AlphaTensor 详解

Discovering faster matrix multiplication algorithms with reinforcement learning

小组：李致远、冯文喆、魏睿、宋泽顷、周启民

同济大学物理科学与工程学院

2025 年 12 月 10 日

AlphaTensor 概述

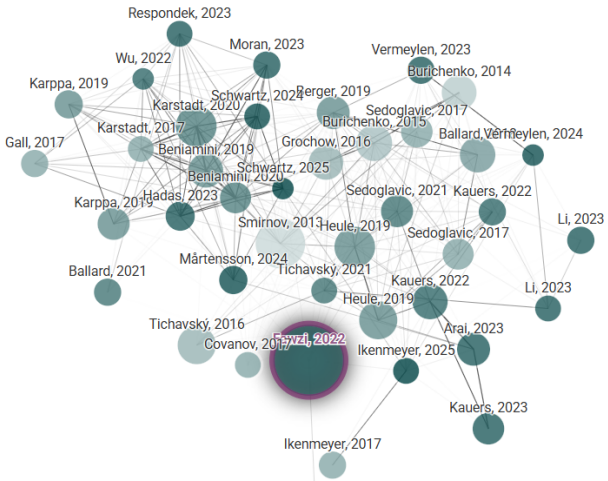
核心理念

“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，通过多次自我博弈，找到这个矩阵 T_n 的最小的低秩分解。
- **解法细节**：使用 transformer 对于矩阵 T_n 提取特征后，policy 头与 value 头计算低秩分解的下一步的潜在概率与得分。使用 MCTS 剪枝搜索，产出一个完整的低秩分解。

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**：论文发表在**Nature(2022)**，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

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 *(*本文研究方向*)*：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**：
  - 简单理解为：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 **3.5 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：
    $O(N^3) \to O(N^{2.807})$

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

Standard algorithm	Strassen's algorithm
$h_1 = a_{1,1}\ b_{1,1}$	$h_1 = (a_{1,1} + a_{2,2})\ (b_{1,1} + b_{2,2})$
$h_2 = a_{1,1}\ b_{1,2}$	$h_2 = (a_{2,1} + a_{2,2})\ b_{1,1}$
$h_3 = a_{1,2}\ b_{2,1}$	$h_3 = a_{1,1}\ (b_{1,2} - b_{2,2})$
$h_4 = a_{1,2}\ b_{2,2}$	$h_4 = a_{2,2}\ (-b_{1,1} + b_{2,1})$
$h_5 = a_{2,1}\ b_{1,1}$	$h_5 = (a_{1,1} + a_{1,2})\ b_{2,2}$
$h_6 = a_{2,1}\ b_{1,2}$	$h_6 = (-a_{1,1} + a_{2,1})\ (b_{1,1} + b_{1,2})$
$h_7 = a_{2,2}\ b_{2,1}$	$h_7 = (a_{1,2} - a_{2,2})(b_{2,1} + b_{2,2})$
$h_8 = a_{2,2}\ b_{2,2}$	
$c_{1,1} = h_1 + h_3$	$c_{1,1} = h_1 + h_4 - h_5 + h_7$
$c_{1,2} = h_2 + h_4$	$c_{1,2} = h_3 + h_5$
$c_{2,1} = h_5 + h_7$	$c_{2,1} = h_2 + h_4$
$c_{2,2} = h_6 + h_8$	$c_{2,2} = h_1 - h_2 + h_3 + h_6$

AlphaTensor 概述

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，通过多次自我博弈，找到这个矩阵 T_n 的最小的低秩分解。
- **解法细节**：使用 transformer 对于矩阵 T_n 提取特征后，policy 头与 value 头计算低秩分解的下一步的潜在概率与得分。使用 MCTS 剪枝搜索，产出一个完整的低秩分解。

问题描述：搜索空间的构建

AlphaTensor 将寻找算法抽象为寻找 **3D 张量分解**的过程。

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：

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

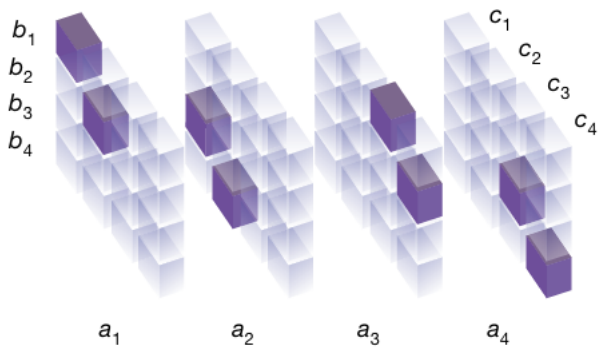
问题描述：直观解释

以 $2 \times 2 \times 2$ 矩阵为例

- **矩阵结构**：列代表乘法数，行代表最终矩阵元素数
- **三个矩阵的含义**：
  - 绿色矩阵 (U)：第一个乘数的线性组合
  - 紫色矩阵 (V)：第二个乘数的线性组合
  - 橙色矩阵 (W)：最终结果的计算方式

