AlphaTensor 详解

Discovering faster matrix multiplication algorithms with reinforcement learning

小组：李致远、冯文喆、魏睿、宋泽颀、周启民

同济大学物理科学与工程学院

2025 年 12 月 5 日

1 背景介绍

2 核心原理

3 实验结果

4 总结展望

1. 背景介绍
2. 核心原理
3. 实验结果
4. 总结展望

AlphaTensor 一句话介绍

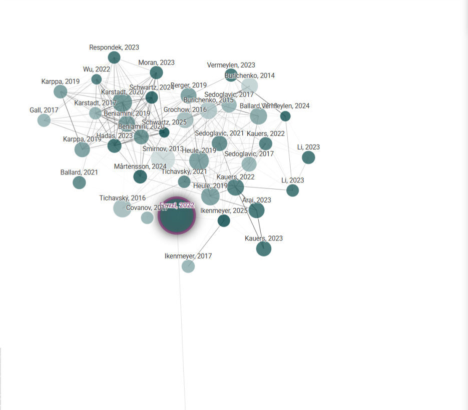
核心理念

“Discovering faster matrix multiplication algorithms with reinforcement learning”利用强化学习对矩阵乘法流程进行自动设计。它的强化学习的思想与基本架构，来自于 alphazero。

- **问题描述**：在一个三维矩阵 $T_n = \sum_{r=1}^{R} u^{(r)} \otimes v^{(r)} \otimes w^{(r)}$ 与一个矩阵乘法之间构建映射。
- **解法思路**：用强化学习，找到这个矩阵 T_n 的最小的低秩分解，因为表征张量的秩等于算法所需的乘法次数。
- **解法细节**：使用 transformer 对于矩阵 T_n 提取特征后，policy 头与 value 头计算下一步的潜在概率与得分。

为什么选择这篇文章?

- **顶尖团队**：本文由 Google DeepMind 团队创作
  - DeepMind 是人工智能领域的领军机构（AlphaGo、AlphaFold 等），DeepMind 团队联合创始人获得 2024 诺贝尔化学奖。
- **学术影响力**：论文引用次数已超过 **500+**



Discovering faster matrix multiplication algorithms with reinforcement learning

Alhussein Fawzi + 11 authors Pushmeet Kohli

2022, Nature

563 Citations Save

Open in:

Improving the efficiency of algorithms for fundamental computations can have a widespread impact, as it can affect the overall speed of a large amount of computations. Matrix multiplication is one such primitive task, occurring in many systems—from neural networks to scientific computing routines. The automatic discovery of algorithms using machine learning offers the prospect of reaching beyond human intuition and outperforming the current best human-designed algorithms. However, automating the algorithm discovery procedure is intricate, as the space of possible algorithms is enormous. Here we report a deep reinforcement learning approach based on AlphaZero1 for discovering efficient and provably correct algorithms for the multiplication of arbitrary matrices. Our agent, AlphaTensor, is trained to play a single-player game where the objective is finding tensor decompositions within a finite factor space. AlphaTensor discovered algorithms that outperform

为什么关注矩阵乘法?

- **核心地位**：矩阵乘法是深度学习（全连接层、Conv 层、transformer 多头注意力）和科学计算的基石。
- **优化维度**：
  - *System* 角度：向量化、访存优化（Cache 命中）、并行计算。
  - *Math* 角度：减少数值乘法的计算次数（乘法开销 $\gg$ 加法开销）。

Strassen 算法的启示 (1969)

对于 2×2 矩阵乘法：

- **朴素算法**：需要 **8 次**乘法 ($O(N^3)$)。
- **Strassen 算法**：只需要 **7 次**乘法 ($O(N^{2.807})$)。

这意味着通过改变算法流程，可以在数学层面实现加速。

Strassen 算法的启示

- **硬件层面的动机**：
  - 远古时期，乘法器就是由加法器堆叠起来的
  - 执行乘法的时间是加法的约 **3 倍**
  - 减少乘法次数 = 实际性能提升
- **Strassen 的贡献 (1969)**：
  - 将 2×2 矩阵乘法从 **8 次**降至 **7 次**
  - 大矩阵可以递归拆分为 2×2 子矩阵
  - 复杂度：$O(N^3) \rightarrow O(N^{2.807})$

$$\begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{bmatrix} \times \begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix} = \begin{bmatrix} c_{1,1} & c_{1,2} \\ c_{2,1} & c_{2,2} \end{bmatrix}$$

