# Applying AI to discover faster matrix multiplication algorithms

Yixuan Wang (BSc Student)
Supervisor: Pedram Hekmati

Summer Research Scholarship

Project Report

Department of Mathematics

The University of Auckland

2023-2024

# Career Development

The University of Auckland summer research was a pivotal experience that deepened my engagement with the mathematical sciences. It solidified my dedication to advancing in the field of Applied Mathematics and Statistics, particularly in developing efficient computational algorithms. My supervisor, Professor Pedram Hekmati, played a crucial role in my academic growth, offering profound insights into complex mathematical theories and the delicate nuances of algorithmic development. His guidance was key in sharpening my skills for the rigorous tasks of mathematical problem-solving and algorithmic efficiency.
Collaborating with peers who share a passion for mathematics, I gained diverse perspectives that enriched my understanding and opened avenues for potential research collaborations. This scholarly exchange has been invaluable, reinforcing my resolve to contribute meaningfully to mathematical research and its applications in technology and science.

# Research Summary and Significance

This report examines the advancement of algorithms for efficient matrix multiplication, a crucial operation in computational mathematics. It revisits Strassen’s algorithm which minimized scalar multiplications, setting a precedent for optimizing computational procedures. The study then transitions to framing matrix multiplication as tensor decomposition, introducing a conceptual basis for further algorithmic developments. This concept is operationalized through TensorGame, a platform that leverages Deep Reinforcement Learning to train AI in optimizing tensor decompositions.
Results showcase the AI-derived algorithms' effectiveness compared to traditional methods, particularly for square matrices, indicating a step forward in computational mathematics. The summary highlights the intersection of classical algorithmic theory with AI, underscoring the potential of this synergy for future computational breakthroughs.

# Abstract

This report investigates the enhancement of matrix multiplication algorithms through tensor decomposition and Deep Reinforcement Learning (DRL). Utilizing the innovative TensorGame, the study showcases the AI's prowess in refining algorithmic efficiency for square matrices. The findings reveal a promising direction for computational mathematics, highlighting the potential of DRL in algorithmic design and optimization.

# Contents

# 1 Introduction

In this section, we will introduce the basic matrix multiplication algorithm and outline Strassen's algorithm for square matrices.

## 1.1 Matrix Multiplication

Standard Algorithm: For an $n \times m$ matrix $A$ and an $m \times p$ matrix $B$, if $C = AB$, then $C$ is an $n \times p$ matrix with entries $c_{ij} = \sum_{k=1}^{m} a_{ik} b_{kj}$.

Standard algorithm

$$
\begin{aligned}
c_1 &= a_1 b_1 + a_2 b_3 \\
c_2 &= a_1 b_2 + a_2 b_4 \\
c_3 &= a_3 b_1 + a_4 b_3 \\
c_4 &= a_3 b_2 + a_4 b_4 \\
&\texttt{return } c_1, c_2, c_3, c_4
\end{aligned}
$$

$$
\underbrace{\begin{pmatrix} a_1 & a_2 \\ a_3 & a_4 \end{pmatrix}}_{\text{A}} \cdot \underbrace{\begin{pmatrix} b_1 & b_2 \\ b_3 & b_4 \end{pmatrix}}_{\text{B}} = \underbrace{\begin{pmatrix} c_1 & c_2 \\ c_3 & c_4 \end{pmatrix}}_{\text{C}}
$$

For example, if we have two $2 \times 2$ matrices $A$ and $B$, then it takes 4 additions and 8 multiplications in total to calculate their product $C = AB$.

$$
\begin{matrix} A & & B & & C \end{matrix}
$$

$$
\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \times \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}
$$

Eight Multiplications

$$
\begin{cases}
1 \times 5 + 2 \times 7 = 19 \\
1 \times 6 + 2 \times 8 = 22 \\
3 \times 5 + 4 \times 7 = 43 \\
3 \times 6 + 4 \times 8 = 50
\end{cases}
$$

Executing eight multiplication steps for two 2×2 matrices may seem simple, but this becomes considerably more complex and time-consuming for larger matrices. The key reason lies in the binary arithmetic used by computers: addition is a straightforward, single-cycle operation for each bit, whereas multiplication involves multiple cycles of addition and bit-shifting. This difference significantly increases the computational load in standard matrix multiplication, underscoring the need for more efficient algorithms in handling larger matrices.

