XX^t Can Be Faster

Dmitry Rybin *[1,2], Yushun Zhang †[1,2], and Zhi-Quan Luo‡[1,2]

[1]The Chinese University of Hong Kong, Shenzhen, China
[2]Shenzhen Research Institute of Big Data

May 16, 2025

Abstract

We present a new algorithm RXTX for computation of the product of matrix by its transpose XX^t. RXTX uses 5% less multiplications and additions and provides accelerations even for small sizes of matrix X. The algorithm was discovered by combining Machine Learning-based search methods with Combinatorial Optimization.

1 Introduction

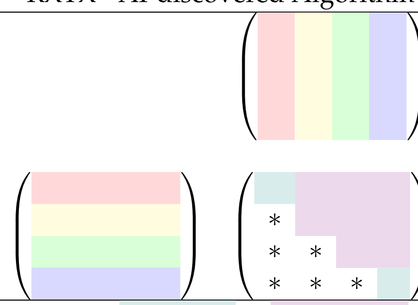
Algorithm	Previous State-of-the-Art for XX^t	RXTX - AI-discovered Algorithm
Matrix Form	$\begin{pmatrix} A & B \\ C & D \end{pmatrix}$ $\begin{pmatrix} A^t & C^t \\ B^t & D^t \end{pmatrix}$ $\begin{pmatrix} AA^t + BB^t & AC^t + BD^t \\ * & CC^t + DD^t \end{pmatrix}$	
Recursion	$S(n) = 4S(n/2) + 2M(n/2)$	$R(n) = 8R(n/4) + 26M(n/4)$
Asymptotic	$S(n) \sim \frac{2}{3}M(n)$	$R(n) \sim \frac{26}{41}M(n)$
4×4 rank	38	34

Table 1: New algorithm (RXTX) is based on recursive 4×4 block matrix multiplication. It uses 8 recursive calls and 26 general products. In comparison, previous SotA uses 16 recursive calls and 24 general products. $R(n), S(n), M(n)$ - are the number of multiplications performed by RXTX, previous SotA, and Strassen algorithm respectively for $n \times n$ matrix X. RXTX asymptotic constant $26/41 \approx 0.6341$ is 5% smaller than $2/3 \approx 0.6666$, which is asymptotic constant of previous SotA.

Finding faster matrix multiplication algorithms is a central challenge in computer science and numerical linear algebra. Since the groundbreaking results of Strassen Strassen [1969] and Winograd Winograd [1968], which demonstrated that the number of multiplications required for a general matrix product AB can be significantly reduced, extensive research has emerged exploring this problem. Techniques in the area

*Correspondence author. Email: dmitryrybin@link.cuhk.edu.cn.
†Email: yushunzhang@link.cuhk.edu.cn.
‡Email: luozq@cuhk.edu.cn

Algorithm 1 RXTX - AI-discovered asymptotic SotA for XX^t

Input: 4×4 block-matrix X

Output: $C = XX^t$ using 8 recursive calls and 26 general products.

