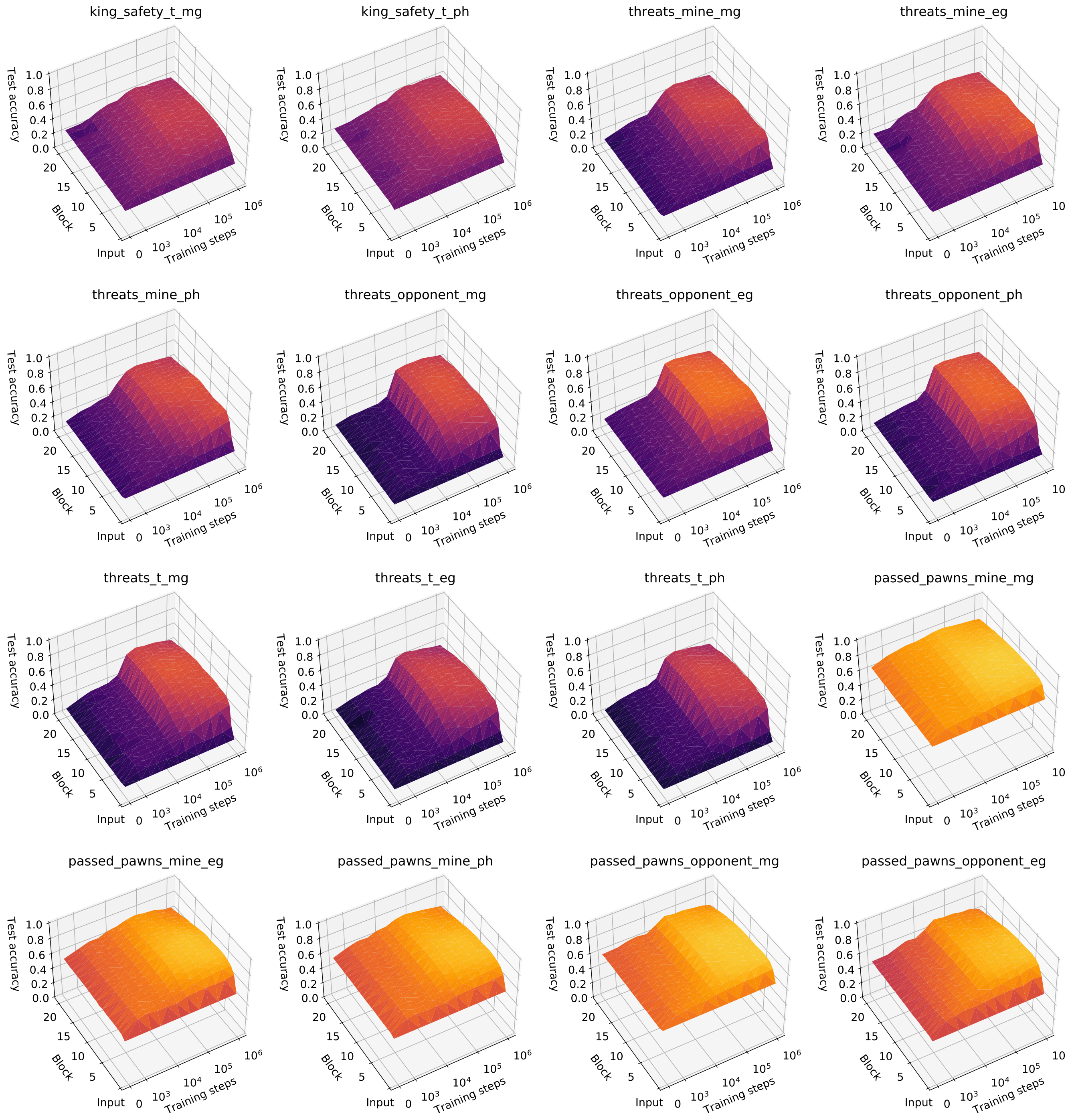
