

Diseñando los algoritmos del futuro con IA

Francisco J. R. Ruiz

06/10/2025



Presentación



Ingeniería de
Telecomunicaciones

Presentación



Ingeniería de
Telecomunicaciones

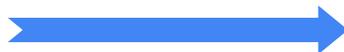


Máster y Doctorado en
Aprendizaje Automático

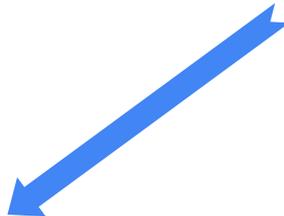
Presentación



Ingeniería de
Telecomunicaciones



Máster y Doctorado en
Aprendizaje Automático



UNIVERSITY OF
CAMBRIDGE

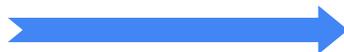


Postdoctorado en
Aprendizaje Automático

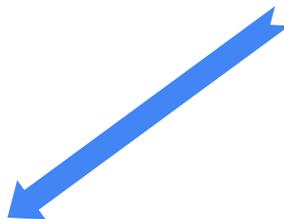
Presentación



Ingeniería de
Telecomunicaciones



Máster y Doctorado en
Aprendizaje Automático



UNIVERSITY OF
CAMBRIDGE



Postdoctorado en
Aprendizaje Automático



Google DeepMind

Investigador en
Google DeepMind

Índice

1. Introducción
2. AlphaTensor
3. FunSearch y AlphaEvolve
4. Conclusiones



Índice

1. **Introducción**
2. AlphaTensor
3. FunSearch y AlphaEvolve
4. Conclusiones



Algoritmos

Euclides
Hallar el MCD
de dos
números



300 a.C.

Al-Khwarizmi
Resolver
ecuaciones de
2º grado



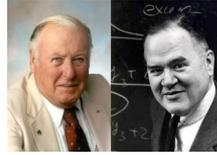
820

Edsger W. Dijkstra
Calcular el camino
más corto



1956

James Cooley &
John Tukey
Transformada
rápida de Fourier
(FFT)



1965

Ron Rivest,
Adi Shamir &
Leonard Adleman
Algoritmo RSA



1977

Peter Schor
Algoritmo
cuántico para
factorización de
números enteros



1994

Larry Page &
Sergey Brin
PageRank para
buscar páginas
web



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

$$h_1 = a_{11} b_{11}$$

$$h_2 = a_{11} b_{12}$$

$$h_3 = a_{12} b_{21}$$

$$h_4 = a_{12} b_{22}$$

$$h_5 = a_{21} b_{11}$$

$$h_6 = a_{21} b_{12}$$

$$h_7 = a_{22} b_{21}$$

$$h_8 = a_{22} b_{22}$$

$$c_{11} = h_1 + h_3$$

$$c_{12} = h_2 + h_4$$

$$c_{21} = h_5 + h_7$$

$$c_{22} = h_6 + h_8$$

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

$$h_1 = a_{11} b_{11}$$

$$h_2 = a_{11} b_{12}$$

$$h_3 = a_{12} b_{21}$$

$$h_4 = a_{12} b_{22}$$

$$h_5 = a_{21} b_{11}$$

$$h_6 = a_{21} b_{12}$$

$$h_7 = a_{22} b_{21}$$

$$h_8 = a_{22} b_{22}$$

$$c_{11} = h_1 + h_3$$

$$c_{12} = h_2 + h_4$$

$$c_{21} = h_5 + h_7$$

$$c_{22} = h_6 + h_8$$

Algoritmo de Strassen

$$h_1 = (a_{11} + a_{22})(b_{11} + b_{22})$$

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

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

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

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

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

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

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

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

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

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

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

$$h_1 = a_{11} b_{11}$$

$$h_2 = a_{11} b_{12}$$

$$h_3 = a_{12} b_{21}$$

$$h_4 = a_{12} b_{22}$$

$$h_5 = a_{21} b_{11}$$

$$h_6 = a_{21} b_{12}$$

$$h_7 = a_{22} b_{21}$$

$$h_8 = a_{22} b_{22}$$

$$c_{11} = h_1 + h_3$$

$$c_{12} = h_2 + h_4$$

$$c_{21} = h_5 + h_7$$

$$c_{22} = h_6 + h_8$$

8 multiplicaciones
de dos escalares

Algoritmo de Strassen

$$h_1 = (a_{11} + a_{22})(b_{11} + b_{22})$$

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

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

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

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

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

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

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

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

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

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

7 multiplicaciones
de dos escalares

Multiplicación de Matrices: Recursividad

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

Multiplicación de Matrices: Recursividad

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

Algoritmo estándar

$$H_1 = A_{11} B_{11}$$

$$H_2 = A_{11} B_{12}$$

$$H_3 = A_{12} B_{21}$$

$$H_4 = A_{12} B_{22}$$

$$H_5 = A_{21} B_{11}$$

$$H_6 = A_{21} B_{12}$$

$$H_7 = A_{22} B_{21}$$

$$H_8 = A_{22} B_{22}$$

$$C_{11} = H_1 + H_3$$

$$C_{12} = H_2 + H_4$$

$$C_{21} = H_5 + H_7$$

$$C_{22} = H_6 + H_8$$

8 multiplicaciones
de dos **matrices**

Algoritmo de Strassen

$$H_1 = (A_{11} + A_{22}) (B_{11} + B_{22})$$

$$H_2 = (A_{21} + A_{22}) B_{11}$$

$$H_3 = A_{11} (B_{12} - B_{22})$$

$$H_4 = A_{22} (-B_{11} + B_{21})$$

$$H_5 = (A_{11} + A_{12}) B_{22}$$

$$H_6 = (-A_{11} + A_{21}) (B_{11} + B_{12})$$

$$H_7 = (A_{12} - A_{22}) (B_{21} + B_{22})$$

$$C_{11} = H_1 + H_4 - H_5 + H_7$$

$$C_{12} = H_3 + H_5$$

$$C_{21} = H_2 + H_4$$

$$C_{22} = H_1 - H_2 + H_3 + H_6$$

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	7	$\mathcal{O}(N^{\log_2 7}) \approx \mathcal{O}(N^{2.81})$
3×3		

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	$\mathcal{O}(N^{\log_2 7}) \approx \mathcal{O}(N^{2.81})$
Problemas abiertos	3×3		
	4×4		
	\vdots	\vdots	\vdots

Multiplicación de Matrices: Complejidad Computacional

	tamaño ($n \times n$)	#multiplicaciones	complejidad ($N \times N$)
Problemas abiertos	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

Multiplicación de Matrices: Complejidad Computacional

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

descubierto por AlphaEvolve

Multiplicación de Matrices: Complejidad Computacional

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

descubierto por AlphaEvolve

Descubrir nuevos algoritmos es una tarea compleja

Índice

1. Introducción
2. **AlphaTensor**
3. FunSearch y AlphaEvolve
4. Conclusiones



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 Descubierta 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 Descubierta 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}(-b_{2,1} - b_{2,5} - b_{3,1}) \\ h_2 &= (a_{2,2} + a_{2,5} - a_{3,5})(-b_{2,5} - b_{5,1}) \\ h_3 &= (-a_{3,1} - a_{4,1} + a_{4,2})(-b_{1,1} + b_{2,5}) \\ h_4 &= (a_{1,2} + a_{1,4} + a_{3,4})(-b_{2,5} - b_{4,1}) \\ h_5 &= (a_{1,5} + a_{2,2} + a_{2,5})(-b_{2,4} + b_{5,1}) \\ h_6 &= (-a_{2,2} - a_{2,5} - a_{4,5})(b_{2,3} + b_{5,1}) \\ h_7 &= (-a_{1,1} + a_{4,1} - a_{4,2})(b_{1,1} + b_{2,4}) \\ h_8 &= (a_{3,2} - a_{3,3} - a_{4,3})(-b_{2,3} + b_{3,1}) \\ h_9 &= (-a_{1,2} - a_{1,4} + a_{4,4})(b_{2,3} + b_{4,1}) \\ h_{10} &= (a_{2,2} + a_{2,5})b_{5,1} \\ h_{11} &= (-a_{2,1} - a_{4,1} + a_{4,2})(-b_{1,1} + b_{2,2}) \\ h_{12} &= (a_{4,1} - a_{4,2})b_{1,1} \\ h_{13} &= (a_{1,2} + a_{1,4} + a_{2,4})(b_{2,2} + b_{4,1}) \\ h_{14} &= (a_{1,3} - a_{3,2} + a_{3,3})(b_{2,4} + b_{3,1}) \\ h_{15} &= (-a_{1,2} - a_{1,4})b_{4,1} \\ h_{16} &= (-a_{3,2} + a_{3,3})b_{3,1} \\ h_{17} &= (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})b_{2,2} \\ h_{18} &= a_{2,1}(b_{1,1} + b_{1,2} + b_{5,2}) \\ h_{19} &= -a_{2,3}(b_{3,1} + b_{3,2} + b_{5,2}) \\ h_{20} &= (-a_{1,5} + a_{2,1} + a_{2,3} - a_{2,5})(-b_{1,1} - b_{1,2} + b_{1,4} - b_{5,2}) \\ h_{21} &= (a_{2,1} + a_{2,3} - a_{2,5})b_{5,2} \\ h_{22} &= (a_{1,3} - a_{1,4} - a_{2,4})(b_{1,1} + b_{1,2} - b_{1,4} - b_{3,1} - b_{3,2} + b_{3,4} + b_{4,4}) \\ h_{23} &= a_{1,3}(-b_{3,1} + b_{3,4} + b_{4,4}) \\ h_{24} &= a_{1,5}(-b_{4,4} - b_{5,1} + b_{5,4}) \\ h_{25} &= -a_{1,1}(b_{1,1} - b_{1,4}) \\ h_{26} &= (-a_{1,3} + a_{1,4} + a_{1,5})b_{4,4} \\ h_{27} &= (a_{1,3} - a_{3,1} + a_{3,3})(b_{1,1} - b_{1,4} + b_{1,5} + b_{3,5}) \\ h_{28} &= (a_{1,3} - a_{3,1} + a_{3,3})(b_{1,1} - b_{1,4} + b_{1,5} + b_{3,5}) \\ h_{29} &= (a_{1,3} - a_{3,1} + a_{3,3})(b_{1,1} - b_{1,4} + b_{1,5} + b_{3,5}) \\ h_{30} &= (a_{1,3} - a_{3,1} + a_{3,3})(b_{1,1} - b_{1,4} + b_{1,5} + b_{3,5}) \\ h_{31} &= (a_{1,3} - a_{3,1} + a_{3,3})(b_{1,1} - b_{1,4} + b_{1,5} + b_{3,5}) \\ h_{32} &= (a_{1,3} - a_{3,1} + a_{3,3})(b_{1,1} - b_{1,4} + b_{1,5} + b_{3,5}) \\ h_{33} &= (a_{1,3} - a_{3,1} + a_{3,3})(b_{1,1} - b_{1,4} + b_{1,5} + b_{3,5}) \\ h_{34} &= (a_{1,3} - a_{3,1} + a_{3,3})(b_{1,1} - b_{1,4} + b_{1,5} + b_{3,5}) \\ h_{35} &= (a_{1,3} - a_{3,1} + a_{3,3})(b_{1,1} - b_{1,4} + b_{1,5} + b_{3,5}) \\ h_{36} &= (a_{1,3} - a_{3,1} + a_{3,3})(b_{1,1} - b_{1,4} + b_{1,5} + b_{3,5}) \\ h_{37} &= (a_{1,3} - a_{3,1} + a_{3,3})(b_{1,1} - b_{1,4} + b_{1,5} + b_{3,5}) \\ h_{38} &= (a_{1,3} - a_{3,1} + a_{3,3})(b_{1,1} - b_{1,4} + b_{1,5} + b_{3,5}) \\ h_{39} &= (a_{1,3} - a_{3,1} + a_{3,3})(b_{1,1} - b_{1,4} + b_{1,5} + b_{3,5}) \\ h_{40} &= (a_{1,3} - a_{3,1} + a_{3,3})(b_{1,1} - b_{1,4} + b_{1,5} + b_{3,5}) \\ h_{41} &= (a_{1,3} - a_{3,1} + a_{3,3})(b_{1,1} - b_{1,4} + b_{1,5} + b_{3,5}) \\ h_{42} &= (a_{1,3} - a_{3,1} + a_{3,3})(b_{1,1} - b_{1,4} + b_{1,5} + b_{3,5}) \\ h_{43} &= (a_{1,3} - a_{3,1} + a_{3,3})(b_{1,1} - b_{1,4} + b_{1,5} + b_{3,5}) \\ h_{44} &= (a_{1,3} - a_{3,1} + a_{3,3})(b_{1,1} - b_{1,4} + b_{1,5} + b_{3,5}) \\ h_{45} &= (a_{1,3} - a_{3,1} + a_{3,3})(b_{1,1} - b_{1,4} + b_{1,5} + b_{3,5}) \\ h_{46} &= (a_{1,3} - a_{3,1} + a_{3,3})(b_{1,1} - b_{1,4} + b_{1,5} + b_{3,5}) \\ h_{47} &= (a_{1,3} - a_{3,1} + a_{3,3})(b_{1,1} - b_{1,4} + b_{1,5} + b_{3,5}) \\ h_{48} &= (a_{1,3} - a_{3,1} + a_{3,3})(b_{1,1} - b_{1,4} + b_{1,5} + b_{3,5}) \\ h_{49} &= (a_{1,3} - a_{3,1} + a_{3,3})(b_{1,1} - b_{1,4} + b_{1,5} + b_{3,5}) \\ h_{50} &= (a_{1,3} - a_{3,1} + a_{3,3})(b_{1,1} - b_{1,4} + b_{1,5} + b_{3,5}) \\ h_{51} &= a_{2,2}(b_{2,1} + b_{2,2} - b_{5,1}) \\ h_{52} &= a_{4,2}(b_{1,1} + b_{2,1} + b_{2,3}) \\ h_{53} &= -a_{1,2}(-b_{2,1} + b_{2,4} + b_{4,1}) \\ h_{54} &= (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})b_{2,3} \\ h_{55} &= (a_{1,4} - a_{4,4})(-b_{2,3} + b_{3,1} + b_{3,3} - b_{3,4} + b_{4,3} - b_{4,4}) \\ h_{56} &= (a_{1,1} - a_{1,5} - a_{4,1} + a_{4,5})(b_{3,1} + b_{3,3} - b_{3,4} + b_{5,1} + b_{5,3} - b_{5,4}) \\ h_{57} &= (-a_{3,1} - a_{4,1})(-b_{1,3} - b_{1,5} - b_{2,5} - b_{5,1} - b_{5,3} - b_{5,5}) \\ h_{58} &= (-a_{1,4} - a_{1,5} - a_{3,4} - a_{3,5})(-b_{5,1} + b_{5,4} - b_{5,5}) \\ h_{59} &= (-a_{3,3} + a_{3,4} - a_{4,3} + a_{4,4})(b_{4,1} + b_{4,3} + b_{4,5} + b_{5,1} + b_{5,3} + b_{5,5}) \\ h_{60} &= (a_{2,5} + a_{4,5})(b_{2,3} - b_{3,1} - b_{3,2} - b_{3,3} - b_{5,2} - b_{5,3}) \\ h_{61} &= (a_{1,4} + a_{3,4})(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}) \\ h_{62} &= (a_{2,1} + a_{4,1})(b_{1,2} + b_{1,3} + b_{2,2} - b_{4,1} - b_{4,2} - b_{4,3}) \\ h_{63} &= (-a_{3,3} - a_{4,3})(-b_{2,3} - b_{3,3} - b_{3,5} - b_{4,1} - b_{4,3} - b_{4,5}) \\ h_{64} &= (a_{1,1} - a_{1,3} - a_{1,4} + a_{3,1} - a_{3,3} - a_{3,4})(b_{1,1} - b_{1,4} + b_{1,5}) \\ h_{65} &= (-a_{1,1} + a_{4,1})(-b_{1,3} + b_{1,4} + b_{2,4} - b_{5,1} - b_{5,3} + b_{5,4}) \\ h_{66} &= (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})b_{2,4} \\ h_{67} &= (a_{2,5} - a_{3,5})(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}) \\ h_{68} &= (a_{1,1} + a_{1,3} - a_{1,4} - a_{1,5} - a_{4,1} - a_{4,3} + a_{4,4} + a_{4,5})(-b_{3,1} - b_{3,3} + b_{3,4}) \\ h_{69} &= (-a_{1,3} + a_{1,4} - a_{2,3} + a_{2,4})(-b_{2,4} - b_{3,1} - b_{3,2} + b_{3,4} - b_{5,2} + b_{5,4}) \\ h_{70} &= (a_{2,3} - a_{2,5} + a_{4,3} - a_{4,5})(-b_{3,1} - b_{3,2} - b_{3,3}) \\ h_{71} &= (-a_{3,1} + a_{3,3} - a_{3,4} + a_{3,5} - a_{4,1} + a_{4,3} - a_{4,4} + a_{4,5})(-b_{5,1} - b_{5,3} - b_{5,5}) \\ h_{72} &= (-a_{2,1} - a_{2,4} - a_{4,1} - a_{4,4})(b_{4,1} + b_{4,2} + b_{4,3}) \\ h_{73} &= (a_{1,3} - a_{1,4} - a_{1,5} + a_{2,3} - a_{2,4} - a_{2,5})(b_{1,1} + b_{1,2} - b_{1,4} + b_{2,4} + b_{5,2} - b_{5,4}) \\ h_{74} &= (a_{2,1} - a_{2,3} + a_{2,4} - a_{3,1} + a_{3,3} - a_{3,4})(b_{4,1} + b_{4,2} + b_{4,5}) \\ h_{75} &= -(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})b_{2,5} \\ h_{76} &= (a_{1,3} + a_{3,3})(-b_{1,1} + b_{1,4} - b_{1,5} + b_{2,4} + b_{3,4} - b_{3,5}) \\ h_{77} &= -h_{10} + h_{12} + h_{14} - h_{15} - h_{16} + h_{53} + h_5 - h_{66} - h_7 \\ h_{78} &= h_{10} + h_{11} - h_{12} + h_{13} + h_{15} + h_{16} - h_{17} - h_{44} + h_{51} \\ h_{79} &= h_{10} + h_{11} - h_{12} + h_{13} + h_{15} + h_{16} - h_{17} - h_{44} + h_{51} \\ h_{80} &= h_{10} + h_{11} - h_{12} + h_{13} + h_{15} + h_{16} - h_{17} - h_{44} + h_{51} \\ h_{81} &= h_{10} + h_{11} - h_{12} + h_{13} + h_{15} + h_{16} - h_{17} - h_{44} + h_{51} \\ h_{82} &= h_{10} + h_{11} - h_{12} + h_{13} + h_{15} + h_{16} - h_{17} - h_{44} + h_{51} \\ h_{83} &= h_{10} + h_{11} - h_{12} + h_{13} + h_{15} + h_{16} - h_{17} - h_{44} + h_{51} \\ h_{84} &= h_{10} + h_{11} - h_{12} + h_{13} + h_{15} + h_{16} - h_{17} - h_{44} + h_{51} \\ h_{85} &= h_{10} + h_{11} - h_{12} + h_{13} + h_{15} + h_{16} - h_{17} - h_{44} + h_{51} \\ h_{86} &= h_{10} + h_{11} - h_{12} + h_{13} + h_{15} + h_{16} - h_{17} - h_{44} + h_{51} \\ h_{87} &= h_{10} + h_{11} - h_{12} + h_{13} + h_{15} + h_{16} - h_{17} - h_{44} + h_{51} \\ h_{88} &= h_{10} + h_{11} - h_{12} + h_{13} + h_{15} + h_{16} - h_{17} - h_{44} + h_{51} \\ h_{89} &= h_{10} + h_{11} - h_{12} + h_{13} + h_{15} + h_{16} - h_{17} - h_{44} + h_{51} \\ h_{90} &= h_{10} + h_{11} - h_{12} + h_{13} + h_{15} + h_{16} - h_{17} - h_{44} + h_{51} \\ h_{91} &= h_{10} + h_{11} - h_{12} + h_{13} + h_{15} + h_{16} - h_{17} - h_{44} + h_{51} \\ h_{92} &= h_{10} + h_{11} - h_{12} + h_{13} + h_{15} + h_{16} - h_{17} - h_{44} + h_{51} \\ h_{93} &= h_{10} + h_{11} - h_{12} + h_{13} + h_{15} + h_{16} - h_{17} - h_{44} + h_{51} \\ h_{94} &= h_{10} + h_{11} - h_{12} + h_{13} + h_{15} + h_{16} - h_{17} - h_{44} + h_{51} \\ h_{95} &= h_{10} + h_{11} - h_{12} + h_{13} + h_{15} + h_{16} - h_{17} - h_{44} + h_{51} \\ h_{96} &= h_{10} + h_{11} - h_{12} + h_{13} + h_{15} + h_{16} - h_{17} - h_{44} + h_{51} \\ h_{97} &= h_{10} + h_{11} - h_{12} + h_{13} + h_{15} + h_{16} - h_{17} - h_{44} + h_{51} \\ h_{98} &= h_{10} + h_{11} - h_{12} + h_{13} + h_{15} + h_{16} - h_{17} - h_{44} + h_{51} \\ h_{99} &= h_{10} + h_{11} - h_{12} + h_{13} + h_{15} + h_{16} - h_{17} - h_{44} + h_{51} \\ h_{100} &= h_{10} + h_{11} - h_{12} + h_{13} + h_{15} + h_{16} - h_{17} - h_{44} + h_{51} \end{aligned}$$