$m_1 = (-X_2 + X_3 - X_4 + X_8) \cdot (X_8 + X_{11})^t$

$m_2 = (X_1 - X_5 - X_6 + X_7) \cdot (X_{15} + X_5)^t$

$m_3 = (-X_2 + X_{12}) \cdot (-X_{10} + X_{16} + X_{12})^t$

$m_4 = (X_9 - X_6) \cdot (X_{13} + X_9 - X_{14})^t$

$m_5 = (X_2 + X_{11}) \cdot (-X_6 + X_{15} - X_7)^t$

$m_6 = (X_6 + X_{11}) \cdot (X_6 + X_7 - X_{11})^t$

$m_7 = X_{11} \cdot (X_6 + X_7)^t$

$m_8 = X_2 \cdot (-X_{14} - X_{10} + X_6 - X_{15} + X_7 + X_{16} + X_{12})^t$

$m_9 = X_6 \cdot (X_{13} + X_9 - X_{14} - X_{10} + X_6 + X_7 - X_{11})^t$

$m_{10} = (X_2 - X_3 + X_7 + X_{11} + X_4 - X_8) \cdot X_{11}^t$

$m_{11} = (X_5 + X_6 - X_7) \cdot X_5^t$

$m_{12} = (X_2 - X_3 + X_4) \cdot X_8^t$

$m_{13} = (-X_1 + X_5 + X_6 + X_3 - X_7 + X_{11}) \cdot X_{15}^t$

$m_{14} = (-X_1 + X_5 + X_6) \cdot (X_{13} + X_9 + X_{15})^t$

$m_{15} = (X_2 + X_4 - X_8) \cdot (X_{11} + X_{16} + X_{12})^t$

$m_{16} = (X_1 - X_8) \cdot (X_9 - X_{16})^t$

$m_{17} = X_{12} \cdot (X_{10} - X_{12})^t$

$m_{18} = X_9 \cdot (X_{13} - X_{14})^t$

$m_{19} = (-X_2 + X_3) \cdot (-X_{15} + X_7 + X_8)^t$

$m_{20} = (X_5 + X_9 - X_8) \cdot X_9^t$

$m_{21} = X_8 \cdot (X_9 - X_8 + X_{12})^t$

$m_{22} = (-X_6 + X_7) \cdot (X_5 + X_7 - X_{11})^t$

$m_{23} = X_1 \cdot (X_{13} - X_5 + X_{16})^t$

$m_{24} = (-X_1 + X_4 + X_{12}) \cdot X_{16}^t$

$m_{25} = (X_9 + X_2 + X_{10}) \cdot X_{14}^t$

$m_{26} = (X_6 + X_{10} + X_{12}) \cdot X_{10}^t$

$s_1 = X_1 X_1^t$

$s_2 = X_2 X_2^t$

$s_3 = X_3 X_3^t$

$s_4 = X_4 X_4^t$

$s_5 = X_{13} X_{13}^t$

$s_6 = X_{14} X_{14}^t$

$s_7 = X_{15} X_{15}^t$

$s_8 = X_{16} X_{16}^t$

$C_{11} = s_1 + s_2 + s_3 + s_4$

$C_{12} = m_2 - m_5 - m_7 + m_{11} + m_{12} + m_{13} + m_{19}$

$C_{13} = m_1 + m_3 + m_{12} + m_{15} + m_{16} + m_{17} + m_{21} - m_{24}$

$C_{14} = m_2 - m_3 - m_5 - m_7 - m_8 + m_{11} + m_{13} - m_{17} + m_{23} + m_{24}$

$C_{22} = m_1 + m_6 - m_7 + m_{10} + m_{11} + m_{12} + m_{22}$

$C_{23} = m_1 - m_4 + m_6 - m_7 - m_9 + m_{10} + m_{12} + m_{18} + m_{20} + m_{21}$

$C_{24} = m_2 + m_4 + m_{11} + m_{14} + m_{16} - m_{18} - m_{20} + m_{23}$

$C_{33} = m_4 - m_6 + m_7 + m_9 - m_{17} - m_{18} + m_{26}$

$C_{34} = m_3 + m_5 + m_7 + m_8 + m_{17} + m_{18} + m_{25}$

$C_{44} = s_5 + s_6 + s_7 + s_8$

return C

range from gradient descent approaches Smirnov [2013] and heuristics Éric Drevet et al. [2011], to group-theoretic methods Ye and Lim [2018], graph-based random walks Kauers and Moosbauer [2022], and deep reinforcement learning Fawzi et al. [2022].

Despite this progress, much less attention has been paid to matrix products with **additional structure**, such as $B = A$ or $B = A^t$, or products involving sparsity or symmetry Dumas et al. [2020, 2023], Arrigoni et al. [2021]. This is surprising given that expressions like AA^t are widely used in fields such as statistics, data analysis, deep learning, and wireless communications. For example, AA^t often represents a covariance or Gram matrix, while in linear regression, the solution for the data pair (X, y) involves the data covariance matrix $X^t X$:

$$\beta = (X^t X)^{-1} X^t y.$$

From a theoretical standpoint, computing XX^t has the same asymptotic complexity as general matrix multiplication. As a result, only constant-factor speedups are possible. The RXTX algorithm, presented in Algorithm 1, achieves such a speedup by exploiting structure specific to XX^t.

1.1 Previous works

Prior work by Ye and Lim [2016, 2018] used representation theory and the Cohn–Umans framework to derive new multiplication schemes for structured matrix products. Reinforcement learning methods have also been applied to this domain. For instance, Fawzi et al. [2022] used deep RL to compute tensor ranks and discover novel multiplication algorithms. Neural Networks wtih proper training setup can rediscover Strassen and Laderman algorithms for small matrices Elser [2016].

More recently, Dumas et al. [2020, 2023] proposed optimized schemes for computing XX^T over finite fields and complex numbers. To the best of our knowledge, the current state-of-the-art approach for real-valued XX^T is due to Arrigoni et al. [2021], which recursively applies Strassen's algorithm to 2×2 block matrices, reducing the problem to general matrix multiplication. In contrast, our approach uses the structure of XX^t in a novel way.