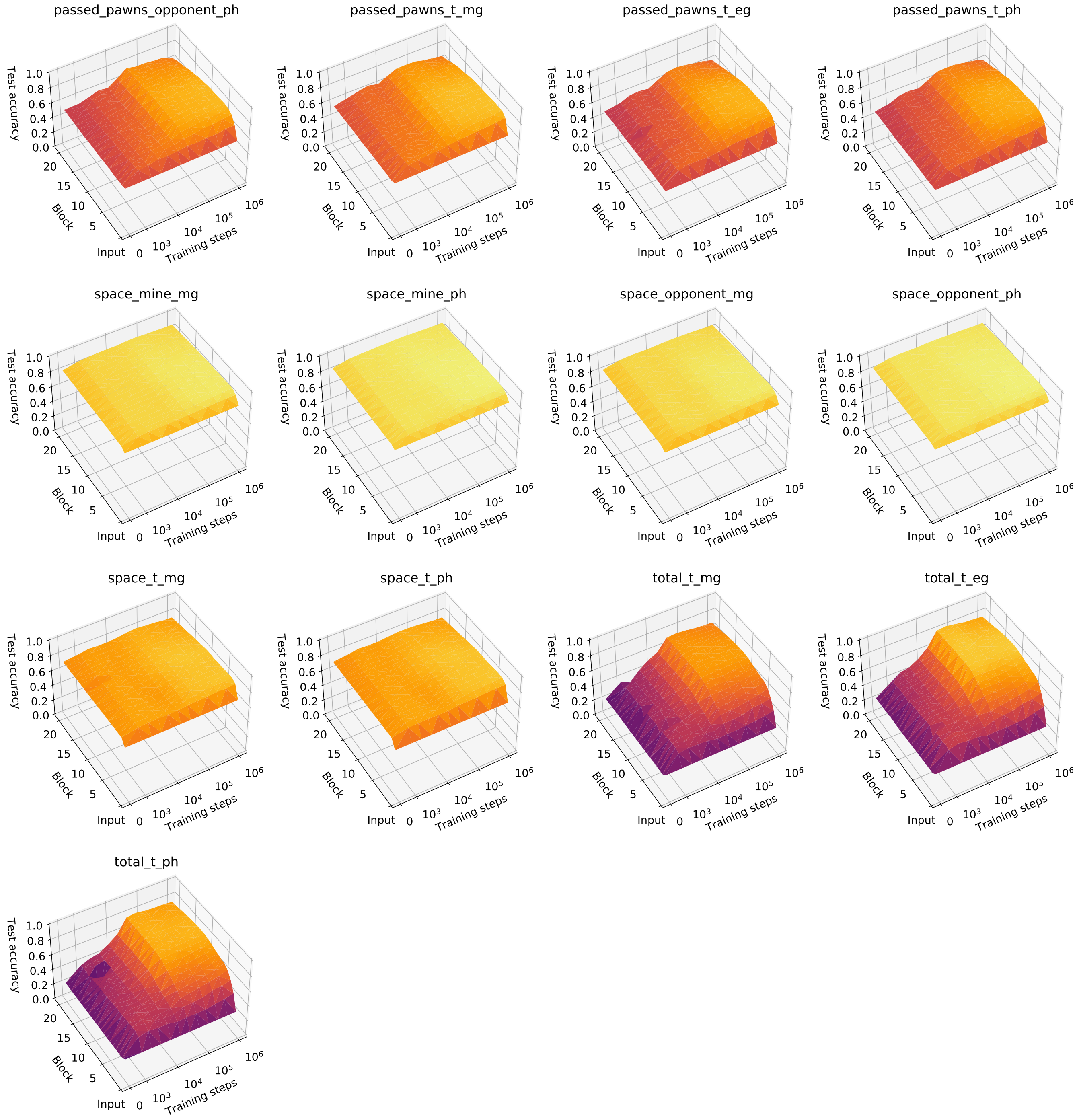