Algoritmo Descubierta 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}(-b_{2,1} - b_{2,5} - b_{3,1}) \\ h_2 &= (a_{2,2} + a_{2,5} - a_{3,5})(-b_{2,5} - b_{5,1}) \\ h_3 &= (-a_{3,1} - a_{4,1} + a_{4,2})(-b_{1,1} + b_{2,5}) \\ h_4 &= (a_{1,2} + a_{1,4} + a_{3,4})(-b_{2,5} - b_{4,1}) \\ h_5 &= (a_{1,5} + a_{2,2} + a_{2,5})(-b_{2,4} + b_{5,1}) \\ h_6 &= (-a_{2,2} - a_{2,5} - a_{4,5})(b_{2,3} + b_{5,1}) \\ h_7 &= (-a_{1,1} + a_{4,1} - a_{4,2})(b_{1,1} + b_{2,4}) \\ h_8 &= (a_{3,2} - a_{3,3} - a_{4,3})(-b_{2,3} + b_{3,1}) \\ h_9 &= (-a_{1,2} - a_{1,4} + a_{4,4})(b_{2,3} + b_{4,1}) \\ h_{10} &= (a_{2,2} + a_{2,5})b_{5,1} \\ h_{11} &= (-a_{2,1} - a_{4,1} + a_{4,2})(-b_{1,1} + b_{2,2}) \\ h_{12} &= (a_{4,1} - a_{4,2})b_{1,1} \\ h_{13} &= (a_{1,2} + a_{1,4} + a_{2,4})(b_{2,2} + b_{4,1}) \\ h_{14} &= (a_{1,3} - a_{3,2} + a_{3,3})(b_{2,4} + b_{3,1}) \\ h_{15} &= (-a_{1,2} - a_{1,4})b_{4,1} \\ h_{16} &= (-a_{3,2} + a_{3,3})b_{3,1} \\ h_{17} &= (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})b_{2,2} \\ h_{18} &= a_{2,1}(b_{1,1} + b_{1,2} + b_{5,2}) \\ h_{19} &= -a_{2,3}(b_{3,1} + b_{3,2} + b_{5,2}) \\ h_{20} &= (-a_{1,5} + a_{2,1} + a_{2,3} - a_{2,5})(-b_{1,1} - b_{1,2} + b_{1,4} - b_{5,2}) \\ h_{21} &= (a_{2,1} + a_{2,3} - a_{2,5})b_{5,2} \\ h_{22} &= (a_{1,3} - a_{1,4} - a_{2,4})(b_{1,1} + b_{1,2} - b_{1,4} - b_{3,1} - b_{3,2} + b_{3,4} + b_{4,4}) \\ h_{23} &= a_{1,3}(-b_{3,1} + b_{3,4} + b_{4,4}) \\ h_{24} &= a_{1,5}(-b_{4,4} - b_{5,1} + b_{5,4}) \\ h_{25} &= -a_{1,1}(b_{1,1} - b_{1,4}) \\ h_{26} &= (-a_{1,3} + a_{1,4} + a_{1,5})b_{4,4} \\ h_{27} &= (a_{1,3} - a_{3,1} + a_{3,3})(b_{1,1} - b_{1,4} + b_{1,5} + b_{3,5}) \\ h_{28} &= (-a_{1,3} + a_{1,4} + a_{1,5} + a_{2,3} - a_{2,4} - a_{2,5})(b_{1,1} + b_{1,2} - b_{1,4} + b_{2,4} + b_{5,2} - b_{5,4}) \\ h_{29} &= (-a_{1,3} + a_{1,4} - a_{1,5} + a_{2,3} - a_{2,4} - a_{2,5})(b_{1,1} + b_{1,2} - b_{1,4} + b_{2,4} + b_{5,2} - b_{5,4}) \\ h_{30} &= (-a_{1,3} + a_{1,4} - a_{1,5} + a_{2,3} - a_{2,4} - a_{2,5})(b_{1,1} + b_{1,2} - b_{1,4} + b_{2,4} + b_{5,2} - b_{5,4}) \\ h_{31} &= (-a_{1,3} + a_{1,4} - a_{1,5} + a_{2,3} - a_{2,4} - a_{2,5})(b_{1,1} + b_{1,2} - b_{1,4} + b_{2,4} + b_{5,2} - b_{5,4}) \\ h_{32} &= (-a_{1,3} + a_{1,4} - a_{1,5} + a_{2,3} - a_{2,4} - a_{2,5})(b_{1,1} + b_{1,2} - b_{1,4} + b_{2,4} + b_{5,2} - b_{5,4}) \\ h_{33} &= (-a_{1,3} + a_{1,4} - a_{1,5} + a_{2,3} - a_{2,4} - a_{2,5})(b_{1,1} + b_{1,2} - b_{1,4} + b_{2,4} + b_{5,2} - b_{5,4}) \\ h_{34} &= (-a_{1,3} + a_{1,4} - a_{1,5} + a_{2,3} - a_{2,4} - a_{2,5})(b_{1,1} + b_{1,2} - b_{1,4} + b_{2,4} + b_{5,2} - b_{5,4}) \\ h_{35} &= (-a_{1,3} + a_{1,4} - a_{1,5} + a_{2,3} - a_{2,4} - a_{2,5})(b_{1,1} + b_{1,2} - b_{1,4} + b_{2,4} + b_{5,2} - b_{5,4}) \\ h_{36} &= (-a_{1,3} + a_{1,4} - a_{1,5} + a_{2,3} - a_{2,4} - a_{2,5})(b_{1,1} + b_{1,2} - b_{1,4} + b_{2,4} + b_{5,2} - b_{5,4}) \\ h_{37} &= (-a_{1,3} + a_{1,4} - a_{1,5} + a_{2,3} - a_{2,4} - a_{2,5})(b_{1,1} + b_{1,2} - b_{1,4} + b_{2,4} + b_{5,2} - b_{5,4}) \\ h_{38} &= (-a_{1,3} + a_{1,4} - a_{1,5} + a_{2,3} - a_{2,4} - a_{2,5})(b_{1,1} + b_{1,2} - b_{1,4} + b_{2,4} + b_{5,2} - b_{5,4}) \\ h_{39} &= (-a_{1,3} + a_{1,4} - a_{1,5} + a_{2,3} - a_{2,4} - a_{2,5})(b_{1,1} + b_{1,2} - b_{1,4} + b_{2,4} + b_{5,2} - b_{5,4}) \\ h_{40} &= (-a_{1,3} + a_{1,4} - a_{1,5} + a_{2,3} - a_{2,4} - a_{2,5})(b_{1,1} + b_{1,2} - b_{1,4} + b_{2,4} + b_{5,2} - b_{5,4}) \\ h_{41} &= (-a_{1,3} + a_{1,4} - a_{1,5} + a_{2,3} - a_{2,4} - a_{2,5})(b_{1,1} + b_{1,2} - b_{1,4} + b_{2,4} + b_{5,2} - b_{5,4}) \\ h_{42} &= (-a_{1,3} + a_{1,4} - a_{1,5} + a_{2,3} - a_{2,4} - a_{2,5})(b_{1,1} + b_{1,2} - b_{1,4} + b_{2,4} + b_{5,2} - b_{5,4}) \\ h_{43} &= (-a_{1,3} + a_{1,4} - a_{1,5} + a_{2,3} - a_{2,4} - a_{2,5})(b_{1,1} + b_{1,2} - b_{1,4} + b_{2,4} + b_{5,2} - b_{5,4}) \\ h_{44} &= (-a_{1,3} + a_{1,4} - a_{1,5} + a_{2,3} - a_{2,4} - a_{2,5})(b_{1,1} + b_{1,2} - b_{1,4} + b_{2,4} + b_{5,2} - b_{5,4}) \\ h_{45} &= (-a_{1,3} + a_{1,4} - a_{1,5} + a_{2,3} - a_{2,4} - a_{2,5})(b_{1,1} + b_{1,2} - b_{1,4} + b_{2,4} + b_{5,2} - b_{5,4}) \\ h_{46} &= (-a_{1,3} + a_{1,4} - a_{1,5} + a_{2,3} - a_{2,4} - a_{2,5})(b_{1,1} + b_{1,2} - b_{1,4} + b_{2,4} + b_{5,2} - b_{5,4}) \\ h_{47} &= (-a_{1,3} + a_{1,4} - a_{1,5} + a_{2,3} - a_{2,4} - a_{2,5})(b_{1,1} + b_{1,2} - b_{1,4} + b_{2,4} + b_{5,2} - b_{5,4}) \\ h_{48} &= (-a_{1,3} + a_{1,4} - a_{1,5} + a_{2,3} - a_{2,4} - a_{2,5})(b_{1,1} + b_{1,2} - b_{1,4} + b_{2,4} + b_{5,2} - b_{5,4}) \\ h_{49} &= (-a_{1,3} + a_{1,4} - a_{1,5} + a_{2,3} - a_{2,4} - a_{2,5})(b_{1,1} + b_{1,2} - b_{1,4} + b_{2,4} + b_{5,2} - b_{5,4}) \\ h_{50} &= (-a_{1,3} + a_{1,4} - a_{1,5} + a_{2,3} - a_{2,4} - a_{2,5})(b_{1,1} + b_{1,2} - b_{1,4} + b_{2,4} + b_{5,2} - b_{5,4}) \\ h_{51} &= a_{2,2}(b_{2,1} + b_{2,2} - b_{5,1}) \\ h_{52} &= a_{4,2}(b_{1,1} + b_{2,1} + b_{2,3}) \\ h_{53} &= -a_{1,2}(-b_{2,1} + b_{2,4} + b_{4,1}) \\ h_{54} &= (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})b_{2,3} \\ h_{55} &= (a_{1,4} - a_{4,4})(-b_{2,3} + b_{3,1} + b_{3,3} - b_{3,4} + b_{4,3} - b_{4,4}) \\ h_{56} &= (a_{1,1} - a_{1,5} - a_{4,1} + a_{4,5})(b_{3,1} + b_{3,3} - b_{3,4} + b_{5,1} + b_{5,3} - b_{5,4}) \\ h_{57} &= (-a_{3,1} - a_{4,1})(-b_{1,3} - b_{1,5} - b_{2,5} - b_{5,1} - b_{5,3} - b_{5,5}) \\ h_{58} &= (-a_{1,4} - a_{1,5} - a_{3,4} - a_{3,5})(-b_{5,1} + b_{5,4} - b_{5,5}) \\ h_{59} &= (-a_{3,3} + a_{3,4} - a_{4,3} + a_{4,4})(b_{4,1} + b_{4,3} + b_{4,5} + b_{5,1} + b_{5,3} + b_{5,5}) \\ h_{60} &= (a_{2,5} + a_{4,5})(b_{2,3} - b_{3,1} - b_{3,2} - b_{3,3} - b_{5,2} - b_{5,3}) \\ h_{61} &= (a_{1,4} + a_{3,4})(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}) \\ h_{62} &= (a_{2,1} + a_{4,1})(b_{1,2} + b_{1,3} + b_{2,2} - b_{4,1} - b_{4,2} - b_{4,3}) \\ h_{63} &= (-a_{3,3} - a_{4,3})(-b_{2,3} - b_{3,3} - b_{3,5} - b_{4,1} - b_{4,3} - b_{4,5}) \\ h_{64} &= (a_{1,1} - a_{1,3} - a_{1,4} + a_{3,1} - a_{3,3} - a_{3,4})(b_{1,1} - b_{1,4} + b_{1,5}) \\ h_{65} &= (-a_{1,1} + a_{4,1})(-b_{1,3} + b_{1,4} + b_{2,4} - b_{5,1} - b_{5,3} + b_{5,4}) \\ h_{66} &= (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})b_{2,4} \\ h_{67} &= (a_{2,5} - a_{3,5})(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}) \\ h_{68} &= (a_{1,1} + a_{1,3} - a_{1,4} - a_{1,5} - a_{4,1} - a_{4,3} + a_{4,4} + a_{4,5})(-b_{3,1} - b_{3,3} + b_{3,4}) \\ h_{69} &= (-a_{1,3} + a_{1,4} - a_{2,3} + a_{2,4})(-b_{2,4} - b_{3,1} - b_{3,2} + b_{3,4} - b_{5,2} + b_{5,4}) \\ h_{70} &= (a_{2,3} - a_{2,5} + a_{4,3} - a_{4,5})(-b_{3,1} - b_{3,2} - b_{3,3}) \\ h_{71} &= (-a_{3,1} + a_{3,3} - a_{3,4} + a_{3,5} - a_{4,1} + a_{4,3} - a_{4,4} + a_{4,5})(-b_{5,1} - b_{5,3} - b_{5,5}) \\ h_{72} &= (-a_{2,1} - a_{2,4} - a_{4,1} - a_{4,4})(b_{4,1} + b_{4,2} + b_{4,3}) \\ h_{73} &= (a_{1,3} - a_{1,4} - a_{1,5} + a_{2,3} - a_{2,4} - a_{2,5})(b_{1,1} + b_{1,2} - b_{1,4} + b_{2,4} + b_{5,2} - b_{5,4}) \\ h_{74} &= (a_{2,1} - a_{2,3} + a_{2,4} - a_{3,1} + a_{3,3} - a_{3,4})(b_{4,1} + b_{4,2} + b_{4,5}) \\ h_{75} &= (-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})b_{2,5} \\ h_{76} &= (a_{1,3} + a_{3,3})(-b_{1,1} + b_{1,4} - b_{1,5} + b_{2,4} + b_{3,4} - b_{3,5}) \\ h_{77} &= -h_{10} + h_{12} + h_{14} - h_{15} - h_{16} + h_{53} + h_5 - h_{66} - h_7 \\ h_{78} &= h_{10} + h_{11} - h_{12} + h_{13} + h_{15} + h_{16} - h_{17} - h_{44} + h_{51} \\ h_{79} &= h_{10} + h_{11} - h_{12} + h_{13} + h_{15} + h_{16} - h_{17} - h_{44} + h_{51} \\ h_{80} &= h_{10} + h_{11} - h_{12} + h_{13} + h_{15} + h_{16} - h_{17} - h_{44} + h_{51} \\ h_{81} &= h_{10} + h_{11} - h_{12} + h_{13} + h_{15} + h_{16} - h_{17} - h_{44} + h_{51} \\ h_{82} &= h_{10} + h_{11} - h_{12} + h_{13} + h_{15} + h_{16} - h_{17} - h_{44} + h_{51} \\ h_{83} &= h_{10} + h_{11} - h_{12} + h_{13} + h_{15} + h_{16} - h_{17} - h_{44} + h_{51} \\ h_{84} &= h_{10} + h_{11} - h_{12} + h_{13} + h_{15} + h_{16} - h_{17} - h_{44} + h_{51} \\ h_{85} &= h_{10} + h_{11} - h_{12} + h_{13} + h_{15} + h_{16} - h_{17} - h_{44} + h_{51} \\ h_{86} &= h_{10} + h_{11} - h_{12} + h_{13} + h_{15} + h_{16} - h_{17} - h_{44} + h_{51} \\ h_{87} &= h_{10} + h_{11} - h_{12} + h_{13} + h_{15} + h_{16} - h_{17} - h_{44} + h_{51} \\ h_{88} &= h_{10} + h_{11} - h_{12} + h_{13} + h_{15} + h_{16} - h_{17} - h_{44} + h_{51} \\ h_{89} &= h_{10} + h_{11} - h_{12} + h_{13} + h_{15} + h_{16} - h_{17} - h_{44} + h_{51} \\ h_{90} &= h_{10} + h_{11} - h_{12} + h_{13} + h_{15} + h_{16} - h_{17} - h_{44} + h_{51} \\ h_{91} &= h_{10} + h_{11} - h_{12} + h_{13} + h_{15} + h_{16} - h_{17} - h_{44} + h_{51} \\ h_{92} &= h_{10} + h_{11} - h_{12} + h_{13} + h_{15} + h_{16} - h_{17} - h_{44} + h_{51} \\ h_{93} &= h_{10} + h_{11} - h_{12} + h_{13} + h_{15} + h_{16} - h_{17} - h_{44} + h_{51} \\ h_{94} &= h_{10} + h_{11} - h_{12} + h_{13} + h_{15} + h_{16} - h_{17} - h_{44} + h_{51} \\ h_{95} &= h_{10} + h_{11} - h_{12} + h_{13} + h_{15} + h_{16} - h_{17} - h_{44} + h_{51} \\ h_{96} &= h_{10} + h_{11} - h_{12} + h_{13} + h_{15} + h_{16} - h_{17} - h_{44} + h_{51} \\ h_{97} &= h_{10} + h_{11} - h_{12} + h_{13} + h_{15} + h_{16} - h_{17} - h_{44} + h_{51} \\ h_{98} &= h_{10} + h_{11} - h_{12} + h_{13} + h_{15} + h_{16} - h_{17} - h_{44} + h_{51} \\ h_{99} &= h_{10} + h_{11} - h_{12} + h_{13} + h_{15} + h_{16} - h_{17} - h_{44} + h_{51} \\ h_{100} &= h_{10} + h_{11} - h_{12} + h_{13} + h_{15} + h_{16} - h_{17} - h_{44} + h_{51} \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