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

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

c

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

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

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

强化学习建模

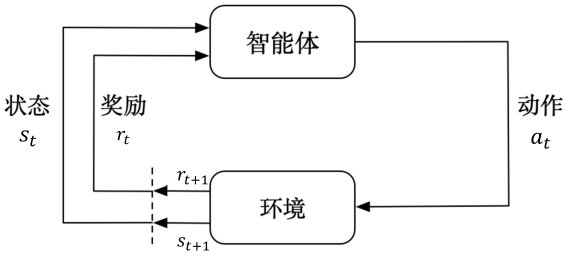
- **游戏目标**：用尽可能少的步数（秩-1 张量）将初始张量 T_n 减为零张量。
- **状态 (State)**：当前剩余的张量 S_t（初始为 T_n）。
- **动作 (Action)**：选择三个向量 u, v, w 构成一个秩-1 张量。
  - 离散化：系数限制在 $\{-2, -1, 0, 1, 2\}$ 中，剪枝搜索空间。
- **状态更新**：$S_{t+1} \leftarrow S_t - u \otimes v \otimes w$

奖励函数 (Reward)

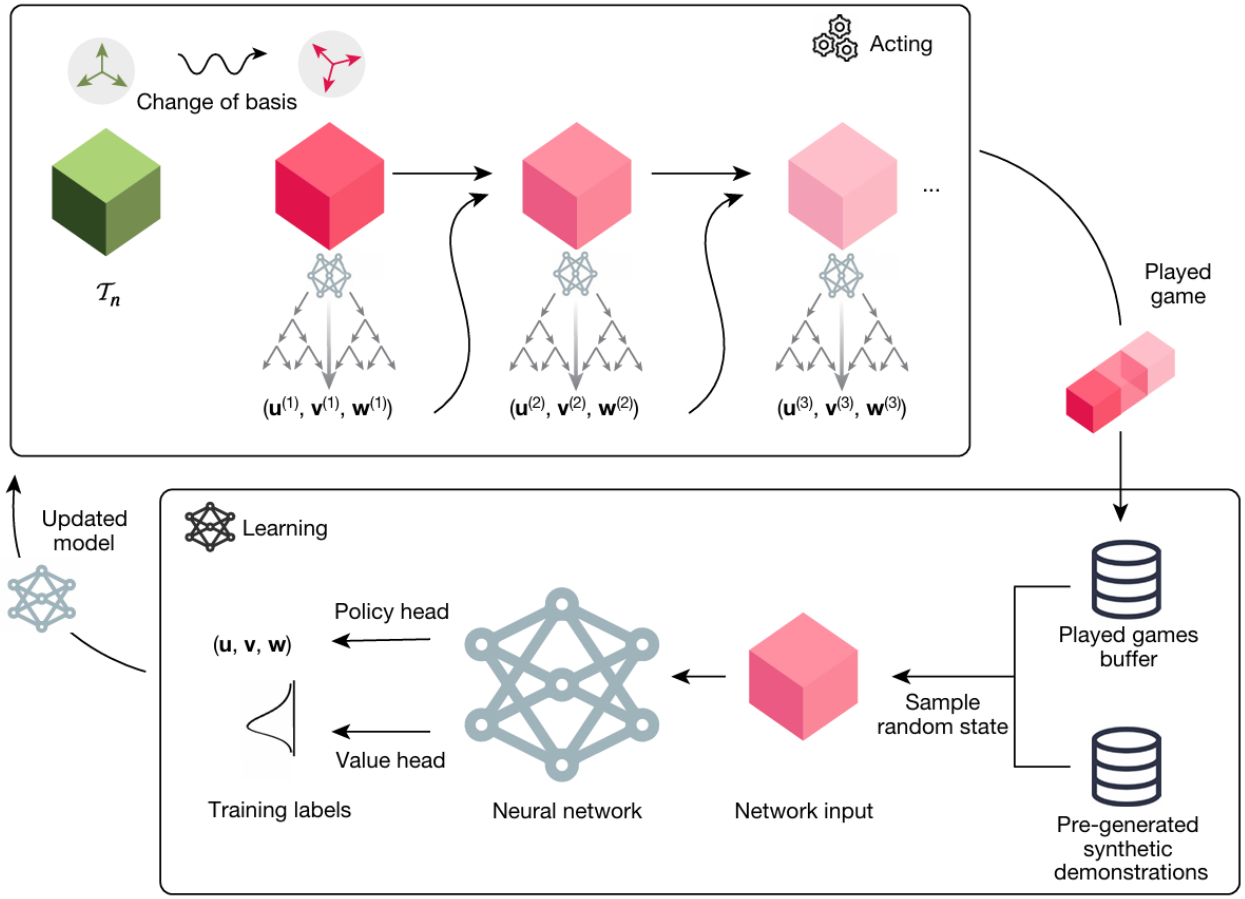
- 每走一步，Reward $=$ -1（鼓励更少步数）。
- 若步骤超限未归零，施加额外惩罚（与剩余张量的秩相关）。
- 可选：加入硬件运行时延时作为负奖励（针对特定硬件优化）。