Standard algorithm

$h_1 = a_{1,1}\, b_{1,1}$

$h_2 = a_{1,1}\, b_{1,2}$

$h_3 = a_{1,2}\, b_{2,1}$

$h_4 = a_{1,2}\, b_{2,2}$

$h_5 = a_{2,1}\, b_{1,1}$

$h_6 = a_{2,1}\, b_{1,2}$

$h_7 = a_{2,2}\, b_{2,1}$

$h_8 = a_{2,2}\, b_{2,2}$

$c_{1,1} = h_1 + h_3$

$c_{1,2} = h_2 + h_4$

$c_{2,1} = h_5 + h_7$

$c_{2,2} = h_6 + h_8$

Strassen's algorithm

$h_1 = (a_{1,1} + a_{2,2})\,(b_{1,1} + b_{2,2})$

$h_2 = (a_{2,1} + a_{2,2})\, b_{1,1}$

$h_3 = a_{1,1}\,(b_{1,2} - b_{2,2})$

$h_4 = a_{2,2}\,(-b_{1,1} + b_{2,1})$

$h_5 = (a_{1,1} + a_{1,2})\, b_{2,2}$

$h_6 = (-a_{1,1} + a_{2,1})\,(b_{1,1} + b_{1,2})$

$h_7 = (a_{1,2} - a_{2,2})(b_{2,1} + b_{2,2})$

$c_{1,1} = h_1 + h_4 - h_5 + h_7$

$c_{1,2} = h_3 + h_5$

$c_{2,1} = h_2 + h_4$

$c_{2,2} = h_1 - h_2 + h_3 + h_6$

AlphaTensor 一句话介绍

核心理念

“Discovering faster matrix multiplication algorithms with reinforcement learning” 利用强化学习对矩阵乘法流程进行自动设计。它的强化学习的思想与基本架构，来自于 alphazero。

- **问题描述**：在一个三维矩阵 $T_n = \sum_{r=1}^{R} u^{(r)} \otimes v^{(r)} \otimes w^{(r)}$ 与一个矩阵乘法之间构建映射。
- **解法思路**：用强化学习，找到这个矩阵 T_n 的最小的低秩分解，因为表征张量的秩等于算法所需的乘法次数。
- **解法细节**：使用 transformer 对于矩阵 T_n 提取特征后，policy 头与 value 头计算下一步的潜在概率与得分。

1 背景介绍

2 核心原理

3 实验结果

4 总结展望

问题描述：搜索空间的构建

AlphaTensor 将寻找算法转化为一个 **3D 张量分解**游戏。

对应关系 ①：矩阵乘定义 $\leftrightarrow$ 表征张量 T_n

- 一种尺寸的矩阵乘法定义（如 2×2）唯一对应一个三维张量 T_n。
- 张量中的元素为 0 或 1，代表结果矩阵中位置的值由哪些输入元素相乘得到。

对应关系 ②：低秩分解 $\leftrightarrow$ 算法流程

- 表征张量的 **秩 (Rank)** $=$ 算法所需的 **乘法次数**。
- 将 T_n 分解为 R 个秩-1 张量的和:

$$T_n = \sum_{r=1}^{R} u^{(r)} \otimes v^{(r)} \otimes w^{(r)}$$

问题描述：直观解释

强化学习建模 (RL Formulation)

- **游戏目标**：用尽可能少的步数（秩-1 张量）将初始张量 T_n 减为零张量。
- **状态 (State)**：当前剩余的张量 S_t（初始为 T_n）。
- **动作 (Action)**：选择三个向量 u, v, w 构成一个秩-1 张量。
  - 离散化：系数限制在 $\{-2, -1, 0, 1, 2\}$ 中，剪枝搜索空间。
- **状态更新**：$S_{t+1} \leftarrow S_t - u \otimes v \otimes w$

奖励函数 (Reward)

- 每走一步，Reward $=$ -1（鼓励更少步数）。
- 若步骤超限未归零，施加额外惩罚（与剩余张量的秩相关）。
- 可选：加入硬件运行时延时作为负奖励（针对特定硬件优化）。