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

$$h_1 = (a_{11} + a_{22}) (b_{11} + b_{22})$$

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

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

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

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

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

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

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

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

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

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

Algoritmo de Strassen Paso a Paso

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

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

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

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

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

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

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

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

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

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

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

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

Algoritmo de Strassen Paso a Paso

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

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

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

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

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

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

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

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

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

1

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

0

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

0

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

1

Algoritmo de Strassen Paso a Paso

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

$$\begin{aligned} h_1 &= (a_{11} + a_{22}) (b_{11} + b_{22}) & \left(\begin{matrix} 1 \cdot a_{11} + 0 \cdot a_{12} + 0 \cdot a_{21} + 1 \cdot a_{22} \end{matrix} \right) \left(\begin{matrix} 1 \cdot b_{11} + 0 \cdot b_{12} + 0 \cdot b_{21} + 1 \cdot b_{22} \end{matrix} \right) \\ h_2 &= (a_{21} + a_{22}) b_{11} & \left(\begin{matrix} 0 \cdot a_{11} + 0 \cdot a_{12} + 1 \cdot a_{21} + 1 \cdot a_{22} \end{matrix} \right) \left(\begin{matrix} 1 \cdot b_{11} + 0 \cdot b_{12} + 0 \cdot b_{21} + 0 \cdot b_{22} \end{matrix} \right) \\ h_3 &= a_{11} (b_{12} - b_{22}) \\ h_4 &= a_{22} (-b_{11} + b_{21}) \\ h_5 &= (a_{11} + a_{12}) b_{22} \\ h_6 &= (-a_{11} + a_{21}) (b_{11} + b_{12}) \\ h_7 &= (a_{12} - a_{22}) (b_{21} + b_{22}) \end{aligned}$$

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

Algoritmo de Strassen Paso a Paso

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

$$\begin{aligned} h_1 &= (a_{11} + a_{22}) (b_{11} + b_{22}) && \begin{pmatrix} 1 \cdot a_{11} + 0 \cdot a_{12} + 0 \cdot a_{21} + 1 \cdot a_{22} \\ 0 \cdot a_{11} + 0 \cdot a_{12} + 1 \cdot a_{21} + 1 \cdot a_{22} \end{pmatrix} \begin{pmatrix} 1 \cdot b_{11} + 0 \cdot b_{12} + 0 \cdot b_{21} + 1 \cdot b_{22} \\ 1 \cdot b_{11} + 0 \cdot b_{12} + 0 \cdot b_{21} + 0 \cdot b_{22} \end{pmatrix} \\ h_2 &= (a_{21} + a_{22}) b_{11} \\ h_3 &= a_{11} (b_{12} - b_{22}) \\ h_4 &= a_{22} (-b_{11} + b_{21}) \\ h_5 &= (a_{11} + a_{12}) b_{22} \\ h_6 &= (-a_{11} + a_{21}) (b_{11} + b_{12}) \\ h_7 &= (a_{12} - a_{22}) (b_{21} + b_{22}) && \begin{pmatrix} 0 \cdot a_{11} + 1 \cdot a_{12} + 0 \cdot a_{21} + (-1) a_{22} \\ 0 \cdot b_{11} + 0 \cdot b_{12} + 1 \cdot b_{21} + 1 \cdot b_{22} \end{pmatrix} \end{aligned}$$

$c_{11} = h_1 + h_4 - h_5 + h_7$	1	0				1
$c_{12} = h_3 + h_5$	0	0	•	•	•	0
$c_{21} = h_2 + h_4$	0	1				0
$c_{22} = h_1 - h_2 + h_3 + h_6$	1	-1				0

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:

$$\begin{array}{cccc|cccc|cccc} 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ a_{11} & a_{12} & a_{21} & a_{22} & b_{11} & b_{12} & b_{21} & b_{22} & c_{11} & c_{12} & c_{21} & c_{22} \end{array}$$

Actualización de estado:

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

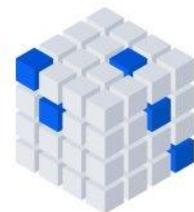
Repetir hasta alcanzar la matriz cero:

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

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:

$$\begin{array}{cccc|cccc|cccc} 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ a_{11} & a_{12} & a_{21} & a_{22} & b_{11} & b_{12} & b_{21} & b_{22} & c_{11} & c_{12} & c_{21} & c_{22} \end{array}$$

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

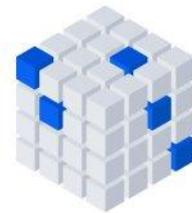
El juego **acaba** en el paso t si se ha alcanzado el estado cero ($S_t = 0$)