king_safety_t_mg
king_safety_t_ph
threats_mine_mg
threats_mine_eg
threats_mine_ph
threats_opponent_mg
threats_opponent_eg
threats_opponent_ph
threats_t_mg
threats_t_eg
threats_t_ph
passed_pawns_mine_mg
passed_pawns_mine_eg
passed_pawns_mine_ph
passed_pawns_opponent_mg
passed_pawns_opponent_eg

Figure 36. Regression results for Stockfish concepts from Table 2, continued.


passed_pawns_opponent_ph

passed_pawns_t_mg

passed_pawns_t_eg

passed_pawns_t_ph

space_mine_mg

space_mine_ph

space_opponent_mg

space_opponent_ph

space_t_mg

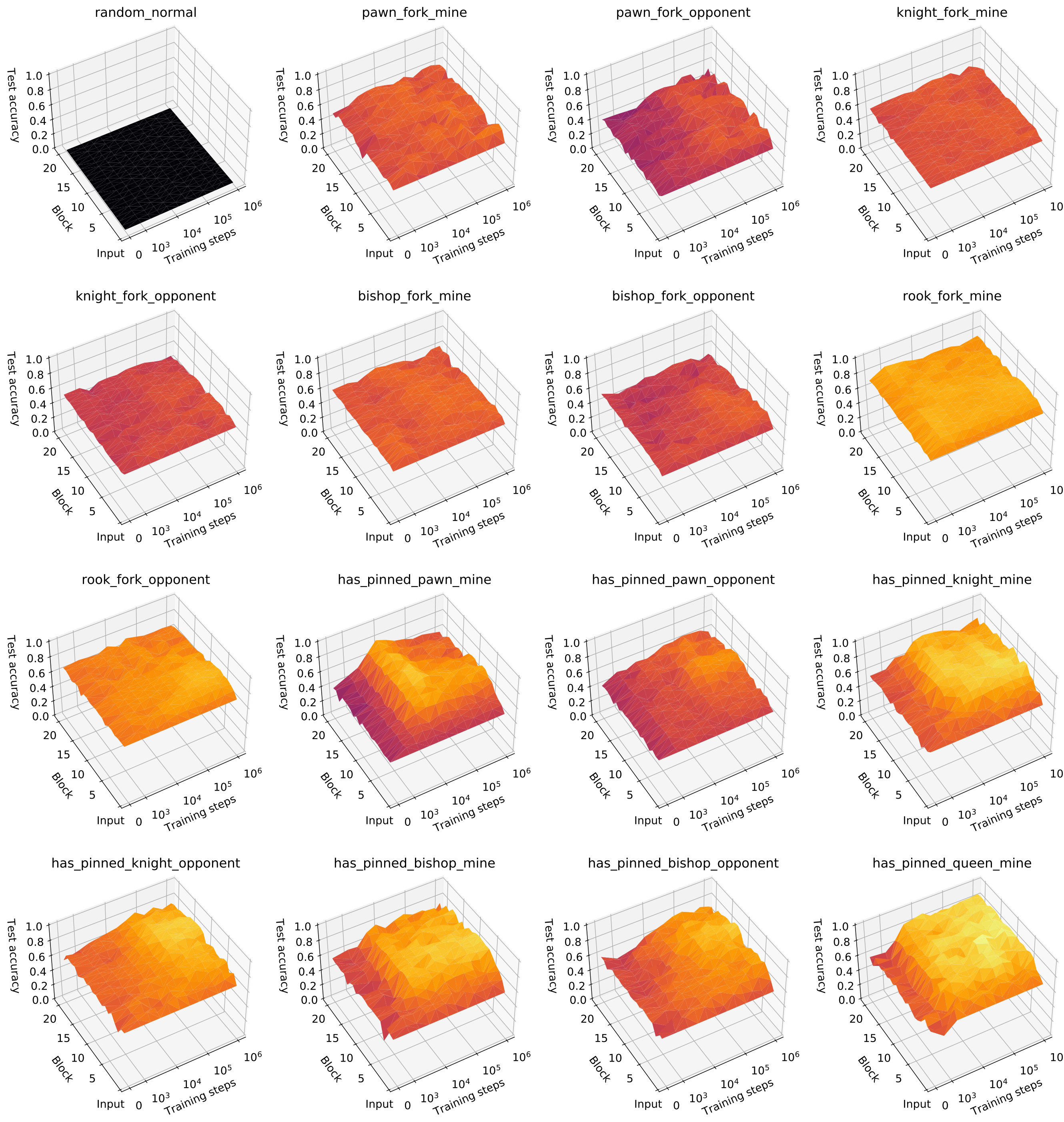
space_t_ph

total_t_mg

total_t_eg

total_t_ph

Figure 37. Regression results for Stockfish concepts from Table 2, continued.


random_normal

pawn_fork_mine

pawn_fork_opponent

knight_fork_mine

knight_fork_opponent

bishop_fork_mine

bishop_fork_opponent

rook_fork_mine

rook_fork_opponent

has_pinned_pawn_mine

has_pinned_pawn_opponent

has_pinned_knight_mine

has_pinned_knight_opponent

has_pinned_bishop_mine

has_pinned_bishop_opponent

has_pinned_queen_mine

Figure 38. Regression results for custom concepts from Table 3, excluding capture-related concepts.