Hence, decreasing the number of multiplications in any computational process, even if it means increasing additions or subtractions, can effectively accelerate the entire operation, leading to a quicker overall implementation.

Consider the following illustration: we compare two algorithms for calculating the difference between the squares of two numbers, a and b. The figure below demonstrates that while both methods yield the identical outcome, the latter is notably more efficient, utilizing only half the number of multiplications required by the former.

**Algorithm 1**

$$m_1 = a \cdot a$$
$$m_2 = b \cdot b$$
$$\texttt{return} \quad m_1 - m_2$$

**Algorithm 2**

$$c_1 = a + b$$
$$c_2 = a - b$$
$$\texttt{return} \quad c_1 \cdot c_2$$

$$(a+b) \cdot (a-b) = a^2 - b^2$$

## 1.2 Strassen's Algorithm

Despite extensive research by mathematicians to find a more efficient algorithm, it was not until 1969 that Volker Strassen made a breakthrough. He introduced an innovative method that required only seven multiplication steps.

**Strassen algorithm**

$$m_1 = (a_1 + a_4)(b_1 + b_4)$$
$$m_2 = (a_3 + a_4)(b_1)$$
$$m_3 = (a_1)(b_2 - b_4)$$
$$m_4 = (a_4)(b_3 - b_1)$$
$$m_5 = (a_1 + a_2)(b_4)$$
$$m_6 = (a_3 - a_1)(b_1 + b_2)$$
$$m_7 = (a_2 - a_4)(b_3 + b_4)$$

$$c_1 = m_1 + m_4 - m_5 + m_7$$
$$c_2 = m_3 + m_5$$
$$c_3 = m_2 + m_4$$
$$c_4 = m_1 - m_2 + m_3 + m_6$$
$$\texttt{return} \ c_1, c_2, c_3, c_4$$

$$\begin{matrix} & A & & B & & C \\ \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} & \times & \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} & = & \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix} \end{matrix}$$

Seven Multiplications

$$\begin{cases} (1+4)\times(5+8)=65 \\ (3+4)\times 5=35 \\ (6-8)\times 1=-2 \\ (-5+7)\times 4=8 \\ (1+2)\times 8=24 \\ (-1+3)\times(5+6)=22 \\ (2-4)\times(7+8)=-30 \end{cases}$$

$$\begin{aligned} 65+8-24+(-30) &= 19 \\ -2+24 &= 22 \\ 35+8 &= 43 \\ 65-35+(-2)+22 &= 50 \end{aligned}$$

Strassen's algorithm optimizes matrix multiplication beyond the basic 2x2 case by exploiting the nature of matrices as arrays of smaller submatrices. Consider a 2,000×2,000 matrix, which can be treated as a 2×2 matrix where each element is itself a 1,000×1,000 submatrix. By recursively breaking these down into 500×500 blocks, Strassen’s algorithm applies its multiplication strategy efficiently at each level. The larger the matrices involved, the more pronounced the benefits: fewer multiplications translate to significant computational efficiency gains, making Strassen's method especially advantageous for large-scale matrix computations.

# 2 Tensor Decomposition

In this section, we explore innovative strategies for discovering new matrix multiplication algorithms. A practical method involves reframing the challenge within the context of tensor decompositions.

## 2.1 Matrix Multiplication as Tensor Decomposition

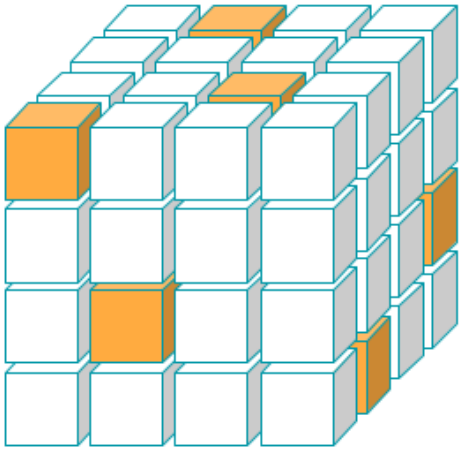
Tensor decomposition, a vital concept in this exploration, refers to the process of breaking down a tensor into simpler, constituent parts. Tensors themselves are an extension of matrices, representing multidimensional arrays of numbers. While matrices, or **2D-tensors**, are organized as two-dimensional grids indexed by two indices i and j, tensors extend this concept to higher dimensions.

$$A = [a_{ij}]$$

For example, a 3D-tensor involves a third index, k, creating a three-dimensional array.