2 Analysis of RXTX

We define

- $R(n)$ - number of multiplications performed by RXTX for $n \times n$ matrix
- $S(n)$ - number of multiplications performed by recursive Strassen Arrigoni et al. [2021] for $n \times n$ matrix
- $M(n)$ - number of multiplications performed by Strassen-Winograd algorithm for general product of $n \times n$ matrices
- $R_+(n)$ - number of additions and multiplications performed by RXTX for $n \times n$ matrix
- $S_+(n)$ - number of additions and multiplications performed by recursive Strassen Arrigoni et al. [2021] for $n \times n$ matrix
- $M_+(n)$ - number of additions and multiplications performed by Strassen-Winograd algorithm for general product of $n \times n$ matrices

The superscript ^{opt} indicates an optimal cutoff: for sufficiently small matrices, standard matrix multiplication is used instead of further recursive calls.

2.1 Number of multiplications

Theorem 1. The number of multiplications for RXTX:

$$R(n) = \frac{26}{41}M(n) + \frac{15}{41}n^{3/2} = \frac{26}{41}n^{\log_2 7} + \frac{15}{41}n^{3/2}.$$

The number of multiplications for recursive Strassen:

$$S(n) = \frac{2}{3}M(n) + \frac{1}{3}n^2 = \frac{2}{3}n^{\log_2 7} + \frac{1}{3}n^2.$$

Proof. The definition of RXTX involves 8 recursive calls and 26 general matrix multiplications. It follows that

$$R(n) = 8R(n/4) + 26M(n/4).$$

The general solutions to this recursive equation has a form Cormen et al. [2009]

$$R(n) = \alpha M(n) + \beta n^{3/2}.$$

Plugging $n = 1$ and $n = 4$ we get

$$1 = \alpha + \beta,$$
$$34 = 49\alpha + 8\beta.$$

Solving this system we obtain

$$\alpha = \frac{26}{41} \approx 0.6341, \qquad \beta = \frac{15}{41} \approx 0.3658.$$

Similarly, recursive Strassen for XX^t uses 4 recursive calls and 2 general matrix multiplications:

$$S(n) = 4S(n/2) + 2M(n/2).$$

General solution form

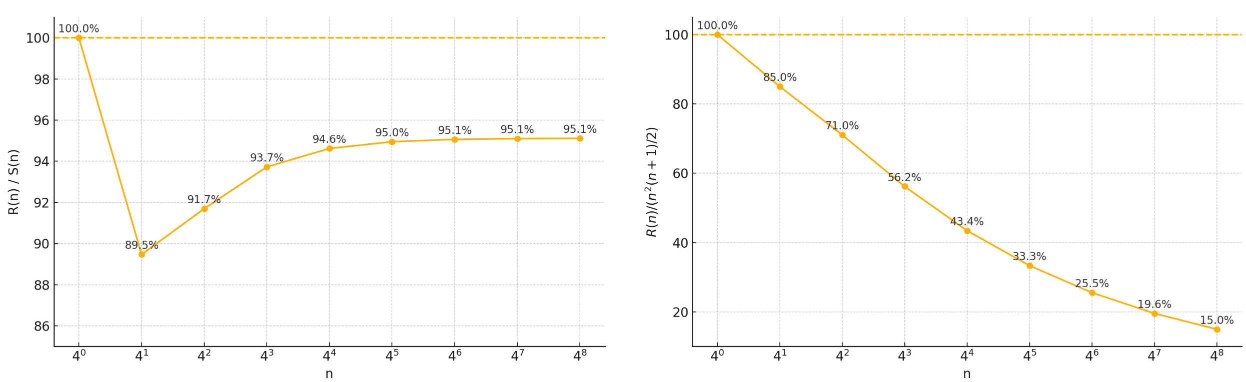
$$S(n) = \gamma M(n) + \delta n^2.$$

Plugging $n = 1$ and $n = 2$ we get

$$1 = \gamma + \delta, \qquad 6 = 7\gamma + 4\delta.$$

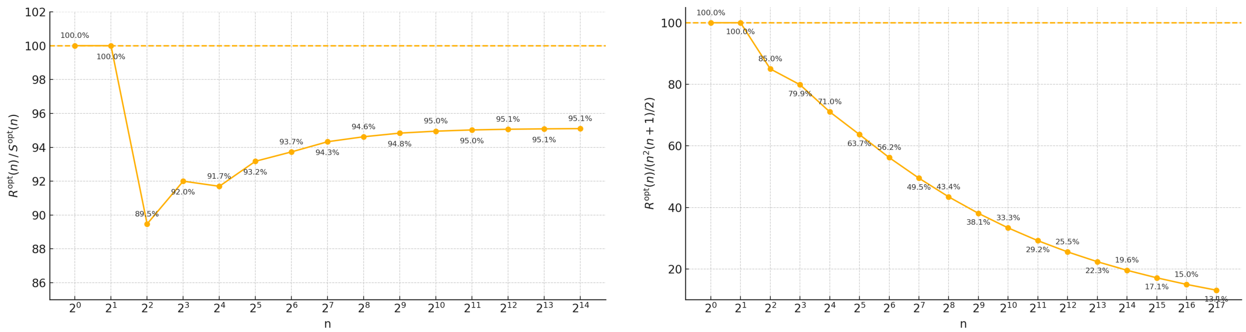
Solving this system we obtain $\gamma = 2/3 \approx 0.6666$ and $\delta = 1/3 \approx 0.3333$. □