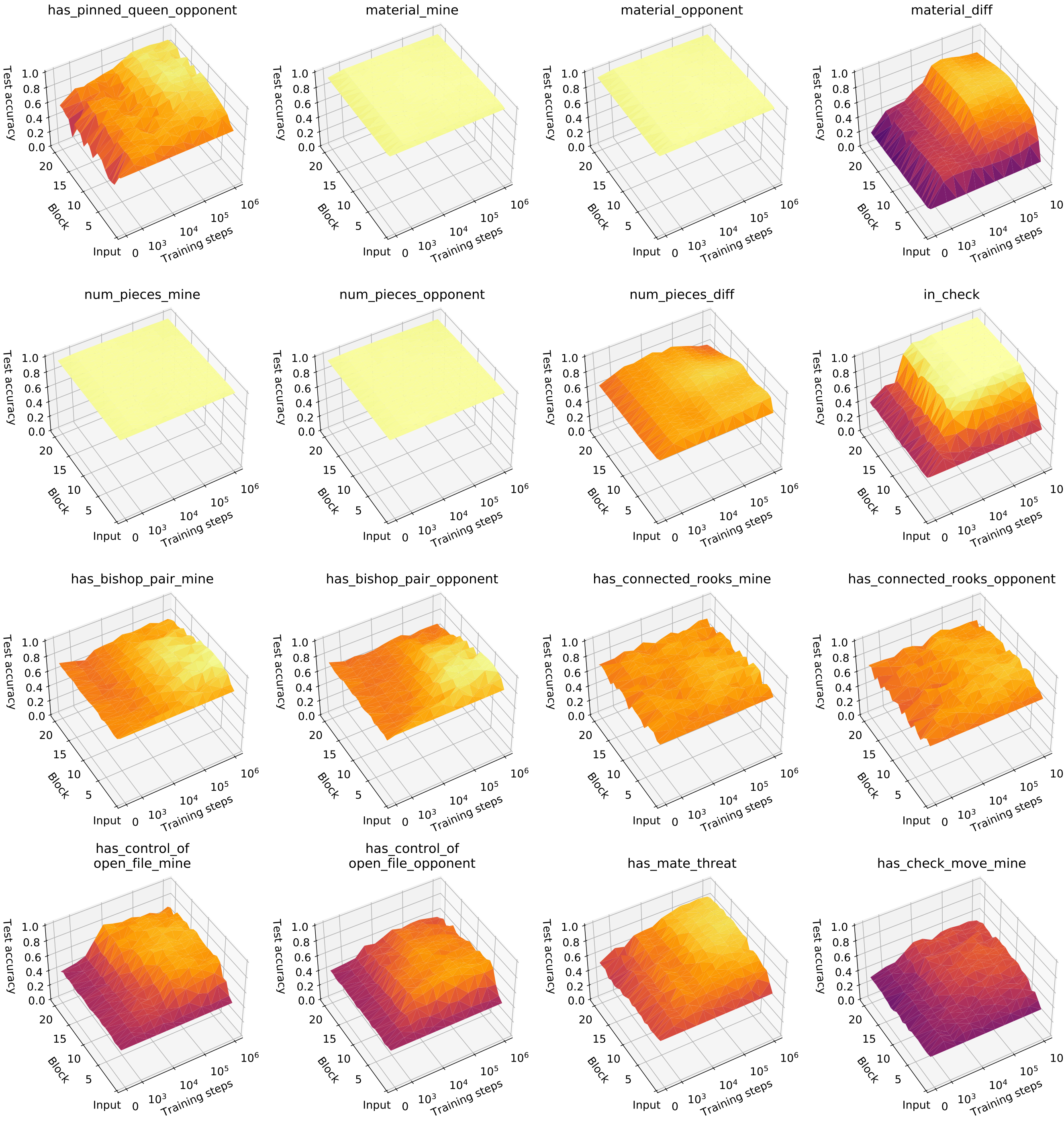


Figure 39. Regression results for custom concepts from Table 3, excluding capture-related concepts, continued.

