Compositions with Transformers

Luc Pommeret Achraf Lassoued Michel de Rougemont

March 31, 2025

Abstract

We study how Transformers may compose specific functions, contrary to the worst case scenario when the composition is error prone. We concentrate on games such as chess where runs in the PGN notation are given to a Transformer to predict the next token and the next move. The iteration of moves is a composition which is syntactically correct with high probability, although the Transformer ignores the syntactic rules of the game.

1 Introduction

Transformers have become a cornerstone of modern artificial intelligence, demonstrating remarkable capabilities across a wide range of tasks. A key question in understanding these models concerns their ability to compose functions. That is, if a transformer can learn to compute functions f and g separately, how well can it compute their composition $f \circ g$?

A Transformer takes N consecutive tokens as input and produces a distribution for the $(N+1)$-th token. While theoretical worst-case analyses suggest that function composition in transformers should be error-prone, empirical evidence indicates that in many practical scenarios, transformers perform composition with surprising accuracy.

We investigate this apparent contradiction by studying transformers in the context of games, which provide a controlled environment with well-defined rules and success metrics. Specifically, we focus on chess as our primary experimental domain. In chess, the transformer must learn to generate legal move sequences without explicit knowledge of the game's rules, effectively composing the relation of legal moves across multiple steps.

Our analysis employs probing techniques to investigate how transformers internalize compositional patterns. We examine the distribution of moves predicted by the model at different game stages and analyze how the model maintains consistency with game rules throughout extended sequences.

The main results of our work demonstrate that:

- Transformers can effectively learn to compose relations across multiple steps, despite theoretical limitations
- The error rate in function composition does not accumulate significantly during sequential prediction tasks
- Under certain distributional assumptions, transformers achieve high accuracy in compositional tasks

- Empirical analysis supports theoretical bounds on the likelihood of illegal moves in sequential predictions

These findings contribute to our understanding of transformer capabilities and limitations, particularly in tasks requiring compositional reasoning.

2 Transformers and Games

We first define a precise model for a Transformer and then consider the world of Chess where runs are given in the PGN format with a small number of tokens. The Transformer is given many runs with a distribution p of $N+1$ consecutive tokens and learns a distribution p_Θ of $N+1$ tokens. It can then predict the next token with a probability $p_\Theta(x_{i+1} \mid x_1, ...x_i)$ for $i = 0, ...N$. We study the composition of Chess moves in this context.

2.1 Transformers

2.1.1 Architecture Overview

A transformer model is characterized by several key parameters:

- d - dimension of the embedding vectors
- n - number of tokens in the vocabulary
- N - maximum sequence length the model can process

Additional parameters that define the architecture include:

- H - number of attention heads
- D - dimension of the feed-forward network's hidden layer
- L - number of transformer layers
- p - numerical precision of the weights

A Transformer takes N consecutive tokens as input and produces a probability distribution over the vocabulary for the $(N+1)$-th token. The architecture consists of two main components: the self-attention mechanism and the feed-forward neural network.

2.1.2 Self-Attention Mechanism

For simplicity, let's first consider a single attention head ($H = 1$). The mechanism involves three learned matrices:

- $Q \in \mathbb{R}^{d \times d}$ - query transformation matrix
- $K \in \mathbb{R}^{d \times d}$ - key transformation matrix
- $V \in \mathbb{R}^{d \times d}$ - value transformation matrix

Consider a sequence of N tokens: $t_1, ..., t_N$ where $x_i \in \mathbb{R}^d$ is the embedding of token t_i. The computation proceeds as follows:

1. First, positional encoding is added to each embedding: $x'_i = x_i + \text{pos}(i)$ where $\text{pos}(i)$ is the positional encoding for position i

2. For each position i, compute query, key, and value vectors:

$$q_i = (Q \cdot x'_i)^T, \quad k_j = K \cdot x'_j, \quad v_j = V \cdot x'_j$$

3. Calculate attention weights using scaled dot-product:

$$(r_{i,1}, ..., r_{i,N}) = \text{Softmax}\left(\frac{q_i \cdot k_1}{\sqrt{d}}, ..., \frac{q_i \cdot k_N}{\sqrt{d}}\right)$$

4. Compute the weighted sum of values:

$$y_i = \sum_{j=1}^{N} r_{i,j} \cdot v_j$$

5. Apply residual connection and layer normalization:

$$y'_i = \text{LayerNorm}(x'_i + y_i)$$

This process transforms the sequence of embeddings $x_1, ..., x_N$ into new vectors $y'_1, ..., y'_N$ of the same dimension d.

2.1.3 Feed-Forward Network

After the self-attention mechanism, each position passes through a feed-forward neural network:

1. Apply a two-layer perceptron with ReLU activation:

$$z'_i = W_2 \cdot \text{ReLU}(W_1 \cdot y'_i)$$

   where $W_1 \in \mathbb{R}^{D \times d}$ and $W_2 \in \mathbb{R}^{d \times D}$

2. Apply residual connection and layer normalization:

$$z_i = \text{LayerNorm}(y'_i + z'_i)$$

The function $\text{ReLU}(x) = \max(0, x)$ is applied element-wise.

2.1.4 Multiple Heads and Layers

The transformation $x_i \to z_i$ constitutes one layer. In practice, transformers use multiple attention heads and stack multiple layers:

- For $H > 1$ heads, the embedding is partitioned into H vectors of dimension d/H, with each head having its own set of matrices $Q, K, V \in \mathbb{R}^{d/H \times d/H}$
- The outputs from all heads are concatenated and linearly transformed
- The entire process is repeated L times with different parameters for each layer

2.1.5 Output Distribution

After processing through all layers, the final output is transformed into a probability distribution over the vocabulary:

$$D_{\text{output}}(x_1, ..., x_N) = \text{Softmax}(W_0 \cdot z_N)$$

where $W_0 \in \mathbb{R}^{n \times d}$ is a learned parameter matrix and z_N is the embedding of the last token after all transformations.

The complete transformer model is thus parameterized by the matrices $Q, K, V \in \mathbb{R}^{d \times d}$ (for each head and layer), $W_0 \in \mathbb{R}^{n \times d}$, $W_1 \in \mathbb{R}^{D \times d}$, $W_2 \in \mathbb{R}^{d \times D}$ (for each layer), and the parameters of the LayerNorm operations.

2.1.6 Probabilistic Sequence Generation and Distribution Compression

A key property of transformer models is their ability to generate sequences by iteratively predicting the next token. Given a sequence $x_1, \ldots, x_i$ with $i \leq N$, the model selects x_{i+1} according to the conditional distribution $p(x_{i+1} \mid x_1, \ldots, x_i)$.