In Figure 1 we can see the ratio $R(n)/S(n)$ for n given by powers of 4. The ratio always stays below 100% and approaches the asymptotic 95%, which indicates a 5% reduction in the number of multiplications. Same happens in Figure 2, where we use optimal cutoff i.e. for small enough matrix sizes we use standard matrix multiplication instead of further recursive calls.



Ratio of $R(n)$ to $S(n)$. Ratio of $R(n)$ to naive algorithm $n^2(n+1)/2$.

Figure 1: Comparison of number of multiplications of RXTX to previous SotA and naive algorithm.



Ratio of $R^{opt}(n)$ to $S^{opt}(n)$.

Ratio of $R^{opt}(n)$ to $n^2(n+1)/2$.

Figure 2: Comparison of number of multiplications of RXTX with optimal cutoff to previous SotA and naive algorithm.

2.2 Total number of operations

Theorem 2. Total number of additions and multiplications for RXTX:

$$R_+(n) = \frac{156}{41} n^{\log_2 7} - \frac{615}{164} n^2 + \frac{155}{164} n^{3/2}.$$

Total number of additions and multiplications for recursive Strassen:

$$S_+(n) = 4n^{\log_2 7} - \frac{7}{4} n^2 \log_2 n - 3n^2.$$

Proof. The definition of RXTX involves 139 additions of $(n/4) \times (n/4)$ matrices. There exist methods in the literature Mårtensson and Wagner [2024] to reduce this number by utilizing common sub-expressions that appear in the algorithm 1 e.g. $X_6 + X_7$. For example, while Strassen algorithm uses 18 additions, its Winograd variant uses only 15 additions. We designed a custom search that allowed us to reduce the number of additions in RXTX from 139 to 100. We provide the resulting addition scheme in Algorithm 2 and Algorithm 3. Assuming 100 additions, we get the recursion

$$R_+(n) = 8R_+(n/4) + 26M_+(n/4) + 100(n/4)^2.$$

General solution has a form

$$R_+(n) = \frac{26}{41} M_+(n) + \alpha n^2 + \beta n^{3/2}.$$

Plugging the value $n = 1$ and $n = 4$ gives

$$\frac{26}{41} + \alpha + \beta = 1.$$

$$\frac{26}{41} \cdot 214 + 16\alpha + 8\beta = 134$$

We conclude that

$$\alpha = -\frac{95}{164} \approx -0.5793, \qquad \beta = \frac{155}{164} \approx 0.9451.$$

Similarly, definition of recursive Strassen gives

$$S_+(n) = 4S_+(n/2) + 2M_+(n) + 3(n/2)^2.$$

Which has a solution of the form

$$S_+(n) = \frac{2}{3} M_+(n) + \gamma n^2 \log_2 n + \delta n^2.$$

Plugging values $n = 1$ and $n = 2$ gives $\gamma = -7/4$ and $\delta = 1/3$. It is known Cenk and Hasan [2017] that

