Google DeepMind

Diseñando los algoritmos del futuro con IA

Francisco J. R. Ruiz
06/10/2025

Presentación

Ingeniería de
Telecomunicaciones

Presentación

Ingeniería de Telecomunicaciones

Máster y Doctorado en Aprendizaje Automático

Presentación

Ingeniería de Telecomunicaciones

Máster y Doctorado en Aprendizaje Automático

Postdoctorado en Aprendizaje Automático

Presentación

Ingeniería de Telecomunicaciones

Máster y Doctorado en Aprendizaje Automático

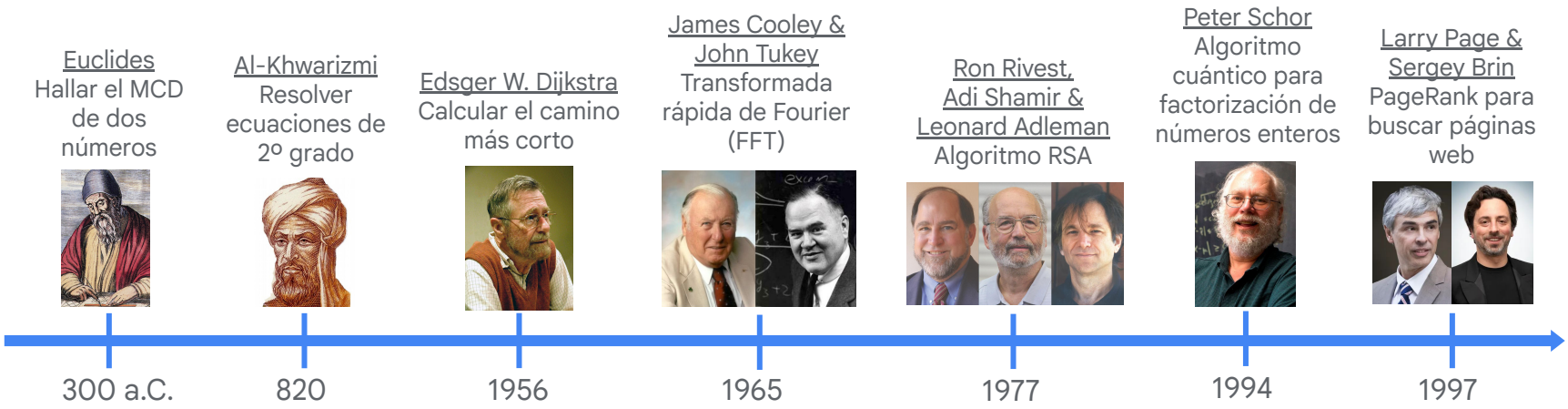
Postdoctorado en Aprendizaje Automático

Investigador en Google DeepMind

Índice

Índice

Algoritmos


Euclides
Hallar el MCD de dos números
Al-Khwarizmi
Resolver ecuaciones de 2º grado
Edsger W. Dijkstra
Calcular el camino más corto
James Cooley & John Tukey
Transformada rápida de Fourier (FFT)
Ron Rivest, Adi Shamir & Leonard Adleman
Algoritmo RSA
Peter Schor
Algoritmo cuántico para factorización de números enteros
Larry Page & Sergey Brin
PageRank para buscar páginas web
300 a.C.
820
1956
1965
1977
1994
1997

“

Descubrir nuevos resultados y algoritmos eficientes para problemas relevantes, usando IA

“

Descubrir nuevos resultados y algoritmos eficientes para problemas relevantes, usando IA

“

Descubrir nuevos resultados y algoritmos eficientes para problemas relevantes, usando IA

“

Descubrir nuevos resultados y algoritmos eficientes para problemas relevantes, usando IA

“

Descubrir nuevos resultados y algoritmos eficientes para problemas relevantes, usando IA

Multiplicación de Matrices

$$\begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \times \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$

Multiplicación de Matrices

$$\begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \times \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$

Algoritmo estándar

$$\begin{aligned} h_1 &= a_{11}\, b_{11} \\ h_2 &= a_{11}\, b_{12} \\ h_3 &= a_{12}\, b_{21} \\ h_4 &= a_{12}\, b_{22} \\ h_5 &= a_{21}\, b_{11} \\ h_6 &= a_{21}\, b_{12} \\ h_7 &= a_{22}\, b_{21} \\ h_8 &= a_{22}\, b_{22} \end{aligned}$$

$$\begin{aligned} c_{11} &= h_1 + h_3 \\ c_{12} &= h_2 + h_4 \\ c_{21} &= h_5 + h_7 \\ c_{22} &= h_6 + h_8 \end{aligned}$$

Multiplicación de Matrices

$$\begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \times \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$

Algoritmo estándar

$$\begin{aligned} h_1 &= a_{11}\, b_{11} \\ h_2 &= a_{11}\, b_{12} \\ h_3 &= a_{12}\, b_{21} \\ h_4 &= a_{12}\, b_{22} \\ h_5 &= a_{21}\, b_{11} \\ h_6 &= a_{21}\, b_{12} \\ h_7 &= a_{22}\, b_{21} \\ h_8 &= a_{22}\, b_{22} \end{aligned}$$

$$\begin{aligned} c_{11} &= h_1 + h_3 \\ c_{12} &= h_2 + h_4 \\ c_{21} &= h_5 + h_7 \\ c_{22} &= h_6 + h_8 \end{aligned}$$

Algoritmo de Strassen

$$\begin{aligned} h_1 &= (a_{11} + a_{22})\,(b_{11} + b_{22}) \\ h_2 &= (a_{21} + a_{22})\, b_{11} \\ h_3 &= a_{11}\,(b_{12} - b_{22}) \\ h_4 &= a_{22}\,(-b_{11} + b_{21}) \\ h_5 &= (a_{11} + a_{12})\, b_{22} \\ h_6 &= (-a_{11} + a_{21})\,(b_{11} + b_{12}) \\ h_7 &= (a_{12} - a_{22})\,(b_{21} + b_{22}) \end{aligned}$$

$$\begin{aligned} c_{11} &= h_1 + h_4 - h_5 + h_7 \\ c_{12} &= h_3 + h_5 \\ c_{21} &= h_2 + h_4 \\ c_{22} &= h_1 - h_2 + h_3 + h_6 \end{aligned}$$

Multiplicación de Matrices

$$\begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \times \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$

8 multiplicaciones de dos escalares

Algoritmo estándar

$$\begin{aligned} h_1 &= a_{11}\, b_{11} \\ h_2 &= a_{11}\, b_{12} \\ h_3 &= a_{12}\, b_{21} \\ h_4 &= a_{12}\, b_{22} \\ h_5 &= a_{21}\, b_{11} \\ h_6 &= a_{21}\, b_{12} \\ h_7 &= a_{22}\, b_{21} \\ h_8 &= a_{22}\, b_{22} \end{aligned}$$

$$\begin{aligned} c_{11} &= h_1 + h_3 \\ c_{12} &= h_2 + h_4 \\ c_{21} &= h_5 + h_7 \\ c_{22} &= h_6 + h_8 \end{aligned}$$

Algoritmo de Strassen

$$\begin{aligned} h_1 &= (a_{11} + a_{22})\,(b_{11} + b_{22}) \\ h_2 &= (a_{21} + a_{22})\, b_{11} \\ h_3 &= a_{11}\,(b_{12} - b_{22}) \\ h_4 &= a_{22}\,(-b_{11} + b_{21}) \\ h_5 &= (a_{11} + a_{12})\, b_{22} \\ h_6 &= (-a_{11} + a_{21})\,(b_{11} + b_{12}) \\ h_7 &= (a_{12} - a_{22})\,(b_{21} + b_{22}) \end{aligned}$$

$$\begin{aligned} c_{11} &= h_1 + h_4 - h_5 + h_7 \\ c_{12} &= h_3 + h_5 \\ c_{21} &= h_2 + h_4 \\ c_{22} &= h_1 - h_2 + h_3 + h_6 \end{aligned}$$

7 multiplicaciones de dos escalares

Multiplicación de Matrices: Recursividad

$$N \times N \qquad \left(\begin{array}{c|c} C_{11} & C_{12} \\ \hline C_{21} & C_{22} \end{array} \right) = \left(\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array} \right) \times \left(\begin{array}{c|c} B_{11} & B_{12} \\ \hline B_{21} & B_{22} \end{array} \right)$$

Multiplicación de Matrices: Recursividad

$$N \times N \qquad \left(\begin{array}{c|c} C_{11} & C_{12} \\ \hline C_{21} & C_{22} \end{array}\right) = \left(\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right) \times \left(\begin{array}{c|c} B_{11} & B_{12} \\ \hline B_{21} & B_{22} \end{array}\right)$$

8 multiplicaciones de dos **matrices**

Algoritmo estándar

$$\begin{aligned} H_1 &= A_{11}\ B_{11} \\ H_2 &= A_{11}\ B_{12} \\ H_3 &= A_{12}\ B_{21} \\ H_4 &= A_{12}\ B_{22} \\ H_5 &= A_{21}\ B_{11} \\ H_6 &= A_{21}\ B_{12} \\ H_7 &= A_{22}\ B_{21} \\ H_8 &= A_{22}\ B_{22} \end{aligned}$$

$$\begin{aligned} C_{11} &= H_1 + H_3 \\ C_{12} &= H_2 + H_4 \\ C_{21} &= H_5 + H_7 \\ C_{22} &= H_6 + H_8 \end{aligned}$$

Algoritmo de Strassen

$$\begin{aligned} H_1 &= (A_{11} + A_{22})\ (B_{11} + B_{22}) \\ H_2 &= (A_{21} + A_{22})\ B_{11} \\ H_3 &= A_{11}\ (B_{12} - B_{22}) \\ H_4 &= A_{22}\ (-B_{11} + B_{21}) \\ H_5 &= (A_{11} + A_{12})\ B_{22} \\ H_6 &= (-A_{11} + A_{21})\ (B_{11} + B_{12}) \\ H_7 &= (A_{12} - A_{22})\ (B_{21} + B_{22}) \end{aligned}$$

$$\begin{aligned} C_{11} &= H_1 + H_4 - H_5 + H_7 \\ C_{12} &= H_3 + H_5 \\ C_{21} &= H_2 + H_4 \\ C_{22} &= H_1 - H_2 + H_3 + H_6 \end{aligned}$$

7 multiplicaciones de dos **matrices**

Multiplicación de Matrices: Complejidad Computacional

tamaño $(n \times n)$	#multiplicaciones	complejidad $(N \times N)$
2×2	7	$\mathcal{O}(N^{\log_2 7}) \approx \mathcal{O}(N^{2.81})$

Multiplicación de Matrices: Complejidad Computacional

tamaño $(n \times n)$	#multiplicaciones	complejidad $(N \times N)$
2×2 3×3	7	$\mathcal{O}(N^{\log_2 7}) \approx \mathcal{O}(N^{2.81})$

Multiplicación de Matrices: Complejidad Computacional

tamaño $(n \times n)$	#multiplicaciones	complejidad $(N \times N)$
2×2	7	$\mathcal{O}(N^{\log_2 7}) \approx \mathcal{O}(N^{2.81})$
3×3		
4×4		
$\vdots$	$\vdots$	$\vdots$

Multiplicación de Matrices: Complejidad Computacional

tamaño $(n \times n)$	#multiplicaciones	complejidad $(N \times N)$
2×2	7	$\mathcal{O}(N^{\log_2 7}) \approx \mathcal{O}(N^{2.81})$
3×3		
4×4		
$\vdots$	$\vdots$	$\vdots$

Problemas abiertos

Multiplicación de Matrices: Complejidad Computacional

tamaño $(n \times n)$	#multiplicaciones	complejidad $(N \times N)$
2×2	7	$\mathcal{O}(N^{\log_2 7}) \approx \mathcal{O}(N^{2.81})$
3×3	23	$\mathcal{O}(N^{\log_3 23}) \approx \mathcal{O}(N^{2.85})$
4×4	48	$\mathcal{O}(N^{\log_4 48}) \approx \mathcal{O}(N^{2.79})$
$\vdots$	$\vdots$	$\vdots$

Problemas abiertos

Multiplicación de Matrices: Complejidad Computacional

tamaño $(n \times n)$	#multiplicaciones	complejidad $(N \times N)$
2×2	7	$\mathcal{O}(N^{\log_2 7}) \approx \mathcal{O}(N^{2.81})$
3×3	23	$\mathcal{O}(N^{\log_3 23}) \approx \mathcal{O}(N^{2.85})$
4×4	48	$\mathcal{O}(N^{\log_4 48}) \approx \mathcal{O}(N^{2.79})$
$\vdots$	$\vdots$	$\vdots$