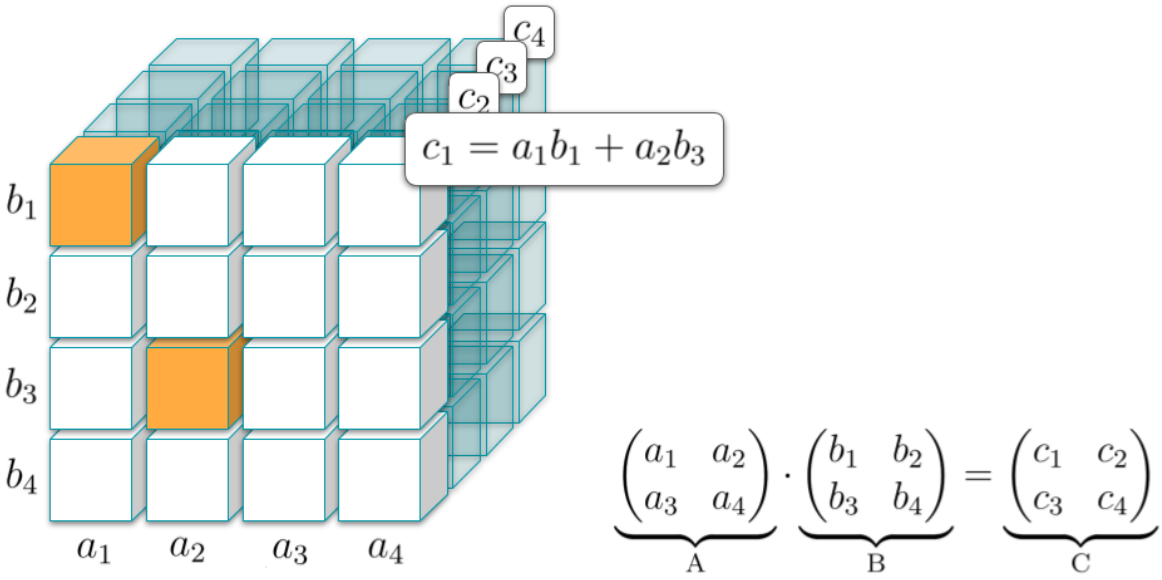
$$\mathcal{T} = [t_{ijk}]$$

A 3D-tensor, expanding upon the concept of a matrix or a 2D-tensor, can be visualized as a cube of numbers, with each element defined by three coordinates. Just like a matrix is represented as a rectangular grid, a 3D-tensor manifests as a three-dimensional array. Particularly when a tensor consists solely of 0s and 1s, this structure allows for a clear visualization: 1s can be marked with colored boxes, while the zeros remain uncolored, as illustrated in the following image:



The concept that a matrix multiplication algorithm equates to a tensor decomposition starts with this key insight: for predetermined matrix sizes, a unique 3D-tensor T, composed only of 0s and 1s, accurately embodies the multiplication of two matrices A and B resulting in C. Intriguingly, any breakdown of this tensor T not only represents the multiplication process but also provides a new method for performing the multiplication. In essence, discovering novel algorithms for matrix multiplication is fundamentally about finding different ways to decompose the relevant tensor.

To demonstrate this concept, let's consider the example of multiplying two 2x2 matrices:




The depicted tensor T in the figure illustrates the process of 2x2 standard matrix multiplication. To expound this in a coordinate-based approach, the $\boldsymbol{a_n}$would be in horizontal way and the $\boldsymbol{b_n}$ would be in vertical way. The highlighted squares in the tensor correspond to the scalar products that need to be computed and summed to obtain the entries of the resulting $c_n$ The orange square means we multiply the corresponding coordinates, and the sum of these two orange squares is our $\boldsymbol{c_1}$.The subsequent segments of T similarly guide the computation of C's other elements.