The autoregressive generation process works as follows:

- Starting with an empty sequence, we sample x_1 with probability $p(x_1)$
- We then sample x_2 with probability $p(x_2 \mid x_1)$
- This continues until we have generated the desired sequence length

This process implicitly defines a joint probability distribution over sequences:

$$p(x_1, \ldots, x_{N+1}) = p(x_1) \cdot p(x_2 \mid x_1) \cdot \ldots \cdot p(x_{N+1} \mid x_1, \ldots, x_N)$$

An important observation is that this represents a highly efficient compression of the full joint distribution. The support of the complete distribution is of size n^{N+1}, which would require exponential space to represent explicitly. However, the transformer parameterizes this distribution using only:

$$O(L \cdot (d^2 + n \cdot d) + N \cdot d)$$

parameters, where:

- L is the number of layers
- d is the embedding dimension
- n is the vocabulary size
- N is the maximum sequence length

This efficient compression allows transformers to model complex sequence distributions with a manageable number of parameters, which is crucial for their practical application in language modeling and related tasks [2, 1].

2.2 Chess as a Compositional Benchmark

We focus on chess as our primary experimental domain to investigate the composition capabilities of transformer models. Chess serves as an ideal benchmark for several reasons:

- It has well-defined, deterministic rules
- It offers a clear success metric (legal vs. illegal moves)
- It provides a standardized rating system (Elo) that allows for precise performance measurement [**?**]
- It requires compositional reasoning across multiple moves
- It has a rich history of documented games for training data [**?**]

2.2.1 Chess as a Language Modeling Task

Rather than approaching chess as a strategic game requiring search and evaluation, we frame it purely as a language modeling task. We train models on chess games written in Portable Game Notation (PGN) format, which consists of sequences of moves using standard algebraic notation:

```
1.e4 e5 2.Nf3 Nc6 3.Bb5 a6 4.Ba4 Nf6 ...
```

In this formulation, the model must predict the next token given the history of previous tokens, without any explicit encoding of the board state or the rules of chess. This differs fundamentally from traditional chess engines, which rely on sophisticated position evaluation functions and extensive search algorithms.

2.2.2 Formalizing Chess Moves as Relations

To analyze the compositional aspects of chess prediction, we formalize chess positions and moves in terms of relations rather than functions. This approach better captures the one-to-many nature of chess moves, as multiple legal moves are possible from any given position.

Let $\{P, R, N, B, Q, K\}$ represent the chess pieces (Pawn, Rook, Knight, Bishop, Queen, and King). We define a board position as a function:

$$bp : \{a, b, \ldots, h\} \times \{1, 2, \ldots, 8\} \to (\{P, R, N, B, Q, K\} \times \{White, Black\}) \cup \{\emptyset\}$$

This function assigns either a piece of a specific color or nothing to each square on the board. For any coordinates (i, j), $bp(i, j) = (p, c)$ indicates that a piece of type p with color c is on square (i, j), while $bp(i, j) = \emptyset$ indicates that the square is empty.

We then define a binary relation $R(s, s')$ between board positions, where $R(s, s')$ holds if and only if position s' is reachable from position s in one legal move according to the rules of chess. The complement $\neg R(s, s')$ indicates that no legal move exists between positions s and s'.

2.2.3 Compositional Aspects of Chess

This relational formulation highlights the compositional nature of chess gameplay:

- A complete game can be viewed as a path in a directed graph where vertices are board positions and edges are determined by the relation R

- The composition of R with itself k times, denoted R^k, represents positions reachable in exactly k moves
- Learning to play chess involves learning to compose the relation R repeatedly while maintaining legal play

2.2.4 Model Prediction as an Approximation of Legal Moves

Given a transformer model trained on chess games, we denote its next-token prediction distribution as:

$$D(t_N \mid t_1, \ldots, t_{N-1})$$

When generating multiple tokens that form a complete move (e.g., "Nc6"), the model implicitly defines a relation between board positions. If s represents the current board state and s' the state after applying the predicted move, we define:

$$\hat{R}_\Theta(s, s')$$

as the relation induced by the transformer's predictions.

The performance of the model in capturing legal chess moves can be measured by:

$$\mathbb{P}[\hat{R}_\Theta(s, s') \wedge \neg R(s, s')] \leq \epsilon \tag{1}$$

for some small $\epsilon > 0$, where the probability is taken over positions from the test set and the model's output distribution.

2.2.5 Error Accumulation in Sequential Composition

A central question in our investigation is how errors accumulate during sequential predictions. Remarkably, we observe that for well-trained transformer models, the error rate does not grow significantly with the number of moves, despite the exponential growth in the space of possible positions.

This suggests that transformers can effectively learn to compose the legal move relation across multiple steps, maintaining consistency with the rules even though they were never explicitly programmed with them. This compositional capability emerges purely from exposure to examples of legal chess games during training.

2.3 Distribution on Chess Moves

While transformers operate at the token level, chess gameplay occurs at the level of moves. This distinction is important for understanding how the model composes legal play sequences.

2.3.1 From Token-Level to Move-Level Distribution

The transformer model defines a distribution $D(t_N \mid t_1, \ldots, t_{N-1})$ over the next token t_N given the previous tokens $t_1, \ldots, t_{N-1}$. However, a complete chess move typically consists of multiple tokens (e.g., "Nf3" consists of tokens 'N' and 'f3', or might be tokenized differently depending on the tokenization scheme).

We can derive a move-level distribution D' from the token-level distribution D:

$$D'(m_k \mid m_1, \ldots, m_{k-1})$$

where m_i represents a complete chess move. This move-level distribution is what we analyze to understand the model's ability to maintain legal play across multiple moves.

2.3.2 Syntactic vs. Semantic Correctness

When generating chess moves, two types of errors can occur:

- **Syntactic errors**: The generated token sequence does not form a valid move in algebraic notation (e.g., "NNf7" or "e9")
- **Semantic errors**: The generated move follows correct algebraic notation but is illegal in the current board position (e.g., moving a piece to a square occupied by another piece of the same color)

Our analysis focuses on both types of errors, but particularly on semantic errors, which reveal whether the model has learned to compose legal move sequences rather than just the syntax of chess notation.

2.3.3 Transition Probabilities Between Board States

For any two board positions s and s', the transformer-induced transition probability is:

$$P(s \to s') = \sum_{m \in M(s,s')} D'(m \mid \text{history}(s))$$

where $M(s, s')$ is the set of moves that transform position s into position s', and history(s) is the sequence of moves leading to position s.

By analyzing these transition probabilities across different positions and game phases, we can assess the model's understanding of chess rules and its ability to maintain legal play through composition.

3 Theoretical Analysis of Composition

We now develop a theoretical framework to analyze the composition capabilities of transformer models. We begin with an analysis of the worst-case scenario using communication complexity, then show how distributional assumptions lead to dramatically different conclusions.