$$M_+(n) = 6n^{\log_2 7} - 5n^2.$$

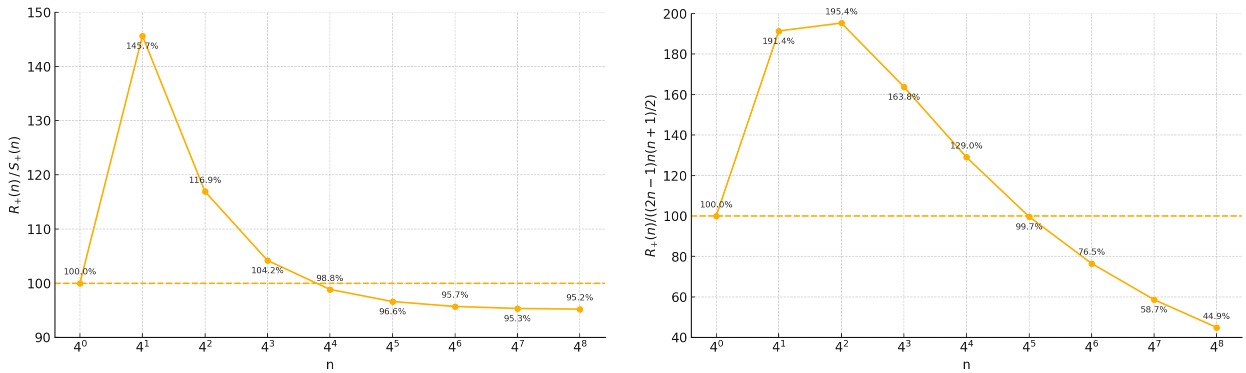
It follows that

$$R_+(n) = \frac{26}{41}(6n^{\log_2 7} - 5n^2) - \frac{95}{164}n^2 + \frac{155}{164}n^{3/2} = \frac{156}{41}n^{\log_2 7} - \frac{615}{164}n^2 + \frac{155}{164}n^{3/2}$$

and

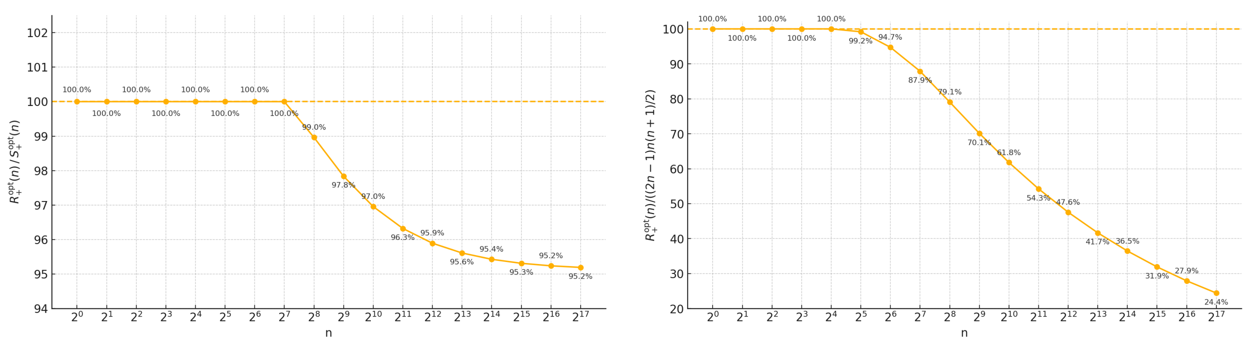
$$S_+(n) = \frac{2}{3}(6n^{\log_2 7} - 5n^2) - \frac{7}{4}n^2 \log_2 n + \frac{1}{3}n^2 = 4n^{\log_2 7} - \frac{7}{4}n^2 \log_2 n - 3n^2.$$

□



Ratio of $R_+(n)$ to $S_+(n)$. Ratio of $R_+(n)$ to $(2n-1)n(n+1)/2$.

Figure 3: Comparison of number of operations of RXTX to recursive Strassen and naive algorithm. RXTX outperforms recursive Strassen for $n \geq 256$ and naive algorithm for $n \geq 1024$.



Ratio of $R_+^{opt}(n)$ to $S_+^{opt}(n)$. Ratio of $R_+^{opt}(n)$ to $(2n-1)n(n+1)/2$.

Figure 4: Comparison of algorithms with optimal cutoffs i.e. for small enough matrices in recursion switch to the algorithm with least operations. RXTX outperforms naive algorithm for $n \geq 32$ and SotA for $n \geq 256$.

Algorithm 2 First stage of optimized addition scheme. The number of additions is reduced from 77 to 53.

Input: $X_1, X_2, ..., X_{16}$
Output: Left elements $L_1, ..., L_{26}$ and right elements $R_1, ..., R_{26}$ of multiplications $m_1, ...m_{26}$
$y_1 \leftarrow X_{13} - X_{14}$
$y_2 \leftarrow X_{12} - X_{10}$
$w_1 \leftarrow X_2 + X_4 - X_8$
$w_2 \leftarrow X_1 - X_5 - X_6$
$w_3 \leftarrow X_6 + X_7$
$w_4 \leftarrow X_{14} + X_{15}$
$w_5 \leftarrow y_2 + X_{16}$
$w_6 \leftarrow X_{10} + X_{11}$
$w_7 \leftarrow X_9 + y_1$
$w_8 \leftarrow X_9 - X_8$
$w_9 \leftarrow X_7 - X_{11}$
$w_{10} \leftarrow X_6 - X_7$
$w_{11} \leftarrow X_2 - X_3$

