Speeding Up Matrix Multiplication with Machine Learning

Manan Bhatia

March 19, 2025

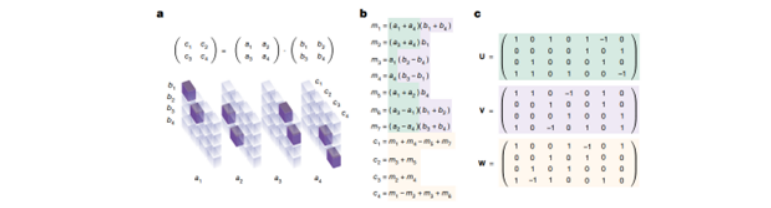
1 Introduction

Matrix multiplication is a fundamental operation in various computational fields, including scientific computing, graphics processing, and particularly in deep learning. Modern neural networks rely heavily on matrix multiplications for tasks such as training large-scale transformer architectures. However, the naive approach to matrix multiplication has a cubic time complexity of $O(N^3)$, making it a computational bottleneck for large matrices.

The quest for faster matrix multiplication algorithms has led to significant breakthroughs. Strassen's algorithm [3] reduced the complexity to $O(N^{2.81})$, demonstrating that improvements beyond the naive method are possible. Further refinements, such as the Coppersmith-Winograd (C-W) algorithm [1], reduced this to approximately $O(N^{2.376})$. Despite these advancements, discovering new and practical multiplication algorithms has been largely a manual process, requiring deep mathematical insights and extensive experimentation.

Recent developments in reinforcement learning (RL) have introduced **AlphaTensor**, a framework that automates the discovery of matrix multiplication algorithms [2]. Inspired by **AlphaZero**, AlphaTensor formulates algorithm discovery as a game, where an RL agent searches for low-rank tensor decompositions that minimise the number of scalar multiplications. This approach has successfully rediscovered classical algorithms and uncovered new ones that outperform existing methods for small matrix sizes.

This project seeks to build upon this work by implementing an RL-based system to discover efficient matrix multiplication algorithms. Additionally, the applicability will be evaluated of existing fast multiplication algorithms, particularly Strassen and C-W, in deep learning architectures such as transformers.


a
$\begin{pmatrix} c_1 & c_2 \\ c_3 & c_4 \end{pmatrix} = \begin{pmatrix} a_1 & a_2 \\ a_3 & a_4 \end{pmatrix} \cdot \begin{pmatrix} b_1 & b_2 \\ b_3 & b_4 \end{pmatrix}$
b_1 b_2 b_3 b_4
c_1 c_2 c_3 c_4
a_1 a_2 a_3 a_4
b
c

$$m_1 = (a_1 + a_4)(b_1 + b_4)$$
$$m_2 = (a_3 + a_4)b_1$$
$$m_3 = a_1(b_2 - b_4)$$
$$m_4 = a_4(b_3 - b_1)$$
$$m_5 = (a_1 + a_2)b_4$$
$$m_6 = (a_3 - a_1)(b_1 + b_2)$$
$$m_7 = (a_2 - a_4)(b_3 + b_4)$$
$$c_1 = m_1 + m_4 - m_5 + m_7$$
$$c_2 = m_3 + m_5$$
$$c_3 = m_2 + m_4$$
$$c_4 = m_1 - m_2 + m_3 + m_6$$

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

Fig. 1 | Matrix multiplication tensor and algorithms. a, Tensor $\mathcal{T}_2$ representing the multiplication of two 2 × 2 matrices. Tensor entries equal to 1 are depicted in purple, and 0 entries are semi-transparent. The tensor specifies which entries from the input matrices to read, and where to write the result. For example, as $c_1 = a_1b_1 + a_2b_3$, tensor entries located at (a_1, b_1, c_1) and (a_2, b_3, c_1) are set to 1. **b**, Strassen's algorithm[2] for multiplying 2 × 2 matrices using 7 multiplications. **c**, Strassen's algorithm in tensor factor representation. The stacked factors **U**, **V** and **W** (green, purple and yellow, respectively) provide a rank-7 decomposition of $\mathcal{T}_2$ (equation (1)). The correspondence between arithmetic operations (**b**) and factors (**c**) is shown by using the aforementioned colours.