Problemas abiertos

descubierto por AlphaEvolve

Multiplicación de Matrices: Complejidad Computacional

tamaño $(n \times n)$	#multiplicaciones	complejidad $(N \times N)$
2×2	7	$\mathcal{O}(N^{\log_2 7}) \approx \mathcal{O}(N^{2.81})$
3×3	23	$\mathcal{O}(N^{\log_3 23}) \approx \mathcal{O}(N^{2.85})$
4×4	48	$\mathcal{O}(N^{\log_4 48}) \approx \mathcal{O}(N^{2.79})$
$\vdots$	$\vdots$	$\vdots$

Problemas abiertos

descubierto por AlphaEvolve

Descubrir nuevos algoritmos es una tarea compleja

Índice

Resultados de AlphaTensor

Tamaño (n, m, p)	Mejor método conocido	Mejor #mult	AlphaTensor Modular	AlphaTensor Estándar
$(2, 2, 2)$	(Strassen, 1969)	7	7	7
$(3, 3, 3)$	(Laderman, 1976)	23	23	23
$(4, 4, 4)$	(Strassen, 1969) $(2, 2, 2) \otimes (2, 2, 2)$	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)	15	15	15
$(2, 3, 4)$	(Hopcroft and Kerr, 1971)	20	20	20
$(2, 3, 5)$	(Hopcroft and Kerr, 1971)	25	25	25
$(2, 4, 4)$	(Hopcroft and Kerr, 1971)	26	26	26
$(2, 4, 5)$	(Hopcroft and Kerr, 1971)	33	33	33
$(2, 5, 5)$	(Hopcroft and Kerr, 1971)	40	40	40
$(3, 3, 4)$	(Smirnov, 2013)	29	29	29
$(3, 3, 5)$	(Smirnov, 2013)	36	36	36
$(3, 4, 4)$	(Smirnov, 2013)	38	38	38
$(3, 4, 5)$	(Smirnov, 2013)	48	**47**	**47**
$(3, 5, 5)$	(Sedoglavic and Smirnov, 2021)	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**

Resultados de AlphaTensor

Tamaño (n, m, p)	Mejor método conocido	Mejor #mult	AlphaTensor Modular	AlphaTensor Estándar
$(2, 2, 2)$	(Strassen, 1969)	7	7	7
$(3, 3, 3)$	(Laderman, 1976)	23	23	23
$(4, 4, 4)$	(Strassen, 1969) $(2, 2, 2) \otimes (2, 2, 2)$	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)	15	15	15
$(2, 3, 4)$	(Hopcroft and Kerr, 1971)	20	20	20
$(2, 3, 5)$	(Hopcroft and Kerr, 1971)	25	25	25
$(2, 4, 4)$	(Hopcroft and Kerr, 1971)	26	26	26
$(2, 4, 5)$	(Hopcroft and Kerr, 1971)	33	33	33
$(2, 5, 5)$	(Hopcroft and Kerr, 1971)	40	40	40
$(3, 3, 4)$	(Smirnov, 2013)	29	29	29
$(3, 3, 5)$	(Smirnov, 2013)	36	36	36
$(3, 4, 4)$	(Smirnov, 2013)	38	38	38
$(3, 4, 5)$	(Smirnov, 2013)	48	**47**	**47**
$(3, 5, 5)$	(Sedoglavic and Smirnov, 2021)	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**

Algoritmo Descubierto por AlphaTensor

$$\begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & a_{1,5} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} & a_{2,5} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} & a_{3,5} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} & a_{4,5} \end{pmatrix} \begin{pmatrix} b_{1,1} & b_{1,2} & b_{1,3} & b_{1,4} & b_{1,5} \\ b_{2,1} & b_{2,2} & b_{2,3} & b_{2,4} & b_{2,5} \\ b_{3,1} & b_{3,2} & b_{3,3} & b_{3,4} & b_{3,5} \\ b_{4,1} & b_{4,2} & b_{4,3} & b_{4,4} & b_{4,5} \\ b_{5,1} & b_{5,2} & b_{5,3} & b_{5,4} & b_{5,5} \end{pmatrix} = \begin{pmatrix} c_{1,1} & c_{1,2} & c_{1,3} & c_{1,4} & c_{1,5} \\ c_{2,1} & c_{2,2} & c_{2,3} & c_{2,4} & c_{2,5} \\ c_{3,1} & c_{3,2} & c_{3,3} & c_{3,4} & c_{3,5} \\ c_{4,1} & c_{4,2} & c_{4,3} & c_{4,4} & c_{4,5} \end{pmatrix}$$

Algoritmo Descubierto por AlphaTensor

$$\begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & a_{1,5} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} & a_{2,5} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} & a_{3,5} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} & a_{4,5} \end{pmatrix} \begin{pmatrix} b_{1,1} & b_{1,2} & b_{1,3} & b_{1,4} & b_{1,5} \\ b_{2,1} & b_{2,2} & b_{2,3} & b_{2,4} & b_{2,5} \\ b_{3,1} & b_{3,2} & b_{3,3} & b_{3,4} & b_{3,5} \\ b_{4,1} & b_{4,2} & b_{4,3} & b_{4,4} & b_{4,5} \\ b_{5,1} & b_{5,2} & b_{5,3} & b_{5,4} & b_{5,5} \end{pmatrix} = \begin{pmatrix} c_{1,1} & c_{1,2} & c_{1,3} & c_{1,4} & c_{1,5} \\ c_{2,1} & c_{2,2} & c_{2,3} & c_{2,4} & c_{2,5} \\ c_{3,1} & c_{3,2} & c_{3,3} & c_{3,4} & c_{3,5} \\ c_{4,1} & c_{4,2} & c_{4,3} & c_{4,4} & c_{4,5} \end{pmatrix}$$

$$\begin{aligned}
h_1 &= a_{3,2}\left(-b_{2,1} - b_{2,5} - b_{3,1}\right) \\
h_2 &= \left(a_{2,2} + a_{2,5} - a_{3,5}\right)\left(-b_{2,5} - b_{5,1}\right) \\
h_3 &= \left(-a_{3,1} - a_{4,1} + a_{4,2}\right)\left(-b_{1,1} + b_{2,5}\right) \\
h_4 &= \left(a_{1,2} + a_{1,4} + a_{3,4}\right)\left(-b_{2,5} - b_{4,1}\right) \\
h_5 &= \left(a_{1,5} + a_{2,2} + a_{2,5}\right)\left(-b_{2,4} + b_{5,1}\right) \\
h_6 &= \left(-a_{2,2} - a_{2,5} - a_{4,5}\right)\left(b_{2,3} + b_{5,1}\right) \\
h_7 &= \left(-a_{1,1} + a_{4,1} - a_{4,2}\right)\left(b_{1,1} + b_{2,4}\right) \\
h_8 &= \left(a_{3,2} - a_{3,3} - a_{4,3}\right)\left(-b_{2,3} + b_{3,1}\right) \\
h_9 &= \left(-a_{1,2} - a_{1,4} + a_{4,4}\right)\left(b_{2,3} + b_{4,1}\right) \\
h_{10} &= \left(a_{2,2} + a_{2,5}\right) b_{5,1} \\
h_{11} &= \left(-a_{2,1} - a_{4,1} + a_{4,2}\right)\left(-b_{1,1} + b_{2,2}\right) \\
h_{12} &= \left(a_{4,1} - a_{4,2}\right) b_{1,1} \\
h_{13} &= \left(a_{1,2} + a_{1,4} + a_{2,4}\right)\left(b_{2,2} + b_{4,1}\right) \\
h_{14} &= \left(a_{1,3} - a_{3,2} + a_{3,3}\right)\left(b_{2,4} + b_{3,1}\right) \\
h_{15} &= \left(-a_{1,2} - a_{1,4}\right) b_{4,1} \\
h_{16} &= \left(-a_{3,2} + a_{3,3}\right) b_{3,1} \\
h_{17} &= \left(a_{1,2} + a_{1,4} - a_{2,1} + a_{2,2} - a_{2,3} + a_{2,4} - a_{3,2} + a_{3,3} - a_{4,1} + a_{4,2}\right) b_{2,2} \\
h_{18} &= a_{2,1}\left(b_{1,1} + b_{1,2} + b_{5,2}\right) \\
h_{19} &= -a_{2,3}\left(b_{3,1} + b_{3,2} + b_{5,2}\right) \\
h_{20} &= \left(-a_{1,5} + a_{2,1} + a_{2,3} - a_{2,5}\right)\left(-b_{1,1} - b_{1,2} + b_{1,4} - b_{5,2}\right) \\
h_{21} &= \left(a_{2,1} + a_{2,3} - a_{2,5}\right) b_{5,2} \\
h_{22} &= \left(a_{1,3} - a_{1,4} - a_{2,4}\right)\left(b_{1,1} + b_{1,2} - b_{1,4} - b_{3,1} - b_{3,2} + b_{3,4} + b_{4,4}\right) \\
h_{23} &= a_{1,3}\left(-b_{3,1} + b_{3,4} + b_{4,4}\right) \\
h_{24} &= a_{1,5}\left(-b_{4,4} - b_{5,1} + b_{5,4}\right) \\
h_{25} &= -a_{1,1}\left(b_{1,1} - b_{1,4}\right) \\
h_{26} &= \left(-a_{1,3} + a_{1,4} + a_{1,5}\right) b_{4,4} \\
h_{27} &= \left(a_{1,3} - a_{3,1} + a_{3,3}\right)\left(b_{1,1} - b_{1,4} + b_{1,5} + b_{3,5}\right)
\end{aligned}$$

$$\begin{aligned}
h_{51} &= a_{2,2}\left(b_{2,1} + b_{2,2} - b_{5,1}\right) \\
h_{52} &= a_{4,2}\left(b_{1,1} + b_{2,1} + b_{2,3}\right) \\
h_{53} &= -a_{1,2}\left(-b_{2,1} + b_{2,4} + b_{4,1}\right) \\
h_{54} &= \left(a_{1,2} + a_{1,4} - a_{2,2} - a_{2,5} - a_{3,2} + a_{3,3} - a_{4,2} + a_{4,3} - a_{4,4} - a_{4,5}\right) b_{2,3} \\
h_{55} &= \left(a_{1,4} - a_{4,4}\right)\left(-b_{2,3} + b_{3,1} + b_{3,3} - b_{3,4} + b_{4,3} - b_{4,4}\right) \\
h_{56} &= \left(a_{1,1} - a_{1,5} - a_{4,1} + a_{4,5}\right)\left(b_{3,1} + b_{3,3} - b_{3,4} + b_{5,1} + b_{5,3} - b_{5,4}\right) \\
h_{57} &= \left(-a_{3,1} - a_{4,1}\right)\left(-b_{1,3} - b_{1,5} - b_{2,5} - b_{5,1} - b_{5,3} - b_{5,5}\right) \\
h_{58} &= \left(-a_{1,4} - a_{1,5} - a_{3,4} - a_{3,5}\right)\left(-b_{5,1} + b_{5,4} - b_{5,5}\right) \\
h_{59} &= \left(-a_{3,3} + a_{3,4} - a_{4,3} + a_{4,4}\right)\left(b_{4,1} + b_{4,3} + b_{4,5} + b_{5,1} + b_{5,3} + b_{5,5}\right) \\
h_{60} &= \left(a_{2,5} + a_{4,5}\right)\left(b_{2,3} - b_{3,1} - b_{3,2} - b_{3,3} - b_{5,2} - b_{5,3}\right) \\
h_{61} &= \left(a_{1,4} + a_{3,4}\right)\left(b_{1,1} - b_{1,4} + b_{1,5} - b_{2,5} - b_{4,4} + b_{4,5} - b_{5,1} + b_{5,4} - b_{5,5}\right) \\
h_{62} &= \left(a_{2,1} + a_{4,1}\right)\left(b_{1,2} + b_{1,3} + b_{2,2} - b_{4,1} - b_{4,2} - b_{4,3}\right) \\
h_{63} &= \left(-a_{3,3} - a_{4,3}\right)\left(-b_{2,3} - b_{3,3} - b_{3,5} - b_{4,1} - b_{4,3} - b_{4,5}\right) \\
h_{64} &= \left(a_{1,1} - a_{1,3} - a_{1,4} + a_{3,1} - a_{3,3} - a_{3,4}\right)\left(b_{1,1} - b_{1,4} + b_{1,5}\right) \\
h_{65} &= \left(-a_{1,1} + a_{4,1}\right)\left(-b_{1,3} + b_{1,4} + b_{2,4} - b_{5,1} - b_{5,3} + b_{5,4}\right) \\
h_{66} &= \left(a_{1,1} - a_{1,2} + a_{1,3} - a_{1,5} - a_{2,2} - a_{2,5} - a_{3,2} + a_{3,3} - a_{4,1} + a_{4,2}\right) b_{2,4} \\
h_{67} &= \left(a_{2,5} - a_{3,5}\right)\left(b_{1,1} + b_{1,2} + b_{1,5} - b_{2,5} - b_{4,1} - b_{4,2} - b_{4,5} + b_{5,2} + b_{5,5}\right) \\
h_{68} &= \left(a_{1,1} + a_{1,3} - a_{1,4} - a_{1,5} - a_{4,1} - a_{4,3} + a_{4,4} + a_{4,5}\right)\left(-b_{3,1} - b_{3,3} + b_{3,4}\right) \\
h_{69} &= \left(-a_{1,3} + a_{1,4} - a_{2,3} + a_{2,4}\right)\left(-b_{2,4} - b_{3,1} - b_{3,2} + b_{3,4} - b_{5,2} + b_{5,4}\right) \\
h_{70} &= \left(a_{2,3} - a_{2,5} + a_{4,3} - a_{4,5}\right)\left(-b_{3,1} - b_{3,2} - b_{3,3}\right) \\
h_{71} &= \left(-a_{3,1} + a_{3,3} - a_{3,4} + a_{3,5} - a_{4,1} + a_{4,3} - a_{4,4} + a_{4,5}\right)\left(-b_{5,1} - b_{5,3} - b_{5,5}\right) \\
h_{72} &= \left(-a_{2,1} - a_{2,4} - a_{4,1} - a_{4,4}\right)\left(b_{4,1} + b_{4,2} + b_{4,3}\right) \\
h_{73} &= \left(a_{1,3} - a_{1,4} - a_{1,5} + a_{2,3} - a_{2,4} - a_{2,5}\right)\left(b_{1,1} + b_{1,2} - b_{1,4} + b_{2,4} + b_{5,2} - b_{5,4}\right) \\
h_{74} &= \left(a_{2,1} - a_{2,3} + a_{2,4} - a_{3,1} + a_{3,3} - a_{3,4}\right)\left(b_{4,1} + b_{4,2} + b_{4,5}\right) \\
h_{75} &= -\left(a_{1,2} + a_{1,4} - a_{2,2} - a_{2,5} - a_{3,1} + a_{3,2} + a_{3,4} + a_{3,5} - a_{4,1} + a_{4,2}\right) b_{2,5} \\
h_{76} &= \left(a_{1,3} + a_{3,3}\right)\left(-b_{1,1} + b_{1,4} - b_{1,5} + b_{2,4} + b_{3,4} - b_{3,5}\right) \\
c_{1,1} &= -h_{10} + h_{12} + h_{14} - h_{15} - h_{16} + h_{53} + h_5 - h_{66} - h_7 \\
c_{2,1} &= h_{10} + h_{11} - h_{12} + h_{13} + h_{15} + h_{16} - h_{17} - h_{44} + h_{51}
\end{aligned}$$