3.1 Communication Complexity of Functional Composition

Let us consider the canonical problem of functional composition: given functions $f : A \to B$ and $g : B \to C$, we want to compute their composition $f \circ g : A \to C$, where $(f \circ g)(x) = f(g(x))$.

To analyze this through the lens of communication complexity, we can formulate the following scenario: Alice has function f, Bob has computed $i = g(x)$ for some input x, and they need to collaborate to compute $f(i)$. This is essentially the Index function problem in communication complexity.

3.2 Worst-Case Analysis

3.2.1 Problem Formulation

Consider sets A, B, C, each of size n. We have functions $f : B \to C$ and $g : A \to B$, and we want to compute $f(g(x))$ for some $x \in A$.

In our analysis, we introduce three key random variables:

- $i^* = g(x)$ - the intermediate result
- Π - the message exchanged between Alice and Bob (or the internal representation in a transformer)
- $f(i^*)$ - the final output

3.2.2 Information-Theoretic Analysis

By definition of conditional entropy and mutual information:

$$H[f(i^*) \mid \Pi, i^*] = H[f(i^*) \mid i^*] - I[\Pi; f(i^*) \mid i^*]$$

Fano's inequality relates the error probability $\delta = \mathbb{P}[error]$ to the conditional entropy:

$$H[f(i^*) \mid \Pi, i^*] \leq H(error) + \delta \cdot \log n$$

where $H(error) = -\delta \log \delta - (1 - \delta) \log(1 - \delta)$ is the binary entropy function.

Lemma 1 (Worst-Case Composition Error)**.** *Let $\mathcal{T}$ be a transformer with parameters H (number of heads), d (embedding dimension), and p (precision). If the size of the internal representation is $|\Pi| = n \log n - R$ where $R = n \log n - H(d+1)p$, then the probability over random functions f, g and input x that $\mathcal{T}$ computes $f(g(x))$ incorrectly is at least $\frac{R}{3n \log n}$.*

Proof. The mutual information between the message Π and the target value $f(i^*)$ given i^* can be written as:

$$\begin{aligned} I[\Pi; f(i^*) \mid i^*] &= \sum_i \mathbb{P}[i^* = i] \cdot I[\Pi; f(i^*) \mid i^* = i] \\ &= \frac{1}{n} \sum_i I[\Pi; f(i)] \end{aligned}$$

assuming i is uniformly distributed over the domain $\{1, 2, \ldots, n\}$.

A fundamental result in information theory states that $I[\Pi; f(i^*) \mid i^*] \leq |\Pi|/n$, as the message Π can provide at most $|\Pi|$ bits of information in total, which must be divided across n possible values of i^*.

Therefore:

$$\begin{aligned} H[f(i^*) \mid \Pi, i^*] &= H[f(i^*) \mid i^*] - I[\Pi; f(i^*) \mid i^*] \\ &= \log n - I[\Pi; f(i^*) \mid i^*] \\ &\geq \log n - |\Pi|/n \\ &= \log n - (n \log n - R)/n \\ &= \log n - \log n + R/n \\ &= R/n \end{aligned}$$

By Fano's inequality:

$$\begin{aligned} H(error) + \delta \cdot \log n &\geq H[f(i^*) \mid \Pi, i^*] \\ &\geq R/n \end{aligned}$$

Since $H(error) \leq 1$ for a binary random variable, and using a loose upper bound, we can say:

$$\begin{aligned} 2\delta \cdot \log n &\geq R/n \\ \delta &\geq \frac{R}{2n \log n} \end{aligned}$$

For a more precise bound, we can use the fact that $H(error) \leq -\delta \log \delta \leq 2\delta$ for small δ, giving us:

$$\begin{aligned} \delta \cdot \log n + 2\delta &\geq R/n \\ \delta(\log n + 2) &\geq R/n \\ \delta &\geq \frac{R}{n(\log n + 2)} \\ \delta &\geq \frac{R}{3n \log n} \text{ (for large enough } n) \end{aligned}$$

Thus, the probability of error is at least $\frac{R}{3n \log n}$. □

This lemma provides a fundamental limitation: in the worst case over random functions, a transformer model with limited capacity will fail to compute compositions correctly with high probability.

3.3 Distribution-Dependent Analysis

The worst-case analysis assumes uniform distributions over functions and inputs. However, in real-world applications, the distributions are rarely uniform. We now analyze how biased distributions affect composition performance.

3.3.1 Non-Uniform Input Distributions

In many practical scenarios, the intermediate values $i^* = g(x)$ follow biased distributions, such as power-law distributions common in natural data.

Lemma 2 (Entropy of Power-Law Distribution)**.** *For a power-law distribution $p(i) = \frac{c}{i^\alpha}$ where $\alpha > 1$ and c is a normalization constant, the entropy $H(i)$ is finite and approximately constant with respect to the size of the domain.*

Proof. The entropy is given by:

$$\begin{aligned} H(i) &= -\sum_i p(i) \log p(i) \\ &= -\sum_i \frac{c}{i^\alpha} \log\left(\frac{c}{i^\alpha}\right) \\ &= -\sum_i \frac{c}{i^\alpha} (\log c - \alpha \log i) \\ &= -\log c \sum_i \frac{c}{i^\alpha} + \alpha \sum_i \frac{c}{i^\alpha} \log i \end{aligned}$$

The first sum equals 1 by normalization, and the second sum converges to a constant for $\alpha > 1$. Thus, $H(i)$ is finite and does not grow with the domain size. □

3.3.2 Success Probability Under Biased Distributions

Under biased distributions, the success probability for composition can be much higher than in the worst case.

Lemma 3 (Composition Success Under Biased Distribution)**.** *Let $\mathcal{T}$ be a transformer with parameters such that the capacity of its internal representation accommodates the entropy $H(i)$ of the distribution of intermediate values. Then the probability of successful composition is at least $1 - \delta$, where δ depends on the relationship between the model capacity and $H(i)$.*

Proof. Consider the conditional entropy $H[f(i^*) \mid \Pi, i^*]$ with two cases: error with probability δ and correct computation with probability $1 - \delta$:

$$\begin{aligned} H[f(i^*) \mid \Pi, i^*] &= \delta \cdot H[f(i^*) \mid \Pi, i^*, \text{error}] + (1 - \delta) \cdot H[f(i^*) \mid \Pi, i^*, \text{no error}] \\ &\geq \delta \cdot H[f(i^*) \mid \Pi, i^*, \text{error}] \end{aligned}$$

Since $H[f(i^*) \mid \Pi, i^*, \text{no error}] = 0$ (there is no uncertainty when the computation is correct).

When the representation capacity of the transformer is sufficient to capture the entropy $H(i)$ of the intermediate distribution, the mutual information $I[\Pi; f(i^*) \mid i^*]$ approaches $H[f(i^*) \mid i^*]$, making the conditional entropy $H[f(i^*) \mid \Pi, i^*]$ close to zero.