强化学习架构

强化学习典型架构



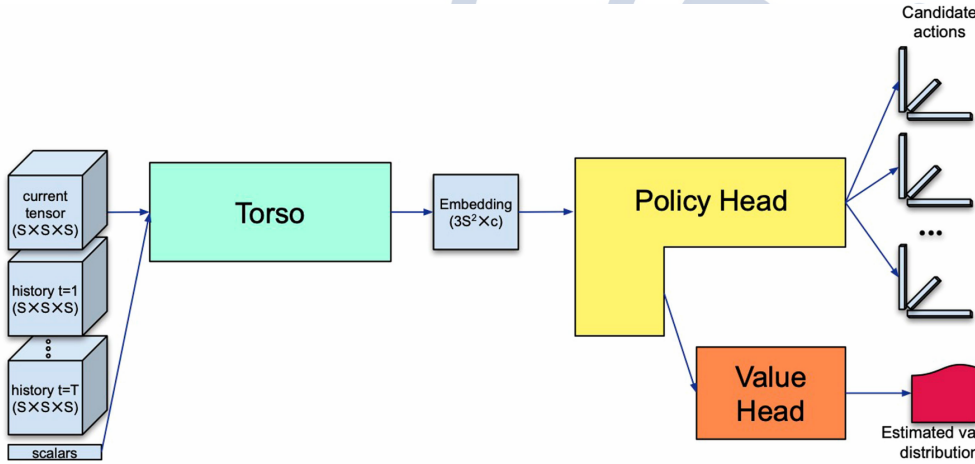
AlphaTensor 架构



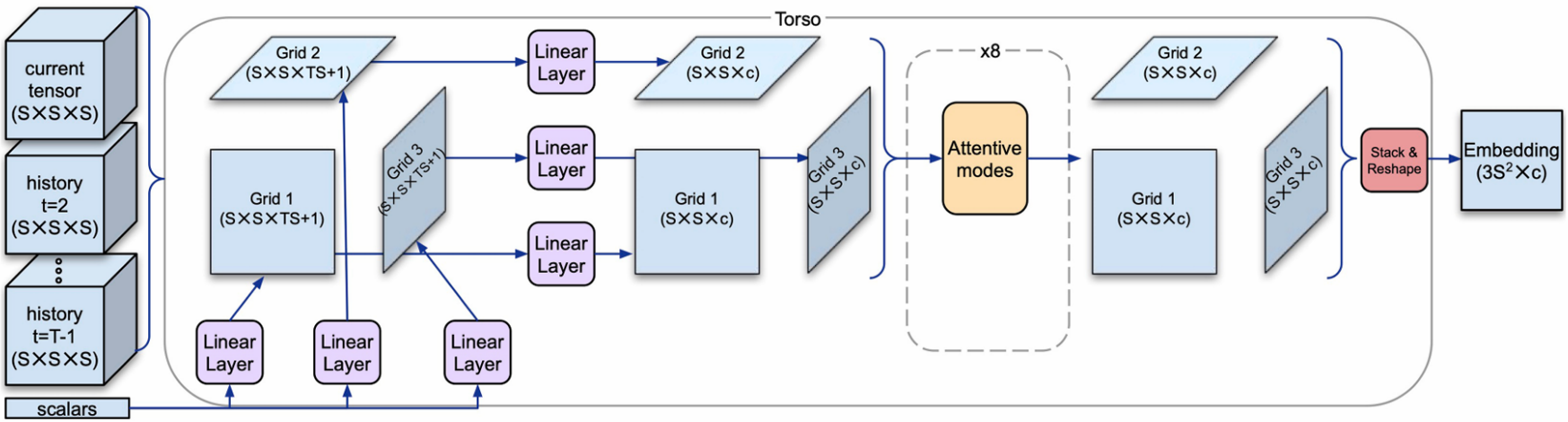
网络架构与搜索算法

AlphaZero 风格的 RL + MCTS

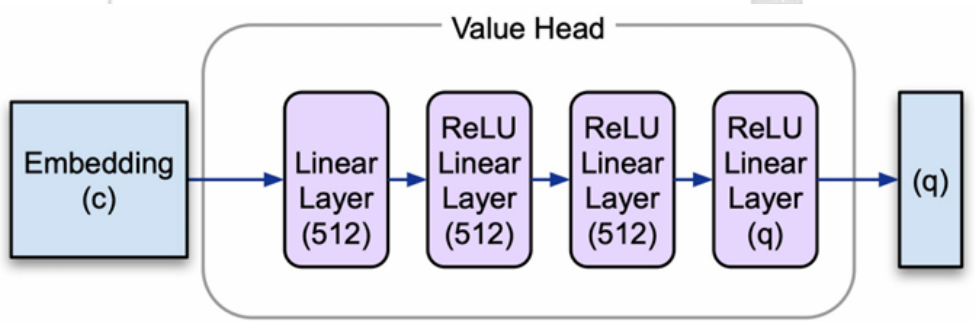
- **MCTS (蒙特卡洛树搜索)**：用于规划下一步动作，平衡探索 (Explore) 与利用 (Exploit)。
- **Policy Network**：基于 Transformer 架构。
  - 输入：当前张量状态。
  - 输出：建议的动作分布。
  - 包含 Cross-attention, Causal self-attention 等机制。
- **Value Network**：预测当前状态归零所需的最小步数。