Here's Strassen's algorithm using tensor decomposition.

$$
\begin{aligned}
m_1 &= (a_1 + a_4)(b_1 + b_4) \\
m_2 &= (a_3 + a_4)\, b_1 \\
m_3 &= a_1\,(b_2 - b_4) \\
m_4 &= a_4\,(b_3 - b_1) \\
m_5 &= (a_1 + a_2)\, b_4 \\
m_6 &= (a_3 - a_1)(b_1 + b_2) \\
m_7 &= (a_2 - a_4)(b_3 + b_4) \\
c_1 &= m_1 + m_4 - m_5 + m_7 \\
c_2 &= m_3 + m_5 \\
c_3 &= m_2 + m_4 \\
c_4 &= m_1 - m_2 + m_3 + m_6
\end{aligned}
$$

$$
\mathbf{U} = \begin{pmatrix}
1 & 0 & 1 & 0 & 1 & -1 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 \\
1 & 1 & 0 & 1 & 0 & 0 & -1
\end{pmatrix}
$$

$$
\mathbf{V} = \begin{pmatrix}
1 & 1 & 0 & -1 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 \\
1 & 0 & -1 & 0 & 1 & 0 & 1
\end{pmatrix}
$$

$$
\mathbf{W} = \begin{pmatrix}
1 & 0 & 0 & 1 & -1 & 0 & 1 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 \\
1 & -1 & 1 & 0 & 0 & 1 & 0
\end{pmatrix}
$$

The matrices U, V, and W in the tensor decomposition correspond to the positions of a, b, c, separately. For example, the first operation utilizes elements a1 and a4 from matrix A, and accordingly, only these positions are marked with 1 in the U matrix to denote their selection. This methodical approach reduces the overall number of scalar multiplications required, thus optimizing the process of matrix multiplication.

Once we fix the dimensions of matrices A and B, constructing tensor T becomes quite straightforward. The pivotal aspect here is that T allows for multiple unique decompositions into sums of outer products of three vectors, as follows:

$$
\mathcal{T} = \sum_{t=1}^{R} \mathbf{u}^{(t)} \otimes \mathbf{v}^{(t)} \otimes \mathbf{w}^{(t)}
$$

A tensor decomposition breaks down tensor T into several components. The total number of these components, denoted as R, is known as the decomposition's rank. This rank is directly tied to the count of multiplications needed in the matrix multiplication algorithm derived from this particular decomposition.

### 2.1.1 Outer Product

An outer product is an operation that takes two vectors and produces a matrix or higher-order tensor. For vectors $\boldsymbol{u}$ and $\mathbf{v}$, their outer product, denoted by $u \otimes v$ **,** results in a matrix where the element at the $i_{th}$ row and $j_{th}$column is the product of the $i_{th}$ element of $\boldsymbol{u}$ and the $j_{th}$ element of $\mathbf{v}$. In the context of tensor decomposition, the outer product

extends to three vectors, generating a 3D-tensor whose elements are products of the corresponding elements of each vector.

Here is how Strassen's algorithm unfolds using tensor decomposition:
Let $A$ and $B$ be $2 \times 2$ matrices that we want to multiply and let $C$ be the resulting matrix. Strassen's algorithm decomposes the multiplication process into seven unique products of linear combinations of the elements of $A$ and $B$.

In Strassen's algorithm, each outer product of vectors $u^{(t)}$, $v^{(t)}$, and $w^{(t)}$ constructs a rank-1 tensor. When these tensors are summed, they form the complete tensor $T$, which represents the maytrix multiplication $AB = C$.

$$T = \sum_{t=1}^{7} u^{(t)} \otimes v^{(t)} \otimes w^{(t)}$$

For each index $t$ from 1 to 7, the vectors $u^{(t)}$, $v^{(t)}$, and $w^{(t)}$ correspond to a unique set of operations. The outer product $u^{(t)} \otimes v^{(t)} \otimes w^{(t)}$ creates a tensor that contributes to a single element of the resulting matrix $C$ when the calculations involving elements of $A$ and $B$ are completed. The sum of these products, across all values of t, yields the reconstructed matrix $C$.