Figure 1: Tensor representation of matrix multiplication as a rank-3 tensor. Lower-rank decompositions reduce scalar multiplications. (Source: AlphaTensor paper)

2 Literature Review

The challenge of improving matrix multiplication algorithms has been widely studied. Strassen's algorithm [3] was the first breakthrough, reducing the number of required multiplications by dividing the problem into seven recursive subproblems instead of eight. The Coppersmith-Winograd algorithm [1] further refined this approach, achieving a complexity of $O(N^{2.376})$. Current theoretical methods push this further to $O(N^{2.37})$, but practical applications still favor Strassen's algorithm due to its efficiency in real-world scenarios.

Tensor decomposition plays a crucial role in understanding and optimising matrix multiplication. It allows matrix multiplication to be expressed as a rank-3 tensor, where reducing the rank translates to fewer multiplications. Strassen's method can be interpreted as a rank-7 decomposition of the $2 \times 2 \times 2$ matrix multiplication tensor. This mathematical perspective provides a framework for both analysing known algorithms and discovering new ones.

Reinforcement learning (RL) has recently emerged as a powerful tool for algorithmic discovery. AlphaZero demonstrated that self-play and Monte Carlo Tree Search (MCTS) could surpass human expertise in complex games like chess and Go. The *AlphaTensor* model [2] extended this approach to algorithm discovery, treating tensor decomposition as a search problem. This method has successfully uncovered novel matrix multiplication algorithms that outperform traditional methods on small-scale problems.

By integrating RL-based discovery with classical algorithm analysis, this research aims to explore both theoretical and practical implications for matrix multiplication in machine learning models.

3 Methodology

This project will follow a structured approach, consisting of three main components: theoretical exploration, reinforcement learning implementation, and empirical evaluation.

The first stage involves a mathematical exploration of matrix multiplication, focusing on tensor decomposition and algorithmic complexity. Existing fast multiplication methods will be analysed and establish benchmarks for evaluating newly discovered algorithms.

In the second stage, a reinforcement learning system will be implemented inspired by AlphaZero. The RL agent will frame matrix multiplication as a search problem, where the goal is to find optimal tensor decompositions. Using Monte Carlo Tree Search (MCTS) for exploration and a neural network for policy evaluation, the agent will iteratively refine its approach through self-play. The reward function will be designed to minimise scalar multiplications while maintaining numerical stability. Training will initially focus on smaller matrices (e.g., 3×3 and 4×4) due to computational constraints.

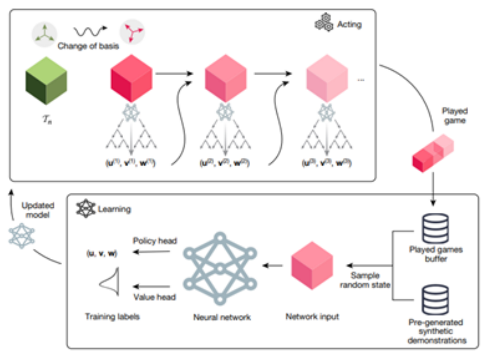

Change of basis
Acting
$\mathcal{T}_n$
$(\mathbf{u}^{(1)}, \mathbf{v}^{(1)}, \mathbf{w}^{(1)})$
$(\mathbf{u}^{(2)}, \mathbf{v}^{(2)}, \mathbf{w}^{(2)})$
$(\mathbf{u}^{(3)}, \mathbf{v}^{(3)}, \mathbf{w}^{(3)})$
Played game
Updated model
Learning
Policy head
(u, v, w)
Value head
Training labels
Neural network
Network input
Sample random state
Played games buffer
Pre-generated synthetic demonstrations

Fig. 2 | Overview of AlphaTensor. The neural network (bottom box) takes as input a tensor $\mathcal{S}_t$, and outputs samples $(\mathbf{u}, \mathbf{v}, \mathbf{w})$ from a distribution over potential next actions to play, and an estimate of the future returns (for example, of $-\text{Rank}(\mathcal{S}_t)$). The network is trained on two data sources: previously played games and synthetic demonstrations. The updated network is sent to the actors (top box), where it is used by the MCTS planner to generate new games.

Figure 2: AlphaTensor's reinforcement learning framework for algorithm discovery. (Source: AlphaTensor paper)

In the final stage, the discovered algorithms will be benchmarked against classical methods. Evaluation metrics will include: - **Computational Complexity:** Measuring the number of scalar multiplications. - **Runtime Performance:** Comparing execution times on standard datasets. - **Practical Applicability:**

Integrating discovered algorithms into transformer models and measuring their impact on self-attention mechanisms.