has_check_move_opponent

can_capture_queen_mine

can_capture_queen_opponent

num_king_attacked squares_mine

num_king_attacked squares_opponent

num_king_attacked squares_diff

has_contested_open_file

has_right_bc ha_promotion_mine

has_right_bc ha_promotion_opponent

num_scb_pawns same_side_mine

num_scb_pawns same_side_opponent

num_scb_pawns same_side_diff

num_ocb_pawns same_side_mine

num_ocb_pawns same_side_opponent

num_ocb_pawns same_side_diff

num_scb_pawns other_side_mine

Figure 40. Regression results for custom concepts from Table 3, excluding capture-related concepts, continued.

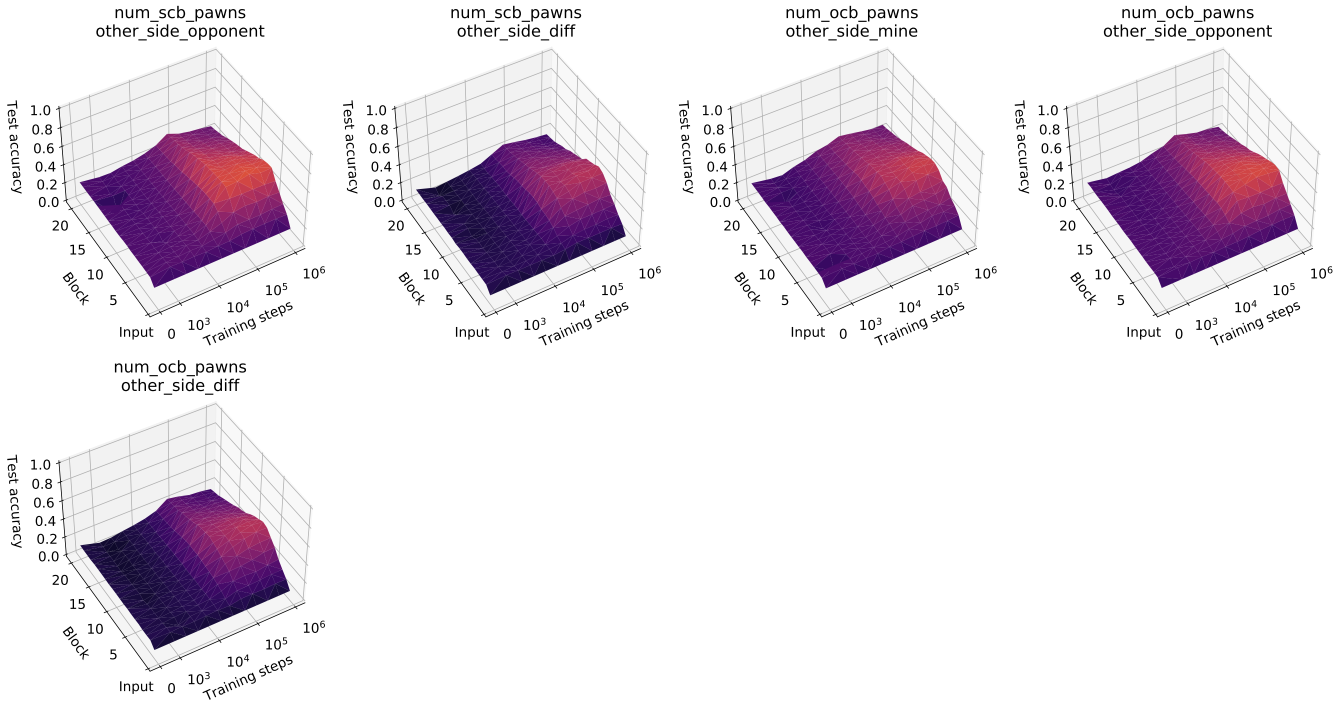
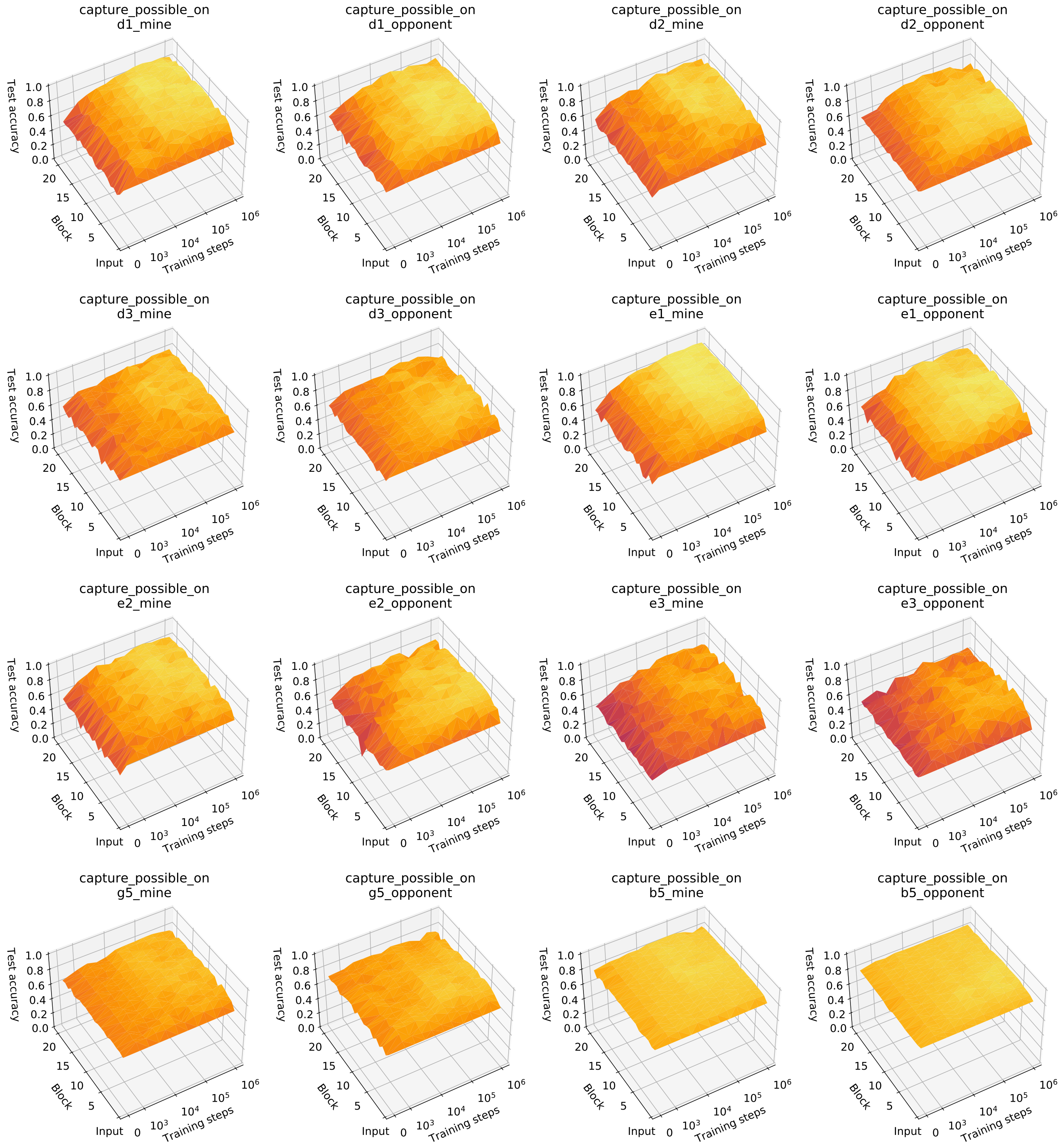


Figure 41. Regression results for custom concepts from Table 3, excluding capture-related concepts, continued.


capture_possible_on d1_mine

capture_possible_on d1_opponent

capture_possible_on d2_mine

capture_possible_on d2_opponent

capture_possible_on d3_mine

capture_possible_on d3_opponent

capture_possible_on e1_mine

capture_possible_on e1_opponent

capture_possible_on e2_mine

capture_possible_on e2_opponent

capture_possible_on e3_mine

capture_possible_on e3_opponent

capture_possible_on g5_mine

capture_possible_on g5_opponent

capture_possible_on b5_mine

capture_possible_on b5_opponent

Figure 42. Regression results for custom concepts from Table 3 related to captures.