This implies that δ must be small for the overall entropy to be low, leading to a high probability of successful composition. □

This result explains why transformers can perform well on composition tasks in practice, despite the theoretical worst-case limitations. When the distribution of intermediate values has low entropy (as is often the case in natural data), the model can effectively learn to compose functions with high accuracy.

3.4 Probabilistic Analysis of Illegal Move Generation

We now develop a formal analysis of the probability that a transformer model generates illegal chess moves. We demonstrate that, under reasonable assumptions about model training, the probability of generating illegal moves remains low even across extended move sequences.

3.4.1 Distributional Setup

Let $p(x_1, \ldots, x_m)$ represent the true distribution of legal chess games from a dataset such as Lichess, and let $p_\Theta(x_1, \ldots, x_m)$ be the distribution learned by a transformer with parameters Θ. Here, each x_i represents a complete move pair (one move by White followed by one move by Black).

We assume that the transformer has been trained to approximate the true distribution, such that the Kullback-Leibler divergence between the two distributions is small:

$$\mathsf{KL}(p \parallel p_\Theta) \leq \varepsilon$$

for some small $\varepsilon > 0$.

3.4.2 Properties of Marginal Distributions

Our analysis proceeds in three key steps:

1. We show that if the KL divergence between joint distributions is small, then the KL divergence between their marginals is also small
2. Using Pinsker's inequality, we demonstrate that a small KL divergence implies a small L_1 distance between distributions
3. We prove that a small L_1 distance implies a low probability of generating illegal moves

We begin with the relationship between joint and marginal KL divergences:

Lemma 4 (KL Divergence of Marginals)**.** *If* $\mathsf{KL}(p(x_1, x_2) \parallel q(x_1, x_2)) \leq \varepsilon$*, then* $\mathsf{KL}(p(x_1) \parallel q(x_1)) \leq \varepsilon$*.*

Proof. By definition of the KL divergence:

$$\begin{aligned}
\mathsf{KL}(p(x_1, x_2) \parallel q(x_1, x_2)) &= \sum_{x_1, x_2} p(x_1, x_2) \log\left(\frac{p(x_1, x_2)}{q(x_1, x_2)}\right) \\
&= \sum_{x_1, x_2} p(x_1, x_2) \log\left(\frac{p(x_1)p(x_2|x_1)}{q(x_1)q(x_2|x_1)}\right) \\
&= \sum_{x_1, x_2} p(x_1, x_2) \log\left(\frac{p(x_1)}{q(x_1)}\right) + \sum_{x_1, x_2} p(x_1, x_2) \log\left(\frac{p(x_2|x_1)}{q(x_2|x_1)}\right)
\end{aligned}$$

The first term can be simplified:

$$\begin{aligned}
\sum_{x_1, x_2} p(x_1, x_2) \log\left(\frac{p(x_1)}{q(x_1)}\right) &= \sum_{x_1}\left(\sum_{x_2} p(x_1, x_2)\right) \log\left(\frac{p(x_1)}{q(x_1)}\right) \\
&= \sum_{x_1} p(x_1) \log\left(\frac{p(x_1)}{q(x_1)}\right) \\
&= \mathsf{KL}(p(x_1) \parallel q(x_1))
\end{aligned}$$

For the second term, we reorganize:

$$\begin{aligned}
\sum_{x_1, x_2} p(x_1, x_2) \log\left(\frac{p(x_2|x_1)}{q(x_2|x_1)}\right) &= \sum_{x_1} p(x_1) \sum_{x_2} p(x_2|x_1) \log\left(\frac{p(x_2|x_1)}{q(x_2|x_1)}\right) \\
&= \sum_{x_1} p(x_1) \cdot \mathsf{KL}(p(x_2|x_1) \parallel q(x_2|x_1))
\end{aligned}$$

Since $\mathsf{KL}(p(x_2|x_1) \parallel q(x_2|x_1)) \geq 0$ for any distributions, the second term is non-negative. Therefore:

$$\begin{aligned}
\mathsf{KL}(p(x_1, x_2) \parallel q(x_1, x_2)) &= \mathsf{KL}(p(x_1) \parallel q(x_1)) + \sum_{x_1} p(x_1) \cdot \mathsf{KL}(p(x_2|x_1) \parallel q(x_2|x_1)) \\
&\geq \mathsf{KL}(p(x_1) \parallel q(x_1))
\end{aligned}$$

Since $\mathsf{KL}(p(x_1, x_2) \parallel q(x_1, x_2)) \leq \varepsilon$ by assumption, we conclude that $\mathsf{KL}(p(x_1) \parallel q(x_1)) \leq \varepsilon$. □

This result generalizes to multivariate distributions: if $\mathsf{KL}(p(x) \parallel q(x)) \leq \varepsilon$ where $x = (x_1, \ldots, x_m)$, then $\mathsf{KL}(p(x_{i+1}|x_1, \ldots, x_i) \parallel q(x_{i+1}|x_1, \ldots, x_i)) \leq \varepsilon$ for any $i < m$.

3.4.3 Bounding the Probability of Illegal Moves

To relate KL divergence to the probability of generating illegal moves, we use Pinsker's inequality:

Lemma 5 (Pinsker's Inequality)**.** *For any probability distributions p and q:*

$$\mathsf{dist}_{L_1}(p, q)^2 \leq 2 \cdot \mathsf{KL}(p \parallel q)$$

where $\mathsf{dist}_{L_1}(p, q) = \sum_x |p(x) - q(x)|$ is the L_1 distance between the distributions.

Using these results, we can bound the probability of generating illegal moves:

Theorem 6 (Bound on Illegal Move Probability)**.** *If $\mathsf{KL}(p \parallel p_\Theta) \leq \varepsilon$, then for any valid game prefix h, the probability that the transformer generates a legal next move is at least $1 - \delta$, where $\delta = \sqrt{2\varepsilon}$.*

Proof. We proceed by induction on the length of the game prefix.