Algoritmo Descubierto por AlphaTensor

$$\begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & a_{1,5} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} & a_{2,5} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} & a_{3,5} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} & a_{4,5} \end{pmatrix} \begin{pmatrix} b_{1,1} & b_{1,2} & b_{1,3} & b_{1,4} & b_{1,5} \\ b_{2,1} & b_{2,2} & b_{2,3} & b_{2,4} & b_{2,5} \\ b_{3,1} & b_{3,2} & b_{3,3} & b_{3,4} & b_{3,5} \\ b_{4,1} & b_{4,2} & b_{4,3} & b_{4,4} & b_{4,5} \\ b_{5,1} & b_{5,2} & b_{5,3} & b_{5,4} & b_{5,5} \end{pmatrix} = \begin{pmatrix} c_{1,1} & c_{1,2} & c_{1,3} & c_{1,4} & c_{1,5} \\ c_{2,1} & c_{2,2} & c_{2,3} & c_{2,4} & c_{2,5} \\ c_{3,1} & c_{3,2} & c_{3,3} & c_{3,4} & c_{3,5} \\ c_{4,1} & c_{4,2} & c_{4,3} & c_{4,4} & c_{4,5} \end{pmatrix}$$

$$\begin{aligned}
h_{1} &= a_{3,2}\left(-b_{2,1}-b_{2,5}-b_{3,1}\right)\\
h_{2} &= \left(a_{2,2}+a_{2,5}-a_{3,5}\right)\left(-b_{2,5}-b_{5,1}\right)\\
h_{3} &= \left(-a_{3,1}-a_{4,1}+a_{4,2}\right)\left(-b_{1,1}+b_{2,5}\right)\\
h_{4} &= \left(a_{1,2}+a_{1,4}+a_{3,4}\right)\left(-b_{2,5}-b_{4,1}\right)\\
h_{5} &= \left(a_{1,5}+a_{2,2}+a_{2,5}\right)\left(-b_{2,4}+b_{5,1}\right)\\
h_{6} &= \left(-a_{2,2}-a_{2,5}-a_{4,5}\right)\left(b_{2,3}+b_{5,1}\right)\\
h_{7} &= \left(-a_{1,1}+a_{4,1}-a_{4,2}\right)\left(b_{1,1}+b_{2,4}\right)\\
h_{8} &= \left(a_{3,2}-a_{3,3}-a_{4,3}\right)\left(-b_{2,3}+b_{3,1}\right)\\
h_{9} &= \left(-a_{1,2}-a_{1,4}+a_{4,4}\right)\left(b_{2,3}+b_{4,1}\right)\\
h_{10} &= \left(a_{2,2}+a_{2,5}\right)b_{5,1}\\
h_{11} &= \left(-a_{2,1}-a_{4,1}+a_{4,2}\right)\left(-b_{1,1}+b_{2,2}\right)\\
h_{12} &= \left(a_{4,1}-a_{4,2}\right)b_{1,1}\\
h_{13} &= \left(a_{1,2}+a_{1,4}+a_{2,4}\right)\left(b_{2,2}+b_{4,1}\right)\\
h_{14} &= \left(a_{1,3}-a_{3,2}+a_{3,3}\right)\left(b_{2,4}+b_{3,1}\right)\\
h_{15} &= \left(-a_{1,2}-a_{1,4}\right)b_{4,1}\\
h_{16} &= \left(-a_{3,2}+a_{3,3}\right)b_{3,1}\\
h_{17} &= \left(a_{1,2}+a_{1,4}-a_{2,1}+a_{2,2}-a_{2,3}+a_{2,4}-a_{3,2}+a_{3,3}-a_{4,1}+a_{4,2}\right)b_{2,2}\\
h_{18} &= a_{2,1}\left(b_{1,1}+b_{1,2}+b_{5,2}\right)\\
h_{19} &= -a_{2,3}\left(b_{3,1}+b_{3,2}+b_{5,2}\right)\\
h_{20} &= \left(-a_{1,5}+a_{2,1}+a_{2,3}-a_{2,5}\right)\left(-b_{1,1}-b_{1,2}+b_{1,4}-b_{5,2}\right)\\
h_{21} &= \left(a_{2,1}+a_{2,3}-a_{2,5}\right)b_{5,2}\\
h_{22} &= \left(a_{1,3}-a_{1,4}-a_{2,4}\right)\left(b_{1,1}+b_{1,2}-b_{1,4}-b_{3,1}-b_{3,2}+b_{3,4}+b_{4,4}\right)\\
h_{23} &= a_{1,3}\left(-b_{3,1}+b_{3,4}+b_{4,4}\right)\\
h_{24} &= a_{1,5}\left(-b_{4,4}-b_{5,1}+b_{5,4}\right)\\
h_{25} &= -a_{1,1}\left(b_{1,1}-b_{1,4}\right)\\
h_{26} &= \left(-a_{1,3}+a_{1,4}+a_{1,5}\right)b_{4,4}\\
h_{27} &= \left(a_{1,3}-a_{3,1}+a_{3,3}\right)\left(b_{1,1}-b_{1,4}+b_{1,5}+b_{3,5}\right)\\
&\vdots
\end{aligned}$$

$$\begin{aligned}
h_{51} &= a_{2,2}\left(b_{2,1}+b_{2,2}-b_{5,1}\right)\\
h_{52} &= a_{4,2}\left(b_{1,1}+b_{2,1}+b_{2,3}\right)\\
h_{53} &= -a_{1,2}\left(-b_{2,1}+b_{2,4}+b_{4,1}\right)\\
h_{54} &= \left(a_{1,2}+a_{1,4}-a_{2,2}-a_{2,5}-a_{3,2}+a_{3,3}-a_{4,2}+a_{4,3}-a_{4,4}-a_{4,5}\right)b_{2,3}\\
h_{55} &= \left(a_{1,4}-a_{4,4}\right)\left(-b_{2,3}+b_{3,1}+b_{3,3}-b_{3,4}+b_{4,3}-b_{4,4}\right)\\
h_{56} &= \left(a_{1,1}-a_{1,5}-a_{4,1}+a_{4,5}\right)\left(b_{3,1}+b_{3,3}-b_{3,4}+b_{5,1}+b_{5,3}-b_{5,4}\right)\\
h_{57} &= \left(-a_{3,1}-a_{4,1}\right)\left(-b_{1,3}-b_{1,5}-b_{2,5}-b_{5,1}-b_{5,3}-b_{5,5}\right)\\
h_{58} &= \left(-a_{1,4}-a_{1,5}-a_{3,4}-a_{3,5}\right)\left(-b_{5,1}+b_{5,4}-b_{5,5}\right)\\
h_{59} &= \left(-a_{3,3}+a_{3,4}-a_{4,3}+a_{4,4}\right)\left(b_{4,1}+b_{4,3}+b_{4,5}+b_{5,1}+b_{5,3}+b_{5,5}\right)\\
h_{60} &= \left(a_{2,5}+a_{4,5}\right)\left(b_{2,3}-b_{3,1}-b_{3,2}-b_{3,3}-b_{5,2}-b_{5,3}\right)\\
h_{61} &= \left(a_{1,4}+a_{3,4}\right)\left(b_{1,1}-b_{1,4}+b_{1,5}-b_{2,5}-b_{4,4}+b_{4,5}-b_{5,1}+b_{5,4}-b_{5,5}\right)\\
h_{62} &= \left(a_{2,1}+a_{4,1}\right)\left(b_{1,2}+b_{1,3}+b_{2,2}-b_{4,1}-b_{4,2}-b_{4,3}\right)\\
h_{63} &= \left(-a_{3,3}-a_{4,3}\right)\left(-b_{2,3}-b_{3,3}-b_{3,5}-b_{4,1}-b_{4,3}-b_{4,5}\right)\\
h_{64} &= \left(a_{1,1}-a_{1,3}-a_{1,4}+a_{3,1}-a_{3,3}-a_{3,4}\right)\left(b_{1,1}-b_{1,4}+b_{1,5}\right)\\
h_{65} &= \left(-a_{1,1}+a_{4,1}\right)\left(-b_{1,3}+b_{1,4}+b_{2,4}-b_{5,1}-b_{5,3}+b_{5,4}\right)\\
h_{66} &= \left(a_{1,1}-a_{1,2}+a_{1,3}-a_{1,5}-a_{2,2}-a_{2,5}-a_{3,2}+a_{3,3}-a_{4,1}+a_{4,2}\right)b_{2,4}\\
h_{67} &= \left(a_{2,5}-a_{3,5}\right)\left(b_{1,1}+b_{1,2}+b_{1,5}-b_{2,5}-b_{4,1}-b_{4,2}-b_{4,5}+b_{5,2}+b_{5,5}\right)\\
h_{68} &= \left(a_{1,1}+a_{1,3}-a_{1,4}-a_{1,5}-a_{4,1}-a_{4,3}+a_{4,4}+a_{4,5}\right)\left(-b_{3,1}-b_{3,3}+b_{3,4}\right)\\
h_{69} &= \left(-a_{1,3}+a_{1,4}-a_{2,3}+a_{2,4}\right)\left(-b_{2,4}-b_{3,1}-b_{3,2}+b_{3,4}-b_{5,2}+b_{5,4}\right)\\
h_{70} &= \left(a_{2,3}-a_{2,5}+a_{4,3}-a_{4,5}\right)\left(-b_{3,1}-b_{3,2}-b_{3,3}\right)\\
h_{71} &= \left(-a_{3,1}+a_{3,3}-a_{3,4}+a_{3,5}-a_{4,1}+a_{4,3}-a_{4,4}+a_{4,5}\right)\left(-b_{5,1}-b_{5,3}-b_{5,5}\right)\\
h_{72} &= \left(-a_{2,1}-a_{2,4}-a_{4,1}-a_{4,4}\right)\left(b_{4,1}+b_{4,2}+b_{4,3}\right)\\
h_{73} &= \left(a_{1,3}-a_{1,4}-a_{1,5}+a_{2,3}-a_{2,4}-a_{2,5}\right)\left(b_{1,1}+b_{1,2}-b_{1,4}+b_{2,4}+b_{5,2}-b_{5,4}\right)\\
h_{74} &= \left(a_{2,1}-a_{2,3}+a_{2,4}-a_{3,1}+a_{3,3}-a_{3,4}\right)\left(b_{4,1}+b_{4,2}+b_{4,5}\right)\\
h_{75} &= -\left(a_{1,2}+a_{1,4}-a_{2,2}-a_{2,5}-a_{3,1}+a_{3,2}+a_{3,4}+a_{3,5}-a_{4,1}+a_{4,2}\right)b_{2,5}\\
h_{76} &= \left(a_{1,3}+a_{3,3}\right)\left(-b_{1,1}+b_{1,4}-b_{1,5}+b_{2,4}+b_{3,4}-b_{3,5}\right)\\
c_{1,1} &= -h_{10}+h_{12}+h_{14}-h_{15}-h_{16}+h_{53}+h_{5}-h_{66}-h_{7}\\
&\vdots
\end{aligned}$$