AlphaTensor 架构细节 1

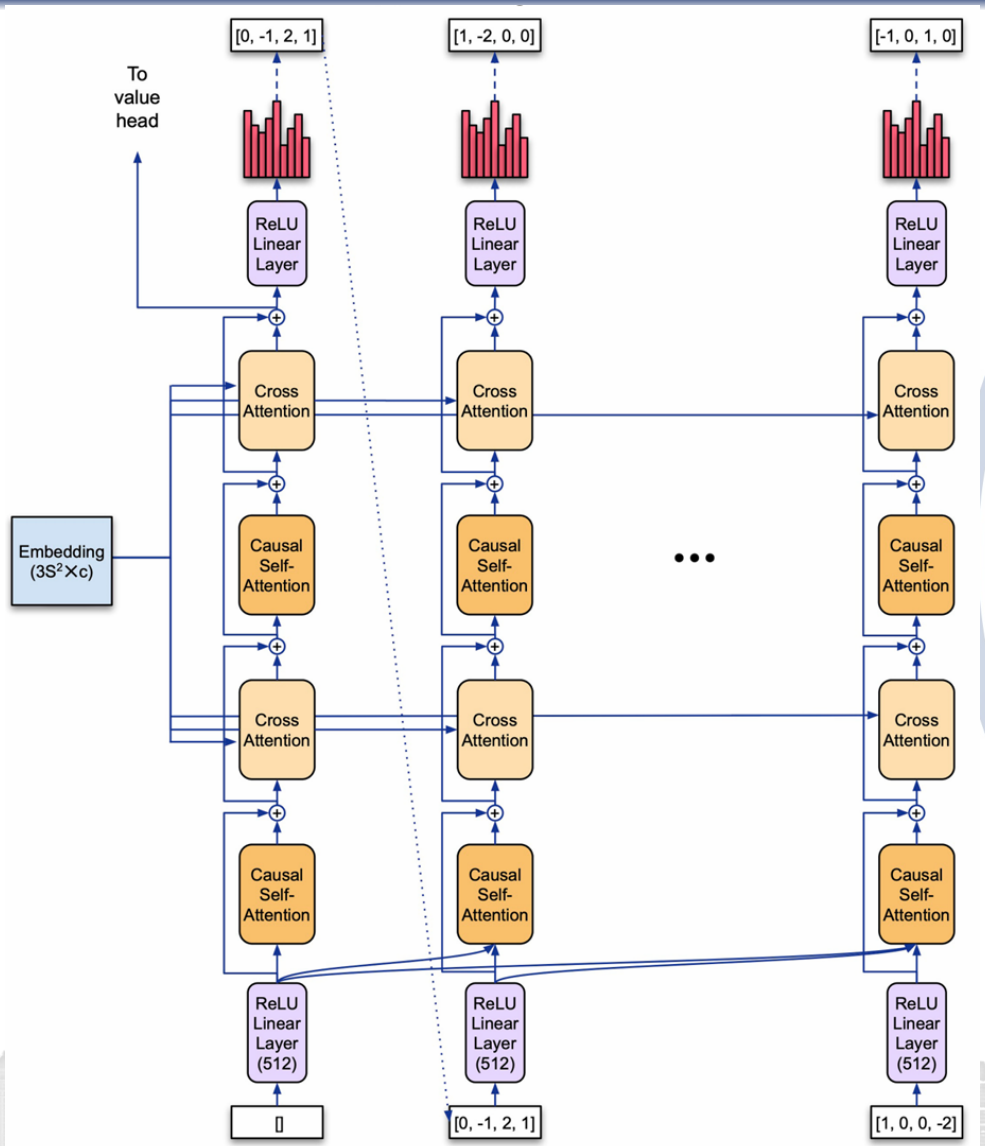


(a) Torso diagram.



(c) Value head diagram.

AlphaTensor 架构细节 2



(b) Policy head diagram. Blocks in the same row share the learning weights.

理论突破：发现更优的秩

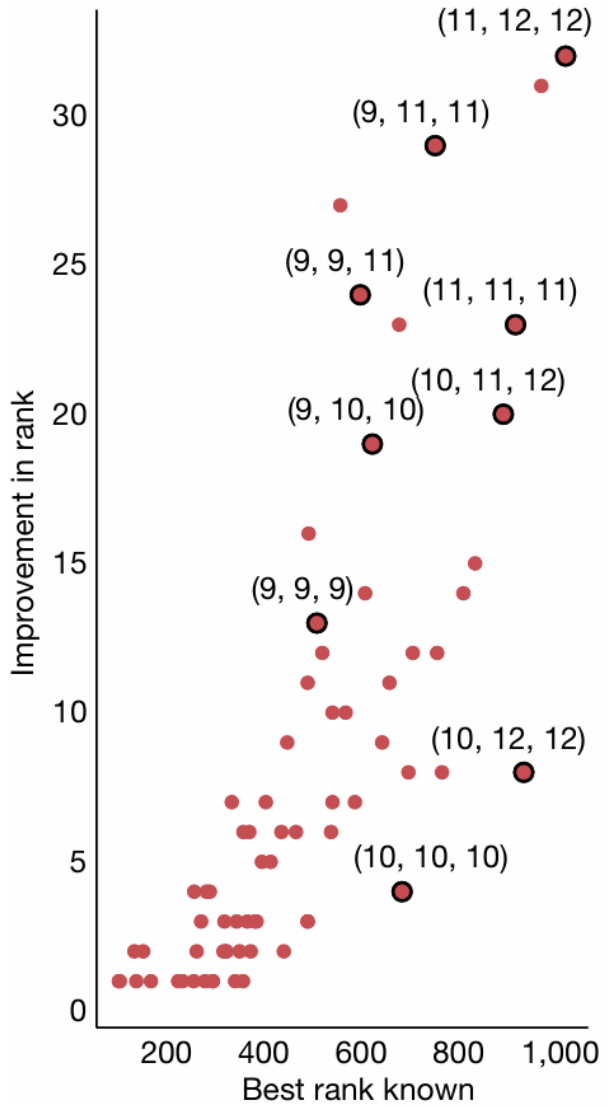
表 1: AlphaTensor 在不同矩阵尺寸下的发现

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。
- **注**：寻找矩阵最小的低秩分解，是 np-hard 问题。3×3 矩阵乘法已知的最小秩为 23，但全局最优解仍然未知，本文也未攻克。

理论突破：不同矩阵乘法的最小秩

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] $(2, 2, 2) \otimes (2, 2, 2)$	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