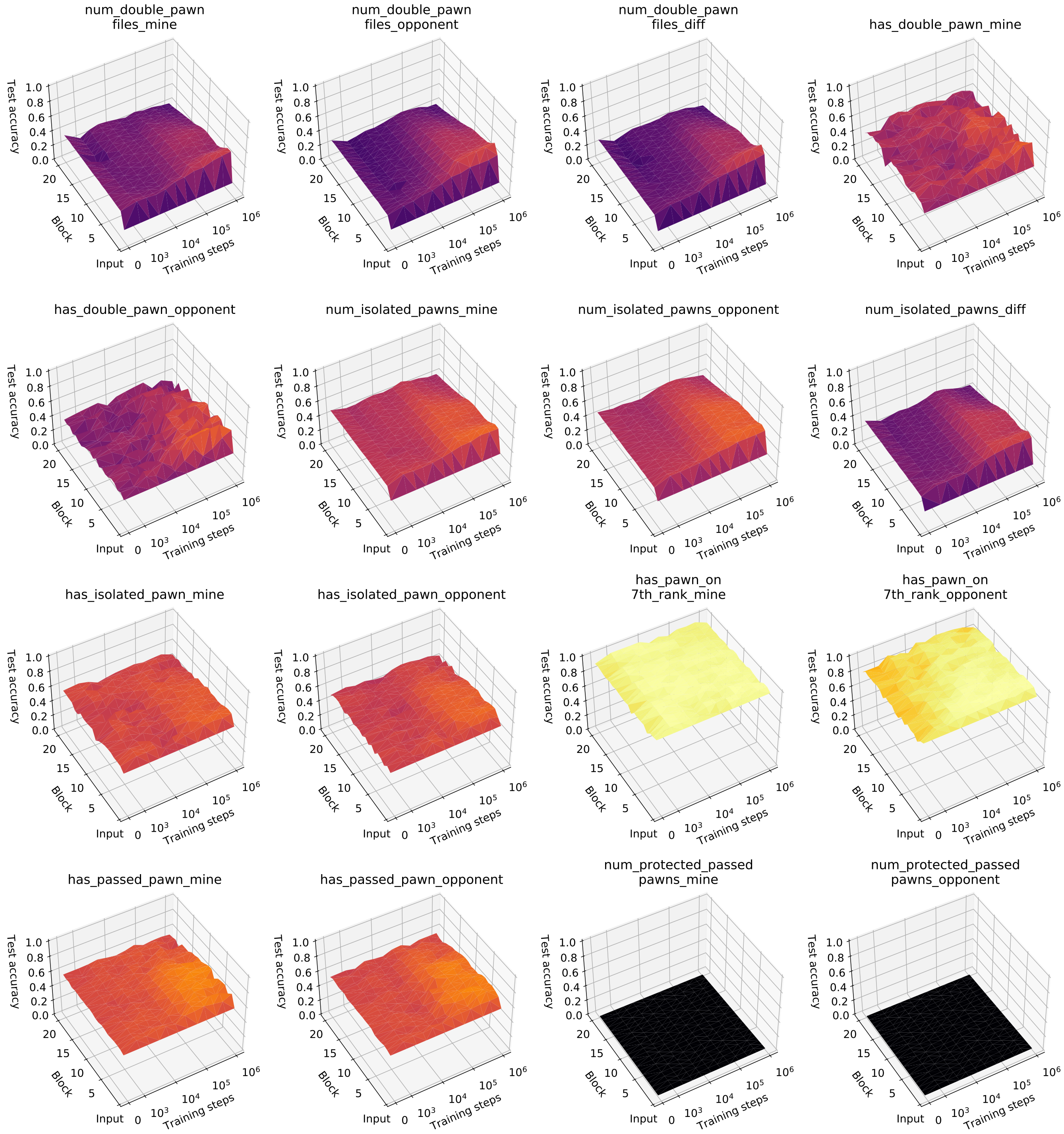


Figure 43. Regression results for custom pawn-related concepts from Table 4.