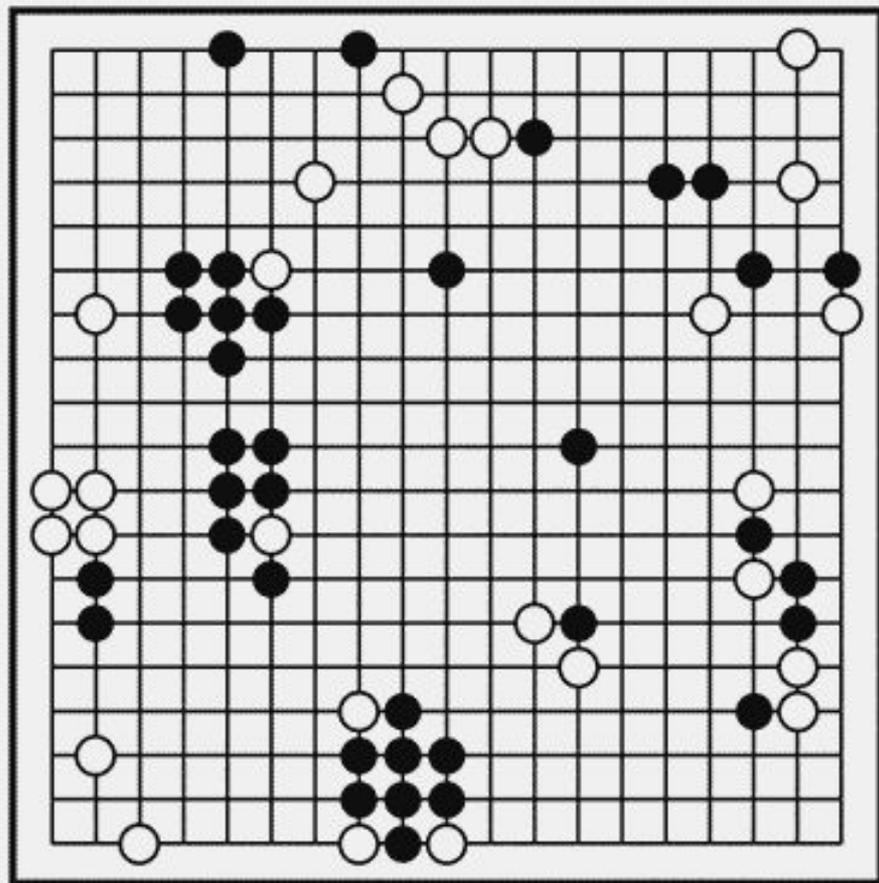
Por construcción, la secuencia de jugadas describe un algoritmo válido

Puntuación: -1 por cada jugada

alta puntuación \Leftrightarrow pocas jugadas \Leftrightarrow
algoritmo eficiente

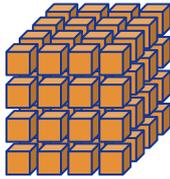
Idea: Usar aprendizaje por refuerzo para entrenar a un agente que aprenda a jugar a TensorGame



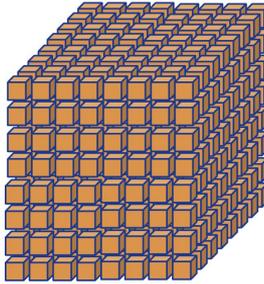


Dificultades de TensorGame

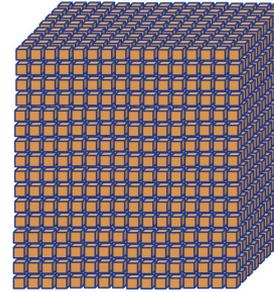
2x2



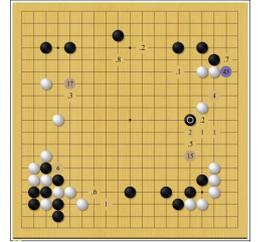
3x3



4x4



Go



Número de jugadas

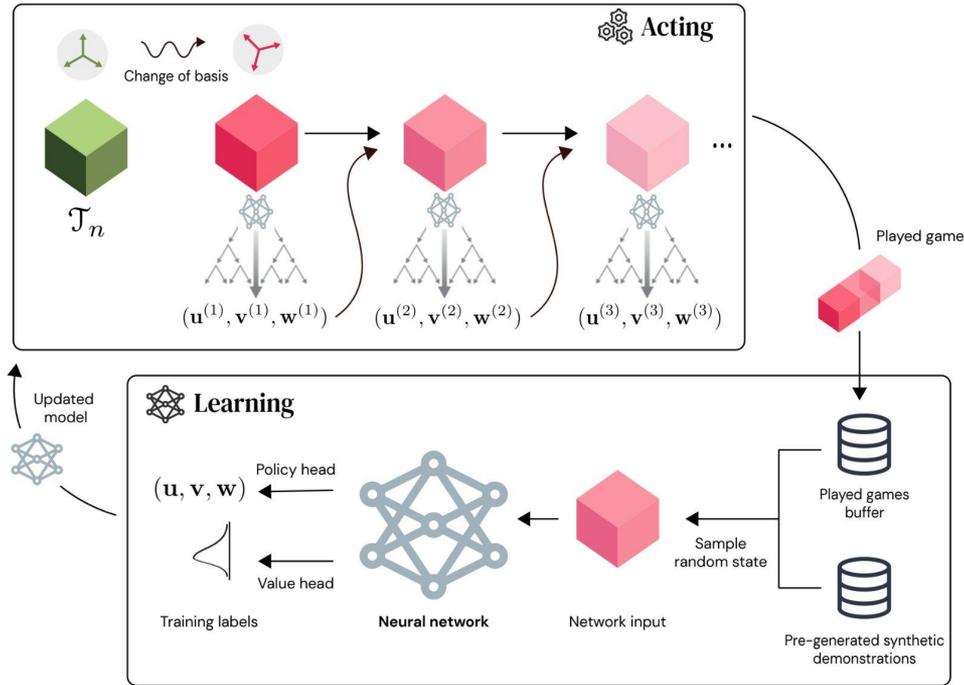
10^8

10^{18}

10^{33}

362

AlphaTensor



Se basa en AlphaZero

Dos **componentes** principales:

→ Actores (juegan continuamente)

→ Aprendiz (actualiza una red neuronal)



Índice

1. Introducción
2. AlphaTensor
- 3. FunSearch y AlphaEvolve**
4. Conclusiones



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

```
[1 1 1 1 1 1 1 1] [2 1 1 1 1 1 1 1] [0 0 1 0 1 0 2 1] [0 1 0 0 2 0 1 2] [1 0 0 0 0 1 1 1] [2 0 0 0 0 1 1 1] [0 0 1 1 0 1 2 1] [1 0 0 1 1 2 0 1]
[1 1 1 1 1 1 2 1] [2 1 1 1 1 1 2 1] [0 0 1 0 1 0 2 2] [0 1 0 0 2 0 2 2] [1 0 0 0 0 1 1 2] [2 0 0 0 0 1 1 2] [0 0 1 1 0 1 2 2] [1 0 0 1 1 2 0 2]
[1 1 1 1 1 2 1 1] [2 1 1 1 1 2 1 1] [0 0 1 0 1 1 0 1] [0 1 0 0 2 1 0 1] [1 0 0 0 0 1 1 0] [2 0 0 0 0 1 1 0] [0 0 1 1 0 2 2 1] [1 0 0 1 1 2 1 0]
[1 1 1 1 2 1 1 1] [2 1 1 1 2 1 1 1] [0 0 1 0 1 1 0 2] [0 1 0 0 2 1 0 2] [1 0 0 0 0 2 1 1] [2 0 0 0 0 2 1 1] [0 0 1 2 0 1 2 1] [1 0 0 1 2 2 0 1]
[1 1 1 2 1 1 1 1] [2 1 1 2 1 1 1 1] [0 0 1 0 1 2 0 1] [0 1 0 0 2 2 0 1] [1 0 0 0 0 2 1 0] [2 0 0 0 0 2 1 0] [0 0 1 2 0 2 2 1] [1 0 0 1 2 2 1 0]
[1 1 1 2 2 1 1 1] [2 1 1 2 2 1 1 1] [0 0 1 0 1 2 0 2] [0 1 0 0 2 2 0 2] [1 0 0 0 0 2 2 1] [2 0 0 0 0 2 2 1] [0 0 1 2 0 2 2 2] [1 0 0 1 2 2 2 0]
[1 1 2 1 1 1 1 1] [2 1 2 1 1 1 1 1] [0 0 1 0 2 0 1 0] [0 1 0 1 1 0 1 0] [1 0 0 1 0 0 1 2] [2 0 0 1 0 0 1 2] [0 0 2 1 0 1 1 2] [1 0 0 2 1 1 0 2]
[1 1 2 1 2 1 1 1] [2 1 2 1 2 1 1 1] [0 0 1 0 2 0 1 1] [0 1 0 1 1 0 1 1] [1 0 0 1 0 0 1 1] [2 0 0 1 0 0 1 1] [0 0 2 1 0 1 1 1] [1 0 0 2 1 1 0 1]
[1 1 2 2 1 1 1 1] [2 1 2 2 1 1 1 1] [0 0 1 0 2 0 1 2] [0 1 0 1 1 0 1 2] [1 0 0 2 0 0 1 1] [2 0 0 2 0 0 1 1] [0 0 2 2 0 1 1 1] [1 0 0 2 2 1 0 1]
[1 1 2 2 2 1 1 1] [2 1 2 2 2 1 1 1] [0 0 1 0 2 0 2 0] [0 1 0 2 0 0 2 0] [1 0 0 2 0 0 2 1] [2 0 0 2 0 0 2 1] [0 0 2 2 0 2 1 1] [1 0 0 2 2 1 0 1]
[1 1 2 2 2 1 2 1] [2 1 2 2 2 1 2 1] [0 0 1 0 2 0 2 1] [0 1 0 2 0 0 2 1] [1 0 0 2 0 0 2 2] [2 0 0 2 0 0 2 2] [0 0 2 2 0 2 2 1] [1 0 0 2 2 1 0 1]
[1 1 2 2 2 2 1 1] [2 1 2 2 2 2 1 1] [0 0 1 0 2 0 2 2] [0 1 0 2 0 0 2 2] [1 0 0 2 0 0 2 2] [2 0 0 2 0 0 2 2] [0 0 2 2 0 2 2 2] [1 0 0 2 2 1 0 2]
[1 1 2 2 2 2 2 1] [2 1 2 2 2 2 2 1] [0 0 1 0 2 0 2 2] [0 1 0 2 0 0 2 2] [1 0 0 2 0 0 2 2] [2 0 0 2 0 0 2 2] [0 0 2 2 0 2 2 2] [1 0 0 2 2 1 0 2]
[1 2 1 1 1 1 1 1] [2 2 1 1 1 1 1 1] [0 0 2 0 1 0 1 0] [0 1 0 1 0 1 0 0] [1 1 0 0 0 0 1 1] [2 1 0 0 0 0 1 1] [0 1 1 2 0 1 1 0] [2 0 0 1 1 2 0 1]
[1 2 1 1 1 1 2 1] [2 2 1 1 1 1 2 1] [0 0 2 0 1 0 1 1] [0 1 0 1 0 1 0 1] [1 1 0 0 0 0 1 2] [2 1 0 0 0 0 1 2] [0 1 1 2 0 1 2 0] [2 0 0 1 1 2 0 2]
[1 2 1 1 1 2 1 1] [2 2 1 1 1 2 1 1] [0 0 2 0 1 0 2 0] [0 1 0 1 0 2 0 0] [1 1 1 0 0 0 2 0] [2 1 1 0 0 0 2 0] [0 1 2 1 0 2 1 0] [2 0 0 1 1 2 1 0]
[1 2 1 1 2 1 1 1] [2 2 1 1 2 1 1 1] [0 0 2 0 2 0 1 0] [0 1 0 2 0 0 1 0] [1 1 1 0 0 0 2 0] [2 1 1 0 0 0 2 0] [0 1 2 1 0 2 1 0] [2 0 0 1 1 2 1 0]
[1 2 1 1 2 1 2 1] [2 2 1 1 2 1 2 1] [0 0 2 0 2 0 1 1] [0 1 0 2 0 0 1 1] [1 1 1 0 0 0 2 1] [2 1 1 0 0 0 2 1] [0 1 2 1 0 2 1 0] [2 0 0 1 1 2 2 0]
[1 2 1 1 2 2 1 1] [2 2 1 1 2 2 1 1] [0 0 2 0 2 0 2 0] [0 1 0 2 0 0 2 0] [1 1 1 0 0 0 2 0] [2 1 1 0 0 0 2 0] [0 1 2 1 0 2 1 0] [2 0 0 1 1 2 2 0]
[1 2 1 1 2 2 2 1] [2 2 1 1 2 2 2 1] [0 0 2 0 2 0 2 1] [0 1 0 2 0 0 2 1] [1 1 1 0 0 0 2 1] [2 1 1 0 0 0 2 1] [0 1 2 1 0 2 1 0] [2 0 0 1 1 2 2 1]
[1 2 2 1 1 1 1 1] [2 2 2 1 1 1 1 1] [0 0 2 0 1 1 0 0] [0 1 0 2 0 1 0 0] [1 1 0 0 0 0 2 0] [2 1 0 0 0 0 2 0] [0 1 2 1 0 2 1 0] [2 0 0 1 1 2 0 1]
[1 2 2 1 1 1 2 1] [2 2 2 1 1 1 2 1] [0 0 2 0 1 1 0 1] [0 1 0 2 0 1 0 1] [1 1 0 0 0 0 2 1] [2 1 0 0 0 0 2 1] [0 1 2 1 0 2 1 0] [2 0 0 1 1 2 0 2]
[1 2 2 1 1 2 1 1] [2 2 2 1 1 2 1 1] [0 0 2 0 1 1 1 0] [0 1 0 2 0 1 1 0] [1 1 0 0 0 0 2 0] [2 1 0 0 0 0 2 0] [0 1 2 1 0 2 1 0] [2 0 0 1 1 2 0 1]
[1 2 2 1 1 2 2 1] [2 2 2 1 1 2 2 1] [0 0 2 0 1 1 2 0] [0 1 0 2 0 1 2 0] [1 1 0 0 0 0 2 0] [2 1 0 0 0 0 2 0] [0 1 2 1 0 2 1 0] [2 0 0 1 1 2 0 2]
[1 2 2 1 1 2 2 1] [2 2 2 1 1 2 2 1] [0 0 2 0 1 1 2 1] [0 1 0 2 0 1 2 1] [1 1 0 0 0 0 2 1] [2 1 0 0 0 0 2 1] [0 1 2 1 0 2 1 0] [2 0 0 1 1 2 0 1]
[1 2 2 1 2 1 1 1] [2 2 2 1 2 1 1 1] [0 0 2 0 2 0 1 0] [0 1 0 2 0 2 0 0] [1 1 1 0 0 0 2 0] [2 1 1 0 0 0 2 0] [0 1 2 1 0 2 1 0] [2 0 0 1 1 2 0 1]
[1 2 2 1 2 1 2 1] [2 2 2 1 2 1 2 1] [0 0 2 0 2 0 1 1] [0 1 0 2 0 2 0 1] [1 1 1 0 0 0 2 1] [2 1 1 0 0 0 2 1] [0 1 2 1 0 2 1 0] [2 0 0 1 1 2 0 2]
[1 2 2 1 2 2 1 1] [2 2 2 1 2 2 1 1] [0 0 2 0 2 0 2 0] [0 1 0 2 0 2 0 0] [1 1 1 0 0 0 2 0] [2 1 1 0 0 0 2 0] [0 1 2 1 0 2 1 0] [2 0 0 1 1 2 0 1]
[1 2 2 1 2 2 2 1] [2 2 2 1 2 2 2 1] [0 0 2 0 2 0 2 1] [0 1 0 2 0 2 0 1] [1 1 1 0 0 0 2 1] [2 1 1 0 0 0 2 1] [0 1 2 1 0 2 1 0] [2 0 0 1 1 2 0 2]
[1 2 2 2 1 1 1 1] [2 2 2 2 1 1 1 1] [0 0 2 0 1 1 0 0] [0 1 0 2 0 1 0 0] [1 1 0 0 0 0 2 0] [2 1 0 0 0 0 2 0] [0 1 2 1 0 2 1 0] [2 0 0 1 1 2 0 1]
[1 2 2 2 1 1 2 1] [2 2 2 2 1 1 2 1] [0 0 2 0 1 1 0 1] [0 1 0 2 0 1 0 1] [1 1 0 0 0 0 2 1] [2 1 0 0 0 0 2 1] [0 1 2 1 0 2 1 0] [2 0 0 1 1 2 0 2]
[1 2 2 2 1 2 1 1] [2 2 2 2 1 2 1 1] [0 0 2 0 1 1 1 0] [0 1 0 2 0 1 1 0] [1 1 0 0 0 0 2 0] [2 1 0 0 0 0 2 0] [0 1 2 1 0 2 1 0] [2 0 0 1 1 2 0 1]
[1 2 2 2 1 2 2 1] [2 2 2 2 1 2 2 1] [0 0 2 0 1 1 2 0] [0 1 0 2 0 1 2 0] [1 1 0 0 0 0 2 0] [2 1 0 0 0 0 2 0] [0 1 2 1 0 2 1 0] [2 0 0 1 1 2 0 2]
[1 2 2 2 1 2 2 1] [2 2 2 2 1 2 2 1] [0 0 2 0 1 1 2 1] [0 1 0 2 0 1 2 1] [1 1 0 0 0 0 2 1] [2 1 0 0 0 0 2 1] [0 1 2 1 0 2 1 0] [2 0 0 1 1 2 0 1]
[1 2 2 2 2 1 1 1] [2 2 2 2 2 1 1 1] [0 0 2 0 2 1 0 0] [0 1 0 2 0 2 0 0] [1 1 2 0 0 0 2 0] [2 1 2 0 0 0 2 0] [0 1 2 1 0 2 1 0] [2 0 0 1 1 2 0 1]
[1 2 2 2 2 1 2 1] [2 2 2 2 2 1 2 1] [0 0 2 0 2 1 0 1] [0 1 0 2 0 2 0 1] [1 1 2 0 0 0 2 1] [2 1 2 0 0 0 2 1] [0 1 2 1 0 2 1 0] [2 0 0 1 1 2 0 2]
[1 2 2 2 2 2 1 1] [2 2 2 2 2 2 1 1] [0 0 2 0 2 1 1 0] [0 1 0 2 0 2 1 0] [1 1 2 0 0 0 2 0] [2 1 2 0 0 0 2 0] [0 1 2 1 0 2 1 0] [2 0 0 1 1 2 0 1]
[1 2 2 2 2 2 2 1] [2 2 2 2 2 2 2 1] [0 0 2 0 2 1 2 0] [0 1 0 2 0 2 2 0] [1 1 2 0 0 0 2 0] [2 1 2 0 0 0 2 0] [0 1 2 1 0 2 1 0] [2 0 0 1 1 2 0 2]
[1 2 2 2 2 2 2 1] [2 2 2 2 2 2 2 1] [0 0 2 0 2 1 2 1] [0 1 0 2 0 2 2 1] [1 1 2 0 0 0 2 1] [2 1 2 0 0 0 2 1] [0 1 2 1 0 2 1 0] [2 0 0 1 1 2 0 1]
```