Algoritmo de Strassen Paso a Paso

$$\begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \times \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$

$$h_1 = (a_{11} + a_{22})\ (b_{11} + b_{22})$$
$$h_2 = (a_{21} + a_{22})\ b_{11}$$
$$h_3 = a_{11}\ (b_{12} - b_{22})$$
$$h_4 = a_{22}\ (-b_{11} + b_{21})$$
$$h_5 = (a_{11} + a_{12})\ b_{22}$$
$$h_6 = (-a_{11} + a_{21})\ (b_{11} + b_{12})$$
$$h_7 = (a_{12} - a_{22})\ (b_{21} + b_{22})$$

$$c_{11} = h_1 + h_4 - h_5 + h_7$$
$$c_{12} = h_3 + h_5$$
$$c_{21} = h_2 + h_4$$
$$c_{22} = h_1 - h_2 + h_3 + h_6$$

Algoritmo de Strassen Paso a Paso

$$\begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \times \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$

$$\begin{aligned}
h_1 &= (a_{11} + a_{22})\,(b_{11} + b_{22}) \\
h_2 &= (a_{21} + a_{22})\,b_{11} \\
h_3 &= a_{11}\,(b_{12} - b_{22}) \\
h_4 &= a_{22}\,(-b_{11} + b_{21}) \\
h_5 &= (a_{11} + a_{12})\,b_{22} \\
h_6 &= (-a_{11} + a_{21})\,(b_{11} + b_{12}) \\
h_7 &= (a_{12} - a_{22})\,(b_{21} + b_{22})
\end{aligned}$$

$$\begin{aligned}
c_{11} &= h_1 + h_4 - h_5 + h_7 \\
c_{12} &= h_3 + h_5 \\
c_{21} &= h_2 + h_4 \\
c_{22} &= h_1 - h_2 + h_3 + h_6
\end{aligned}$$

Algoritmo de Strassen Paso a Paso

$$\begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \times \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$

$$h_1 = (a_{11} + a_{22})\,(b_{11} + b_{22}) \qquad (1 \cdot a_{11} + 0 \cdot a_{12} + 0 \cdot a_{21} + 1 \cdot a_{22})(1 \cdot b_{11} + 0 \cdot b_{12} + 0 \cdot b_{21} + 1 \cdot b_{22})$$

$$h_2 = (a_{21} + a_{22})\,b_{11}$$

$$h_3 = a_{11}\,(b_{12} - b_{22})$$

$$h_4 = a_{22}\,(-b_{11} + b_{21})$$

$$h_5 = (a_{11} + a_{12})\,b_{22}$$

$$h_6 = (-a_{11} + a_{21})\,(b_{11} + b_{12})$$

$$h_7 = (a_{12} - a_{22})\,(b_{21} + b_{22})$$

$$c_{11} = h_1 + h_4 - h_5 + h_7$$

$$c_{12} = h_3 + h_5$$

$$c_{21} = h_2 + h_4$$

$$c_{22} = h_1 - h_2 + h_3 + h_6$$

Algoritmo de Strassen Paso a Paso

$$\begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \times \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$

$$h_1 = (a_{11} + a_{22})\,(b_{11} + b_{22}) \qquad (1 \cdot a_{11} + 0 \cdot a_{12} + 0 \cdot a_{21} + 1 \cdot a_{22})(1 \cdot b_{11} + 0 \cdot b_{12} + 0 \cdot b_{21} + 1 \cdot b_{22})$$

$$h_2 = (a_{21} + a_{22})\,b_{11}$$

$$h_3 = a_{11}\,(b_{12} - b_{22})$$

$$h_4 = a_{22}\,(-b_{11} + b_{21})$$

$$h_5 = (a_{11} + a_{12})\,b_{22}$$

$$h_6 = (-a_{11} + a_{21})\,(b_{11} + b_{12})$$

$$h_7 = (a_{12} - a_{22})\,(b_{21} + b_{22})$$

$$c_{11} = h_1 + h_4 - h_5 + h_7 \qquad 1$$

$$c_{12} = h_3 + h_5 \qquad 0$$

$$c_{21} = h_2 + h_4 \qquad 0$$

$$c_{22} = h_1 - h_2 + h_3 + h_6 \qquad 1$$

Algoritmo de Strassen Paso a Paso

$$\begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \times \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$

$$h_1 = (a_{11} + a_{22})\ (b_{11} + b_{22})$$
$$h_2 = (a_{21} + a_{22})\ b_{11}$$
$$h_3 = a_{11}\ (b_{12} - b_{22})$$
$$h_4 = a_{22}\ (-b_{11} + b_{21})$$
$$h_5 = (a_{11} + a_{12})\ b_{22}$$
$$h_6 = (-a_{11} + a_{21})\ (b_{11} + b_{12})$$
$$h_7 = (a_{12} - a_{22})\ (b_{21} + b_{22})$$

$$\begin{pmatrix} 1 \cdot a_{11} + 0 \cdot a_{12} + 0 \cdot a_{21} + 1 \cdot a_{22} \\ 0 \cdot a_{11} + 0 \cdot a_{12} + 1 \cdot a_{21} + 1 \cdot a_{22} \end{pmatrix} \begin{pmatrix} 1 \cdot b_{11} + 0 \cdot b_{12} + 0 \cdot b_{21} + 1 \cdot b_{22} \\ 1 \cdot b_{11} + 0 \cdot b_{12} + 0 \cdot b_{21} + 0 \cdot b_{22} \end{pmatrix}$$

$$c_{11} = h_1 + h_4 - h_5 + h_7$$
$$c_{12} = h_3 + h_5$$
$$c_{21} = h_2 + h_4$$
$$c_{22} = h_1 - h_2 + h_3 + h_6$$

1	0
0	0
0	1
1	-1

Algoritmo de Strassen Paso a Paso

$$\begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \times \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$

$$h_1 = (a_{11} + a_{22})\ (b_{11} + b_{22}) \qquad (1 \cdot a_{11} + 0 \cdot a_{12} + 0 \cdot a_{21} + 1 \cdot a_{22})(1 \cdot b_{11} + 0 \cdot b_{12} + 0 \cdot b_{21} + 1 \cdot b_{22})$$
$$h_2 = (a_{21} + a_{22})\ b_{11} \qquad (0 \cdot a_{11} + 0 \cdot a_{12} + 1 \cdot a_{21} + 1 \cdot a_{22})(1 \cdot b_{11} + 0 \cdot b_{12} + 0 \cdot b_{21} + 0 \cdot b_{22})$$
$$h_3 = a_{11}\ (b_{12} - b_{22}) \qquad \vdots$$
$$h_4 = a_{22}\ (-b_{11} + b_{21})$$
$$h_5 = (a_{11} + a_{12})\ b_{22}$$
$$h_6 = (-a_{11} + a_{21})\ (b_{11} + b_{12})$$
$$h_7 = (a_{12} - a_{22})\ (b_{21} + b_{22}) \qquad (0 \cdot a_{11} + 1 \cdot a_{12} + 0 \cdot a_{21} + (-1) a_{22})(0 \cdot b_{11} + 0 \cdot b_{12} + 1 \cdot b_{21} + 1 \cdot b_{22})$$

$$c_{11} = h_1 + h_4 - h_5 + h_7 \qquad \begin{matrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \\ 1 & -1 \end{matrix} \quad \cdots \quad \begin{matrix} 1 \\ 0 \\ 0 \\ 0 \end{matrix}$$
$$c_{12} = h_3 + h_5$$
$$c_{21} = h_2 + h_4$$
$$c_{22} = h_1 - h_2 + h_3 + h_6$$

TensorGame

Estado:

Jugada:

Actualización de estado:

Repetir hasta alcanzar la matriz cero:

TensorGame

Estado: $$S_t = \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{bmatrix}$$

Jugada:

Actualización de estado:

Repetir hasta alcanzar la matriz cero:

TensorGame

Estado:

$$S_t = \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{bmatrix}$$

Jugada:

1	0	0	1	1	0	0	1	1	0	0	1
a_{11}	a_{12}	a_{21}	a_{22}	b_{11}	b_{12}	b_{21}	b_{22}	c_{11}	c_{12}	c_{21}	c_{22}

Actualización de estado:

Repetir hasta alcanzar la matriz cero:

TensorGame

Estado:

$$S_t = \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{bmatrix}$$

Jugada:

1	0	0	1	1	0	0	1	1	0	0	1
a_{11}	a_{12}	a_{21}	a_{22}	b_{11}	b_{12}	b_{21}	b_{22}	c_{11}	c_{12}	c_{21}	c_{22}

Actualización de estado:

$$S_{t+1} = S_t - \begin{bmatrix} (a_{11} + a_{22})(b_{11} + b_{22}) & 0 \\ 0 & (a_{11} + a_{22})(b_{11} + b_{22}) \end{bmatrix}$$

Repetir hasta alcanzar la matriz cero:

TensorGame

Estado:

$$S_t = \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{bmatrix}$$

Jugada:

1	0	0	1	1	0	0	1	1	0	0	1
a_{11}	a_{12}	a_{21}	a_{22}	b_{11}	b_{12}	b_{21}	b_{22}	c_{11}	c_{12}	c_{21}	c_{22}

Actualización de estado:

$$S_{t+1} = S_t - \begin{bmatrix} (a_{11} + a_{22})(b_{11} + b_{22}) & 0 \\ 0 & (a_{11} + a_{22})(b_{11} + b_{22}) \end{bmatrix}$$

Repetir hasta alcanzar la matriz cero:

$$\begin{bmatrix} -a_{11}b_{22} + a_{12}b_{21} - a_{22}b_{11} - a_{22}b_{22} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & -a_{11}b_{11} - a_{11}b_{22} + a_{21}b_{12} - a_{22}b_{11} \end{bmatrix}$$

TensorGame

Estado:

$$S_t = \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{bmatrix}$$

Jugada:

1	0	0	1	1	0	0	1	1	0	0	1
a_{11}	a_{12}	a_{21}	a_{22}	b_{11}	b_{12}	b_{21}	b_{22}	c_{11}	c_{12}	c_{21}	c_{22}

Actualización de estado:

$$S_{t+1} = S_t - \begin{bmatrix} (a_{11} + a_{22})(b_{11} + b_{22}) & 0 \\ 0 & (a_{11} + a_{22})(b_{11} + b_{22}) \end{bmatrix}$$

Repetir hasta alcanzar la matriz cero:

$$\begin{bmatrix} -a_{11}b_{22} + a_{12}b_{21} - a_{22}b_{11} - a_{22}b_{22} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & -a_{11}b_{11} - a_{11}b_{22} + a_{21}b_{12} - a_{22}b_{11} \end{bmatrix}$$

TensorGame

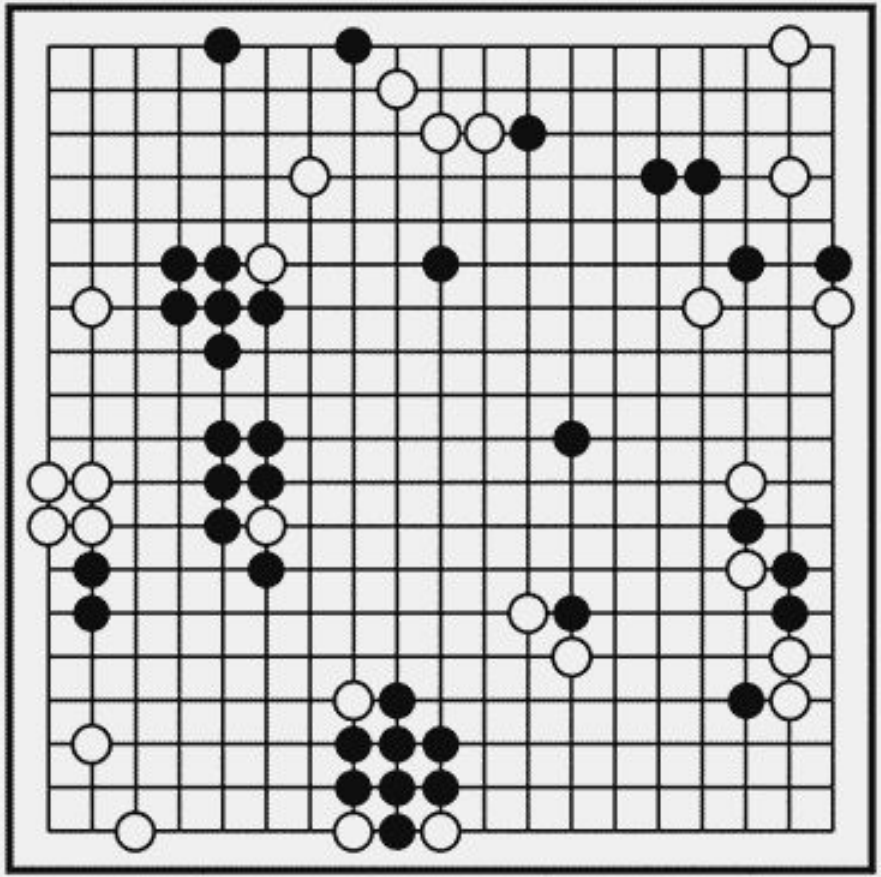
El juego **acaba** en el paso *t* si se ha alcanzado el estado cero ($S_t = 0$)