理论突破：4*4*4 矩阵的低秩分解

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

$$\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

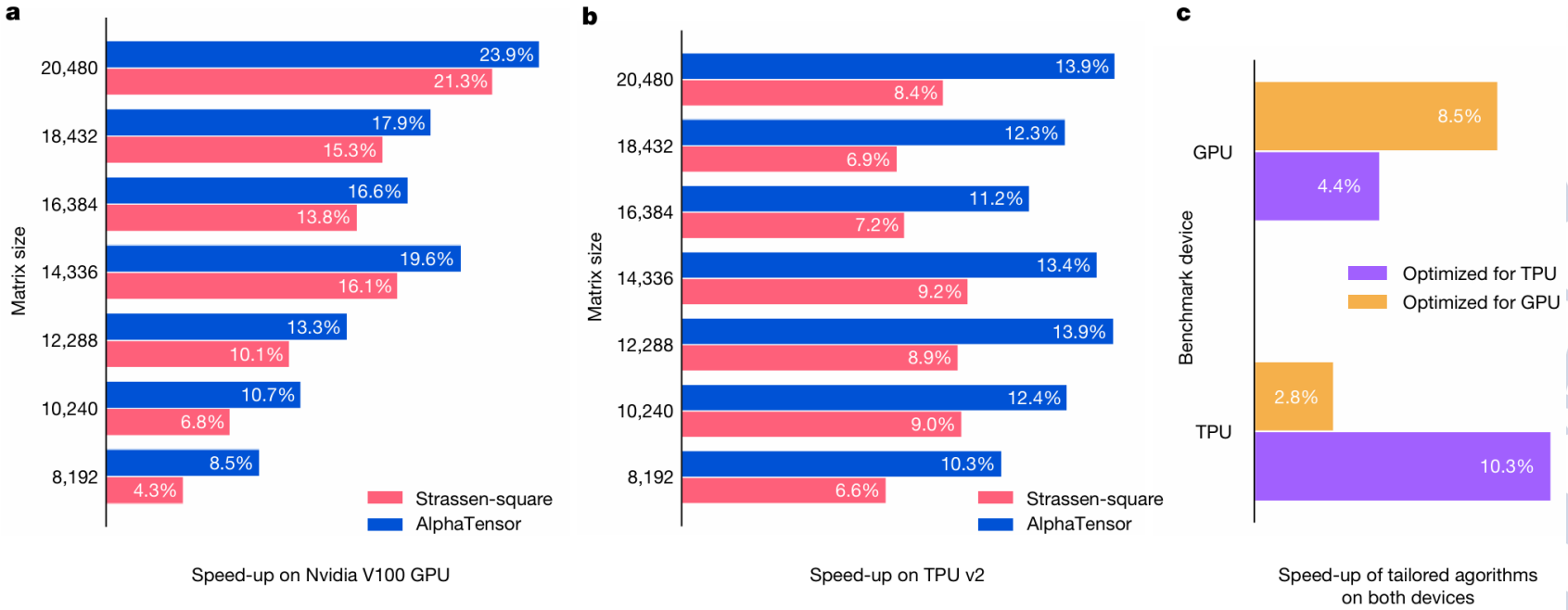
AlphaTensor 不仅理论上减少了计算次数，还能针对特定硬件 (GPU V100, TPU v3) 进行优化。

Runtime 优化

- 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。

- 在矩阵尺寸增大时，加速比显著提升。
- 证明了 AI 发现的算法具有更强的加速效果。

实际应用：加速效果


a
Matrix size
20,480
18,432
16,384
14,336
12,288
10,240
8,192
23.9%
21.3%
17.9%
15.3%
16.6%
13.8%
19.6%
16.1%
13.3%
10.1%
10.7%
6.8%
8.5%
4.3%
Strassen-square
AlphaTensor
Speed-up on Nvidia V100 GPU
b
Matrix size
20,480
18,432
16,384
14,336
12,288
10,240
8,192
13.9%
8.4%
12.3%
6.9%
11.2%
7.2%
13.4%
9.2%
13.9%
8.9%
12.4%
9.0%
10.3%
6.6%
Strassen-square
AlphaTensor
Speed-up on TPU v2
c
Benchmark device
GPU
TPU
8.5%
4.4%
2.8%
10.3%
Optimized for TPU
Optimized for GPU
Speed-up of tailored agorithms
on both devices

消融实验

消融条件	发现的秩
Without *synthetic demonstrations*	64
Without *selfplay* (supervised only)	60
Without *change of basis*	58
Without *signed permutations*	53
Without retraining on best games 10% of the time	52
Without *QR head* (using a *categorical head*)	52
AlphaTensor (完整版)	**49**

结论

完整的 AlphaTensor 达到最优性能（Rank = 49）。移除任何组件都会导致性能下降，
其中 *synthetic demonstrations*（合成演示）的影响最大。

复现结果: 训练部分

```
(base) root@autodl-container-b2b911ba00-d8d87f98:~/li# conda activate opentensor
(opentensor) root@autodl-container-b2b911ba00-d8d87f98:~/li# nvidia-smi
Wed Dec 10 17:44:22 2025
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 525.89.02    Driver Version: 525.89.02    CUDA Version: 12.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla V100-PCIE...  On   | 00000000:65:04.0 Off |                  Off |
| N/A   32C    P0    26W / 250W |      0MiB / 32768MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