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

```
[1 1 1 1 1 1 1] [2 1 1 1 1 1 1] [0 0 1 0 1 0 2 1] [0 1 0 0 2 0 1 2] [1 0 0 0 0 1 1 1] [2 0 0 0 0 1 1 1] [0 0 1 1 0 1 2 1] [1 0 0 1 1 2 0 2]
[1 1 1 1 1 1 2] [2 1 1 1 1 1 2] [0 0 1 0 1 0 2 2] [0 1 0 0 2 0 2 2] [1 0 0 0 0 1 1 2] [2 0 0 0 0 1 1 2] [0 0 1 1 0 1 2 2] [1 0 0 1 1 2 0 2]
[1 1 1 1 2 1 1] [2 1 1 1 2 1 1] [0 0 1 0 1 1 0 2] [0 1 0 0 2 1 0 2] [1 0 0 0 0 1 1 2] [2 0 0 0 0 1 1 2] [0 0 1 1 0 1 2 2] [1 0 0 1 1 2 0 2]
[1 1 1 2 1 1 1] [2 1 1 2 1 1 1] [0 0 1 0 1 1 2 0] [0 1 0 0 2 1 2 0] [1 0 0 0 0 2 1 1] [2 0 0 0 0 2 1 1] [0 0 1 2 0 1 2 1] [1 0 0 1 2 2 0 1]
[1 1 1 2 1 2 1] [2 1 1 2 1 2 1] [0 0 1 0 1 2 0 2] [0 1 0 0 2 2 0 2] [1 0 0 0 0 2 2 1] [2 0 0 0 0 2 2 1] [0 0 1 2 0 2 2 1] [1 0 0 1 2 2 0 1]
[1 1 2 1 1 1 1] [2 1 2 1 1 1 1] [0 0 1 0 1 2 0 2] [0 1 0 1 1 0 1 2] [1 0 0 1 0 0 1 2] [2 0 0 1 0 0 1 2] [0 0 2 1 0 1 1 2] [1 0 0 2 1 1 0 2]
[1 1 2 1 2 1 1] [2 1 2 1 2 1 1] [0 0 1 0 2 0 2 2] [0 1 0 1 1 0 1 2] [1 0 0 1 0 0 2 2] [2 0 0 1 0 0 2 2] [0 0 2 1 0 1 2 2] [1 0 0 2 1 1 0 2]
[1 1 2 2 1 1 1] [2 1 2 2 1 1 1] [0 0 1 0 2 1 0 2] [0 1 0 1 1 0 2 2] [1 0 0 1 0 0 2 2] [2 0 0 1 0 0 2 2] [0 0 2 1 0 2 2 2] [1 0 0 2 1 1 0 2]
[1 1 2 2 1 2 1] [2 1 2 2 1 2 1] [0 0 1 0 2 1 0 2] [0 1 0 1 1 0 2 2] [1 0 0 1 0 0 2 2] [2 0 0 1 0 0 2 2] [0 0 2 1 0 2 2 2] [1 0 0 2 1 1 0 2]
[1 1 2 2 2 1 1] [2 1 2 2 2 1 1] [0 0 1 0 2 2 0 2] [0 1 0 1 2 0 0 2] [1 0 0 2 0 0 2 2] [2 0 0 2 0 0 2 2] [0 0 2 2 0 2 2 2] [1 0 0 2 2 1 0 2]
[1 1 2 2 2 1 2] [2 1 2 2 2 1 2] [0 0 1 0 2 2 0 2] [0 1 0 1 2 0 0 2] [1 0 0 2 0 0 2 2] [2 0 0 2 0 0 2 2] [0 0 2 2 0 2 2 2] [1 0 0 2 2 1 0 2]
[1 1 2 2 2 2 1] [2 1 2 2 2 2 1] [0 0 1 0 2 2 0 2] [0 1 0 1 2 0 0 2] [1 0 0 2 0 0 2 2] [2 0 0 2 0 0 2 2] [0 0 2 2 0 2 2 2] [1 0 0 2 2 1 0 2]
[1 1 2 2 2 2 2] [2 1 2 2 2 2 2] [0 0 1 0 2 2 0 2] [0 1 0 1 2 0 0 2] [1 0 0 2 0 0 2 2] [2 0 0 2 0 0 2 2] [0 0 2 2 0 2 2 2] [1 0 0 2 2 1 0 2]
[1 2 1 1 1 1 1] [2 2 1 1 1 1 1] [0 0 1 1 0 0 1 1] [0 1 0 2 1 0 2 1] [1 0 1 0 0 0 2 1] [2 0 1 0 0 0 2 1] [0 1 0 1 0 1 2 1] [1 0 1 1 1 2 0 1]
[1 2 1 1 1 2 1] [2 2 1 1 1 2 1] [0 0 1 1 0 0 2 1] [0 1 0 2 1 0 2 1] [1 0 1 0 0 0 2 1] [2 0 1 0 0 0 2 1] [0 1 0 1 0 1 2 1] [1 0 1 1 1 2 0 1]
[1 2 1 1 2 1 1] [2 2 1 1 2 1 1] [0 0 1 1 0 0 2 1] [0 1 0 2 1 0 2 1] [1 0 1 0 0 0 2 1] [2 0 1 0 0 0 2 1] [0 1 0 1 0 1 2 1] [1 0 1 1 1 2 0 1]
[1 2 1 1 2 2 1] [2 2 1 1 2 2 1] [0 0 1 1 0 0 2 1] [0 1 0 2 1 0 2 1] [1 0 1 0 0 0 2 1] [2 0 1 0 0 0 2 1] [0 1 0 1 0 1 2 1] [1 0 1 1 1 2 0 1]
[1 2 1 2 1 1 1] [2 2 1 2 1 1 1] [0 0 1 1 0 0 2 1] [0 1 0 2 1 0 2 1] [1 0 1 0 0 0 2 1] [2 0 1 0 0 0 2 1] [0 1 0 1 0 1 2 1] [1 0 1 1 1 2 0 1]
[1 2 1 2 1 2 1] [2 2 1 2 1 2 1] [0 0 1 1 0 0 2 1] [0 1 0 2 1 0 2 1] [1 0 1 0 0 0 2 1] [2 0 1 0 0 0 2 1] [0 1 0 1 0 1 2 1] [1 0 1 1 1 2 0 1]
[1 2 1 2 2 1 1] [2 2 1 2 2 1 1] [0 0 1 1 0 0 2 1] [0 1 0 2 1 0 2 1] [1 0 1 0 0 0 2 1] [2 0 1 0 0 0 2 1] [0 1 0 1 0 1 2 1] [1 0 1 1 1 2 0 1]
[1 2 1 2 2 2 1] [2 2 1 2 2 2 1] [0 0 1 1 0 0 2 1] [0 1 0 2 1 0 2 1] [1 0 1 0 0 0 2 1] [2 0 1 0 0 0 2 1] [0 1 0 1 0 1 2 1] [1 0 1 1 1 2 0 1]
[1 2 2 1 1 1 1] [2 2 2 1 1 1 1] [0 0 1 1 1 0 0 1] [0 1 0 2 1 0 2 1] [1 0 1 0 0 0 2 1] [2 0 1 0 0 0 2 1] [0 1 0 1 0 1 2 1] [1 0 1 1 1 2 0 1]
[1 2 2 1 1 2 1] [2 2 2 1 1 2 1] [0 0 1 1 1 0 0 1] [0 1 0 2 1 0 2 1] [1 0 1 0 0 0 2 1] [2 0 1 0 0 0 2 1] [0 1 0 1 0 1 2 1] [1 0 1 1 1 2 0 1]
[1 2 2 1 2 1 1] [2 2 2 1 2 1 1] [0 0 1 1 1 0 0 1] [0 1 0 2 1 0 2 1] [1 0 1 0 0 0 2 1] [2 0 1 0 0 0 2 1] [0 1 0 1 0 1 2 1] [1 0 1 1 1 2 0 1]
[1 2 2 1 2 2 1] [2 2 2 1 2 2 1] [0 0 1 1 1 0 0 1] [0 1 0 2 1 0 2 1] [1 0 1 0 0 0 2 1] [2 0 1 0 0 0 2 1] [0 1 0 1 0 1 2 1] [1 0 1 1 1 2 0 1]
[1 2 2 2 1 1 1] [2 2 2 2 1 1 1] [0 0 1 1 1 0 0 1] [0 1 0 2 1 0 2 1] [1 0 1 0 0 0 2 1] [2 0 1 0 0 0 2 1] [0 1 0 1 0 1 2 1] [1 0 1 1 1 2 0 1]
[1 2 2 2 1 2 1] [2 2 2 2 1 2 1] [0 0 1 1 1 0 0 1] [0 1 0 2 1 0 2 1] [1 0 1 0 0 0 2 1] [2 0 1 0 0 0 2 1] [0 1 0 1 0 1 2 1] [1 0 1 1 1 2 0 1]
[1 2 2 2 2 1 1] [2 2 2 2 2 1 1] [0 0 1 1 1 0 0 1] [0 1 0 2 1 0 2 1] [1 0 1 0 0 0 2 1] [2 0 1 0 0 0 2 1] [0 1 0 1 0 1 2 1] [1 0 1 1 1 2 0 1]
[1 2 2 2 2 2 1] [2 2 2 2 2 2 1] [0 0 1 1 1 0 0 1] [0 1 0 2 1 0 2 1] [1 0 1 0 0 0 2 1] [2 0 1 0 0 0 2 1] [0 1 0 1 0 1 2 1] [1 0 1 1 1 2 0 1]
[1 2 2 2 2 2 2] [2 2 2 2 2 2 2] [0 0 1 0 0 1 2 0] [0 1 0 2 2 0 2 0] [1 0 2 0 0 0 2 0] [2 0 1 0 0 0 2 0] [0 1 0 1 0 2 2 0] [1 0 1 1 2 0 2 0]
[1 2 2 2 2 2 2] [2 2 2 2 2 2 2] [0 0 1 0 0 1 2 0] [0 1 0 2 2 0 2 0] [1 0 2 0 0 0 2 0] [2 0 1 0 0 0 2 0] [0 1 0 1 0 2 2 0] [1 0 1 1 2 0 2 0]
```

```
78 def get_capset(n: int) -> CapSet:
79     """Returns a 512-cap in AG(8, 3)."""
80     V = np.array(list(itertools.product(range(3), repeat=n)), dtype=np.int32)
81     reflections = lambda v: sum(1 for i in range(1, n // 2) if v[i] == v[-i])
82
83     # First we list 128 weight-8 vectors with >= 2 reflections.
84     weight8_points = [v for v in V
85                       if np.count_nonzero(v) == 8 # Weight is 8.
86                          and reflections(v) >= 2] # At least 2 reflections.
87
88     # Then we list 256 weight-4 vectors with allowed support and <= 1 reflections.
89     allowed_supports = [
90         (0, 1, 2, 3), (0, 1, 2, 5), (0, 1, 2, 7), (0, 1, 2, 6), (0, 1, 3, 7),
91         (0, 1, 6, 7), (0, 3, 6, 7), (0, 5, 6, 7), (0, 1, 5, 7), (1, 3, 4, 6),
92         (1, 4, 5, 6), (0, 2, 3, 6), (2, 3, 4, 7), (2, 4, 5, 7), (0, 2, 6, 7),
93         (0, 2, 5, 6), (1, 2, 4, 7), (1, 2, 4, 6), (1, 3, 4, 7), (1, 4, 6, 7),
94         (1, 4, 5, 7), (2, 3, 4, 6), (2, 4, 6, 7), (2, 4, 5, 6),
95     ]
96     weight4_points = [
97         v for v in V
98         if np.count_nonzero(v) == 4 # Weight is 4.
99         and tuple(i for i in range(n) if v[i] != 0) in allowed_supports
100         and reflections(v) <= 1] # At most 1 reflection.
101
102     # Finally we add 128 weight-5 vectors with <= 1 reflections.
103     allowed_zeros = [(0, 4, 7), (0, 2, 4), (0, 1, 4), (0, 4, 6),
104                     (1, 2, 6), (2, 6, 7), (1, 2, 7), (1, 6, 7)]
105     weight5_points = [
106         v for v in V
107         if np.count_nonzero(v) == 5 # Weight is 4.
108         and tuple(i for i in range(n) if v[i] == 0) in allowed_zeros
109         and reflections(v) <= 1 # At most 1 reflection.
110         and (v[1] * v[7]) % 3 != 1 and (v[2] * v[6]) % 3 != 1] # Mod conditions.
111
112     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