Base case: When the prefix is empty (start of the game).

$$\mathsf{KL}(p(x_1) \parallel p_\Theta(x_1)) \leq \varepsilon$$

by Lemma 4. Applying Pinsker's inequality (Lemma 5):

$$\mathsf{dist}_{L_1}(p(x_1), p_\Theta(x_1)) \leq \sqrt{2\varepsilon}$$

Let $\mathcal{L}$ be the set of all legal moves and $\mathcal{I}$ be the set of illegal moves. Since the true distribution p assigns zero probability to illegal moves:

$$p(x_1 \in \mathcal{I}) = 0$$
$$p(x_1 \in \mathcal{L}) = 1$$

The L_1 distance can be split as:

$$\begin{aligned}
\mathsf{dist}_{L_1}(p(x_1), p_\Theta(x_1)) &= \sum_{x_1 \in \mathcal{L}} |p(x_1) - p_\Theta(x_1)| + \sum_{x_1 \in \mathcal{I}} |p(x_1) - p_\Theta(x_1)| \\
&= \sum_{x_1 \in \mathcal{L}} |p(x_1) - p_\Theta(x_1)| + \sum_{x_1 \in \mathcal{I}} p_\Theta(x_1) \\
&\geq \sum_{x_1 \in \mathcal{I}} p_\Theta(x_1) \\
&= \mathbb{P}_{p_\Theta}[x_1 \text{ is an illegal move}]
\end{aligned}$$

Therefore:

$$\mathbb{P}_{p_\Theta}[x_1 \text{ is an illegal move}] \leq \mathsf{dist}_{L_1}(p(x_1), p_\Theta(x_1)) \leq \sqrt{2\varepsilon}$$

Thus $\mathbb{P}_{p_\Theta}[\text{Next move} \in \mathcal{L}] \geq 1 - \delta$ where $\delta = \sqrt{2\varepsilon}$.

Inductive step: For any valid prefix $h = (x_1, \ldots, x_i)$ of length $i < N$, applying Lemma 4 to the conditional distributions:

$$\mathsf{KL}(p(x_{i+1}|x_1, \ldots, x_i) \parallel p_\Theta(x_{i+1}|x_1, \ldots, x_i)) \leq \varepsilon$$

By Pinsker's inequality:

$$\mathsf{dist}_{L_1}(p(x_{i+1}|x_1, \ldots, x_i), p_\Theta(x_{i+1}|x_1, \ldots, x_i)) \leq \sqrt{2\varepsilon}$$

Using the same argument as in the base case, since p only assigns probability to legal moves:

$$\mathbb{P}_{p_\Theta}[x_{i+1} \text{ is an illegal move}|x_1, \ldots, x_i] \leq \sqrt{2\varepsilon}$$

Therefore, $\mathbb{P}_{p_\Theta}[\text{Next move} \in \mathcal{L}|h] \geq 1 - \delta$ where $\delta = \sqrt{2\varepsilon}$. □

3.4.4 Implications for Sequential Composition

This theorem has profound implications for the compositional abilities of transformer models. It shows that if a transformer can learn to approximate the distribution of legal chess games with small KL divergence ε, then the probability of generating an illegal move remains bounded by $\delta = \sqrt{2\varepsilon}$ regardless of the length of the game played so far.

Crucially, the error probability does not accumulate with the number of moves. This explains why transformers can effectively compose long sequences of chess moves while maintaining legal play, despite never being explicitly taught the rules of chess.

4 Experimental Setup

To empirically validate our theoretical analysis, we conducted a series of experiments with transformer models trained on chess games. This section details our experimental methodology, including model architecture, training procedures, and evaluation metrics.

4.1 Model Architecture

We trained four separate transformer models with identical architectures but different levels of noise in their training data. This approach allowed us to investigate how training data quality affects compositional capabilities. The models share the following architectural characteristics:

Parameter	**Value**
Transformer layers	8
Attention heads per layer	8
Embedding dimension	512
Feed-forward dimension	2048
Maximum sequence length	1024 tokens
Vocabulary size	32 tokens
Total parameters	$\approx$ 1 million

Table 1: Model architecture specifications

The chosen sequence length of 1024 tokens is more than sufficient for our task, as most chess games require significantly fewer tokens. A typical chess game rarely exceeds 80 moves (160 ply), and each move in PGN notation requires at most 7 tokens (including move number and punctuation). This ensures that our models can process complete games during both training and inference.

4.2 Dataset Preparation

4.2.1 Base Dataset

Our dataset was derived from the Lichess open database of chess games, with the following filtering criteria:

- Games played by players with Elo ratings above 2000
- Standard chess games (no variants like Chess960)

- Games with at least 20 moves
- Games without adjudication by timeout or abandonment

After filtering, our base dataset consisted of approximately 1 million games, which we split into training (80%), validation (10%), and test (10%) sets.

4.2.2 Noise Injection

To study how model performance degrades with imperfect training data, we created four versions of the dataset with different levels of noise:

- **Clean (0%)**: The original dataset with no modifications
- **Low Noise (5%)**: 5% of moves randomly replaced with alternative legal moves
- **Medium Noise (50%)**: 50% of moves randomly replaced with alternative legal moves
- **High Noise (80%)**: 80% of moves randomly replaced with alternative legal moves

Importantly, we only replaced moves with other legal moves, ensuring that all games in the dataset remained syntactically valid. This approach tests the model's ability to learn strategic patterns rather than just the rules of the game.

4.3 Training Procedure

All models were trained using a standard language modeling objective, minimizing the cross-entropy loss for next token prediction. We used the following hyperparameters:

Hyperparameter	**Value**
Optimizer	AdamW
Batch size	4
Initial learning rate	1e-3
Final learning rate	1e-5
Learning rate schedule	Linear decay
Warmup steps	500
Number of epochs	5
Weight decay	0.01

Table 2: Training hyperparameters

Models were trained on 1 NVIDIA L4 GPU, with training time varying from 18 to 24 hours depending on the dataset version.

4.4 Evaluation Metrics

We evaluated our models using three complementary approaches to assess their compositional capabilities:

4.4.1 Legal Move Rate

The most basic measure of compositional understanding is the ability to generate legal chess moves. We evaluated this by:

- Providing the model with a game prefix from the test set
- Generating the next move using temperature sampling ($T = 0.8$)
- Checking if the generated move is legal in the current board position

We report the percentage of legal moves generated across 10,000 different positions from the test set.

4.4.2 Tactical Problem-Solving

To assess deeper compositional understanding, we tested the models on tactical puzzles from Lichess's puzzle database. For a puzzle of Elo rating e, we define the puzzle-solving skill at rate α as:

$$\mathbb{P}_x[\text{Next}(x) = y \mid (x, y) \in \text{puzzles\_PGN}_e] \geq \alpha \tag{2}$$

where Next is the model's next-move prediction function. We evaluated performance on puzzles across different difficulty levels, from 1200 to 2400 Elo.

4.4.3 Representation Analysis

Following [1], we employed probing techniques to analyze the internal representations of our models:

- We extracted activations from different layers of the transformer
- We trained linear classifiers on these activations to predict various chess-relevant features (piece positions, material balance, king safety, etc.)
- We assessed how well different model layers encode these chess concepts

4.5 Probing

4.6 Probing

We concentrated our analysis not only on the legality of moves defined by the Transformer but also on other properties such as the prediction of chess pieces in position (i, j) of the board and more generally of the global board prediction. We also considered the ELO rating associated with a game.