Por construcción, la secuencia de jugadas describe un algoritmo válido

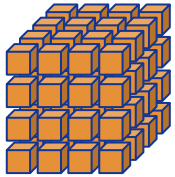
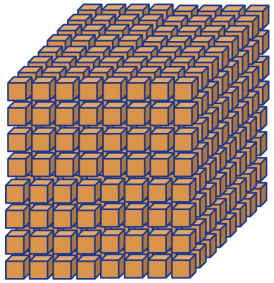
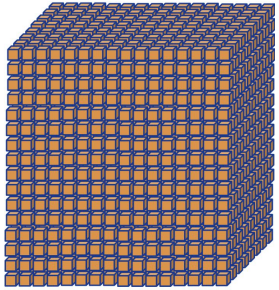
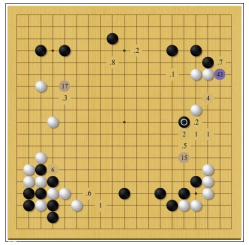
Puntuación: -1 por cada jugada

alta puntuación ⇔ pocas jugadas ⇔ algoritmo eficiente

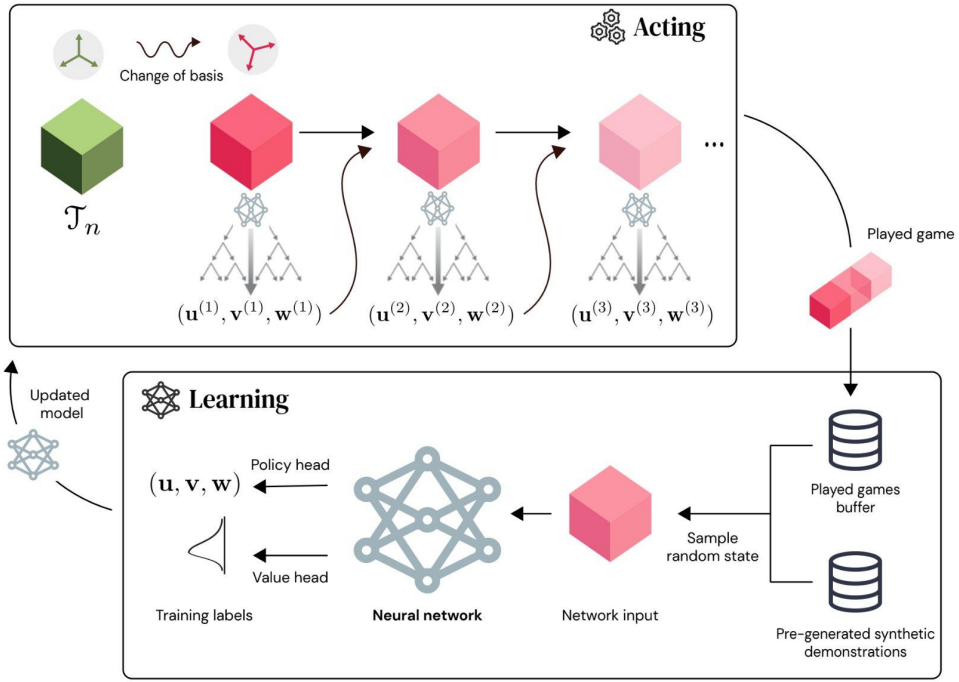
Idea: Usar aprendizaje por refuerzo para entrenar a un agente que aprenda a jugar a TensorGame



Dificultades de TensorGame

	2x2	3x3	4x4	Go
				
Número de jugadas	10^{8}	10^{18}	10^{33}	362

AlphaTensor



Se basa en AlphaZero

Dos **componentes** principales:

→ Actores (juegan continuamente)

→ Aprendiz (actualiza una red neuronal)

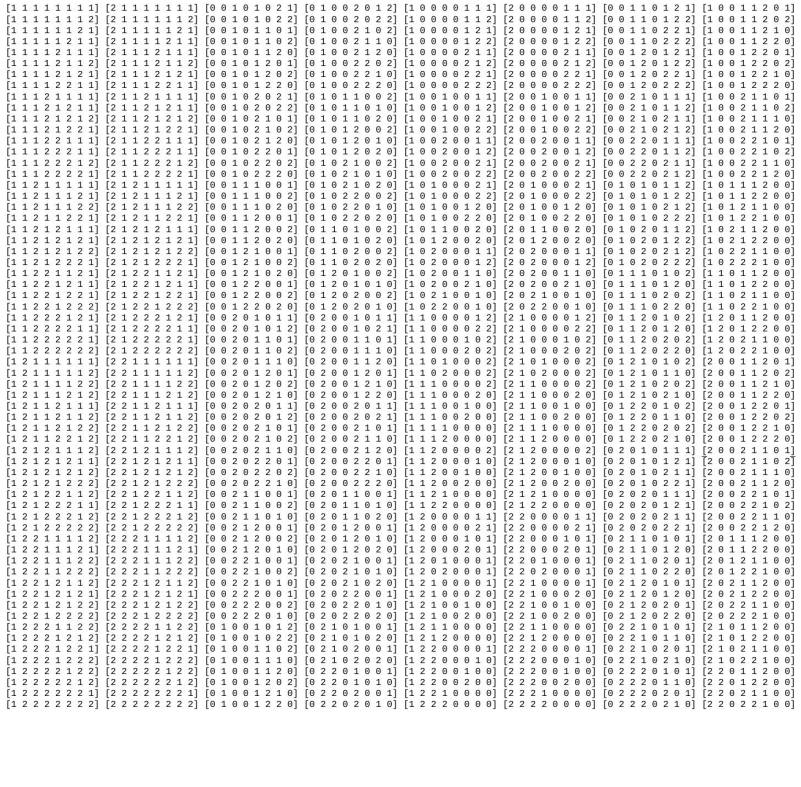
Índice

Espacio de Soluciones vs. Espacio de Código

Problema: Encuentra 512 vectores en $\mathbb{Z}_3^8$ de manera que no haya tres cuya suma (módulo 3) sea cero

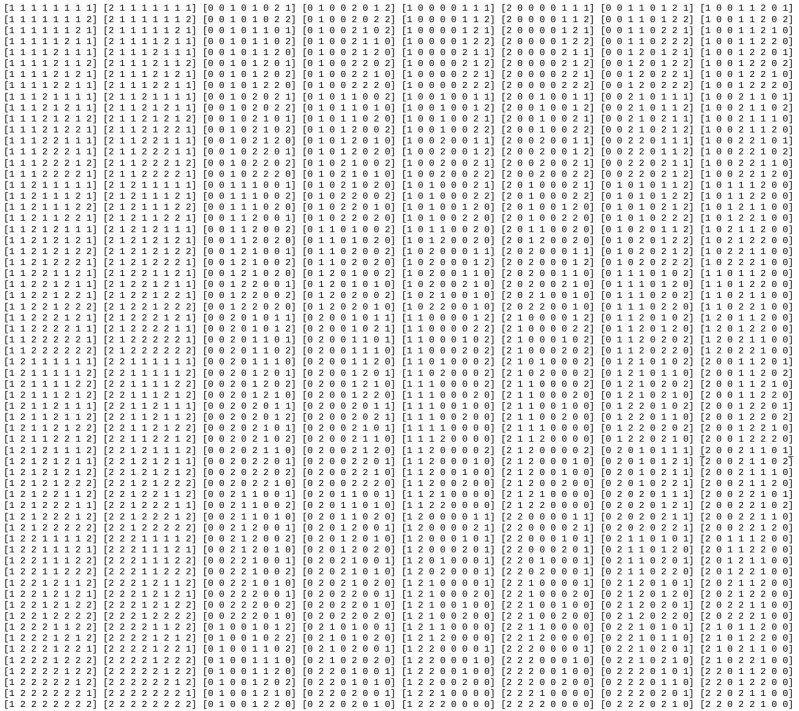
Espacio de Soluciones vs. Espacio de Código

Problema: Encuentra 512 vectores en $\mathbb{Z}_3^8$ de manera que no haya tres cuya suma (módulo 3) sea cero



Espacio de Soluciones vs. Espacio de Código

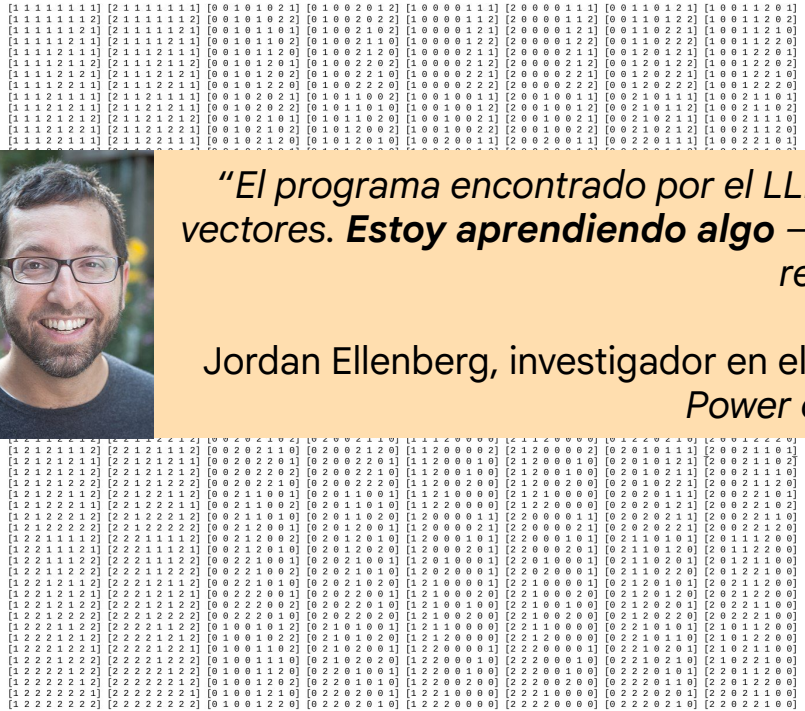
Problema: Encuentra 512 vectores en $\mathbb{Z}_3^8$ de manera que no haya tres cuya suma (módulo 3) sea cero



```python
def get_capset(n: int) -> CapSet:
  """Returns a 512-cap in AG(8, 3)."""
  V = np.array(list(itertools.product(range(3), repeat=n)), dtype=np.int32)
  reflections = lambda v: sum(1 for i in range(1, n // 2) if v[i] == v[-i])

  # First we list 128 weight-8 vectors with >= 2 reflections.
  weight8_points = [v for v in V
                    if np.count_nonzero(v) == 8  # Weight is 8.
                    and reflections(v) >= 2]  # At least 2 reflections.

  # Then we list 256 weight-4 vectors with allowed support and <= 1 reflections.
  allowed_supports = [
      (0, 1, 2, 3), (0, 1, 2, 5), (0, 1, 2, 7), (0, 1, 2, 6), (0, 1, 3, 7),
      (0, 1, 6, 7), (0, 3, 6, 7), (0, 5, 6, 7), (0, 1, 5, 7), (1, 3, 4, 6),
      (1, 4, 5, 6), (0, 2, 3, 6), (2, 3, 4, 7), (2, 4, 5, 7), (0, 2, 6, 7),
      (0, 2, 5, 6), (1, 2, 4, 7), (1, 2, 4, 6), (1, 3, 4, 7), (1, 4, 6, 7),
      (1, 4, 5, 7), (2, 3, 4, 6), (2, 4, 6, 7), (2, 4, 5, 6),
  ]
  weight4_points = [
      v for v in V
      if np.count_nonzero(v) == 4  # Weight is 4.
      and tuple(i for i in range(n) if v[i] != 0) in allowed_supports
      and reflections(v) <= 1]  # At most 1 reflection.

  # Finally we add 128 weight-5 vectors with <= 1 reflections.
  allowed_zeros = [(0, 4, 7), (0, 2, 4), (0, 1, 4), (0, 4, 6),
                   (1, 2, 6), (2, 6, 7), (1, 2, 7), (1, 6, 7)]
  weight5_points = [
      v for v in V
      if np.count_nonzero(v) == 5  # Weight is 4.
      and tuple(i for i in range(n) if v[i] == 0) in allowed_zeros
      and reflections(v) <= 1  # At most 1 reflection.
      and (v[1] * v[7]) % 3 != 1 and (v[2] * v[6]) % 3 != 1]  # Mod conditions.

  return weight8_points + weight4_points + weight5_points
```