网络架构与搜索算法

AlphaZero 风格的 RL + MCTS

- **MCTS (蒙特卡洛树搜索)**：用于规划下一步动作，平衡探索 (Explore) 与利用 (Exploit)。
- **Policy Network**：基于 Transformer 架构。
    - 输入：当前张量状态。
    - 输出：建议的动作分布。
    - 包含 Cross-attention，Causal self-attention 等机制。
- **Value Network**：预测当前状态归零所需的最小步数。

RL Agent

↓

MCTS Search

↓

Decomposition

流程示意图

1. 背景介绍
2. 核心原理
3. **实验结果**
4. 总结展望

理论突破：发现更优的秩

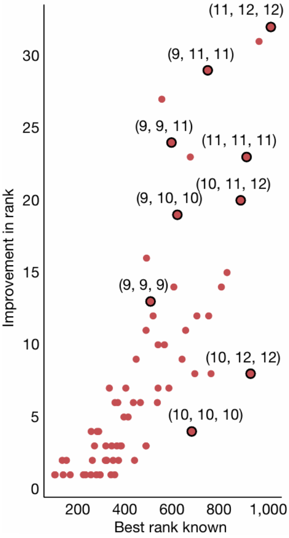
表 1: AlphaTensor 在不同矩阵尺寸下的发现

矩阵尺寸 (n, m, k)	现有最佳秩 (Human)	AlphaTensor
$4, 4, 4$	49 (Strassen^2)	**47**
$5, 5, 5$	98	**96**
$4, 5, 5$	80	**76**

- AlphaTensor 在多种尺寸下发现了比人类已知算法更少的乘法次数。
- 对于较大矩阵，其优势呈递增趋势。
- **注**：3×3 的全局最优解仍是数学界的未解之谜，AlphaTensor 也未完全攻克。

理论突破：发现更优的秩

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) ⊗ (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) ⊗ (2, 1, 1)	80	76	76



理论突破：发现更优的秩

Article

$$\begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} \end{pmatrix} \begin{pmatrix} b_{1,1} & b_{1,2} & b_{1,3} & b_{1,4} \\ b_{2,1} & b_{2,2} & b_{2,3} & b_{2,4} \\ b_{3,1} & b_{3,2} & b_{3,3} & b_{3,4} \\ b_{4,1} & b_{4,2} & b_{4,3} & b_{4,4} \end{pmatrix} = \begin{pmatrix} c_{1,1} & c_{1,2} & c_{1,3} & c_{1,4} \\ c_{2,1} & c_{2,2} & c_{2,3} & c_{2,4} \\ c_{3,1} & c_{3,2} & c_{3,3} & c_{3,4} \\ c_{4,1} & c_{4,2} & c_{4,3} & c_{4,4} \end{pmatrix}$$