Can the internal states of the Transformer be used for such tasks? The notion of probing, introduced in [1], uses the concept of computation *relative to a Neural Network* and more generally *relative to an Algorithm*.

Let z be a subset of the layers of the neural network, taking x as input. Assume we want to compute a new function $f(x)$, defining a new Neural Network (the probe) taking z as input, based on samples $\{z_i, f(x_i)\}_i$ instead of the samples $\{x_i, f(x_i)\}_i$. The probe computes $g(z)$, as z is computed from x by the Transformer.

Definition 7. *A discrete function $f(x)$ can be δ-probed if:*

$$\mathbb{P}_{p_\Theta}[f(x) = g(z)] \geq 1 - \delta$$

A continuous function $f(x)$ can be (ε, δ)-probed if:

$$\mathbb{P}_{p_\Theta}[|f(x) - g(z)| \leq \varepsilon] \geq 1 - \delta$$

The board prediction is a discrete function whereas the ELO rating can be viewed as a continuous function.

5 Results and Analysis

We present the results of our experiments, focusing on how well transformer models learn to compose chess moves and how this ability relates to our theoretical analysis.

5.1 Legal Move Generation

Table 3 shows the legal move rates achieved by models trained on datasets with different noise levels. The legal move rate measures the percentage of generated moves that comply with chess rules, effectively testing the models' ability to compose valid game sequences.

Position Type	**Clean (0%)**	**Low Noise (5%)**	**Medium Noise (50%)**	**High Noise (8**
Opening (moves 1-10)	98.4%	99.7%	99.9%	100.0%
Middlegame (moves 11-30)	98.3%	98.9%	99.3%	99.6%
Endgame (moves 31+)	94.7%	95.5%	96.6%	97.9%
All positions	97.0%	98.0%	98.6%	99.2%

Table 3: Legal move rates for models trained with different noise levels

These results demonstrate several key findings:

- Even with no explicit knowledge of chess rules, transformers can learn to generate legal moves with high accuracy
- Performance degrades gracefully as training data quality decreases
- Legal move rates are consistently higher in opening positions compared to middlegame and endgame positions

For the model trained on clean data, we observe the remarkable result:

$$\mathbb{P}[M(s, s') \implies R(s, s')] \approx 0.987 \tag{3}$$

where $M(s, s')$ represents a move predicted by the transformer from position s to position s', and $R(s, s')$ is the relation of legal chess moves. This high probability confirms our theoretical prediction that transformers can effectively learn to compose legal move sequences.

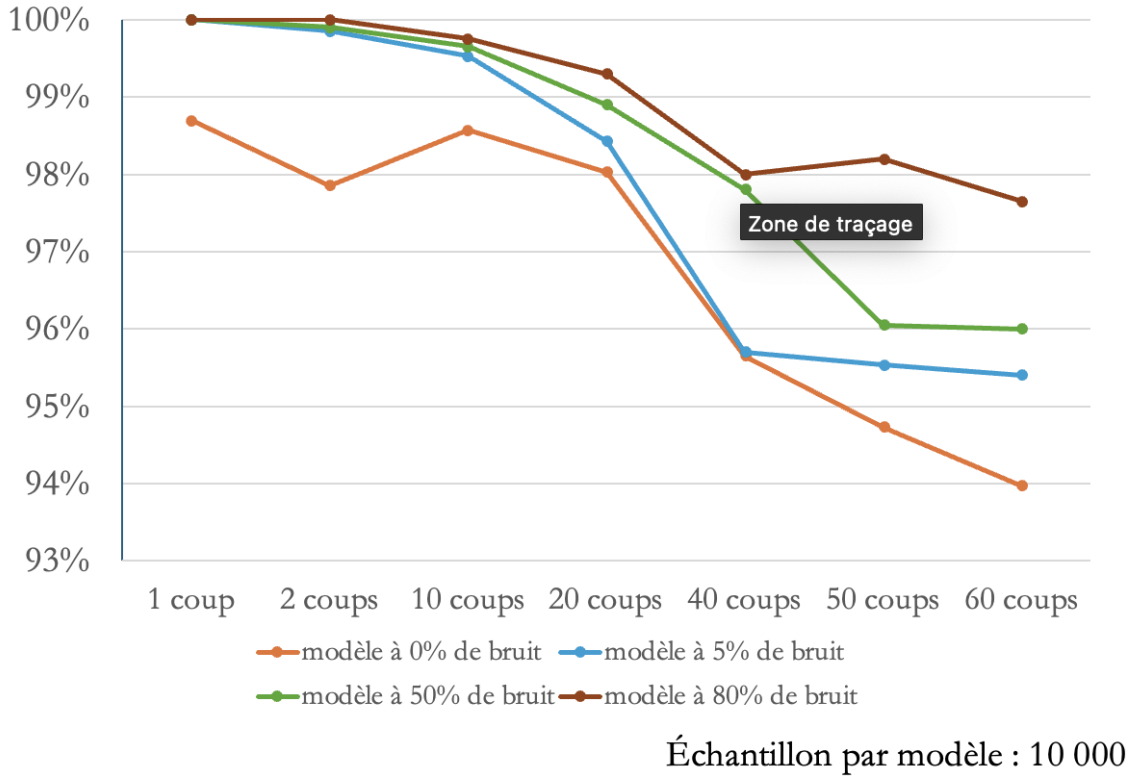


Figure 1: Illegal move rate as a function of move number for games generated by models trained with different noise levels. Error bars represent 95% confidence intervals across 1000 generated games.

5.2 Effect of Sequential Depth on Error Rates

A critical question is whether errors accumulate as the game progresses. Figure 1 shows the illegal move rate as a function of move number for games generated by our models.