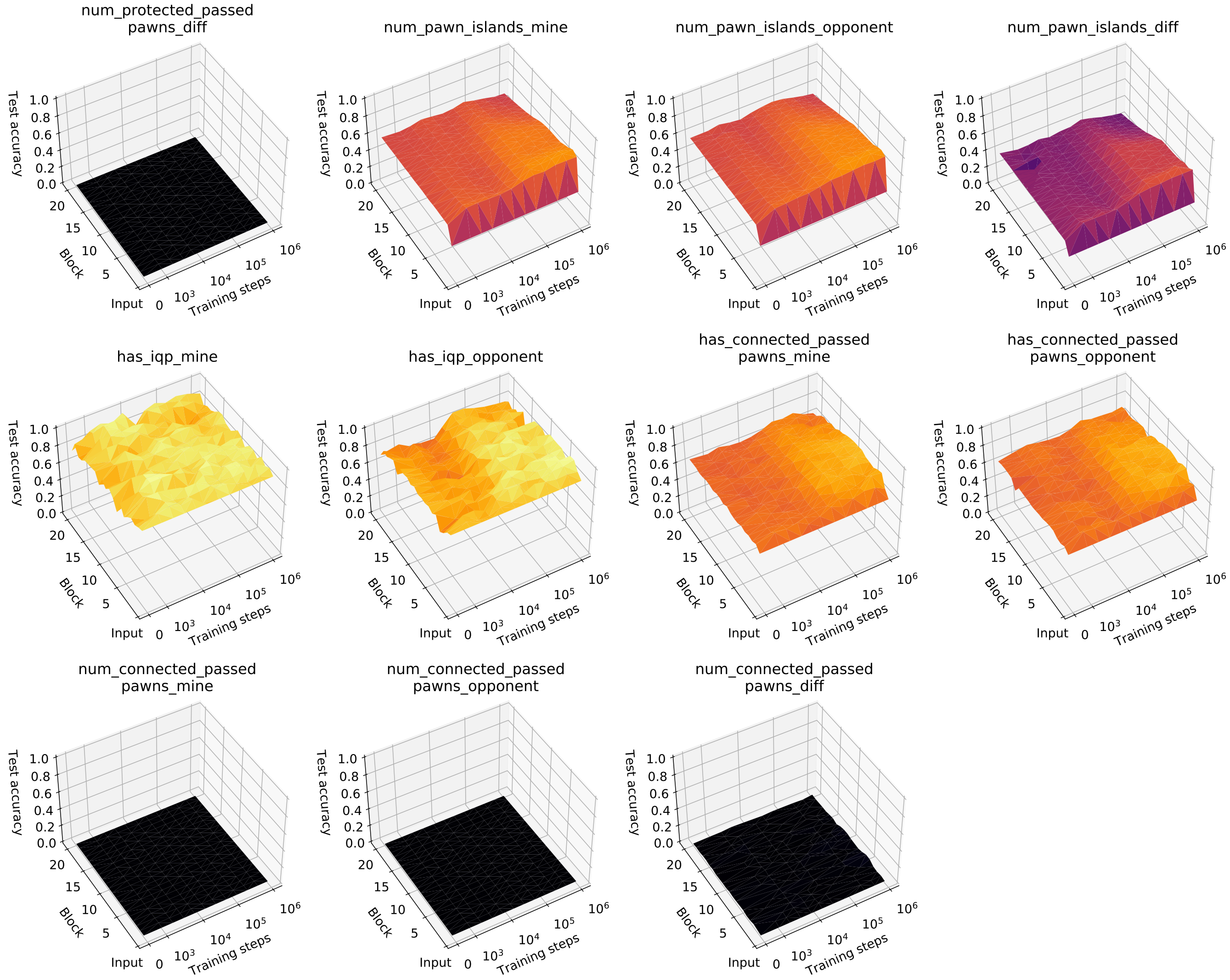


Figure 44. Regression results for custom pawn-related concepts from Table 4, continued.