Espacio de Soluciones vs. Espacio de Código

Problema: Encuentra 512 vectores en $\mathbb{Z}_3^8$ de manera que no haya tres cuya suma (módulo 3) sea cero



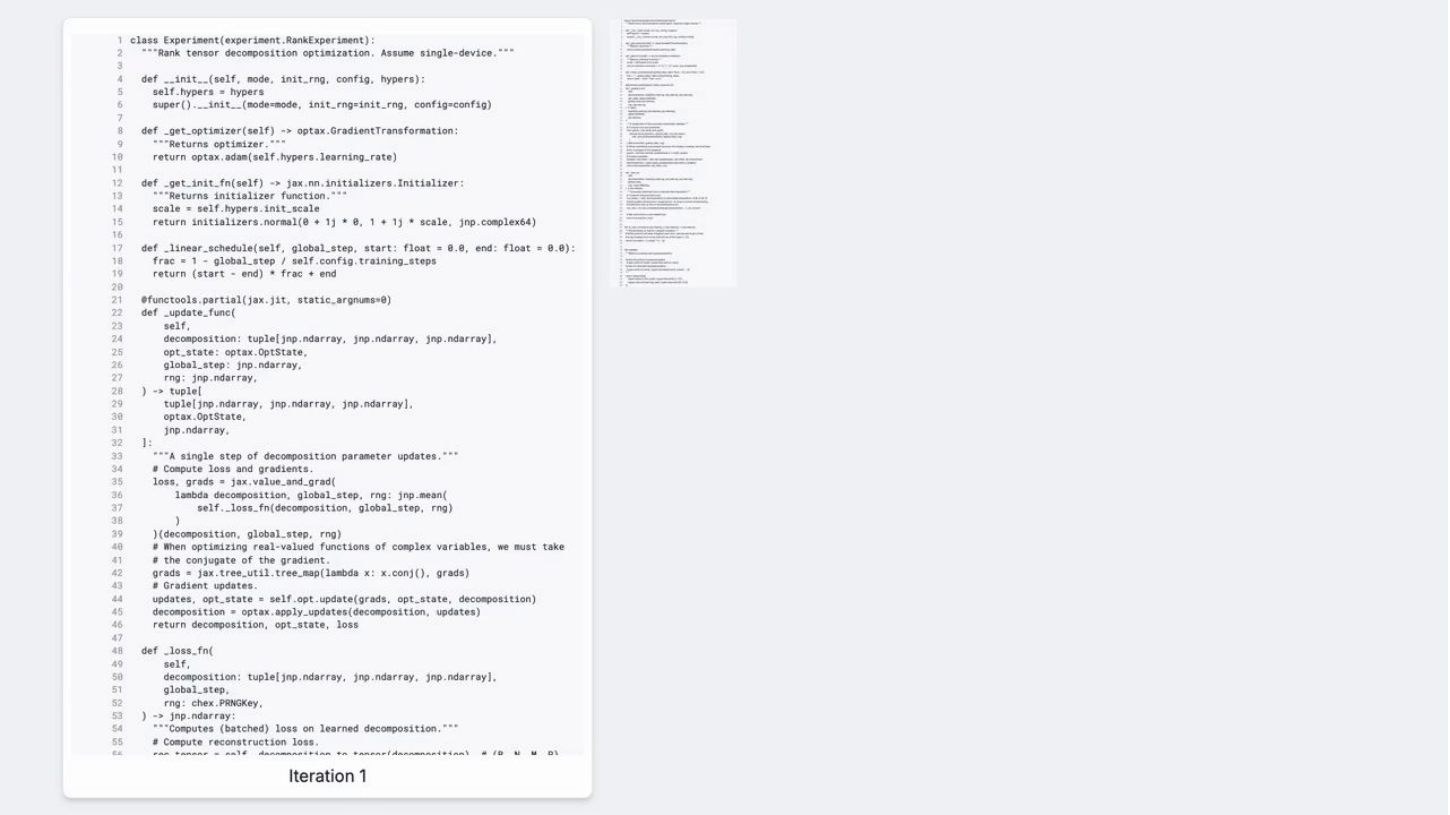
```
def get_capset(n: int) -> CapSet:
  """Returns a 512-cap in AG(8, 3)."""
  V = np.array(list(itertools.product(range(3), repeat=n)), dtype=np.int32)
  reflections = lambda v: sum(1 for i in range(1, n // 2) if v[i] == v[-i])

  # First we list 128 weight-8 vectors with >= 2 reflections.
  weight8_points = [v for v in V
      and tuple(i for i in range(n)) if v[i] == 0) in allowed_supports
      and reflections(v) <= 1]  # At most 1 reflection.

  # Finally we add 128 weight-5 vectors with <= 1 reflections.
  allowed_zeros = [(0, 4, 7), (0, 2, 4), (0, 1, 4), (0, 4, 6),
                   (1, 2, 6), (2, 6, 7), (1, 2, 7), (1, 6, 7)]
  weight5_points = [
      v for v in V
      if np.count_nonzero(v) == 5  # Weight is 4.
      and tuple(i for i in range(n)) if v[i] == 0) in allowed_zeros
      and reflections(v) <= 1  # At most 1 reflection.
      and (v[1] * v[7]) % 3 != 1 and (v[2] * v[6]) % 3 != 1]  # Mod conditions.

  return weight8_points + weight4_points + weight5_points
```


*"El programa encontrado por el LLM es conceptualmente más rico que una mera lista de vectores. **Estoy aprendiendo algo** — por ejemplo, la idea de clasificar en base al número de reflexiones es nueva."*

Jordan Ellenberg, investigador en el área y autor del bestseller *"How Not to be Wrong: The Power of Mathematical Thinking"*

AlphaEvolve

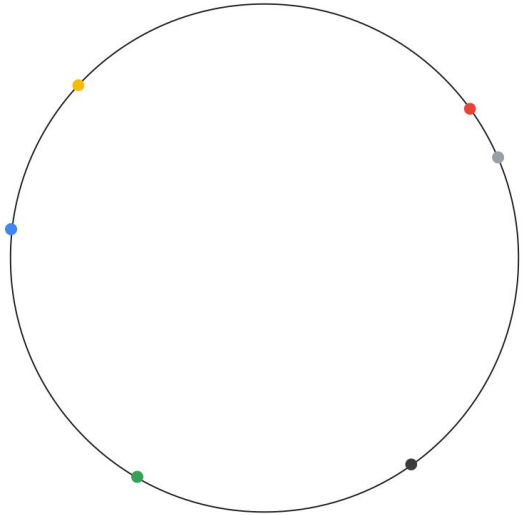
Un **agente de código** que usa un algoritmo **evolutivo** para optimizar y descubrir algoritmos



Ingredientes de AlphaEvolve

LLMs

🌸 **creatividad** (generación de código)



Evaluadores

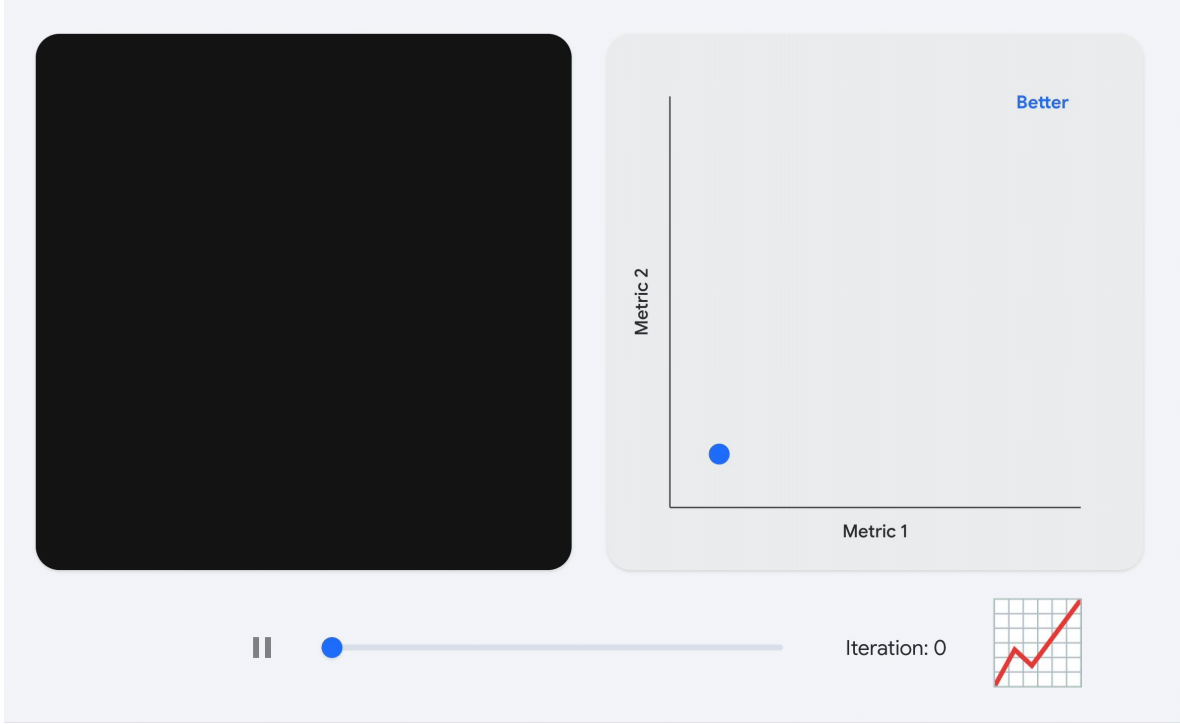
🤖 **verificación** rigurosa

Evolución

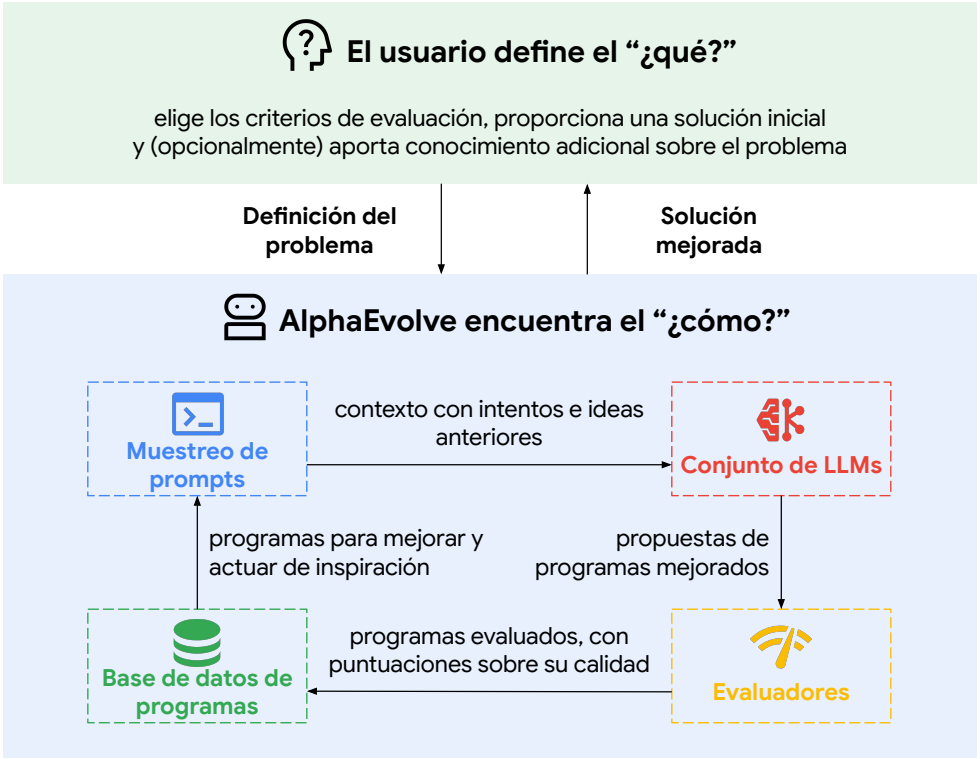
📈 mejora **iterativa**

Ingredientes de AlphaEvolve

Thanks, Gemini Canvas!