$L_1 \leftarrow -w_1 + X_3$	$R_1 \leftarrow X_8 + X_{11}$
$L_2 \leftarrow w_2 + X_7$	$R_2 \leftarrow X_{15} + X_5$
$L_3 \leftarrow -X_2 + X_{12}$	$R_3 \leftarrow w_5$
$L_4 \leftarrow X_9 - X_6$	$R_4 \leftarrow w_7$
$L_5 \leftarrow X_2 + X_{11}$	$R_5 \leftarrow X_{15} - w_3$
$L_6 \leftarrow X_6 + X_{11}$	$R_6 \leftarrow w_3 - X_{11}$
$L_7 \leftarrow X_{11}$	$R_7 \leftarrow w_3$
$L_8 \leftarrow X_2$	$R_8 \leftarrow w_3 - w_4 + w_5$
$L_9 \leftarrow X_6$	$R_9 \leftarrow w_7 - w_6 + w_3$
$L_{10} \leftarrow w_1 - X_3 + X_7 + X_{11}$	$R_{10} \leftarrow X_{11}$
$L_{11} \leftarrow X_5 + w_{10}$	$R_{11} \leftarrow X_5$
$L_{12} \leftarrow w_{11} + X_4$	$R_{12} \leftarrow X_8$
$L_{13} \leftarrow -w_2 + X_3 - w_9$	$R_{13} \leftarrow X_{15}$
$L_{14} \leftarrow -w_2$	$R_{14} \leftarrow w_7 + w_4$
$L_{15} \leftarrow w_1$	$R_{15} \leftarrow w_6 + w_5$
$L_{16} \leftarrow X_1 - X_8$	$R_{16} \leftarrow X_9 - X_{16}$
$L_{17} \leftarrow X_{12}$	$R_{17} \leftarrow -y_2$
$L_{18} \leftarrow X_9$	$R_{18} \leftarrow y_1$
$L_{19} \leftarrow -w_{11}$	$R_{19} \leftarrow -X_{15} + X_7 + X_8$
$L_{20} \leftarrow X_5 + w_8$	$R_{20} \leftarrow X_9$
$L_{21} \leftarrow X_8$	$R_{21} \leftarrow X_{12} + w_8$
$L_{22} \leftarrow -w_{10}$	$R_{22} \leftarrow X_5 + w_9$
$L_{23} \leftarrow X_1$	$R_{23} \leftarrow X_{13} - X_5 + X_{16}$
$L_{24} \leftarrow -X_1 + X_4 + X_{12}$	$R_{24} \leftarrow X_{16}$
$L_{25} \leftarrow X_9 + X_2 + X_{10}$	$R_{25} \leftarrow X_{14}$
$L_{26} \leftarrow X_6 + X_{10} + X_{12}$	$R_{26} \leftarrow X_{10}$

Algorithm 3 Second stage of optimized addition scheme. The number of additions is reduced from 62 to 47.

Input: $m_1, m_2, ..., m_{26}$ and $s_1, ...s_8$.
Output: Entries C_{ij} using 47 additions.
$z_1 \leftarrow m_7 - m_{11} - m_{12}$
$z_2 \leftarrow m_1 + m_{12} + m_{21}$
$z_3 \leftarrow m_3 + m_{17} - m_{24}$
$z_4 \leftarrow m_2 + m_{11} + m_{23}$
$z_5 \leftarrow m_5 + m_7 + m_8$
$z_6 \leftarrow m_4 - m_{18} - m_{20}$
$z_7 \leftarrow m_6 - m_7 - m_9$
$z_8 \leftarrow m_{17} + m_{18}$
$C_{11} \leftarrow s_1 + s_2 + s_3 + s_4$
$C_{12} \leftarrow m_2 - m_5 - z_1 + m_{13} + m_{19}$
$C_{13} \leftarrow z_2 + z_3 + m_{15} + m_{16}$
$C_{14} \leftarrow z_4 - z_3 - z_5 + m_{13}$
$C_{22} \leftarrow m_1 + m_6 - z_1 + m_{10} + m_{22}$
$C_{23} \leftarrow z_2 - z_6 + z_7 + m_{10}$
$C_{24} \leftarrow z_4 + z_6 + m_{14} + m_{16}$
$C_{33} \leftarrow m_4 - z_7 - z_8 + m_{26}$
$C_{34} \leftarrow m_3 + z_5 + z_8 + m_{25}$
$C_{44} \leftarrow s_5 + s_6 + s_7 + s_8$

2.3 Runtime of RXTX

We verify that RXTX gives speed-up in practice for large sizes of matrix X. Figure 5 shows histogram of runtimes in the following setup:

- 6144×6144 dense matrix is sampled 1000 times with random normal $\mathcal{N}(0, 1)$ entries. Here $6144 = 3 \cdot 2^{12}$.
- RXTX is implemented as depth-1 recursion i.e. we directly use BLAS routines to compute 26 general matrix multiplications and 8 symmetric products of matrices of size 1536×1536.
- Default is direct call of BLAS-3 routine for XX^t.
- single thread CPU 10th Gen Intel Core i7-10510U Processor, 1.8 GHz 4 cores.