To compose the tensor $T$, we add up all the rank-1 tensors created by these outer products. This product is not an approximation but an exact representation of the multiplication process:

$$\mathcal{T} = \mathbf{u}^{(1)} \otimes \mathbf{v}^{(1)} \otimes \mathbf{w}^{(1)} + \cdots + \mathbf{u}^{(7)} \otimes \mathbf{v}^{(7)} \otimes \mathbf{w}^{(7)}$$

Each rank-1 tensor produced by these outer products represents a single multiplication step in Strassen's algorithm, and their sum constitutes the tensor $T$ that fully represents the matrix multiplication $AB = C$.

Let's get back to our previous example:

Adding up all the rank-1 tensors created by these outer products:

$$T = \left(u^{(1)} \otimes v^{(1)}\right) \otimes w^{(1)} + \left(u^{(2)} \otimes v^{(2)}\right) \otimes w^{(2)} \ldots + \left(u^{(7)} \otimes v^{(7)}\right) \otimes w^{(7)}$$

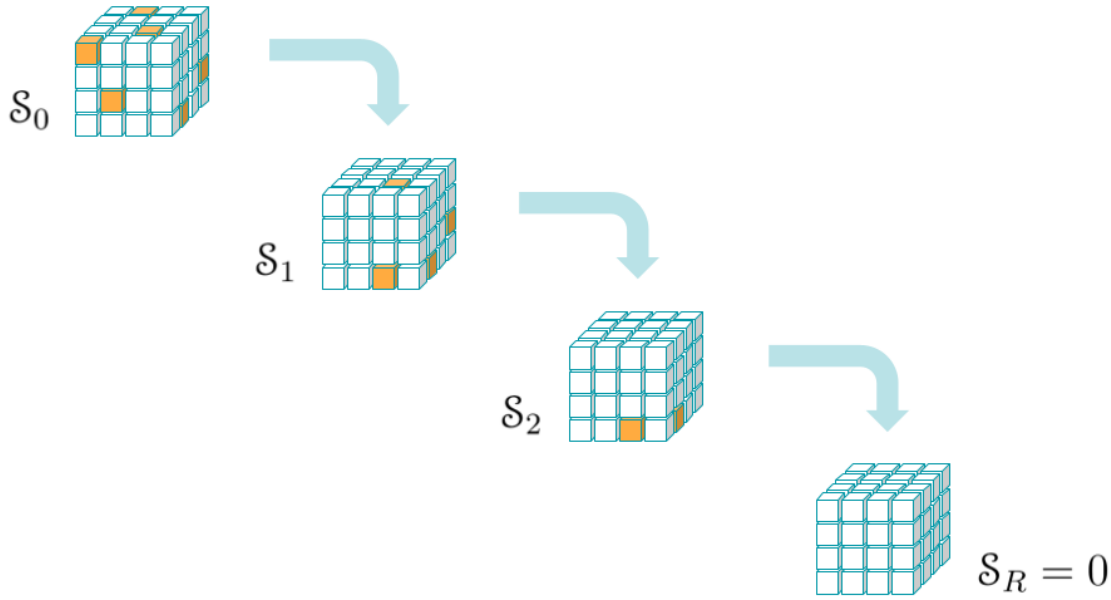
To encapsulate, unearthing a new algorithm that necessitates exactly $R$ multiplications can be achieved by decomposing the tensor $T$ into a sum of $R$ exact products of the form $u \otimes v \otimes w$ as mentioned previously.
The challenge lies in the sheer volume of potential decompositions, which is as vast as the plethora of matrix multiplication algorithms in existence. Navigating this expansive combinatorial landscape demands innovative and sophisticated approaches. For instance, Alpha Tensor addresses this by engaging with a three-dimensional board game called Tensor Game.

# 3 Tensor Game

In this section, we investigate the concept of Tensor Game, a novel approach that casts tensor decomposition as a reinforcement learning challenge within a 3D board game framework.