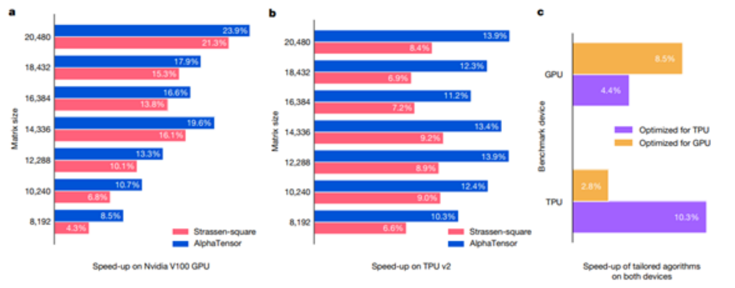


Fig. 5 | Speed-ups of the AlphaTensor-discovered algorithm. a,**b**, Speed-ups (%) of the AlphaTensor-discovered algorithms tailored for a GPU (**a**) and a TPU (**b**), optimized for a matrix multiplication of size 8,192 × 8,192. Speed-ups are measured relative to standard (for example, cuBLAS for the GPU) matrix multiplication on the same hardware. Speed-ups are reported for various matrix sizes (despite optimizing the algorithm only on one matrix size). We also report the speed-up of the Strassen-square algorithm. The median speed-up is reported over 200 runs. The standard deviation over runs is <0.4 percentage points (see Supplementary Information for more details). **c**, Speed-up of both algorithms (tailored to a GPU and a TPU) benchmarked on both devices.

Figure 3: Speed ups of the AlphaTensor-discovered algorithm (Source: AlphaTensor paper)

The approach creates a systematic investigation of matrix multiplication algorithms through machine learning which extends human intuition capabilities beyond current boundaries.

4 Contribution

The research focuses on optimising matrix multiplication techniques to enhance their application in machine learning especially within reinforcing learning frameworks and transformer systems. The following aspects will be the focus of my work in implementation using both Python and Julia:

1. Exploring the mathematics of matrix multiplication and the significance of optimisation

I will start the analysis by examining basic mathematical principles of traditional matrix multiplication algorithms followed by their computational difficulty evaluation. The research will examine fundamental algorithms starting from the straightforward $O(N^3)$ approach alongside fastest methods that include Strassen's and Coppersmith-Winograd algorithms. The implemented algorithms through Python will enable performance evaluation and statistical analysis of their computational requirements. The analysis will include matrix optimisation approaches alongside the implementation of pruning and sparsity exploitation through Julia since it excels at running essential large numerical calculations.

2. Test reinforcement learning models inspired by the AlphaZero framework to discover novel matrix multiplication algorithms.

I will use the AlphaZero framework to develop matrix multiplication algorithms through adapting its framework from its successful application to game-playing AI. The reinforcement learning environment will deploy Python together with OpenAI Gym or Stable Baselines3 platform to let agents choose matrix multiplication approaches while obtaining rewards from lower computation time and time complexity evaluation. Through this approach RL should discover improved matrix multiplication algorithms which will grow better with each subsequent iteration and learning cycle. When analysing time complexities of training algorithms during this task Julia will serve as the benchmarking tool.

3. Assess the performance of the algorithms identified by the model and explore potential improvements.

The benchmark testing phase will assess the new algorithm variants through dual criteria of matrix multiplication speedups along with CPU/GPU time and memory consumption evaluations. The benchmark tests will be built using Python with NumPy and TensorFlow as popular data science libraries to operate on deep learning matrices. Due to its superior capabilities in high-performance numerical computations the Julia language will serve as the primary tool for extensive performance evaluations involving large matrix multiplications needed for neural network dataset training on MNIST and CIFAR-10. Analysis of the results will demonstrate which parts of the RL reward structure or performance tuning parameters need adjustment for peak performance enhancement.

4. Explore using matrix multiplication algorithms for self-attention in transformers.

The study aims to determine how optimised matrix multiplication methods can be implemented for transformers especially inside their self-attention framework. I will modify the attention mechanism using Python and TensorFlow or PyTorch to integrate the optimised matrix multiplication algorithms that allow evaluation of performance effects on training duration and system memory consumption. The model inference times along with redundant calculation reduction in self-attention will be evaluated using Julia during heavy computation operations. The implementation of these modifications will prove the effectiveness of optimised matrix multiplication techniques for improving Transformer models during their training period and operational deployment.