We didn't perform search for the smallest matrix size where RXTX outperforms other methods since runtime is highly sensitive to hardware, computation graph organization, and memory management. Figure 4 suggests that RXTX can be faster than recursive Strassen for $n \geq 256$.

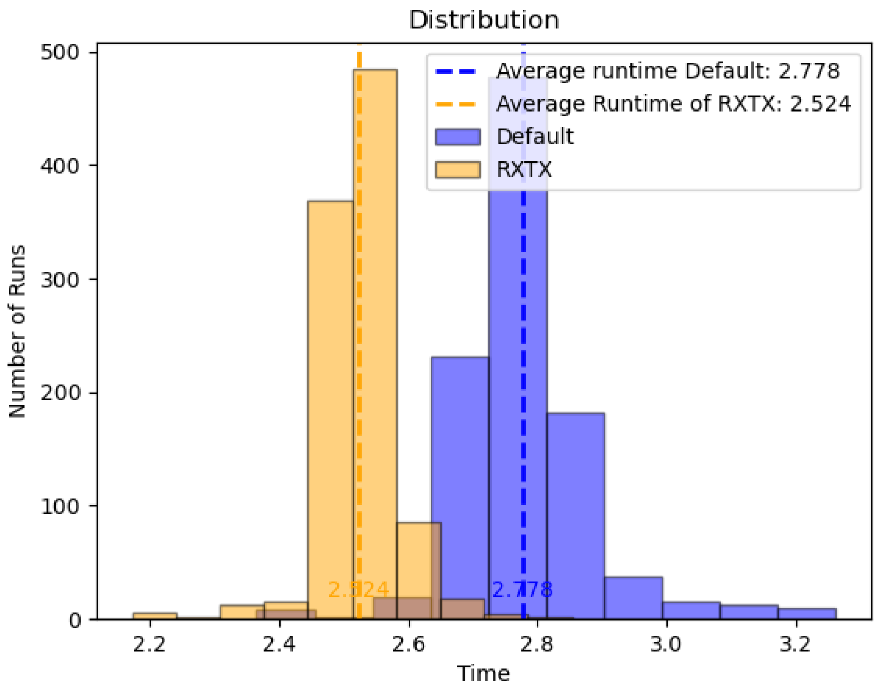


Figure 5: The average runtime for RXTX is 2.524s, which is 9% faster than average runtime of specific BLAS routine 2.778s. RXTX was faster in 99% of the runs.

3 Discovery Methodology

3.1 Description of RL-guided Large Neighborhood Search

In this section we briefly present our methodology. Full methodology with other discovered accelerations will be described in Rybin et al. [2025]. We combine RL-guided Large Neighborhood Search Wu et al. [2021], Addanki et al. [2020] with a two–level MILP pipeline:

1. The RL agent proposes a (potentially redundant) set of rank-1 bilinear products;
2. MILP-A exhaustively enumerates tens of thousands of linear relations between these candidate rank-1 bilinear products and target expressions;
3. MILP-B then selects the smallest subset of products whose induced relations cover every target expression of XX^t.

The loop iterates under a Large Neighborhood Search regime. One way to view this pipeline is a simplification of AlphaTensor RL approach Fawzi et al. [2022]: instead of sampling tensors from $\mathbb{R}^{n^2} \otimes \mathbb{R}^{n^2} \otimes \mathbb{R}^{n^2}$, we sample candidate tensors from $\mathbb{R}^{n^2} \otimes \mathbb{R}^{n^2}$ and let the MILP solver find optimal linear combinations of sampled candidates.

3.2 Example: matrix times transpose algorithm search for 2-by-2 matrix

Consider the example for 2×2 matrix X. We want to perform the computation of XX^t:

$$\begin{pmatrix} x_1 & x_2 \\ x_3 & x_4 \end{pmatrix} \cdot \begin{pmatrix} x_1 & x_3 \\ x_2 & x_4 \end{pmatrix} = \begin{pmatrix} x_1^2 + x_2^2 & x_1x_3 + x_2x_4 \\ x_1x_3 + x_2x_4 & x_3^2 + x_4^2 \end{pmatrix}$$

We identify 3 target expressions

$$T = \{x_1^2 + x_2^2, x_3^2 + x_4^2, x_1x_3 + x_2x_4\}.$$

We randomly sample thousands of products $p_1, ..., p_m$, each one given by

$$\left(\sum_{i=1}^{4} \alpha_i x_i \right) \cdot \left(\sum_{j=1}^{4} \beta_j x_j \right)$$

with $\alpha_i, \beta_j \in \{-1, 0, +1\}$ chosen by RL policy π_θ. MILP-A enumerates ways to write target expressions from T as linear combinations of sampled products $\sum \gamma_i p_i$. MILP-B selects minimal number of sampled products such that every target expression can be obtained as their linear combination. Key observation is that MILP-A and MILP-B are rapidly solvable with solvers like Gurobi Gurobi Optimization, LLC [2024].