## 3.1 Goal of Tensor Game

The primary aim with the game is to determine a decomposition of a given tensor $T$, into a sum of exactly $R$ outer products, where $R$ represents the minimal number of multiplicative operations required in the matrix multiplication algorithm. This pursuit aligns with the overarching goal of enhancing computational efficiency in matrix multiplication tasks.




## 3.2 Steps of Tensor Game

The Tensor Game is executed in a sequence of strategic actions, each geared towards deconstructing the designated tensor $T$. The procedural steps are:
Initial Step (t=0): We begin with the tensor $T$, which we aim to decompose:

$$\mathcal{S}_0 = \mathcal{T}$$

Progressive Steps (t=1,2,3,…): In each round, the player chooses a trio of vectors $u^{(t)}$, $v^{(t)}$, and $w^{(t)}$.

The state of play, $S_t$, evolves by deducting the outer product of these vectors from the preceding state, calculated as:

$$\mathcal{S}_t = \mathcal{S}_{t-1} - \mathbf{u}^{(t)} \otimes \mathbf{v}^{(t)} \otimes \mathbf{w}^{(t)}$$

Conclusion of the Game: After $R$ iterations, the game reaches its end, with the objective of transforming the tensor into a null state:

$$\mathcal{S}_R = 0 \quad \Rightarrow \quad \mathcal{S}_0 - \sum_{t=1}^{R} \mathbf{u}^{(t)} \otimes \mathbf{v}^{(t)} \otimes \mathbf{w}^{(t)} = 0$$

This endpoint signifies a successful decomposition of the original tensor $T$ into a series of outer products. The target is to accomplish this with a minimal number of steps, reflecting the count of multiplications in the algorithm being designed. A system of incentives penalizes unnecessary moves to promote the most direct path to the zero-tensor. Exceeding a predetermined number of moves incurs additional penalties, emphasizing the importance of optimization in the decomposition process.

# 4 Deep Reinforcement Learning

In this section, we delve into Deep Reinforcement Learning (DRL), exploring its application as a powerful tool in optimizing tensor decomposition strategies within the framework of the Tensor Game.

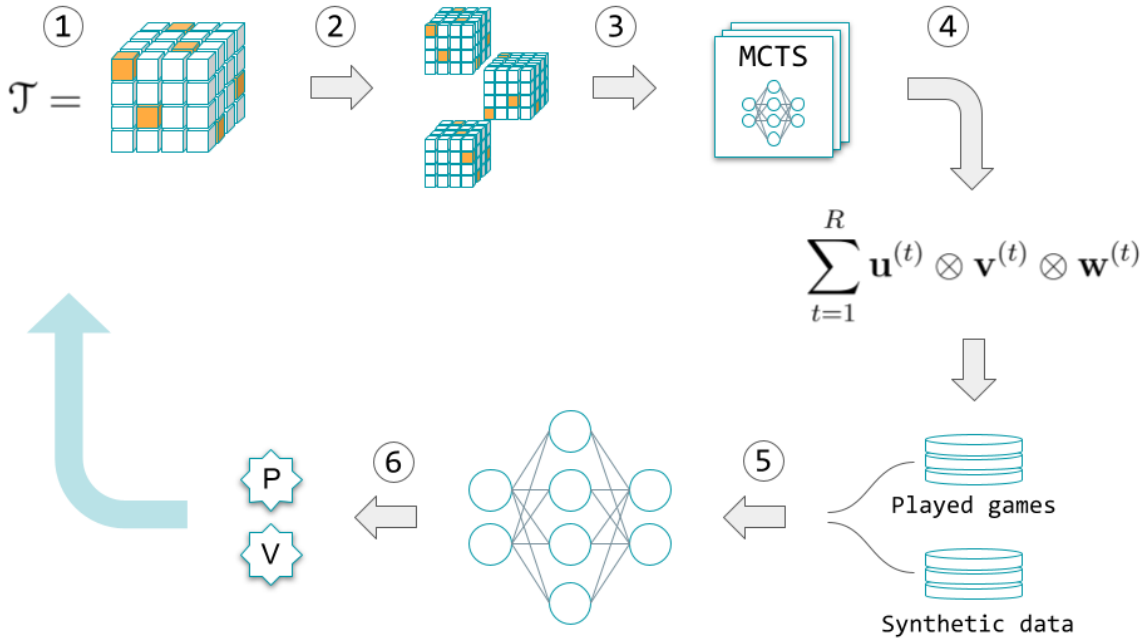
## 4.1 Definition of Deep Reinforcement Learning