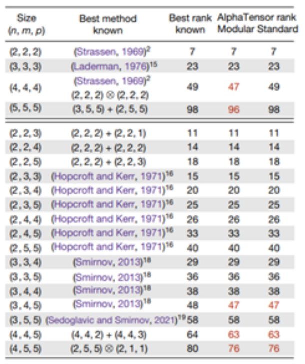
I purpose to create an optimised line of matrix multiplication algorithms which will enhance computational efficiency and provide useful applications for machine learning particularly within reinforcement learning and transformer models. By using Python and Julia the project takes advantage of machine learning tools while gaining access to necessary computational power for processing large numerical data.

5 Expected Results and Analysis

This project aims to explore the potential of reinforcement learning in discovering efficient matrix multiplication algorithms. Initially,the RL agent is expected to rediscover Strassen's algorithm and potentially uncover novel multiplication strategies that reduce the number of scalar multiplications compared to naive methods.

A key aspect of the evaluation will be benchmarking the discovered algorithms against classical techniques, focusing on Strassen's and Coppersmith-Winograd's methods. The primary performance metrics include:

- **Computational Complexity:** Measuring the reduction in multiplication steps compared to existing algorithms.
- **Runtime Efficiency:** Analysing execution times when applied to standard matrix operations and deep learning workloads.
- **Numerical Stability:** Ensuring the discovered algorithms maintain accuracy and do not introduce significant floating-point errors.
- **Practical Applicability:** Integrating the algorithms into transformer self-attention computations to assess real-world impact.



Size (n, m, p)	Best method known	Best rank known	AlphaTensor rank Modular	AlphaTensor rank 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
(2, 2, 3)	(2, 2, 2) + (2, 2, 1)	11	11	11
(2, 2, 4)	(2, 2, 2) + (2, 2, 2)	14	14	14
(2, 2, 5)	(2, 2, 2) + (2, 2, 3)	18	18	18
(2, 3, 3)	(Hopcroft and Kerr, 1971)[16]	15	15	15
(2, 3, 4)	(Hopcroft and Kerr, 1971)[16]	20	20	20
(2, 3, 5)	(Hopcroft and Kerr, 1971)[16]	25	25	25
(2, 4, 4)	(Hopcroft and Kerr, 1971)[16]	26	26	26
(2, 4, 5)	(Hopcroft and Kerr, 1971)[16]	33	33	33
(2, 5, 5)	(Hopcroft and Kerr, 1971)[16]	40	40	40
(3, 3, 4)	(Smirnov, 2013)[18]	29	29	29
(3, 3, 5)	(Smirnov, 2013)[18]	36	36	36
(3, 4, 4)	(Smirnov, 2013)[18]	38	38	38
(3, 4, 5)	(Smirnov, 2013)[18]	48	47	47
(3, 5, 5)	(Sedoglavic and Smirnov, 2021)[19]	58	58	58
(4, 4, 5)	(4, 4, 2) + (4, 4, 3)	64	63	63
(4, 5, 5)	$(2,5,5) \otimes (2,1,1)$	80	76	76

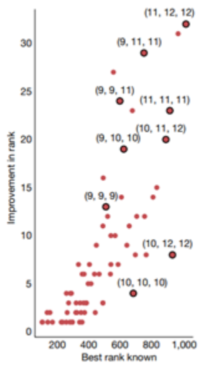
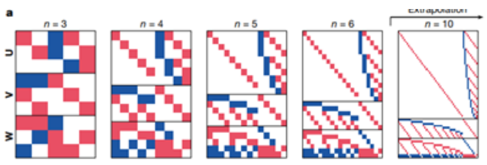


Fig. 3 | Comparison between the complexity of previously known matrix multiplication algorithms and the ones discovered by AlphaTensor. Left: column (n, m, p) refers to the problem of multiplying $n \times m$ with $m \times p$ matrices. The complexity is measured by the number of scalar multiplications (or equivalently, the number of terms in the decomposition of the tensor). 'Best rank known' refers to the best known upper bound on the tensor rank (before this paper), whereas 'AlphaTensor rank' reports the rank upper bounds obtained with our method, in modular arithmetic ($\mathbb{Z}_2$) and standard arithmetic. In all cases, AlphaTensor discovers algorithms that match or improve over known state of the art (improvements are shown in red). See Extended Data Figs. 1 and 2 for examples of algorithms found with AlphaTensor. Right: results (for arithmetic in $\mathbb{R}$) of applying AlphaTensor-discovered algorithms on larger tensors. Each red dot represents a tensor size, with a subset of them labelled. See Extended Data Table 1 for the results in table form. State-of-the-art results are obtained from the list in ref.[64].