Implementación de AlphaEvolve



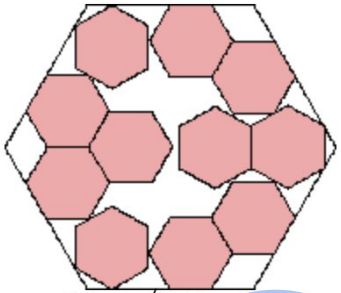
Ejemplo: Un Problema de Empaquetamiento

Problema: ¿Cuál es la dimensión del hexágono regular más pequeño que contenga 11 hexágonos regulares de lado unidad?

Ejemplo: Un Problema de Empaquetamiento

Problema: ¿Cuál es la dimensión del hexágono regular más pequeño que contenga 11 hexágonos regulares de lado unidad?

Estado del arte



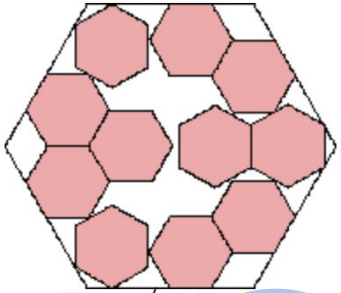
$s = 5(3 + \sqrt{3})/6 = 3.943+$

Construcción encontrada por Maurizio Morandi en 2015

Ejemplo: Un Problema de Empaquetamiento

Problema: ¿Cuál es la dimensión del hexágono regular más pequeño que contenga 11 hexágonos regulares de lado unidad?

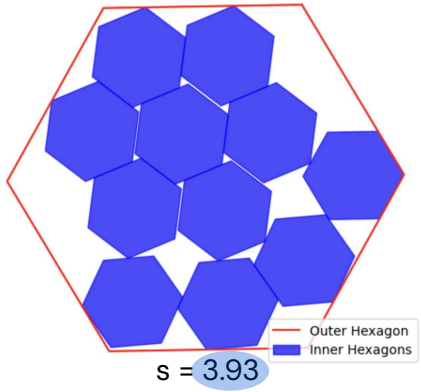
Estado del arte



$s = 5(3 + \sqrt{3})/6 = 3.943+$

Construcción encontrada por Maurizio Morandi en 2015

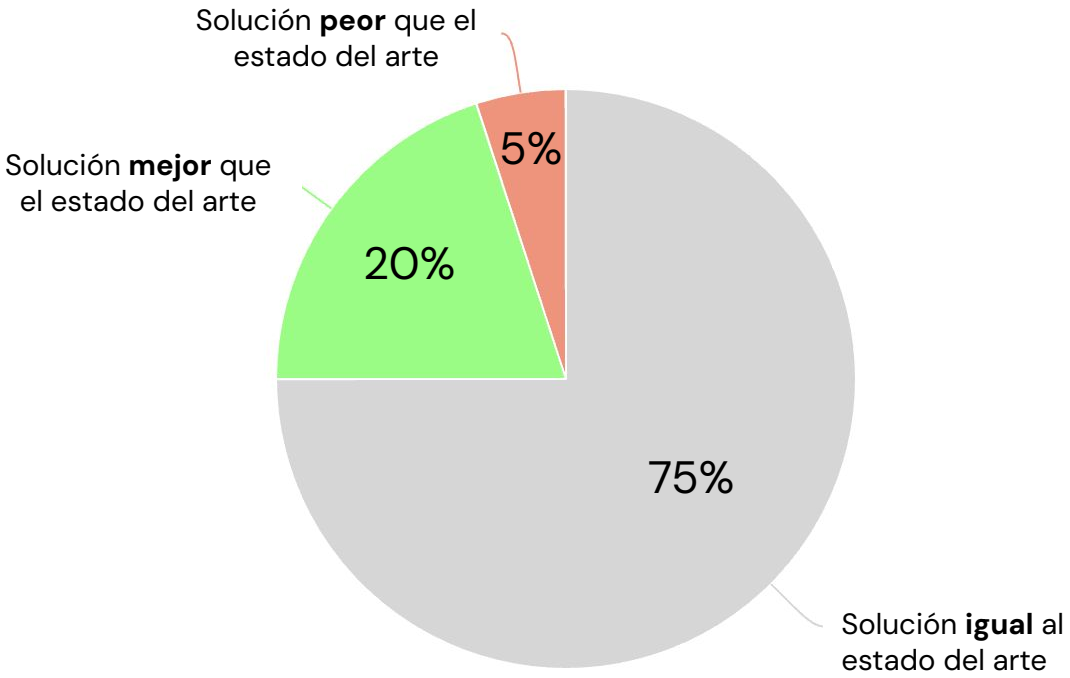
AlphaEvolve



s = 3.93

Resultados en Problemas Matemáticos

Aplicamos AlphaEvolve a un conjunto de 50+ problemas matemáticos en análisis, geometría y combinatoria



Resultados en Multiplicación de Matrices

Aplicamos AlphaEvolve para mejorar un algoritmo de descomposición de tensores (basado en descenso por gradiente)

```
@@ -45,9 +45,14 @@
   # EVOLVE-BLOCK-START
   def _get_optimizer(self) -> optax.GradientTransformation:
     """Returns optimizer."""
-    return optax.adam(self.hypers.learning_rate)
+    return optax.adamw(
+        self.hypers.learning_rate, weight_decay=self.hypers.weight_decay
+    )

   def _get_init_fn(self) -> jax.nn.initializers.Initializer:
     """Returns initializer function."""
-    return initializers.normal(0.0, self.hypers.init_scale, jnp.complex64)
+    # Initialize with a smaller scale to encourage finding low-rank solutions.
+    # Increase scale slightly for better exploration.
+    scale = self.hypers.init_scale
+    return initializers.normal(0 + 1j * 0, scale * 0.2, jnp.complex64)

@@ -80,6 +85,66 @@
     # Gradient updates.
     updates, opt_state = self.opt.update(grads, opt_state, decomposition)
     decomposition = optax.apply_updates(decomposition, updates)
+    # Add a small amount of gradient noise to help with exploration
+    rng, g_noise_rng = jax.random.split(rng)
+    decomposition = jax.tree_util.tree_map(
+        lambda x: x
+        + self.hypers.grad_noise_std * jax.random.normal(g_noise_rng, x.shape),
+        decomposition,
+    )
+
+    # Add noise to the decomposition parameters (exploration).
+    _, noise_rng = jax.random.split(rng)
+    noise_std = self._linear_schedule(
+        global_step, start=self.hypers.noise_std, end=0.0
+    )
+    decomposition = jax.tree_util.tree_map(
+        lambda x: x + noise_std * jax.random.normal(noise_rng, x.shape),
+        decomposition,
+    )
+
+    # Cyclical annealing for clipping threshold.
+    cycle_length = 2000  # Number of steps per cycle
+    cycle_progress = (
+        global_step % cycle_length
+    ) / cycle_length  # Normalized progress within the current cycle [0, 1)
+
+    # Map cycle progress to a sinusoidal curve. Ranges from 0 to 1.
+    clip_threshold_multiplier = (1 + jnp.cos(2 * jnp.pi * cycle_progress)) / 2
+
+    clip_threshold = self.hypers.clip_min + clip_threshold_multiplier * (
+        self.hypers.clip_max - self.hypers.clip_min
+    )
+
+    def soft_clip(x, threshold):
+      # Clipping the real and imaginary parts separately.
+      x_re = jnp.real(x)
+      x_im = jnp.imag(x)
+
+      x_re_clipped = jnp.where(
+          x_re > threshold, threshold + (x_re - threshold) * 0.1, x_re
+      )
+      x_re_clipped = jnp.where(
+          x_re_clipped < -threshold,
+          -threshold + (x_re_clipped + threshold) * 0.1,
+          x_re_clipped,
+      )
+
+      x_im_clipped = jnp.where(
+          x_im > threshold, threshold + (x_im - threshold) * 0.1, x_im
+      )
+      x_im_clipped = jnp.where(
+          x_im_clipped < -threshold,
+          -threshold + (x_im_clipped + threshold) * 0.1,
+          x_im_clipped,
+      )
+
+      return x_re_clipped + 1j * x_im_clipped
+
+    decomposition = jax.tree_util.tree_map(
+        lambda x: soft_clip(x, clip_threshold), decomposition
+    )

     return decomposition, opt_state, loss

   def _loss_fn(
@@ -91,13 +156,86 @@
     """Computes (batched) loss on learned decomposition."""
     # Compute reconstruction loss.
     rec_tensor = self._decomposition_to_tensor(decomposition)  # (B, N, M, P)

+    # Add noise to the target tensor (robustness).
+    rng, noise_rng = jax.random.split(rng)
+    target_noise = self.hypers.target_noise_std * jax.random.normal(
+        noise_rng, self.target_tensor.shape
+    )
```

Resultados en Multiplicación de Matrices

Aplicamos AlphaEvolve para mejorar un algoritmo de descomposición de tensores (basado en descenso por gradiente)

```
@@ -45,9 +45,14 @@
   # EVOLVE-BLOCK-START
   def _get_optimizer(self) -> optax.GradientTransformation:
     """Returns optimizer."""
-    return optax.adam(self.hypers.learning_rate)
+    return optax.adamw(
+        self.hypers.learning_rate, weight_decay=self.hypers.
    weight_decay
+    )

   def _get_init_fn(self) -> jax.nn.initializers.Initializer:
     """Returns initializer function."""
-    return initializers.normal(0.0, self.hypers.init_scale, jnp.
    complex64)
+    # Initialize with a smaller scale to encourage finding low-rank
    solutions.
+    # Increase scale slightly for better exploration.
+    scale = self.hypers.init_scale
+    return initializers.normal(0 + 1j * 0, scale * 0.2, jnp.complex64)
```

Resultados en Multiplicación de Matrices

Aplicamos AlphaEvolve para mejorar un algoritmo de descomposición de tensores (basado en descenso por gradiente)

```
@@ -91,13 +156,86 @@
     """Computes (batched) loss on learned decomposition."""
     # Compute reconstruction loss.
     rec_tensor = self._decomposition_to_tensor(decomposition)  # (B, N, M, P)
...
+    # Discretization loss (encourage entries to be multiples of 1/2 or integer).
+    def dist_to_half_ints(x):
...
+
+    def dist_to_ints(x):
...
+    discretization_loss = 0.0
+    for factor in decomposition:
+      discretization_loss += jnp.mean(dist_to_half_ints(factor))
+      discretization_loss += jnp.mean(dist_to_ints(factor))
+
+    discretization_loss /= (
+        len(decomposition) * 2
+    )  # average across all factors and loss components
+
+    discretization_weight = self._linear_schedule(
+        global_step, start=0.0, end=self.hypers.discretization_weight
+    )
+
+    # Cosine annealing for half-integer loss.
+    cycle_length = self.config.training_steps // 4  # Number of steps per cycle
+    cycle_progress = (
+        global_step % cycle_length
+    ) / cycle_length  # Normalized progress within the current cycle [0, 1)
+    half_int_multiplier = (1 + jnp.cos(jnp.pi * cycle_progress)) / 2
+    half_int_multiplier = (
+        1 - self.hypers.half_int_start
+    ) * half_int_multiplier + self.hypers.half_int_start
+
+    total_loss = (
+        rec_loss
+        + discretization_weight * discretization_loss * half_int_multiplier
+    )
...
```

Resultados en Multiplicación de Matrices

Aplicamos AlphaEvolve para mejorar un algoritmo de descomposición de tensores (basado en descenso por gradiente)

```
@@ -117,6 +255,18 @@
   return hyper.zipit([
-      hyper.uniform('init_scale', hyper.interval(0.2, 1.5)),
-      hyper.uniform('learning_rate', hyper.interval(0.05, 0.3)),
+      hyper.uniform('init_scale', hyper.interval(0.1, 1.0)),
+      hyper.uniform('learning_rate', hyper.interval(0.01, 0.2)),
+      hyper.uniform('discretization_weight', hyper.interval(0.0, 0.1)),
+      hyper.uniform('hallucination_prob', hyper.interval(0.0, 0.2)),
+      hyper.uniform('hallucination_scale', hyper.interval(0.0, 0.2)),
+      hyper.uniform('noise_std', hyper.interval(0.0, 0.01)),
+      hyper.uniform('target_noise_std', hyper.interval(0.0, 0.01)),
+      hyper.uniform('weight_decay', hyper.interval(0.00001, 0.001)),
+      hyper.uniform('clip_min', hyper.interval(0.0, 0.5)),
+      hyper.uniform('clip_max', hyper.interval(1.0, 3.0)),
+      hyper.uniform('large_value_penalty_weight', hyper.interval(0.0, 0.01)),
+      # Add noise to the gradient to aid in exploration.
+      hyper.uniform('grad_noise_std', hyper.interval(0.0, 0.001)),
+      hyper.uniform('half_int_start', hyper.interval(0.0, 1.0)),
   ])
 # EVOLVE-BLOCK-END
```