```
[1 1 1 1 1 1 1 1] [2 1 1 1 1 1 1 1] [0 0 1 0 1 0 2 1] [0 1 0 0 2 0 1 2] [1 0 0 0 0 1 1 1] [2 0 0 0 0 1 1 1] [0 0 1 1 0 1 2 1] [1 0 0 1 1 1 2 0 2]
[1 1 1 1 1 1 2 2] [2 1 1 1 1 1 2 2] [0 0 1 0 1 0 2 2] [0 1 0 0 2 0 2 2] [1 0 0 0 0 1 2 2] [2 0 0 0 0 1 2 2] [0 0 1 1 0 1 2 2] [1 0 0 1 1 1 2 2]
[1 1 1 1 2 1 1 1] [2 1 1 1 2 1 1 1] [0 0 1 0 1 1 0 2] [0 1 0 0 2 1 1 0] [1 0 0 0 0 1 2 2] [2 0 0 0 0 1 2 2] [0 0 1 1 0 2 2 2] [1 0 0 1 1 2 2 0]
[1 1 1 2 1 1 1 1] [2 1 1 2 1 1 1 1] [0 0 1 0 1 1 1 0] [0 1 0 0 2 1 2 0] [1 0 0 0 0 2 1 1] [2 0 0 0 0 2 1 1] [0 0 1 2 0 1 2 1] [1 0 0 1 2 2 0 1]
[1 1 1 2 1 2 1 1] [2 1 1 2 1 2 1 1] [0 0 1 0 1 2 0 2] [0 1 0 0 2 2 1 0] [1 0 0 0 0 2 2 1] [2 0 0 0 0 2 2 1] [0 0 1 2 0 2 2 2] [1 0 0 1 2 2 2 0]
[1 1 1 2 2 1 1 1] [2 1 1 2 2 1 1 1] [0 0 1 0 1 2 2 0] [0 1 0 0 2 2 2 0] [1 0 0 0 0 2 2 2] [2 0 0 0 0 2 2 2] [0 0 1 2 0 2 2 2] [1 0 0 1 2 2 2 0]
[1 1 2 1 2 1 1 1] [2 1 2 1 2 1 1 1] [0 0 1 0 2 0 2 2] [0 1 0 1 1 0 1 2] [1 0 0 1 0 0 1 2] [2 0 0 1 0 0 1 2] [0 0 2 1 0 1 1 2] [1 0 0 2 1 1 0 2]
[1 1 2 1 2 2 1 1] [2 1 2 1 2 2 1 1] [0 0 1 0 2 1 0 2] [0 1 0 1 1 0 2 0] [1 0 0 1 0 0 2 1] [2 0 0 1 0 0 2 1] [0 0 2 1 0 2 1 2] [1 0 0 2 1 1 1 0]
[1 1 2 2 1 2 1 1] [2 1 2 2 1 2 1 1] [0 0 1 0 2 1 2 0] [0 1 0 1 2 0 1 0] [1 0 0 1 0 0 2 2] [2 0 0 1 0 0 2 2] [0 0 2 1 0 2 2 2] [1 0 0 2 1 1 1 0]
[1 1 2 2 2 1 1 1] [2 1 2 2 2 1 1 1] [0 0 1 0 2 2 1 0] [0 1 0 1 2 0 1 0] [1 0 0 1 0 0 1 1] [2 0 0 1 0 0 1 1] [0 0 2 2 0 1 1 1] [1 0 0 2 2 0 1 1]
```

```
78 def get_capset(n: int) -> CapSet:
79     """Returns a 512-cap in AG(8, 3)."""
80     V = np.array(list(itertools.product(range(3), repeat=n)), dtype=np.int32)
81     reflections = lambda v: sum(1 for i in range(1, n // 2) if v[i] == v[-i])
82
83     # First we list 128 weight-8 vectors with >= 2 reflections.
84     weight8_points = [v for v in V
```



*“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”*

```
[1 2 1 2 1 1 2 2] [2 2 1 2 1 1 2 2] [0 0 2 0 2 1 1 0] [0 2 0 0 2 1 2 0] [1 1 2 0 0 0 0 2] [2 1 2 0 0 0 0 2] [0 2 0 1 0 1 1 1] [2 0 0 2 1 1 0 2]
[1 2 1 2 1 2 1 2] [2 2 1 2 1 2 1 2] [0 0 2 0 2 2 0 2] [0 2 0 0 2 2 0 1] [1 1 2 0 0 1 0 0] [2 1 2 0 0 1 0 0] [0 2 0 1 0 2 1 1] [2 0 0 2 1 1 1 0]
[1 2 1 2 1 2 2 2] [2 2 1 2 1 2 2 2] [0 0 2 0 2 2 2 0] [0 2 0 0 2 2 2 0] [1 1 2 0 0 2 0 0] [2 1 2 0 0 2 0 0] [0 2 0 1 0 2 2 1] [2 0 0 2 1 1 2 0]
[1 2 2 1 2 1 1 2] [2 2 2 1 2 1 1 2] [0 0 2 1 1 0 0 2] [0 2 0 1 1 0 0 1] [1 1 2 1 0 0 0 0] [2 1 2 1 0 0 0 0] [0 2 0 2 0 1 1 0] [2 0 0 2 2 1 0 0]
[1 2 2 1 2 2 1 1] [2 2 2 1 2 2 1 1] [0 0 2 1 1 0 0 2] [0 2 0 1 1 0 0 1] [1 1 2 2 0 0 0 0] [2 1 2 2 0 0 0 0] [0 2 0 2 0 1 2 1] [2 0 0 2 2 1 2 0]
[1 2 2 2 1 1 2 1] [2 2 2 2 1 1 2 1] [0 0 2 1 1 0 0 2] [0 2 0 1 1 0 0 1] [1 1 2 2 0 0 0 0] [2 1 2 2 0 0 0 0] [0 2 0 2 0 1 2 1] [2 0 0 2 2 1 2 0]
[1 2 2 2 1 1 2 2] [2 2 2 2 1 1 2 2] [0 0 2 1 2 0 0 1] [0 2 0 2 1 0 0 0] [1 2 0 0 0 0 0 1] [2 2 0 0 0 0 0 1] [0 2 1 1 0 1 2 0] [2 0 1 1 2 2 0 0]
[1 2 2 2 1 2 1 2] [2 2 2 2 1 2 1 2] [0 0 2 1 2 0 0 1] [0 2 0 2 1 0 0 0] [1 2 0 0 0 0 0 1] [2 2 0 0 0 0 0 1] [0 2 1 1 0 2 0 0] [2 0 1 2 1 1 0 0]
[1 2 2 2 1 2 2 2] [2 2 2 2 1 2 2 2] [0 0 2 1 2 0 0 1] [0 2 0 2 1 0 0 0] [1 2 0 0 0 0 0 1] [2 2 0 0 0 0 0 1] [0 2 1 1 0 2 0 0] [2 0 1 2 1 1 0 0]
[1 2 2 2 2 1 1 2] [2 2 2 2 2 1 1 2] [0 0 2 1 2 1 0 0] [0 2 0 2 1 1 0 0] [1 2 1 0 0 0 0 0] [2 2 1 0 0 0 0 0] [0 2 1 2 0 1 0 1] [2 0 2 1 1 2 0 0]
[1 2 2 2 2 2 1 1] [2 2 2 2 2 2 1 1] [0 0 2 1 2 1 0 0] [0 2 0 2 1 1 0 0] [1 2 1 0 0 0 0 0] [2 2 1 0 0 0 0 0] [0 2 1 2 0 1 0 1] [2 0 2 1 1 2 0 0]
[1 2 2 2 2 2 2 1] [2 2 2 2 2 2 2 1] [0 0 2 1 2 1 0 0] [0 2 0 2 1 1 0 0] [1 2 1 0 0 0 0 0] [2 2 1 0 0 0 0 0] [0 2 1 2 0 1 0 1] [2 0 2 1 1 2 0 0]
[1 2 2 2 2 2 2 2] [2 2 2 2 2 2 2 2] [0 0 2 0 2 1 0 0] [0 2 0 2 0 1 0 0] [1 2 1 0 0 0 0 0] [2 2 1 0 0 0 0 0] [0 2 1 2 0 1 0 1] [2 0 2 2 0 2 0 0]
```

```
99     and tuple(1 for i in range(n) if v[i] != 0) in allowed_supports
100     and reflections(v) <= 1] # At most 1 reflection.
101
102     # Finally we add 128 weight-5 vectors with <= 1 reflections.
103     allowed_zeros = [(0, 4, 7), (0, 2, 4), (0, 1, 4), (0, 4, 6),
104                     (1, 2, 6), (2, 6, 7), (1, 2, 7), (1, 6, 7)]
105     weight5_points = [
106         v for v in V
107         if np.count_nonzero(v) == 5 # Weight is 4.
108         and tuple(i for i in range(n) if v[i] == 0) in allowed_zeros
109         and reflections(v) <= 1 # At most 1 reflection.
110         and (v[1] * v[7]) % 3 != 1 and (v[2] * v[6]) % 3 != 1] # Mod conditions.
111
112     return weight8_points + weight4_points + weight5_points
```

AlphaEvolve

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