4 Acknowledgements

The work of Z.-Q. Luo was supported by the Guangdong Major Project of Basic and Applied Basic Research (No.2023B0303000001), the Guangdong Provincial Key Laboratory of Big Data Computing, and the National Key Research and Development Project under grant 2022YFA1003900.

References

R. Addanki, V. Nair, and M. Alizadeh. Neural large neighborhood search. In *Learning Meets Combinatorial Algorithms @ NeurIPS 2020*, 2020.

V. Arrigoni, F. Maggioli, A. Massini, and E. Rodolà. Efficiently parallelizable strassen-based multiplication of a matrix by its transpose. In *Proceedings of the 50th International Conference on Parallel Processing*, ICPP '21, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450390682.

M. Cenk and M. A. Hasan. On the arithmetic complexity of strassen-like matrix multiplications. *Journal of Symbolic Computation*, 80:484–501, 2017. ISSN 0747-7171. doi: https://doi.org/10.1016/j.jsc.2016.07.004. URL https://www.sciencedirect.com/science/article/pii/S0747717116300359.

T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009. ISBN 978-0-262-03384-8. URL http://mitpress.mit.edu/books/introduction-algorithms.

J.-G. Dumas, C. Pernet, and A. Sedoglavic. On fast multiplication of a matrix by its transpose. In *Proceedings of the 45th International Symposium on Symbolic and Algebraic Computation*, ISSAC '20, page 162–169, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450371001.

J.-G. Dumas, C. Pernet, and A. Sedoglavic. Some fast algorithms multiplying a matrix by its adjoint. *Journal of Symbolic Computation*, 115:285–315, 2023.

V. Elser. A network that learns strassen multiplication. *Journal of Machine Learning Research*, 17:1–13, 2016.

A. Fawzi, M. Balog, A. Huang, T. Hubert, B. Romera-Paredes, M. Barekatain, A. Novikov, F. J. R Ruiz, J. Schrittwieser, G. Swirszcz, et al. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 610(7930):47–53, 2022.

Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2024. URL https://www.gurobi.com.

M. Kauers and J. Moosbauer. The fbhhrbnrssshk-algorithm for multiplication in $\mathbb{Z}_2^{5\times 5}$ is still not the end of the story. *ArXiv*, abs/2210.04045, 2022. URL https://api.semanticscholar.org/CorpusID:252780122.

E. Mårtensson and P. S. Wagner. The number of the beast: Reducing additions in fast matrix multiplication algorithms for dimensions up to 666. Cryptology ePrint Archive, Paper 2024/2063, 2024. URL https://eprint.iacr.org/2024/2063.

D. Rybin, Y. Zhang, and Z.-Q. Luo. Accelerating structured matrix computations with machine learning based search. in progress, 2025.

A. V. Smirnov. The bilinear complexity and practical algorithms for matrix multiplication. *Computational Mathematics and Mathematical Physics*, 53(12):1781–1795, Dec. 2013. ISSN 1555-6662. doi: 10.1134/s0965542513120129. URL http://dx.doi.org/10.1134/S0965542513120129.

V. Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13(4):354–356, Aug. 1969. ISSN 0945-3245. doi: 10.1007/bf02165411. URL http://dx.doi.org/10.1007/BF02165411.

S. Winograd. A new algorithm for inner product. *IEEE Transactions on Computers*, 100(7):693–694, 1968.

Y. Wu, W. Song, Z. Cao, and J. Zhang. Learning large neighborhood search policy for integer programming. *arXiv preprint arXiv:2111.03466*, 2021.

K. Ye and L.-H. Lim. Algorithms for structured matrix-vector product of optimal bilinear complexity. In *2016 IEEE Information Theory Workshop (ITW)*, pages 310–314. IEEE, 2016.

K. Ye and L.-H. Lim. Fast structured matrix computations: tensor rank and cohn–umans method. *Foundations of Computational Mathematics*, 18:45–95, 2018.

C. Éric Drevet, M. Nazrul Islam, and Éric Schost. Optimization techniques for small matrix multiplication. *Theoretical Computer Science*, 412(22):2219–2236, 2011. ISSN 0304-3975. doi: https://doi.org/10.1016/j.tcs.2010.12.012. URL https://www.sciencedirect.com/science/article/pii/S0304397510007036.