```
(opentensor) root@autodl-container-b2b911ba00-d8d87f98:~/li/OpenTensor# python main.py --config ./config/my_conf.yaml --mode train
Preprocessing dataset...
0it [00:00, ?it/s]
  0%|                                                                                                | 0/100000 [00:00<?, ?it/s]/root/li/OpenTensor/codes/dataset/dataset.py:64: F
utureWarning: arrays to stack must be passed as a "sequence" type such as list or tuple. Support for non-sequence iterables such as generators is deprecated as
of NumPy 1.16 and will raise an error in the future.
  self._prepare_examples_from_trajs()
100%|████████████████████████████████████████████████████████████████████████████████| 100000/100000 [01:32<00:00, 1080.39it/s]
  0%|                                                                                                                  | 0/3000000 [00:00<?, ?it/s]/root/li/OpenTensor/codes/tr
ainer/Trainer.py:362: FutureWarning: `torch.nn.utils.clip_grad_norm` is now deprecated in favor of `torch.nn.utils.clip_grad_norm_`.
  torch.nn.utils.clip_grad_norm(self.net.parameters(),
  0%|                                                                                                | 318/3000000 [04:59<763:18:20,  1.09it/s]Permutate!
/root/li/OpenTensor/codes/trainer/Trainer.py:332: FutureWarning: arrays to stack must be passed as a "sequence" type such as list or tuple. Support for non-sequ
ence iterables such as generators is deprecated as of NumPy 1.16 and will raise an error in the future.
  dataloader.dataset._permutate_traj()
Epoch: 0 finish.
  0%|                                                                                                | 636/3000000 [12:29<788:13:07,  1.06it/s]Permutate!
Epoch: 1 finish.
  0%|                                                                                                | 954/3000000 [19:56<763:45:14,  1.09it/s]Permutate!
Epoch: 2 finish.
  0%|                                                                                               | 1272/3000000 [27:21<778:21:31,  1.07it/s]Permutate!
  0%|                                                                                               | 1272/3000000 [27:49<1093:07:33,  1.31s/it]
```

复现结果: 运算部分

```
(alphatensor) root@autodl-container-b2b911ba00-d8d87f98:~
/li/pre_alphatensor# python3 benchmarking/run_gpu_benchma
rk.py
Fixing GPU clock frequency to 1530 to reduce benchmarking
 variance...
Done.
Multiplying 8192 x 8192 matrices
=====================================
Strassen^2 vs `jnp.dot`: 5.35% speedup
AlphaTensor GPU-optimized vs `jnp.dot`: 8.55% speedup
AlphaTensor TPU-optimized vs `jnp.dot`: 4.99% speedup


Multiplying 10240 x 10240 matrices
=====================================
Strassen^2 vs `jnp.dot`: 15.98% speedup
AlphaTensor GPU-optimized vs `jnp.dot`: 19.11% speedup
AlphaTensor TPU-optimized vs `jnp.dot`: 15.97% speedup


Multiplying 12288 x 12288 matrices
=====================================
Strassen^2 vs `jnp.dot`: 14.53% speedup
AlphaTensor GPU-optimized vs `jnp.dot`: 17.09% speedup
AlphaTensor TPU-optimized vs `jnp.dot`: 14.39% speedup


Multiplying 14336 x 14336 matrices
=====================================
Strassen^2 vs `jnp.dot`: 18.53% speedup
AlphaTensor GPU-optimized vs `jnp.dot`: 20.82% speedup
AlphaTensor TPU-optimized vs `jnp.dot`: 18.44% speedup
```

```
Multiplying 14336 x 14336 matrices
=====================================
Strassen^2 vs `jnp.dot`: 18.53% speedup
AlphaTensor GPU-optimized vs `jnp.dot`: 20.82% speedup
AlphaTensor TPU-optimized vs `jnp.dot`: 18.44% speedup


Multiplying 16384 x 16384 matrices
=====================================
Strassen^2 vs `jnp.dot`: 17.61% speedup
AlphaTensor GPU-optimized vs `jnp.dot`: 19.60% speedup
AlphaTensor TPU-optimized vs `jnp.dot`: 17.60% speedup