Deep Reinforcement Learning (DRL) is a sophisticated computational approach that empowers machines to learn optimal strategies through direct interaction with their environment, refining their actions based on successively received feedback to achieve a predefined objective. Central to DRL is the concept of an agent that, when faced with a problem as complex as finding faster matrix multiplication algorithms, learns to navigate the intricate landscape of possible solutions.

In the specific context of the Tensor Game, DRL is utilized to train an agent to discover efficient tensor decompositions, which directly translate to faster matrix multiplication methods. By simulating the game environment, the DRL agent experiments with various sequences of outer products, effectively learning to optimize the steps required to reach the goal state. The deep neural networks within the DRL framework serve as the agent's evolving 'brain', enabling it to discern patterns and make increasingly informed decisions, ultimately leading to the formulation of more efficient algorithms for matrix multiplication.

## 4.2 Overview of the model

Alpha Tensor employs a deep reinforcement learning framework, centred around an intelligent agent tasked with exploring the realm of tensor decompositions through the gameplay of Tensor Game. The model's decision-making process hinges on a well-defined policy guided by reward functions, with each game played contributing valuable data. This data is then processed through a neural network, which iteratively refines and enhances the agent's policy and value estimations. The workflow of Alpha Tensor involves the following key steps:

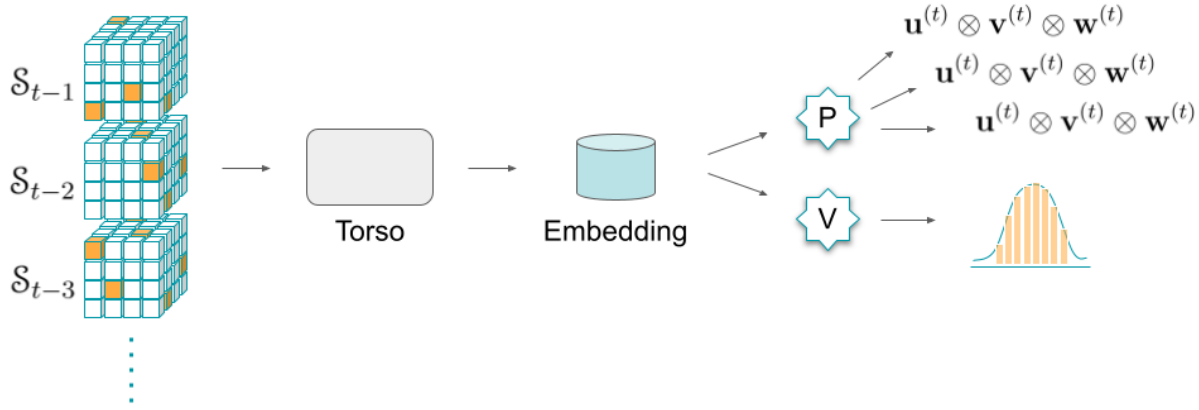



- Tensor Game Initiation: A three-dimensional tensor T, which represents the matrix multiplication $\boldsymbol{AB} = \boldsymbol{C}$, serves as the starting point.
- Data Augmentation: The tensor T undergoes a transformation through a randomized selection of basis changes, diversifying the representation of the tensor and, by extension, the gameplay. This allows **Alpha Tensor** to explore various decompositions in parallel, infusing the learning process with a rich diversity of scenarios.
- Monte Carlo Tree Search (MCTS): This is a heuristic search algorithm used for making decisions at each step, combining it with reinforcement learning to choose the next action. MCTS iteratively builds a search tree based on the exploration of possible moves and uses the policy function for decision-making and a value function to evaluate the potential of each move.
- Gameplay Output: The series of actions taken, equating to a decomposition of T, is recorded as a played game.
- Deep Reinforcement Learning Phase: The model samples states from the played games or synthetic data and feeds them into a neural network, which learns and updates the policy and value functions that inform the MCTS.