$$\begin{aligned}
h_1 &= a_{1,1}b_{1,3} \\
h_2 &= (a_{1,1} + a_{3,1} + a_{3,3})(b_{1,1} + b_{3,1} + b_{3,3}) \\
h_3 &= (a_{1,1} + a_{3,1} + a_{3,4})(b_{1,2} + b_{4,2} + b_{4,3}) \\
h_4 &= (a_{1,3} + a_{2,1} + a_{2,3})(b_{1,3} + b_{1,4} + b_{3,4}) \\
h_5 &= (a_{1,1} + a_{3,1})(b_{1,1} + b_{1,2} + b_{1,3} + b_{3,1} + b_{3,3} + b_{4,2} + b_{4,3}) \\
h_6 &= (a_{1,3} + a_{2,3})(b_{1,3} + b_{1,4} + b_{3,2} + b_{3,3} + b_{3,4} + b_{4,2} + b_{4,3}) \\
h_7 &= (a_{1,4} + a_{4,3} + a_{4,4})(b_{3,1} + b_{3,3} + b_{4,1}) \\
h_8 &= (a_{1,4} + a_{4,1} + a_{4,4})(b_{1,3} + b_{1,4} + b_{4,4}) \\
h_9 &= (a_{1,3} + a_{2,3} + a_{2,4})(b_{3,2} + b_{4,2} + b_{4,3}) \\
h_{10} &= (a_{1,4} + a_{4,4})(b_{1,3} + b_{1,4} + b_{3,1} + b_{3,3} + b_{4,1} + b_{4,3} + b_{4,4}) \\
h_{11} &= a_{3,3}(b_{1,1} + b_{2,2} + b_{2,3} + b_{3,1} + b_{3,2}) \\
h_{12} &= (a_{1,2} + a_{3,2} + a_{3,3})(b_{2,2} + b_{2,3} + b_{3,2}) \\
h_{13} &= a_{3,4}(b_{1,2} + b_{2,1} + b_{2,3} + b_{4,1} + b_{4,2}) \\
h_{14} &= (a_{1,2} + a_{3,2})(b_{2,1} + b_{2,2} + b_{2,3} + b_{3,2} + b_{4,1}) \\
h_{15} &= (a_{1,2} + a_{3,2} + a_{3,4})(b_{2,1} + b_{2,3} + b_{4,1}) \\
h_{16} &= a_{2,1}(b_{1,2} + b_{1,4} + b_{2,2} + b_{2,3} + b_{3,4}) \\
h_{17} &= (a_{1,2} + a_{2,1} + a_{2,2})(b_{1,2} + b_{2,2} + b_{2,3}) \\
h_{18} &= (a_{1,2} + a_{2,2})(b_{1,2} + b_{2,2} + b_{2,3} + b_{2,4} + b_{4,4}) \\
h_{19} &= a_{2,4}(b_{2,3} + b_{2,4} + b_{3,2} + b_{4,2} + b_{4,4}) \\
h_{20} &= (a_{1,2} + a_{2,3} + a_{2,4} + a_{3,2} + a_{3,3})b_{3,2} \\
h_{21} &= (a_{1,2} + a_{2,2} + a_{2,4})(b_{2,3} + b_{2,4} + b_{4,4}) \\
h_{22} &= a_{4,3}(b_{2,3} + b_{2,4} + b_{3,1} + b_{3,4} + b_{4,1}) \\
h_{23} &= (a_{1,1} + a_{1,3} + a_{1,4} + a_{2,3} + a_{2,4} + a_{3,1} + a_{3,4})(b_{4,2} + b_{4,3}) \\
h_{24} &= (a_{1,2} + a_{4,2} + a_{4,3})(b_{2,3} + b_{2,4} + b_{3,4}) \\
h_{25} &= (a_{1,2} + a_{4,2})(b_{1,1} + b_{2,1} + b_{2,3} + b_{2,4} + b_{3,4}) \\
h_{26} &= (a_{1,2} + a_{4,1} + a_{4,2})(b_{1,1} + b_{2,1} + b_{2,3}) \\
h_{27} &= a_{1,4}b_{4,3} \\
h_{28} &= (a_{1,2} + a_{2,1} + a_{2,2} + a_{3,1} + a_{3,4})b_{1,2} \\
h_{29} &= (a_{1,2} + a_{2,1} + a_{2,3} + a_{4,2} + a_{4,3})b_{3,4} \\
h_{30} &= (a_{1,2} + a_{3,1} + a_{3,3} + a_{4,1} + a_{4,2})b_{1,1} \\
h_{31} &= a_{4,1}(b_{1,1} + b_{1,4} + b_{2,1} + b_{2,3} + b_{4,4}) \\
h_{32} &= (a_{1,2} + a_{3,2} + a_{3,4} + a_{4,3} + a_{4,4})b_{4,1} \\
h_{33} &= (a_{1,2} + a_{2,2} + a_{2,4} + a_{4,1} + a_{4,4})b_{4,4} \\
h_{34} &= (a_{2,1} + a_{3,1} + a_{4,1})(b_{1,1} + b_{1,2} + b_{1,4}) \\