Multiplying 18432 x 18432 matrices
=====================================
Strassen^2 vs `jnp.dot`: 19.28% speedup
AlphaTensor GPU-optimized vs `jnp.dot`: 20.99% speedup
AlphaTensor TPU-optimized vs `jnp.dot`: 19.18% speedup


Multiplying 20480 x 20480 matrices
=====================================
Strassen^2 vs `jnp.dot`: 24.04% speedup
AlphaTensor GPU-optimized vs `jnp.dot`: 25.84% speedup
```

复现结果: 运算结果对照

矩阵尺寸	Strassen^2		AlphaTensor	
	复现	原版	复现	原版
8192	5.35	4.3	8.55	8.5
10240	15.98	6.8	19.11	10.7
12288	14.53	10.1	17.09	13.3
14336	18.53	16.1	20.82	19.6
16384	17.61	13.8	19.60	16.6
18432	19.28	15.3	20.99	17.9
20480	24.04	21.3	25.84	23.9

表 2: 不同算法相对于 jnp.dot 的加速比（单位：%）

文章局限：工业实践角度

硬件指令集的冲突

- 所有**现代 GPU** 以Tensor Cores为计算核心，这些硬件单元在物理电路层面，针对标准的 $O(N^3)$ 算法逻辑进行高度优化。
- 在 Tensor Cores 上，" 标准乘法" 极快。虽然 AlphaTensor 减少了乘法操作，但引入了更多加法操作和内存移动开销，在计算密集型硬件上往往得不偿失。

数值稳定性

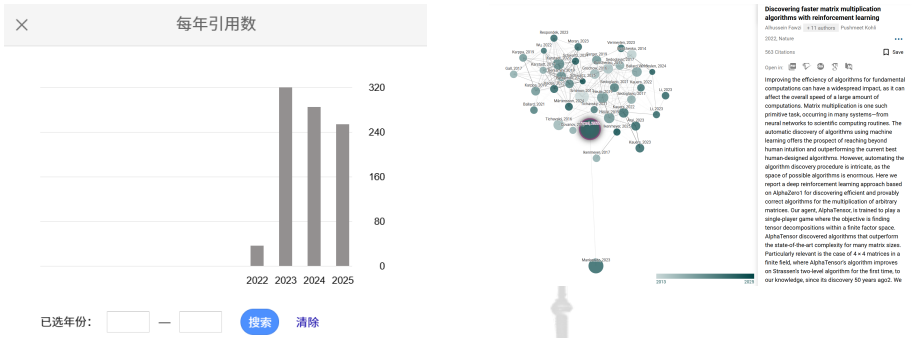
- AlphaTensor 通过复杂的加减法组合减少乘法，在浮点运算中容易引入精度损失，数值稳定性差。
- 标准库倾向于选择**最稳定**的算法而非理论上最快的。

小矩阵 vs. 大矩阵

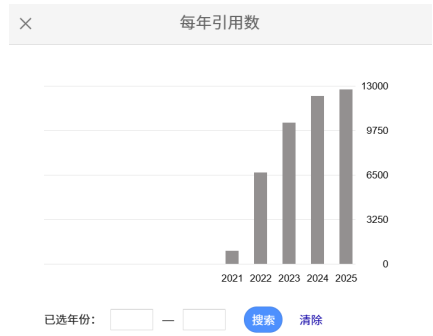
- AlphaTensor 主要优化极小矩阵核（如 4×4），大矩阵可通过分块计算。
- 标准库使用高度定制的汇编代码处理分块，将 AlphaTensor 算法强行集成到现有流水线需要重写底层汇编优化。

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



AlphaFold

文章贡献

- **矩阵加速**：
  - AlphaTensor 将大矩阵乘法次数的下界从 $O(N^{2.807})$ 降低到了 $O(N^{2.777})$。
- **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。是强化学习的又一次成功应用。
- **未来展望**：
  - 优化其他经典算法/理论计算机科学方法的算法流程?
  - 解决其他 np-hard 问题?
  - 把强化学习这种思想，全面引入传统理工科?

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.

[HL13] Christopher J. Hillar and Lek-Heng Lim.
Most tensor problems are np-hard.
J. ACM, 60(6):45, 2013.

[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.

参考资料 II

[SL24] Yiwen Sun and Wenye Li.
Opentensor: Reproducing faster matrix multiplication discovering algorithms, 2024.

[Str69] Volker Strassen.
Gaussian elimination is not optimal.
Numerische Mathematik, 13(4):354–356, 1969.

[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

项目地址:

- https://gitee.com/lychee-garden/pre_alphatensor.git
- https://gitee.com/lychee-garden/pre_opentensor.git

Thanks!