```
1 class Experiment(experiment.RankExperiment):
2     """Rank tensor decomposition optimization. Assume single-device."""
3
4     def __init__(self, mode, init_rng, config, hypers):
5         self.hypers = hypers
6         super().__init__(mode=mode, init_rng=init_rng, config=config)
7
8     def _get_optimizer(self) -> optax.GradientTransformation:
9         """Returns optimizer."""
10        return optax.adam(self.hypers.learning_rate)
11
12    def _get_init_fn(self) -> jax.nn.initializers.Initializer:
13        """Returns initializer function."""
14        scale = self.hypers.init_scale
15        return initializers.normal(0 + 1j * 0, 1 + 1j * scale, jnp.complex64)
16
17    def _linear_schedule(self, global_step, start: float = 0.0, end: float = 0.0):
18        frac = 1 - global_step / self.config.training_steps
19        return (start - end) * frac + end
20
21    @functools.partial(jax.jit, static_argnums=0)
22    def _update_func(
23        self,
24        decomposition: tuple[jnp.ndarray, jnp.ndarray, jnp.ndarray],
25        opt_state: optax.OptState,
26        global_step: jnp.ndarray,
27        rng: jnp.ndarray,
28    ) -> tuple[
29        tuple[jnp.ndarray, jnp.ndarray, jnp.ndarray],
30        optax.OptState,
31        jnp.ndarray,
32    ]:
33        """A single step of decomposition parameter updates."""
34        # Compute loss and gradients.
35        loss, grads = jax.value_and_grad(
36            lambda decomposition, global_step, rng: jnp.mean(
37                self._loss_fn(decomposition, global_step, rng)
38            )
39        )(decomposition, global_step, rng)
40        # When optimizing real-valued functions of complex variables, we must take
41        # the conjugate of the gradient.
42        grads = jax.tree_util.tree_map(lambda x: x.conj(), grads)
43        # Gradient updates.
44        updates, opt_state = self.opt.update(grads, opt_state, decomposition)
45        decomposition = optax.apply_updates(decomposition, updates)
46        return decomposition, opt_state, loss
47
48    def _loss_fn(
49        self,
50        decomposition: tuple[jnp.ndarray, jnp.ndarray, jnp.ndarray],
51        global_step,
52        rng: cheX.PRNGKey,
53    ) -> jnp.ndarray:
54        """Computes (batched) loss on learned decomposition."""
55        # Compute reconstruction loss.
56        # ... = self._reconstruction_loss(decomposition) # / 0.0 # 0.0
```

Iteration 1

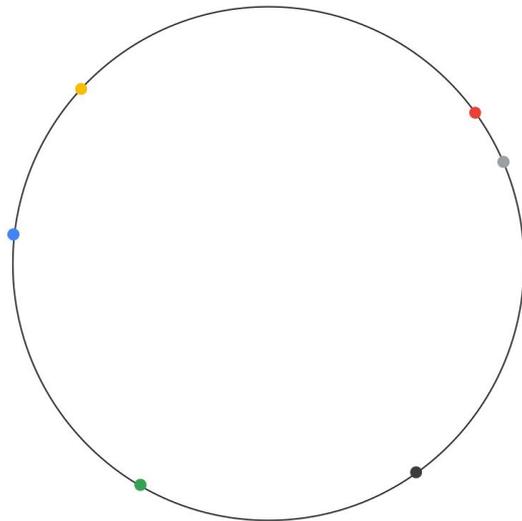


Ingredientes de AlphaEvolve

LLMs



creatividad
(generación de
código)



Evaluadores



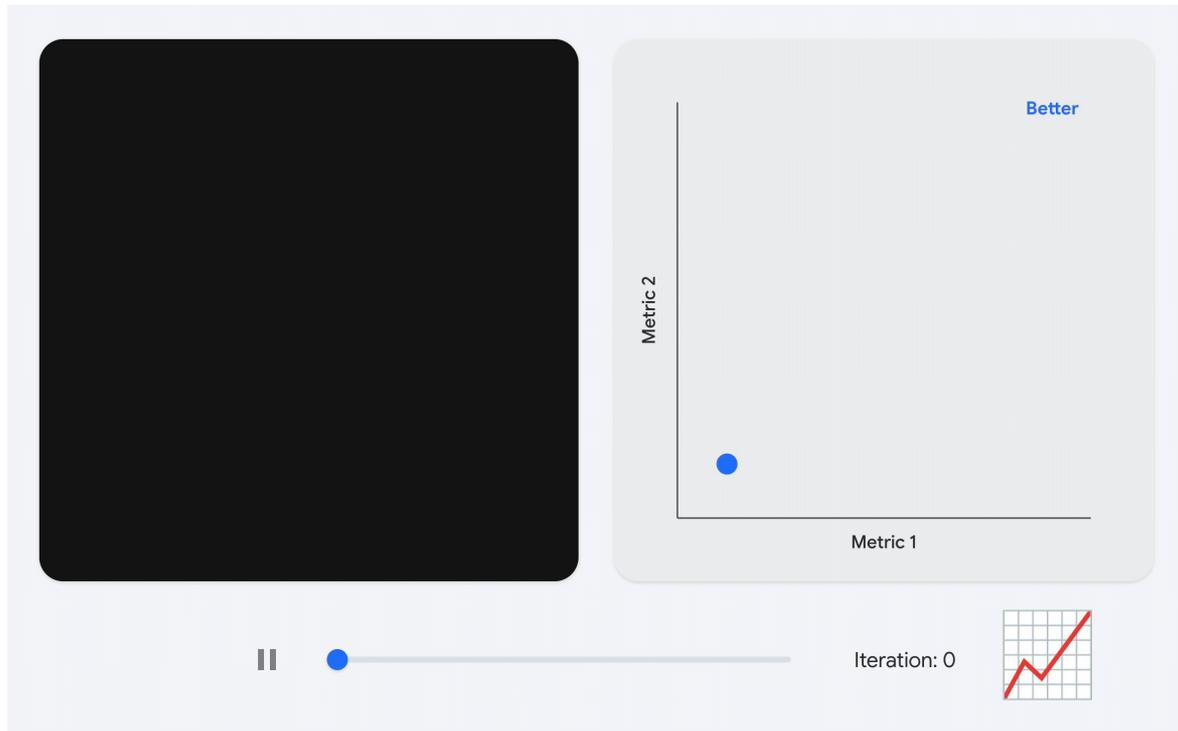
verificación
rigurosa

Evolución



mejora iterativa

Ingredientes de AlphaEvolve



Thanks,
[Gemini Canvas!](#)



Implementación de AlphaEvolve



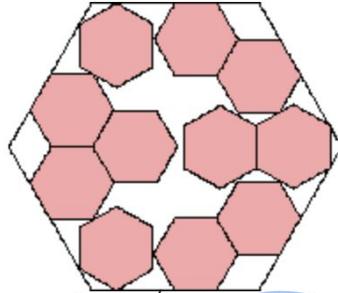
Ejemplo: Un Problema de Empaquetamiento

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



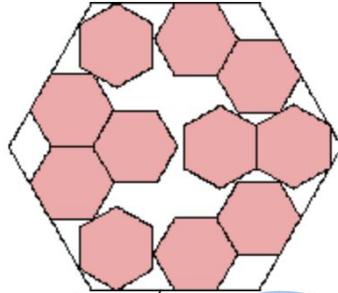
$$s = 5(3 + \sqrt{3})/6 = 3.943+$$

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?

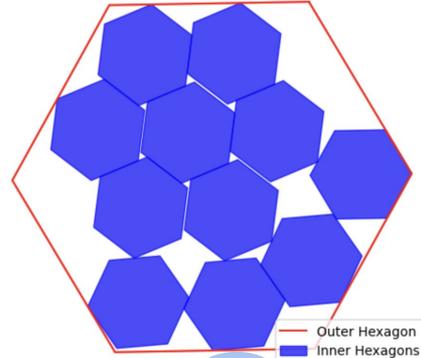
Estado del arte



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

Construcción encontrada por
Maurizio Morandi en 2015

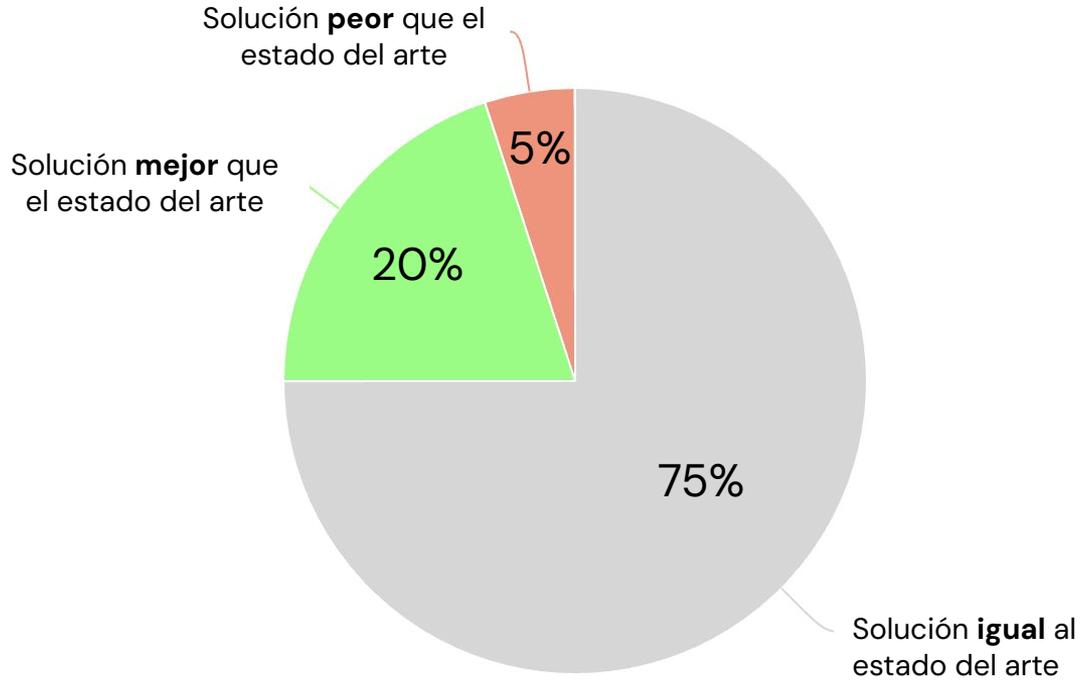
AlphaEvolve



$$s = 3.93$$

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)

```
80 -45.0 +45.14 00
81 # EIGLE=BLOCK-START
82 def _get_optimizer(self) -> optax.GradientTransformation:
83     """Returns optimizer."""
84     return optax.adam(self.hypers.learning_rate)
85 + return optax.adamw
86 + self.hypers.learning_rate, weight_decay=self.hypers.weight_decay
87 }
88
89 def _get_init_fn(self) -> jax.nn.initializers.Initializer:
90     """Returns initializer function."""
91     return initializers.normal(0.0, self.hypers.init_scale, jnp.complex64)
92 + # Initialize with a smaller scale to encourage finding low-rank solutions.
93 + # Increase scale slightly for better exploration.
94 + scale = self.hypers.init_scale
95 + return initializers.normal(0 - 1j * 0, scale * 0.2, jnp.complex64)
96
97
98 -80.0 -80.00 00
99 # Gradient updates.
100 updates, opt_state = self.opt.update(grads, opt_state, decomposition)
101 decomposition = optax.apply_updates(decomposition, updates)
102 + # Add a small amount of gradient noise to help with exploration
103 + rng, g_noise_rng = jax.random.split(rng)
104 decomposition = jax.tree_util.tree_map(
105     lambda x: x
106     + self.hypers.grad_noise_std * jax.random.normal(g_noise_rng, x.shape),
107     decomposition,
108 )
109
110 + # Add noise to the decomposition parameters (exploration).
111 + noise_rng = jax.random.split(rng)
112 + noise_std = self._linear_schedule(
113     global_step, start=self.hypers.noise_std, end=0.0
114 )
115 decomposition = jax.tree_util.tree_map(
116     lambda x: x + noise_std * jax.random.normal(noise_rng, x.shape),
117     decomposition,
118 )
119
120 + # Cyclical annealing for clipping threshold.
121 cycle_length = 2000 # Number of steps per cycle
122 cycle_progress = (
123     global_step % cycle_length
124 ) / cycle_length # Normalized progress within the current cycle [0, 1]
125
126 + # Map cycle progress to a sinusoidal curve. Ranges from 0 to 1.
127 clip_threshold_multiplier = (1 + jnp.cos(2 * jnp.pi * cycle_progress)) / 2
128 clip_threshold = self.hypers.clip_min + clip_threshold_multiplier * (
129     self.hypers.clip_max - self.hypers.clip_min
130 )
131
132 def soft_clip(x, threshold):
133     # Clipping the real and imaginary parts separately.
134     x_re = jnp.real(x)
135     x_im = jnp.imag(x)
136
137     x_re_clipped = jnp.where(
138         x_re > threshold, threshold + (x_re - threshold) * 0.1, x_re
139     )
140     x_re_clipped = jnp.where(
141         x_re_clipped < -threshold,
142         -threshold + (x_re_clipped + threshold) * 0.1,
143         x_re_clipped,
144     )
145
146     x_im_clipped = jnp.where(
147         x_im > threshold, threshold + (x_im - threshold) * 0.1, x_im
148     )
149     x_im_clipped = jnp.where(
150         x_im_clipped < -threshold,
151         -threshold + (x_im_clipped + threshold) * 0.1,
152         x_im_clipped,
153     )
154
155     return x_re_clipped + 1j * x_im_clipped
156
157 decomposition = jax.tree_util.tree_map(
158     lambda x: soft_clip(x, clip_threshold), decomposition
159 )
160
161 return decomposition, opt_state, loss
162
163 def loss_fn(
164     self,
165     **kwargs
166 ):
167     """Computes (batched) loss on learned decomposition."""
168     # Compute reconstruction loss.
169     rec_tensor = self._decomposition_to_tensor(decomposition) # (B, W, M, P)
170
171     + # Add noise to the target tensor (robustness).
172     + rng, noise_rng = jax.random.split(rng)
173     + target_noise = self.hypers.target_noise_std * jax.random.normal(
174     + noise_rng, self.target_tensor.shape
175     )
```


Resultados en Multiplicación de Matrices

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

```
80 -45,9 +45,14 88
81 # EIGLVE-BLOCK-START
82 def _get_optimizer(self) -> optax.GradientTransformation:
83     """Returns optimizer."""
84     return optax.adamw(self.hypers.learning_rate)
85 + return optax.adamw
86 + self.hypers.learning_rate, weight_decay=self.hypers.weight_decay
87 }
88
89 def _get_init_fn(self) -> jax.nn.initializers.Initializer:
90     """Returns initializer function."""
91     return initializers.normal(0, self.hypers.init_scale, jnp.complex64)
92 + # Initialize with a smaller scale to encourage finding low-rank solutions.
93 + # Increase scale slightly for better exploration.
94 + scale = self.hypers.init_scale
95 + return initializers.normal(0 = 1j * 0, scale = 0.2, jnp.complex64)
96
97 @-80,0 -80,0 88
98 # Gradient updates.
99 updates, opt_state = self.opt.update(grade, opt_state, decomposition)
100 decomposition = optax.apply_updates(decomposition, updates)
101 + # Add a small amount of gradient noise to help with exploration
102 + noise_rng = jax.random.split(rng)
103 + decomposition = jax.tree_util.tree_map(
104     lambda x: x
105     + self.hypers.grad_noise_std * jax.random.normal(noise_rng, x.shape),
106     decomposition,
107 )
108
109 + # Add noise to the decomposition parameters (exploration).
110 + noise_rng = jax.random.split(rng)
111 + noise_std = self._linear_schedule(
112     global_step, start=self.hypers.noise_std, end=0.0
113 )
114 + decomposition = jax.tree_util.tree_map(
115     lambda x: x + noise_std * jax.random.normal(noise_rng, x.shape),
116     decomposition,
117 )
118
119 + # Cyclical annealing for clipping threshold.
120 cycle_length = 2000 # Number of steps per cycle
121 cycle_progress = (
122     global_step % cycle_length
123 ) / cycle_length # Normalized progress within the current cycle [0, 1)
124
125 + # Map cycle progress to a sinusoidal curve, ranges from 0 to 1.
126 clip_threshold_multiplier = (1 + jnp.cos(jnp.pi * cycle_progress)) / 2
127
128 + clip_threshold = self.hypers.clip_min + clip_threshold_multiplier * (
129     self.hypers.clip_max - self.hypers.clip_min
130 )
131
132 def _soft_clip(x, threshold):
133     # Clipping the real and imaginary parts separately.
134     x_re = jnp.real(x)
135     x_im = jnp.imag(x)
136
137     x_re_clipped = jnp.where(
138         x_re > threshold, threshold + (x_re - threshold) * 0.1, x_re
139     )
140     x_re_clipped = jnp.where(
141         x_re_clipped < -threshold,
142         -threshold + (x_re_clipped + threshold) * 0.1,
143         x_re_clipped,
144     )
145
146     x_im_clipped = jnp.where(
147         x_im > threshold, threshold + (x_im - threshold) * 0.1, x_im
148     )
149     x_im_clipped = jnp.where(
150         x_im_clipped < -threshold,
151         -threshold + (x_im_clipped + threshold) * 0.1,
152         x_im_clipped,
153     )
154
155     return x_re_clipped + 1j * x_im_clipped
156
157 decomposition = jax.tree_util.tree_map(
158     lambda x: soft_clip(x, clip_threshold), decomposition
159 )
160
161 return decomposition, opt_state, loss
162
163 def _loss_fn(
164     @-91,13 +156,86 88
165     """Computes (batched) loss on learned decomposition."""
166     # Compute reconstruction loss.
167     rec_tensor = self._decomposition_to_tensor(decomposition) # (B, N, M, P)
168
169     + # Add noise to the target tensor (robustness).
170     + rng, noise_rng = jax.random.split(rng)
171     + target_noise = self.hypers.target_noise_std * jax.random.normal(
172     + noise_rng, self.target_tensor.shape
173 )
```