h_{35} &= (a_{1,2} + a_{2,1} + a_{2,2} + a_{3,2} + a_{3,3})(b_{2,2} + b_{2,3}) \\
h_{36} &= (a_{1,2} + a_{2,4} + a_{3,2} + a_{4,3})(b_{2,3} + b_{2,4} + b_{3,2} + b_{4,1}) \\
h_{37} &= (a_{1,2} + a_{2,1} + a_{3,3} + a_{4,2})(b_{1,1} + b_{2,2} + b_{2,3} + b_{3,4}) \\
h_{38} &= (a_{2,2} + a_{3,2} + a_{4,2})(b_{2,1} + b_{2,2} + b_{2,4}) \\
h_{39} &= a_{1,2}b_{2,3} \\
h_{40} &= a_{1,3}b_{3,3} \\
h_{41} &= (a_{1,1} + a_{1,3} + a_{1,4} + a_{2,1} + a_{2,3} + a_{4,1} + a_{4,4})(b_{1,3} + b_{1,4}) \\
h_{42} &= (a_{1,2} + a_{3,2} + a_{3,4} + a_{4,1} + a_{4,2})(b_{2,1} + b_{2,3}) \\
h_{43} &= (a_{2,4} + a_{3,4} + a_{4,4})(b_{4,1} + b_{4,2} + b_{4,4}) \\
h_{44} &= (a_{2,3} + a_{3,3} + a_{4,3})(b_{3,1} + b_{3,2} + b_{3,4}) \\
h_{45} &= (a_{1,1} + a_{1,3} + a_{1,4} + a_{3,1} + a_{3,3} + a_{4,3} + a_{4,4})(b_{3,1} + b_{3,3}) \\
h_{46} &= (a_{1,2} + a_{2,2} + a_{3,4} + a_{4,1})(b_{1,2} + b_{2,1} + b_{2,3} + b_{4,4}) \\
h_{47} &= (a_{1,2} + a_{2,2} + a_{2,4} + a_{4,2} + a_{4,3})(b_{2,3} + b_{2,4}) \\
c_{1,1} &= h_{15} + h_{26} + h_2 + h_{30} + h_{32} + h_{39} + h_{40} + h_{42} + h_{45} + h_7 \\
c_{2,1} &= h_{11} + h_{12} + h_{14} + h_{20} + h_{22} + h_{24} + h_{25} + h_{29} + h_{35} + h_{36} + h_{37} + h_{38} + h_{44} + h_{47} \\
c_{3,1} &= h_{11} + h_{12} + h_{14} + h_{15} + h_{26} + h_{30} + h_{39} + h_{42} \\
c_{4,1} &= h_{15} + h_{22} + h_{24} + h_{25} + h_{26} + h_{32} + h_{39} + h_{42} \\
c_{1,2} &= h_{12} + h_{17} + h_{20} + h_{23} + h_{27} + h_{28} + h_{35} + h_{39} + h_3 + h_9 \\
c_{2,2} &= h_{12} + h_{17} + h_{18} + h_{19} + h_{20} + h_{21} + h_{35} + h_{39} \\
c_{3,2} &= h_{12} + h_{13} + h_{14} + h_{15} + h_{17} + h_{28} + h_{35} + h_{39} \\
c_{4,2} &= h_{13} + h_{14} + h_{15} + h_{18} + h_{19} + h_{21} + h_{32} + h_{33} + h_{36} + h_{38} + h_{42} + h_{43} + h_{46} + h_{47} \\
c_{1,3} &= h_1 + h_{27} + h_{39} + h_{40} \\
c_{2,3} &= h_{16} + h_{17} + h_{18} + h_{19} + h_{21} + h_{39} + h_{40} + h_4 + h_6 + h_9 \\
c_{3,3} &= h_{11} + h_{12} + h_{13} + h_{14} + h_{15} + h_1 + h_2 + h_{39} + h_3 + h_5 \\
c_{4,3} &= h_{10} + h_{22} + h_{24} + h_{25} + h_{26} + h_{27} + h_{31} + h_{39} + h_7 + h_8 \\
c_{1,4} &= h_1 + h_{21} + h_{24} + h_{29} + h_{33} + h_{39} + h_{41} + h_{47} + h_4 + h_8 \\
c_{2,4} &= h_{16} + h_{17} + h_{18} + h_{21} + h_{24} + h_{29} + h_{39} + h_{47} \\
c_{3,4} &= h_{16} + h_{17} + h_{18} + h_{25} + h_{26} + h_{28} + h_{30} + h_{31} + h_{34} + h_{35} + h_{37} + h_{38} + h_{42} + h_{46} \\
c_{4,4} &= h_{21} + h_{24} + h_{25} + h_{26} + h_{31} + h_{33} + h_{39} + h_{47}
\end{aligned}$$