- Model Updates: With new insights gained from played games and synthetic data, the model updates the policy and value functions, readying itself for a new iteration of improved decision-making.

### 4.2.1 Data Augmentation and Monte Carlo Tree Search(MCTS)

Alpha Tensor leverages data augmentation and linear algebra to manipulate the core tensor T, preserving its rank R across different basis transformations. This process not only diversifies the tensor's representation but also ensures its fundamental properties remain unchanged. In each iteration of the Tensor Game, the model deploys Monte Carlo Tree Search (MCTS) alongside reinforcement learning to navigate the decision space. This combination allows the model to evaluate potential decompositions efficiently, using policy and value functions to guide the search towards optimal solutions.




The game's iterations are invaluable, generating tensor-factorization pairs that serve as a foundation for both reinforcement and supervised learning within the neural network. This hybrid training approach refines the model's strategy, incrementally improving its capability to decompose tensors. Through this iterative process, Alpha Tensor not only enriches its dataset with diverse tensor decompositions but also sharpens its neural network's ability to predict and evaluate the most efficient pathways for matrix multiplication.

The transformer-based neural network architecture of Alpha Tensor acts as a sophisticated guide in this process. With an axial attention mechanism at its heart, the model effectively encodes the state of the game, providing a nuanced understanding that informs both the policy and value heads. As a result, the agent's skill in identifying and executing more effective matrix multiplication algorithms is continuously enhanced, driving advancements in computational mathematics.

# 5 Results

In this section, we present the empirical findings of Alpha Tensor's computational experiments, focusing on the complexity of matrix multiplication for square matrices. The evaluation benchmarks Alpha Tensor's algorithms against the established standards and observes their performance in both modular and standard arithmetic frameworks.

| Size $(n, m, p)$ | Best method known | Best rank known | AlphaTensor rank | |
|---|---|---|---|---|
| | | | Modular | Standard |
| $(2, 2, 2)$ | (Strassen, 1969)[2] | 7 | 7 | 7 |
| $(3, 3, 3)$ | (Laderman, 1976)[15] | 23 | 23 | 23 |
| $(4, 4, 4)$ | (Strassen, 1969)[2] $(2, 2, 2) \otimes (2, 2, 2)$ | 49 | 47 | 49 |
| $(5, 5, 5)$ | $(3, 5, 5) + (2, 5, 5)$ | 98 | 96 | 98 |

The comparison reveals Alpha Tensor's capability to either meet or improve upon the best-known tensor ranks, which signify the total number of scalar multiplications required. For example, in the 4x4 matrix multiplication case, Alpha Tensor demonstrates optimization by achieving a rank of 47, against the previously best-known rank of 49. These findings, though incremental in their nature, are a testament to the potential of Alpha Tensor's approach and the strides it is making in computational mathematics.

It's crucial to note, however, that these results are derived from synthetic data simulations. The true efficacy and efficiency of Alpha Tensor's algorithms in real-world scenarios, especially those involving specialized matrix structures such as sparse matrices, have yet to be validated. Additionally, while speed enhancements are of clear interest, the importance of numerical stability in practical matrix multiplication remains a paramount concern. This aspect has not been thoroughly investigated, indicating an avenue for future research and exploration within Alpha Tensor's framework.

Conclusively, this section illustrates the contributions of Alpha Tensor to the advancement of matrix multiplication algorithms. Although not revolutionary, the system's methodical improvements and its potential to revolutionize algorithmic design in computational tasks are promising. The anticipation of applying Alpha Tensor's methodologies to a wider array of mathematical problems provides an exciting direction for future developments in the field.

# References

[1] Marco Ramponi (2022). DeepMind's AlphaTensor Explained. Assembly AI

[2] Alhussein Fawzi et al (2022). Discovering faster matrix multiplication algorithms with reinforcement learning. Nature

[3] Ben Brubaker (2022). AI Reveals New Possibilities in Matrix Multiplication. Quanta Magazine