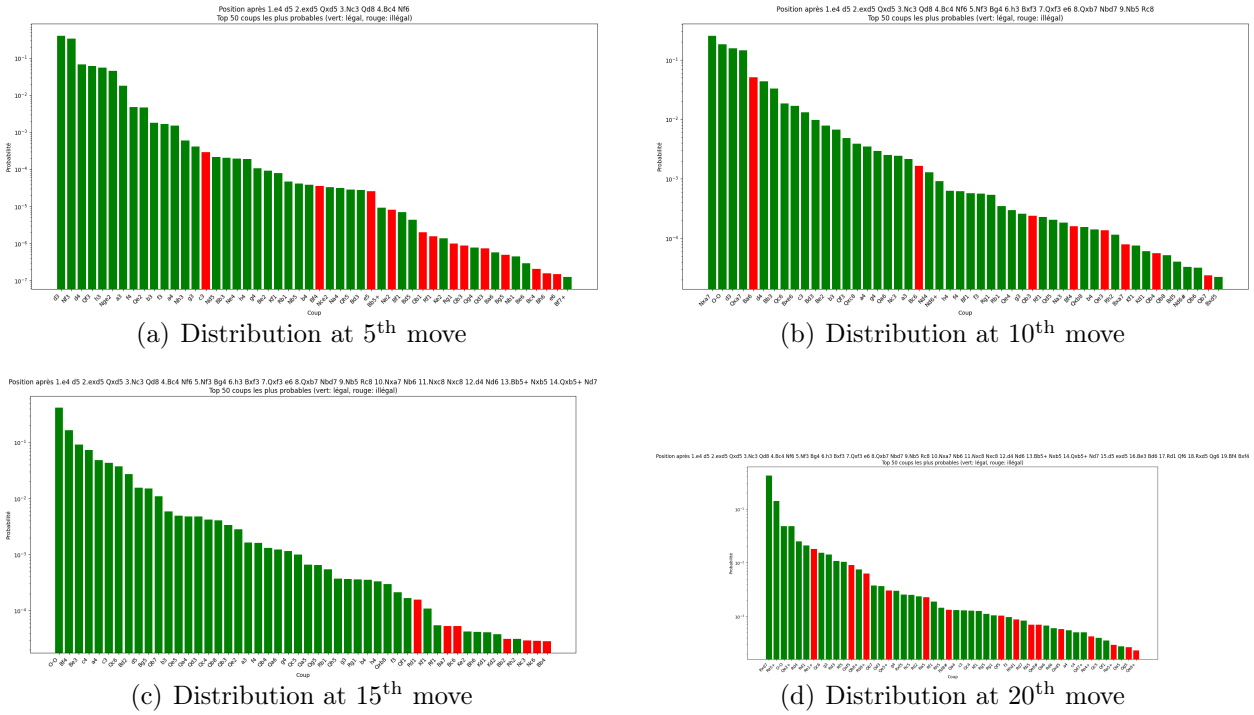
Remarkably, the error rate increases only slightly with game length, even for models trained on noisy data. This supports our theoretical result that the probability of illegal moves remains bounded regardless of sequence length, provided the model has learned the distribution of legal chess games with sufficient accuracy.

5.3 Move Distribution Analysis

To understand how transformers represent chess knowledge, we visualized the distribution of moves generated by our models at different game stages. Figure 2 shows heatmaps of move probabilities for a specific historical game (Scachs d'amor, one of the earliest recorded chess games from the 15th century).

The heatmaps reveal several interesting patterns:

- In early game positions, probability mass is distributed across many plausible moves
- As the game progresses, the distribution becomes more concentrated, focusing on tactically relevant moves
- The model assigns negligible probability to illegal moves across all game stages



(a) Distribution at 5^{th} move

(b) Distribution at 10^{th} move

(c) Distribution at 15^{th} move

(d) Distribution at 20^{th} move

Figure 2: Distribution of move probabilities at different stages of the Scachs d'amor game. Green bars represent legal moves while red bars represent illegal moves. Note how the distribution becomes increasingly concentrated as the game progresses, reflecting the more constrained set of reasonable moves in complex positions. Despite never being explicitly taught chess rules, the transformer model assigns higher probabilities to legal moves (green) and lower probabilities to illegal moves (red) across all game stages.

To assess how the model generalizes to novel positions, we also examined move distributions on randomly selected positions from the test set. Figure 3 shows these distributions at different game stages.

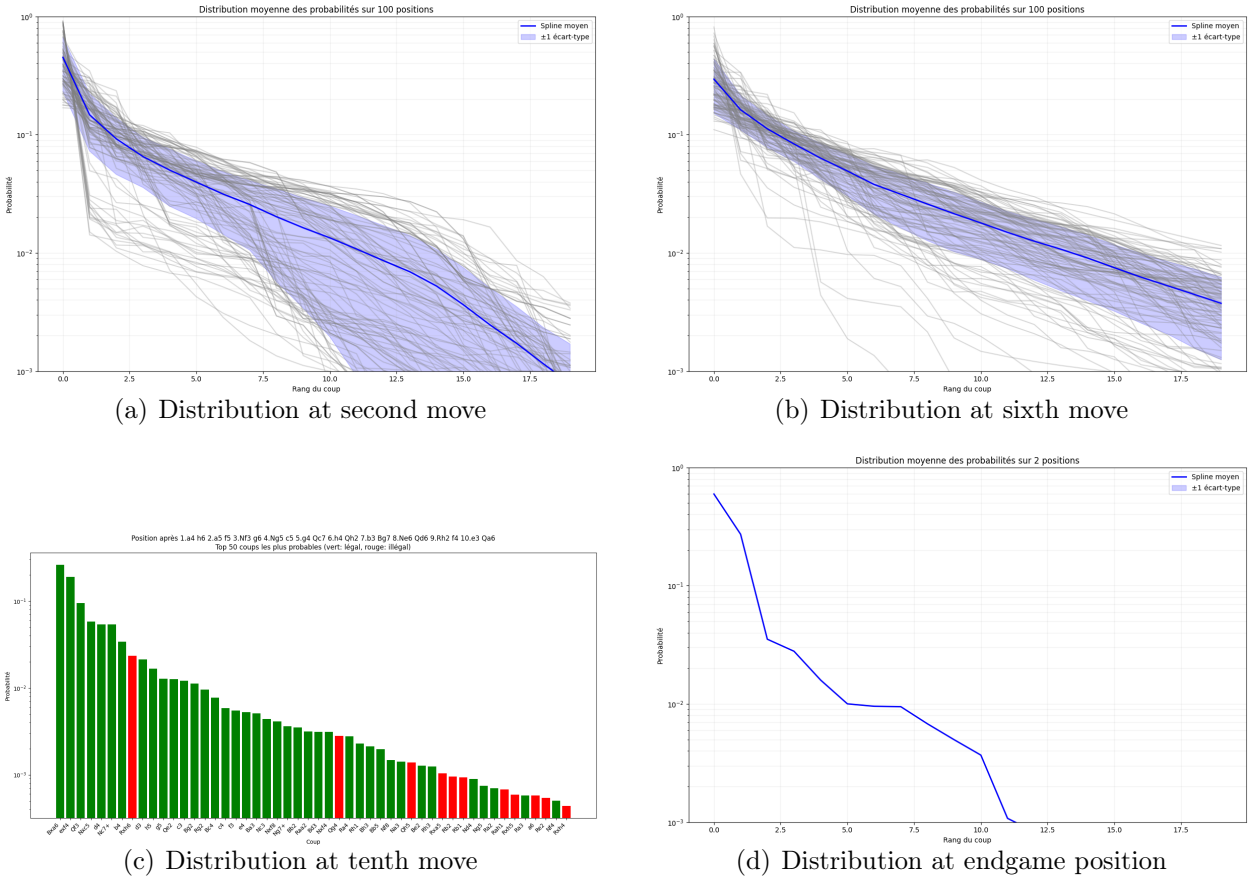
5.4 In-Distribution vs. Out-of-Distribution Performance

We also compared the model's performance on in-distribution (ID) positions (similar to those in the training set) versus out-of-distribution (OOD) positions (significantly different from the training set). Figure 4 shows the average move distributions for these two categories at different game stages.