Figure 4: Comparison between the complexity of previous known matrix multiplication algorithms and the ones discovered by AlphaTensor. (Source: AlphaTensor paper)

One expected challenge is ensuring that the RL model generalises beyond small matrices while maintaining stability. If the discovered algorithms perform well for moderate-sized matrices, further investigations could explore their scalability and potential hardware optimisations. The outcomes of this study will provide insights into whether reinforcement learning is a viable approach for discovering new, efficient linear algebra algorithms.



b

Input: $n \times n$ skew-symmetric matrix $\mathbf{A}$, vector $\mathbf{b}$.
Output: The resulting vector $\mathbf{c} = \mathbf{Ab}$ computed in $\frac{(n-1)(n+2)}{2}$ multiplications.

```
(1) for $i = 1, \ldots, n-2$ do
(2)     for $j = i+1, \ldots, n$ do
(3)         $w_{ij} = a_{ij}(b_j - b_i)$        ▷ Computing the first $(n-2)(n+1)/2$ intermediate products
(4) for $i = 1, \ldots, n$ do
(5)     $q_i = b_i \sum_{j=1}^{n} a_{ij}$        ▷ Computing the final $n$ intermediate products
(6) for $i = 1, \ldots, n-2$ do
(7)     $c_i = \sum_{j=1}^{i-1} w_{ji} + \sum_{j=i+1}^{n} w_{ij} - q_i$
(8) $c_{n-1} = -\sum_{i=1}^{n-2} \sum_{j=i+1}^{n-2} w_{ij} - \sum_{j=1}^{n-2} w_{jn} + \sum_{i=1, i \neq n-1}^{n} q_i$
(9) $c_n = -\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} w_{ij} + \sum_{i=1}^{n-1} q_i$
```

Fig. 4 | Algorithm discovery beyond standard matrix multiplication. **a**, Decompositions found by AlphaTensor for the tensors of size $\frac{n(n-1)}{2} \times n \times n$ (with $n = 3, 4, 5, 6$) representing the skew-symmetric matrix-vector multiplication. The red pixels denote 1, the blue pixels denote −1 and the white pixels denote 0. Extrapolation to $n = 10$ is shown in the rightmost figure. **b**, Skew-symmetric matrix-by-vector multiplication algorithm, obtained from the examples solved by AlphaTensor. The w_{ij} and q_i terms in steps 3 and 5 correspond to the m_r terms in Algorithm 1. It is noted that steps 6–9 do not involve any multiplications.

Figure 5: Algorithm discovery beyond standard matrix multiplication (Source: AlphaTensor paper)

This study aims to demonstrate the practicality of deploying machine learning for algorithm discovery because it results in improved performance of matrix-based deep learning operations.

6 Conclusion

This work seeks to explore whether reinforcement learning techniques can find accelerated matrix multiplication algorithms based on AlphaZero framework success stories in discovering algorithms. The project starts with thorough investigation of matrix multiplication theoretical groundwork then creates a reinforcement learning system which approaches algorithm discovery as a gameplay format. Both theoretical and practical evaluations will assess the performance of discovered algorithms with special focus on their implementation in self-attention layers of transformers.

The successful execution of this project will yield new algorithms which minimise matrix product scalar multiplication requirements based on subsequent efficiency enhancements to speed-up machine learning model training. Smaller matrices will be the subject of first experiments, but the research aims to determine practical implementation potential of these discoveries for real-world systems.

The research direction includes improving both the RL training procedure and hardware optimisation strategies while expanding the solution approach towards additional linear algebra calculations. This paper wants to offer both insights about matrix multiplication along with new methods for using machine learning in basic algorithm investigation.

7 References

References

[1] Don Coppersmith and Shmuel Winograd. "Matrix multiplication via arithmetic progressions". In: *Journal of Symbolic Computation* 9.3 (1990), pp. 251–280.

[2] Alhussein Fawzi et al. "Discovering faster matrix multiplication algorithms with reinforcement learning". In: *Nature* 610.7930 (2022), pp. 47–53.

[3] Volker Strassen. "Gaussian elimination is not optimal". In: *Numerische Mathematik* 13.4 (1969), pp. 354–356.