```
1 @@ -91,13 +156,86 @@
2     """Computes (batched) loss on learned decomposition."""
3     # Compute reconstruction loss.
4     rec_tensor = self._decomposition_to_tensor(decomposition) # (B, N,
5     M, P)
6     + # Discretization loss (encourage entries to be multiples of 1/2 or
7     + integer).
8     def dist_to_half_ints(x):
9     +
10    + def dist_to_ints(x):
11    +
12    + discretization_loss = 0.0
13    + for factor in decomposition:
14    +     discretization_loss += jnp.mean(dist_to_half_ints(factor))
15    +     discretization_loss += jnp.mean(dist_to_ints(factor))
16    +
17    + discretization_loss /= (
18    +     len(decomposition) * 2
19    + ) # average across all factors and loss components
20    +
21    + discretization_weight = self._linear_schedule(
22    +     global_step, start=0.0, end=self.hypers.discretization_weight
23    + )
24    +
25    + # Cosine annealing for half-integer loss.
26    + cycle_length = self.config.training_steps // 4 # Number of steps
27    + per cycle
28    + cycle_progress = (
29    +     global_step % cycle_length
30    + ) / cycle_length # Normalized progress within the current cycle
31    + [0, 1)
32    + half_int_multiplier = (1 + jnp.cos(jnp.pi * cycle_progress)) / 2
33    + half_int_multiplier = (
34    +     1 - self.hypers.half_int_start
35    + ) * half_int_multiplier + self.hypers.half_int_start
36    +
37    + total_loss = (
38    +     rec_loss
39    +     + discretization_weight * discretization_loss *
40    +     half_int_multiplier
```

Resultados en Multiplicación de Matrices

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

```
## -48,9 +48,14 ##
# EVOLVE-BLOCK-START
def get_optimizer(self) -> optax.GradientTransformation:
    """Returns optimizer."""
    return optax.adamw(self.hyper.learning_rate)
+ return optax.adamw
+ self.hyper.learning_rate, weight_decay=self.hyper.weight_decay
}
}

def get_init_fn(self) -> jax.nn.initializers.Initializer:
    """Returns initializer function."""
    return initializers.normal(0.0, self.hyper.init_scale, jnp.complex64)
+ # Initialize with a smaller scale to encourage finding low-rank solutions.
+ # Increase scale slightly for better exploration.
+ scale = self.hyper.init_scale
+ return initializers.normal(0.1j * 0.0, scale * 0.2, jnp.complex64)

## -80,0 +80,80 ##
# 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.hyper.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.hyper.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, range from 0 to 1.
+ clip_threshold_multiplier = (1 + jnp.cos(2 * jnp.pi * cycle_progress)) / 2
+ clip_threshold = self.hyper.clip_min + clip_threshold_multiplier * (
+     self.hyper.clip_max - self.hyper.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

## -91,13 +96,86 ##
def loss_fn(
    """Computes (batched) loss on learned decomposition."""
    # Compute reconstruction loss.
    rec_tensor = self._decomposition_to_tensor(decomposition) # (B, W, M, P)
+ # Add noise to the target tensor (robustness).
+ rng, noise_rng = jax.random.split(rng)
+ target_noise = self.hyper.target_noise_std * jax.random.normal(
+     noise_rng, self.target_tensor.shape
+ )
+ )
+ )
```

```
1 @@ -117,6 +255,18 @@
2 return hyper.zipit([
3     hyper.uniform('init_scale', hyper.interval(0.2, 1.5)),
4     hyper.uniform('learning_rate', hyper.interval(0.05, 0.3)),
5     hyper.uniform('init_scale', hyper.interval(0.1, 1.0)),
6 + hyper.uniform('learning_rate', hyper.interval(0.01, 0.2)),
7 + hyper.uniform('discretization_weight', hyper.interval(0.0, 0.1))
8 +
9 + hyper.uniform('hallucination_prob', hyper.interval(0.0, 0.2)),
10 + hyper.uniform('hallucination_scale', hyper.interval(0.0, 0.2)),
11 + hyper.uniform('noise_std', hyper.interval(0.0, 0.01)),
12 + hyper.uniform('target_noise_std', hyper.interval(0.0, 0.01)),
13 + hyper.uniform('weight_decay', hyper.interval(0.00001, 0.001)),
14 + hyper.uniform('clip_min', hyper.interval(0.0, 0.5)),
15 + hyper.uniform('clip_max', hyper.interval(1.0, 3.0)),
16 + hyper.uniform('large_value_penalty_weight', hyper.interval(0.0,
17 + 0.01)),
18 + # Add noise to the gradient to aid in exploration.
19 + hyper.uniform('grad_noise_std', hyper.interval(0.0, 0.001)),
20 + hyper.uniform('half_int_start', hyper.interval(0.0, 1.0)),
21 ])
```

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	$\langle m, n, p \rangle$	mejor #mult	AlphaEvolve	$\langle m, n, p \rangle$	mejor #mult	AlphaEvolve
$\langle 2, 2, 2 \rangle$	7 [95]	7	$\langle 2, 3, 6 \rangle$	30 [93]	30	$\langle 3, 4, 4 \rangle$	38 [93]	38
$\langle 2, 2, 3 \rangle$	11 [93]	11	$\langle 2, 3, 7 \rangle$	35 [93]	35	$\langle 3, 4, 5 \rangle$	47 [26]	47
$\langle 2, 2, 4 \rangle$	14 [93]	14	$\langle 2, 3, 8 \rangle$	40 [93]	40	$\langle 3, 4, 6 \rangle$	56 [48]	54
$\langle 2, 2, 5 \rangle$	18 [93]	18	$\langle 2, 3, 9 \rangle$	45 [93]	45	$\langle 3, 4, 7 \rangle$	66 [91]	63
$\langle 2, 2, 6 \rangle$	21 [93]	21	$\langle 2, 3, 10 \rangle$	50 [93]	50	$\langle 3, 4, 8 \rangle$	75 [91]	74
$\langle 2, 2, 7 \rangle$	25 [93]	25	$\langle 2, 4, 4 \rangle$	26 [93]	26	$\langle 3, 5, 5 \rangle$	58 [91]	58
$\langle 2, 2, 8 \rangle$	28 [93]	28	$\langle 2, 4, 5 \rangle$	33 [42]	32	$\langle 3, 5, 6 \rangle$	70 [48]	68
$\langle 2, 2, 9 \rangle$	32 [93]	32	$\langle 2, 4, 6 \rangle$	39 [93]	39	$\langle 3, 5, 7 \rangle$	82 [91]	80
$\langle 2, 2, 10 \rangle$	35 [93]	35	$\langle 2, 4, 7 \rangle$	46 [93]	45	$\langle 4, 4, 4 \rangle$	49 [95]	48
$\langle 2, 2, 11 \rangle$	39 [93]	39	$\langle 2, 4, 8 \rangle$	52 [93]	51	$\langle 4, 4, 5 \rangle$	62 [47]	61
$\langle 2, 2, 12 \rangle$	42 [93]	42	$\langle 2, 5, 5 \rangle$	40 [93]	40	$\langle 4, 4, 6 \rangle$	73 [48]	73
$\langle 2, 2, 13 \rangle$	46 [93]	46	$\langle 2, 5, 6 \rangle$	48 [93]	47	$\langle 4, 4, 7 \rangle$	87 [93, 95]	85
$\langle 2, 2, 14 \rangle$	49 [93]	49	$\langle 3, 3, 3 \rangle$	23 [52]	23	$\langle 4, 4, 8 \rangle$	98 [95]	96
$\langle 2, 2, 15 \rangle$	53 [93]	53	$\langle 3, 3, 4 \rangle$	29 [93]	29	$\langle 4, 4, 9 \rangle$	104 [92]	108
$\langle 2, 2, 16 \rangle$	56 [93]	56	$\langle 3, 3, 5 \rangle$	36 [93]	36	$\langle 4, 5, 5 \rangle$	76 [26]	76
$\langle 2, 3, 3 \rangle$	15 [93]	15	$\langle 3, 3, 6 \rangle$	40 [93]	40	$\langle 4, 5, 6 \rangle$	93 [48]	90
$\langle 2, 3, 4 \rangle$	20 [93]	20	$\langle 3, 3, 7 \rangle$	49 [93]	49	$\langle 5, 5, 5 \rangle$	93 [72]	93
$\langle 2, 3, 5 \rangle$	25 [93]	25	$\langle 3, 3, 8 \rangle$	55 [93]	55	$\langle 6, 6, 6 \rangle$	153 [72]	156

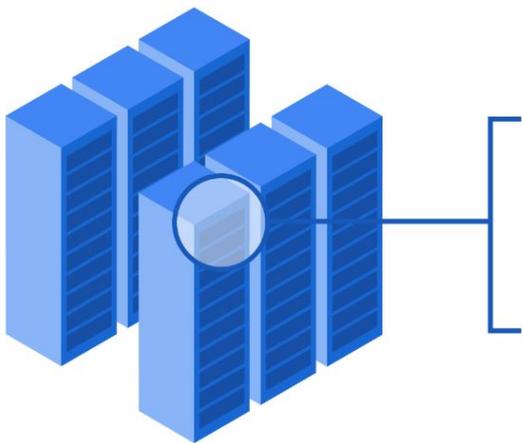
Resultados en Multiplicación de Matrices

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

$\langle m, n, p \rangle$	mejor #mult	AlphaEvolve	$\langle m, n, p \rangle$	mejor #mult	AlphaEvolve	$\langle m, n, p \rangle$	mejor #mult	AlphaEvolve
$\langle 2, 2, 2 \rangle$	7 [95]	7	$\langle 2, 3, 6 \rangle$	30 [93]	30	$\langle 3, 4, 4 \rangle$	38 [93]	38
$\langle 2, 2, 3 \rangle$	11 [93]	11	$\langle 2, 3, 7 \rangle$	35 [93]	35	$\langle 3, 4, 5 \rangle$	47 [26]	47
$\langle 2, 2, 4 \rangle$	14 [93]	14	$\langle 2, 3, 8 \rangle$	40 [93]	40	$\langle 3, 4, 6 \rangle$	56 [48]	54
$\langle 2, 2, 5 \rangle$	18 [93]	18	$\langle 2, 3, 9 \rangle$	45 [93]	45	$\langle 3, 4, 7 \rangle$	66 [91]	63
$\langle 2, 2, 6 \rangle$	21 [93]	21	$\langle 2, 3, 10 \rangle$	50 [93]	50	$\langle 3, 4, 8 \rangle$	75 [91]	74
$\langle 2, 2, 7 \rangle$	25 [93]	25	$\langle 2, 4, 4 \rangle$	26 [93]	26	$\langle 3, 5, 5 \rangle$	58 [91]	58
$\langle 2, 2, 8 \rangle$	28 [93]	28	$\langle 2, 4, 5 \rangle$	33 [42]	32	$\langle 3, 5, 6 \rangle$	70 [48]	68
$\langle 2, 2, 9 \rangle$	32 [93]	32	$\langle 2, 4, 6 \rangle$	39 [93]	39	$\langle 3, 5, 7 \rangle$	82 [91]	80
$\langle 2, 2, 10 \rangle$	35 [93]	35	$\langle 2, 4, 7 \rangle$	46 [93]	45	$\langle 4, 4, 4 \rangle$	49 [95]	48
$\langle 2, 2, 11 \rangle$	39 [93]	39	$\langle 2, 4, 8 \rangle$	52 [93]	51	$\langle 4, 4, 5 \rangle$	62 [47]	61
$\langle 2, 2, 12 \rangle$	42 [93]	42	$\langle 2, 5, 5 \rangle$	40 [93]	40	$\langle 4, 4, 6 \rangle$	73 [48]	73
$\langle 2, 2, 13 \rangle$	46 [93]	46	$\langle 2, 5, 6 \rangle$	48 [93]	47	$\langle 4, 4, 7 \rangle$	87 [93, 95]	85
$\langle 2, 2, 14 \rangle$	49 [93]	49	$\langle 3, 3, 3 \rangle$	23 [52]	23	$\langle 4, 4, 8 \rangle$	98 [95]	96
$\langle 2, 2, 15 \rangle$	53 [93]	53	$\langle 3, 3, 4 \rangle$	29 [93]	29	$\langle 4, 4, 9 \rangle$	104 [92]	108
$\langle 2, 2, 16 \rangle$	56 [93]	56	$\langle 3, 3, 5 \rangle$	36 [93]	36	$\langle 4, 5, 5 \rangle$	76 [26]	76
$\langle 2, 3, 3 \rangle$	15 [93]	15	$\langle 3, 3, 6 \rangle$	40 [93]	40	$\langle 4, 5, 6 \rangle$	93 [48]	90
$\langle 2, 3, 4 \rangle$	20 [93]	20	$\langle 3, 3, 7 \rangle$	49 [93]	49	$\langle 5, 5, 5 \rangle$	93 [72]	93
$\langle 2, 3, 5 \rangle$	25 [93]	25	$\langle 3, 3, 8 \rangle$	55 [93]	55	$\langle 6, 6, 6 \rangle$	153 [72]	156

Mejorando el Ecosistema de Computación de Google

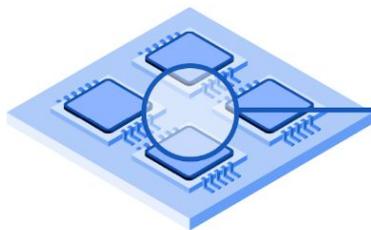
Usamos AlphaEvolve para mejorar algoritmos que ya están en **producción** dentro de Alphabet



Optimizar centros de datos
Distribución de tareas en Borg

0.7%

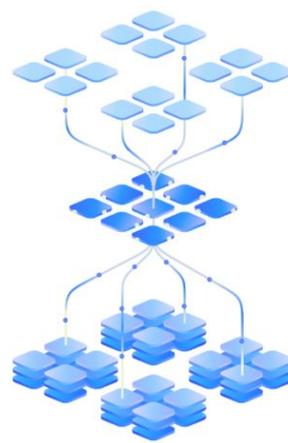
recursos recuperados



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 un sistema distribuido que puede ser computacionalmente costoso
 - Generación de soluciones mediante LLMs + Evaluación automática

Índice

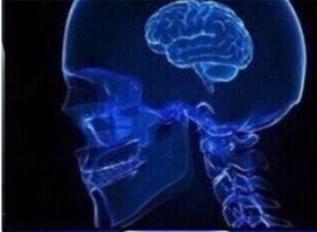
1. Introducción
2. AlphaTensor
3. FunSearch y AlphaEvolve
4. **Conclusiones**



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!