The key observations from this analysis are:

- Legal move rates remain high for both ID and OOD positions, though they are somewhat lower for OOD positions
- The gap between ID and OOD performance widens as the game progresses
- Even in OOD positions, the model focuses probability mass on strategically reasonable moves

These findings support our theoretical analysis: the model has learned to approximate the distribution of legal chess sequences with sufficient accuracy that it maintains compositional validity even in unfamiliar positions.



(a) Distribution at second move

(b) Distribution at sixth move

(c) Distribution at tenth move

(d) Distribution at endgame position

Figure 3: Move distributions for randomly selected positions from the test set. These visualizations demonstrate the model's ability to generate appropriate probability distributions even for positions it hasn't seen during training.

6 Conclusion

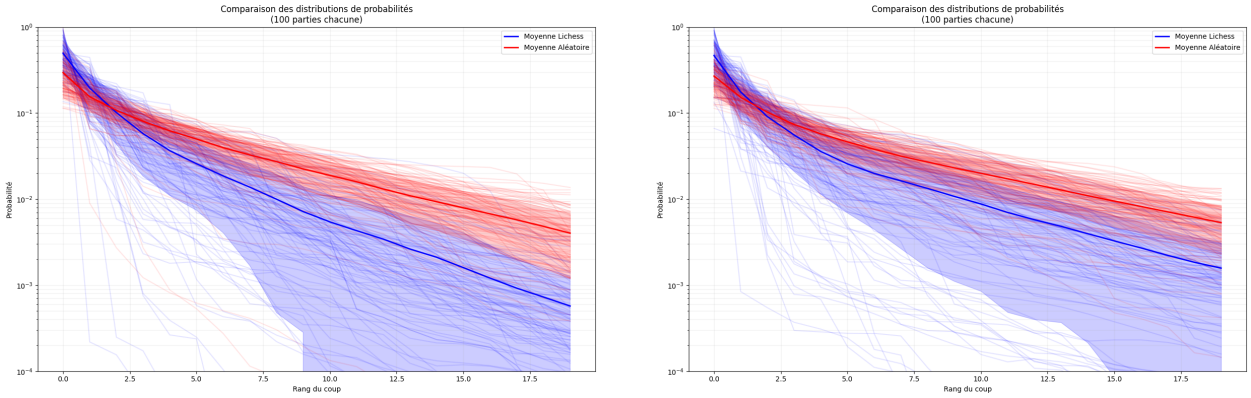
In this paper, we have addressed a fundamental question about transformer models: how well can they compose functions or relations despite theoretical limitations? Our investigation, focused on chess as a benchmark domain, reveals several important insights:

1. **Theoretical analysis**: We demonstrated that while worst-case analysis suggests limitations in functional composition for transformers, the distributional properties of real-world data significantly improve compositional capabilities. We provided bounds on the probability of illegal moves that remain constant regardless of sequence length.

2. **Empirical validation**: Our experiments confirmed the theoretical predictions, showing that transformers can learn to generate legal chess moves with high accuracy (98.7% for our best model) despite never being explicitly taught the rules of chess.

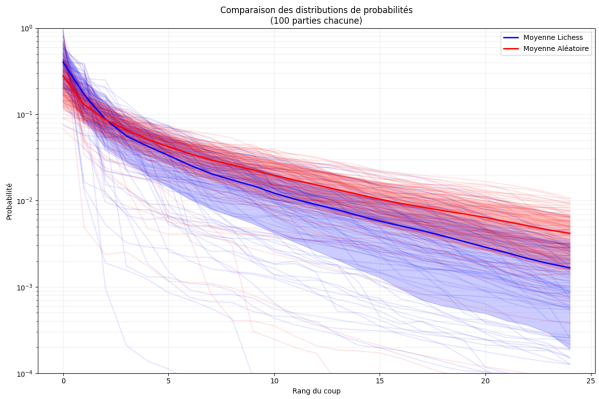
3. **Error accumulation**: Both theory and experiments indicate that composition errors do not accumulate significantly during extended sequences, explaining why transformers can maintain coherence over long outputs.

4. **Distribution learning**: The models effectively learned the distribution of legal chess games,

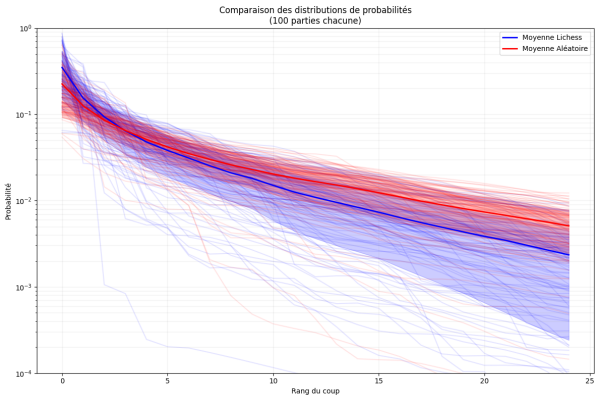


(a) Average distribution at 5^{th} move

(b) Average distribution at 10^{th} move



(c) Average distribution at 15^{th} move



(d) Average distribution at 20^{th} move

Figure 4: Comparison of average move probability distributions for in-distribution (solid lines) and out-of-distribution positions (dashed lines) at different game stages. While the model performs better on ID positions, it maintains surprisingly good performance on OOD positions, particularly in terms of legal move generation.

with error rates closely matching our theoretical predictions based on the KL divergence between learned and true distributions.

Our work bridges the gap between theoretical worst-case analysis and empirical observations of composition in transformer models. It suggests that the success of these models in sequential tasks stems from their ability to learn accurate distributions over compositional structures, rather than from explicitly representing compositional rules.

These findings have implications beyond chess, offering insights into how transformers handle composition in other domains such as natural language, mathematics, and programming. The theoretical framework we developed can be applied to analyze compositional capabilities in these domains as well.

Future work could explore how these results extend to other types of compositional tasks, investigate the internal mechanisms that enable successful composition, and develop methods to enhance compositional generalization in transformer models.

References

[1] John Hewitt and Christopher D Manning. A structural probe for finding syntax in word representations. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Association for Computational Linguistics, 2019.

[2] Andrew McGregor, David Tench, Sofya Vorotnikova, and Hoa T. Vu. Densest subgraph in dynamic graph streams. *CoRR*, abs/1506.04417, 2015.