Extended Data Fig. 1 | Algorithm for multiplying 4 × 4 matrices in modular arithmetic ($\mathbb{Z}_2$) with 47 multiplications. This outperforms the two-level Strassen's algorithm, which involves $7^2 = 49$ multiplications.

实际应用：硬件加速

AlphaTensor 不仅理论上减少了计算次数，还能针对特定硬件（GPU V100, TPU v3）进行优化。

Runtime 优化

- 通过将实际运行时间 (Timeit) 加入 Reward。
- 在大尺寸矩阵 (> 8192) 上，AlphaTensor 发现的算法在 GPU/TPU 上均有显著加速。
- 超越了标准的 cuBLAS 库性能。

- 左图（脑补）：在矩阵尺寸增大时，加速比显著提升。
- 证明了 AI 发现的算法具有实际工程落地价值。

实际应用：硬件加速

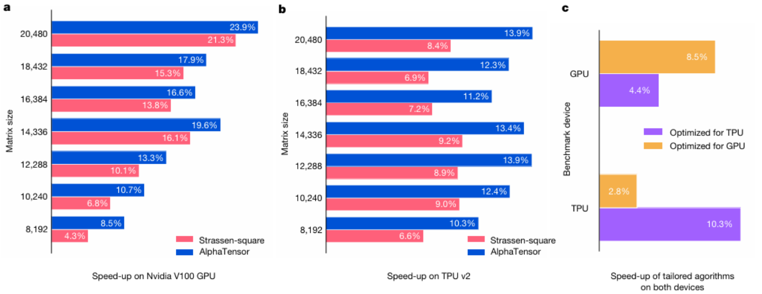


Fig. 5 | Speed-ups of the AlphaTensor-discovered algorithm. a,**b**, Speed-ups (%) of the AlphaTensor-discovered algorithms tailored for a GPU (**a**) and a TPU (**b**), optimized for a matrix multiplication of size 8,192 × 8,192. Speed-ups are measured relative to standard (for example, cuBLAS for the 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.

1 背景介绍

2 核心原理

3 实验结果

4 总结展望

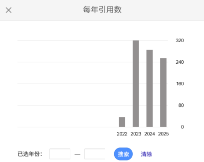
文章局限：工业实践角度

- CPU、GPU 运算时，**执行**只是"取址-译码-执行-访存-写回"流程中的一步
- 就算是执行步骤，现代硬件**加法器与乘法器性能差距已大大缩小**

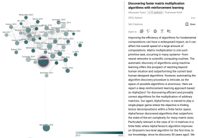
运算单元	**典型结构**	**关键路径延时**	**核心思想**
加法器	行波进位	$\approx N \times t_{FA}$	进位信号逐位串行传
	超前进位	$\approx k\log(N) \times t_{GATE}$	并行计算进位
乘法器	阵列式	$\approx (2N-1) \times t_{FA} + t_{CPA}$	累加路径长，线性增
	Wallace 树	$\approx O(\log N) \times t_{FA} + t_{CPA}$	树形结构，对数增长

文章局限：学术引用角度

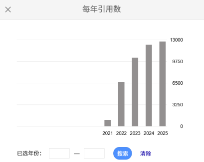
- AlphaTensor 之后，**再也没出过有影响力的优化矩阵计算的工作**
- 论文引用数**逐年减少**，学术热度下降



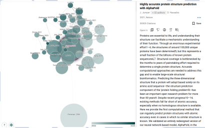
AlphaTensor 引用趋势



AlphaTensor 引用网络



AlphaFold 引用趋势



AlphaFold 引用网络

宏观定位与意义

- **AI for Science (Math/CS)**：
  - 继 AlphaGo (围棋)、AlphaFold (蛋白质) 之后，DeepMind 在基础数学/理论计算机领域的突破。
- **AutoAlgo (自动算法设计)**：
  - 与 AutoML（设计机器学习模型）对应，AlphaTensor 展示了 AI 自动设计/优化经典算法的潜力。
- **未来展望**：
  - 是否可以用强化学习优化其他经典算法？(如数值计算方法)。
  - 难点在于如何为其他算法（如 Dijkstra, DP）建立类似的搜索空间 (Formulation)。

参考资料 I

[FBH+22] Alhussein Fawzi, Matej Balog, Aja Huang, Thomas Hubert, and Bernardino Romera-Paredes et al.
Discovering faster matrix multiplication algorithms with reinforcement learning.
Nature, 2022.

[SHea18] David Silver, Thomas Hubert, and Julian Schrittwieser et al.
Mastering chess and shogi by self-play with a general reinforcement learning algorithm.
Science, 2018.

[Tia22] Keyu Tian.
Comprehensive analysis of deepmind alphatensor, 2022.

人员分工

李致远	选题，演讲
冯文喆	背景调研部分
魏睿	核心原理部分
宋泽顷	实验结果与总结展望部分
周启民	幻灯片

Thanks!