Resultados en Multiplicación de Matrices

Aplicamos AlphaEvolve para mejorar un algoritmo de descomposición de tensores (basado en descenso por gradiente)

$\langle m, n, p \rangle$	mejor #mult	AlphaEvolve
$\langle 2, 2, 2 \rangle$	7 [95]	7
$\langle 2, 2, 3 \rangle$	11 [93]	11
$\langle 2, 2, 4 \rangle$	14 [93]	14
$\langle 2, 2, 5 \rangle$	18 [93]	18
$\langle 2, 2, 6 \rangle$	21 [93]	21
$\langle 2, 2, 7 \rangle$	25 [93]	25
$\langle 2, 2, 8 \rangle$	28 [93]	28
$\langle 2, 2, 9 \rangle$	32 [93]	32
$\langle 2, 2, 10 \rangle$	35 [93]	35
$\langle 2, 2, 11 \rangle$	39 [93]	39
$\langle 2, 2, 12 \rangle$	42 [93]	42
$\langle 2, 2, 13 \rangle$	46 [93]	46
$\langle 2, 2, 14 \rangle$	49 [93]	49
$\langle 2, 2, 15 \rangle$	53 [93]	53
$\langle 2, 2, 16 \rangle$	56 [93]	56
$\langle 2, 3, 3 \rangle$	15 [93]	15
$\langle 2, 3, 4 \rangle$	20 [93]	20
$\langle 2, 3, 5 \rangle$	25 [93]	25

$\langle m, n, p \rangle$	mejor #mult	AlphaEvolve
$\langle 2, 3, 6 \rangle$	30 [93]	30
$\langle 2, 3, 7 \rangle$	35 [93]	35
$\langle 2, 3, 8 \rangle$	40 [93]	40
$\langle 2, 3, 9 \rangle$	45 [93]	45
$\langle 2, 3, 10 \rangle$	50 [93]	50
$\langle 2, 4, 4 \rangle$	26 [93]	26
$\langle 2, 4, 5 \rangle$	33 [42]	**32**
$\langle 2, 4, 6 \rangle$	39 [93]	39
$\langle 2, 4, 7 \rangle$	46 [93]	**45**
$\langle 2, 4, 8 \rangle$	52 [93]	**51**
$\langle 2, 5, 5 \rangle$	40 [93]	40
$\langle 2, 5, 6 \rangle$	48 [93]	**47**
$\langle 3, 3, 3 \rangle$	23 [52]	23
$\langle 3, 3, 4 \rangle$	29 [93]	29
$\langle 3, 3, 5 \rangle$	36 [93]	36
$\langle 3, 3, 6 \rangle$	40 [93]	40
$\langle 3, 3, 7 \rangle$	49 [93]	49
$\langle 3, 3, 8 \rangle$	55 [93]	55

$\langle m, n, p \rangle$	mejor #mult	AlphaEvolve
$\langle 3, 4, 4 \rangle$	38 [93]	38
$\langle 3, 4, 5 \rangle$	47 [26]	47
$\langle 3, 4, 6 \rangle$	56 [48]	**54**
$\langle 3, 4, 7 \rangle$	66 [91]	**63**
$\langle 3, 4, 8 \rangle$	75 [91]	**74**
$\langle 3, 5, 5 \rangle$	58 [91]	58
$\langle 3, 5, 6 \rangle$	70 [48]	**68**
$\langle 3, 5, 7 \rangle$	82 [91]	**80**
$\langle 4, 4, 4 \rangle$	49 [95]	**48**
$\langle 4, 4, 5 \rangle$	62 [47]	**61**
$\langle 4, 4, 6 \rangle$	73 [48]	73
$\langle 4, 4, 7 \rangle$	87 [93, 95]	**85**
$\langle 4, 4, 8 \rangle$	98 [95]	**96**
$\langle 4, 4, 9 \rangle$	104 [92]	108
$\langle 4, 5, 5 \rangle$	76 [26]	76
$\langle 4, 5, 6 \rangle$	93 [48]	**90**
$\langle 5, 5, 5 \rangle$	93 [72]	93
$\langle 6, 6, 6 \rangle$	153 [72]	156

Resultados en Multiplicación de Matrices

Aplicamos AlphaEvolve para mejorar un algoritmo de descomposición de tensores (basado en descenso por gradiente)

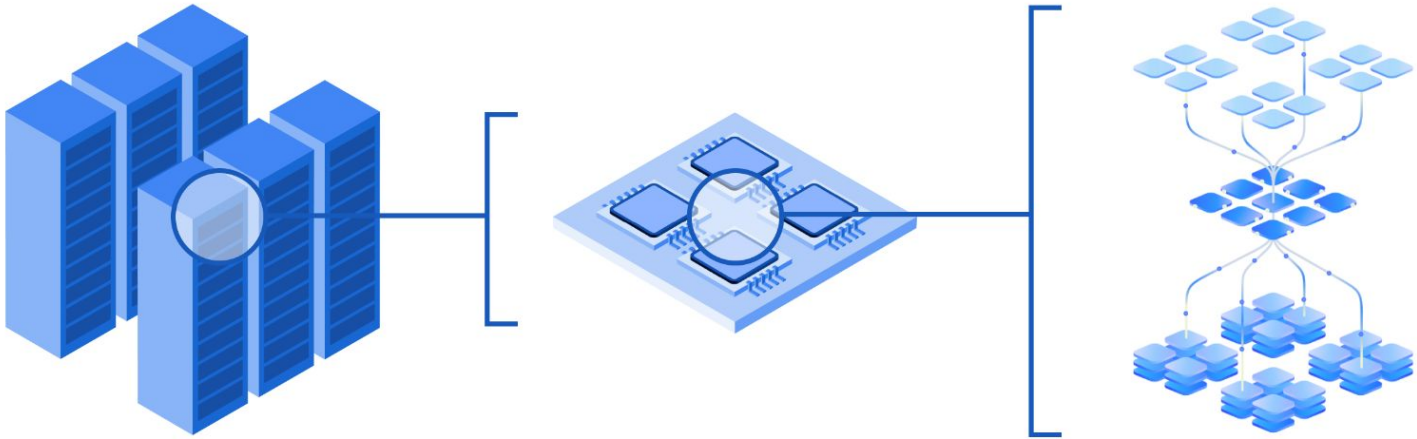
$\langle m, n, p \rangle$	mejor #mult	AlphaEvolve
$\langle 2, 2, 2 \rangle$	7 [95]	7
$\langle 2, 2, 3 \rangle$	11 [93]	11
$\langle 2, 2, 4 \rangle$	14 [93]	14
$\langle 2, 2, 5 \rangle$	18 [93]	18
$\langle 2, 2, 6 \rangle$	21 [93]	21
$\langle 2, 2, 7 \rangle$	25 [93]	25
$\langle 2, 2, 8 \rangle$	28 [93]	28
$\langle 2, 2, 9 \rangle$	32 [93]	32
$\langle 2, 2, 10 \rangle$	35 [93]	35
$\langle 2, 2, 11 \rangle$	39 [93]	39
$\langle 2, 2, 12 \rangle$	42 [93]	42
$\langle 2, 2, 13 \rangle$	46 [93]	46
$\langle 2, 2, 14 \rangle$	49 [93]	49
$\langle 2, 2, 15 \rangle$	53 [93]	53
$\langle 2, 2, 16 \rangle$	56 [93]	56
$\langle 2, 3, 3 \rangle$	15 [93]	15
$\langle 2, 3, 4 \rangle$	20 [93]	20
$\langle 2, 3, 5 \rangle$	25 [93]	25

$\langle m, n, p \rangle$	mejor #mult	AlphaEvolve
$\langle 2, 3, 6 \rangle$	30 [93]	30
$\langle 2, 3, 7 \rangle$	35 [93]	35
$\langle 2, 3, 8 \rangle$	40 [93]	40
$\langle 2, 3, 9 \rangle$	45 [93]	45
$\langle 2, 3, 10 \rangle$	50 [93]	50
$\langle 2, 4, 4 \rangle$	26 [93]	26
$\langle 2, 4, 5 \rangle$	33 [42]	**32**
$\langle 2, 4, 6 \rangle$	39 [93]	39
$\langle 2, 4, 7 \rangle$	46 [93]	**45**
$\langle 2, 4, 8 \rangle$	52 [93]	**51**
$\langle 2, 5, 5 \rangle$	40 [93]	40
$\langle 2, 5, 6 \rangle$	48 [93]	**47**
$\langle 3, 3, 3 \rangle$	23 [52]	23
$\langle 3, 3, 4 \rangle$	29 [93]	29
$\langle 3, 3, 5 \rangle$	36 [93]	36
$\langle 3, 3, 6 \rangle$	40 [93]	40
$\langle 3, 3, 7 \rangle$	49 [93]	49
$\langle 3, 3, 8 \rangle$	55 [93]	55

$\langle m, n, p \rangle$	mejor #mult	AlphaEvolve
$\langle 3, 4, 4 \rangle$	38 [93]	38
$\langle 3, 4, 5 \rangle$	47 [26]	47
$\langle 3, 4, 6 \rangle$	56 [48]	**54**
$\langle 3, 4, 7 \rangle$	66 [91]	**63**
$\langle 3, 4, 8 \rangle$	75 [91]	**74**
$\langle 3, 5, 5 \rangle$	58 [91]	58
$\langle 3, 5, 6 \rangle$	70 [48]	**68**
$\langle 3, 5, 7 \rangle$	82 [91]	**80**
$\langle 4, 4, 4 \rangle$	49 [95]	**48**
$\langle 4, 4, 5 \rangle$	62 [47]	**61**
$\langle 4, 4, 6 \rangle$	73 [48]	73
$\langle 4, 4, 7 \rangle$	87 [93, 95]	**85**
$\langle 4, 4, 8 \rangle$	98 [95]	**96**
$\langle 4, 4, 9 \rangle$	104 [92]	108
$\langle 4, 5, 5 \rangle$	76 [26]	76
$\langle 4, 5, 6 \rangle$	93 [48]	**90**
$\langle 5, 5, 5 \rangle$	93 [72]	93
$\langle 6, 6, 6 \rangle$	153 [72]	156

Mejorando el Ecosistema de Computación de Google

Usamos AlphaEvolve para mejorar algoritmos que ya están en **producción** dentro de Alphabet



Optimizar centros de datos
Distribución de tareas en Borg

0.7%

recursos recuperados

Optimizar hardware
Diseño de circuitos en TPUs

23%

aceleración en el kernel de multiplicación de matrices

Optimizar software
Entrenamiento de Gemini

1%

aceleración end-to-end

¿Cuándo Usar AlphaEvolve?

- **Código.** El algoritmo a mejorar debe estar expresado en forma de código

- **Evaluación.** AlphaEvolve requiere una manera automática de medir la calidad (“score”) de una solución dada
  - La evaluación no debe ser demasiado lenta, o si lo es, debe de ser paralelizable

- **Experiencia humana.** Requiere conocimientos de programación
  - Configuración inicial, definición del problema, desarrollo iterativo, etc.

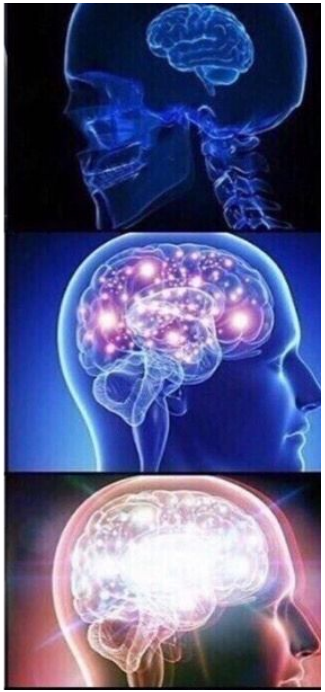
- **Computación.** AlphaEvolve es an sistema distribuido que puede ser computacionalmente costoso
  - Generación de soluciones mediante LLMs + Evaluación automática

Índice

Conclusiones

- Descubrir algoritmos es una tarea compleja
- La IA puede ayudar a progresar e incluso encontrar soluciones que van más allá de las capacidades humanas
- Métodos más generales pueden funcionar mejor que métodos especializados

Maneras de Afrontar Problemas



Buscar la solución directamente
(*dame una lista de vectores*)

Buscar una función que construya la solución directamente
(*dame una función que genere una lista de vectores*)

Buscar un algoritmo de búsqueda que encuentre la solución
(*dame un algoritmo que busque una lista de vectores*)

Limitaciones y Futuras líneas de investigación

- La evaluación automática no está siempre disponible (por ejemplo, en ciencias naturales)
- La evaluación automática, aunque sea posible, puede ser costosa

- Seguir mejorando los componentes de AlphaEvolve
- Combinar con otras herramientas de IA (por ejemplo, Co-Scientist)
- Explorar más problemas!