ICCS 311
FINAL PROJECT PRESENTATION

BY ENZE YU (U6580537)
QI ZHOU (U6580536)

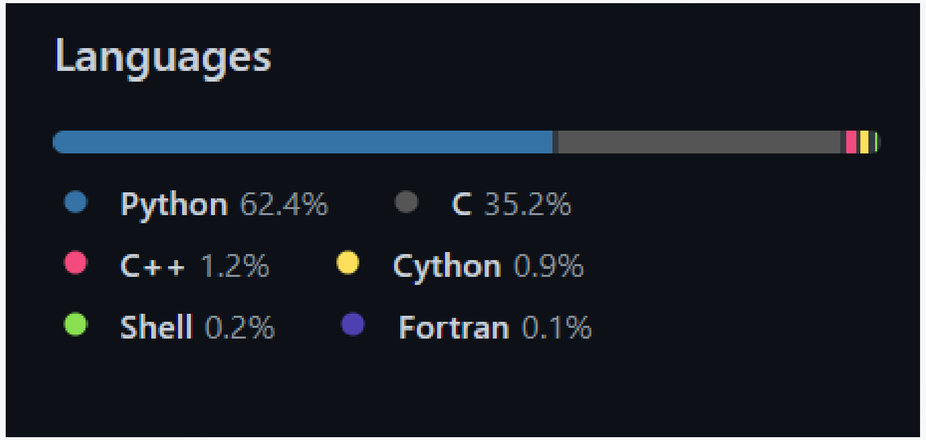
CONTENT

OBJECTIVE

- This project aims to significantly enhance the computational performance of 4 functions from the Numpy library by reimplementing it in Rust/C with parallel processing capabilities and providing a Python wrapper for easy integration.

ABOUT NUMPY

- Have you ever wondered how NumPy performs its complex computations so quickly?
- NumPy is implemented in C which allows NumPy to perform operations directly on contiguous memory blocks, reducing the need for intermediate data structures. It minimizes overhead and executes computations much faster than pure Python.



WRAPPER

We used wrapper such as Maturin tools to extend our function written in Rust/C to Python. This allow us to import our Rust/C code like a package and call the function in Python.

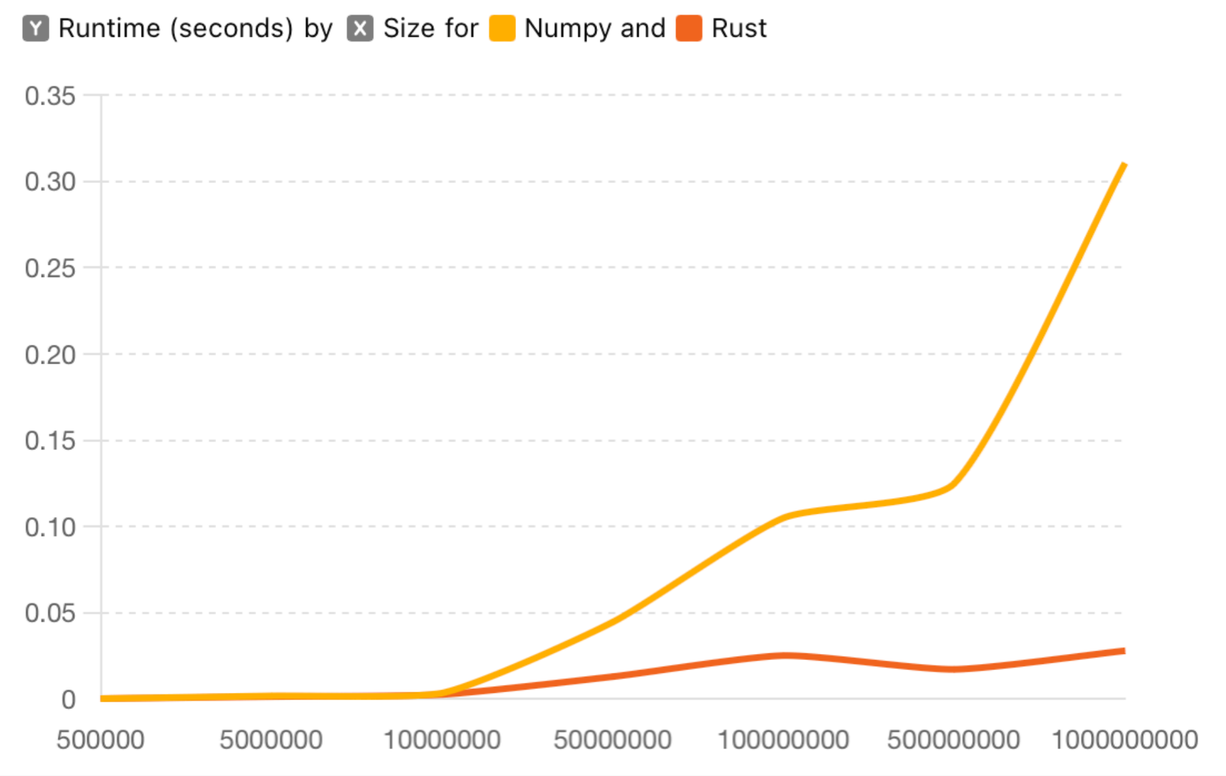
DOT PRODUCT

$$\langle x, y \rangle = \sum_{i=1}^{n} x_i y_i$$

$$\vec{A}^T = \begin{bmatrix} A_1 & A_2 & A_3 \end{bmatrix} \qquad \vec{B} = \begin{bmatrix} B_1 \\ B_2 \\ B_3 \end{bmatrix}$$

$$\begin{bmatrix} A_1 & A_2 & A_3 \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ B_3 \end{bmatrix} = A_1B_1 + A_2B_2 + A_3B_3 = \vec{A}.\vec{B}$$

Implemented by Rust using SIMD



	Size	Numpy Time (s)	Rust Time (s)	Speedup Factor
1	500 thousand	0.000154	0.000144	1.07
2	5 million	0.001758	0.001332	1.32
3	10 million	0.003408	0.002571	1.33
4	50 million	0.044587	0.01291	3.45
5	100 million	0.104933	0.025128	4.18
6	500 million	0.124933	0.017128	7.29
7	1 billion	0.310547	0.027921	11.12

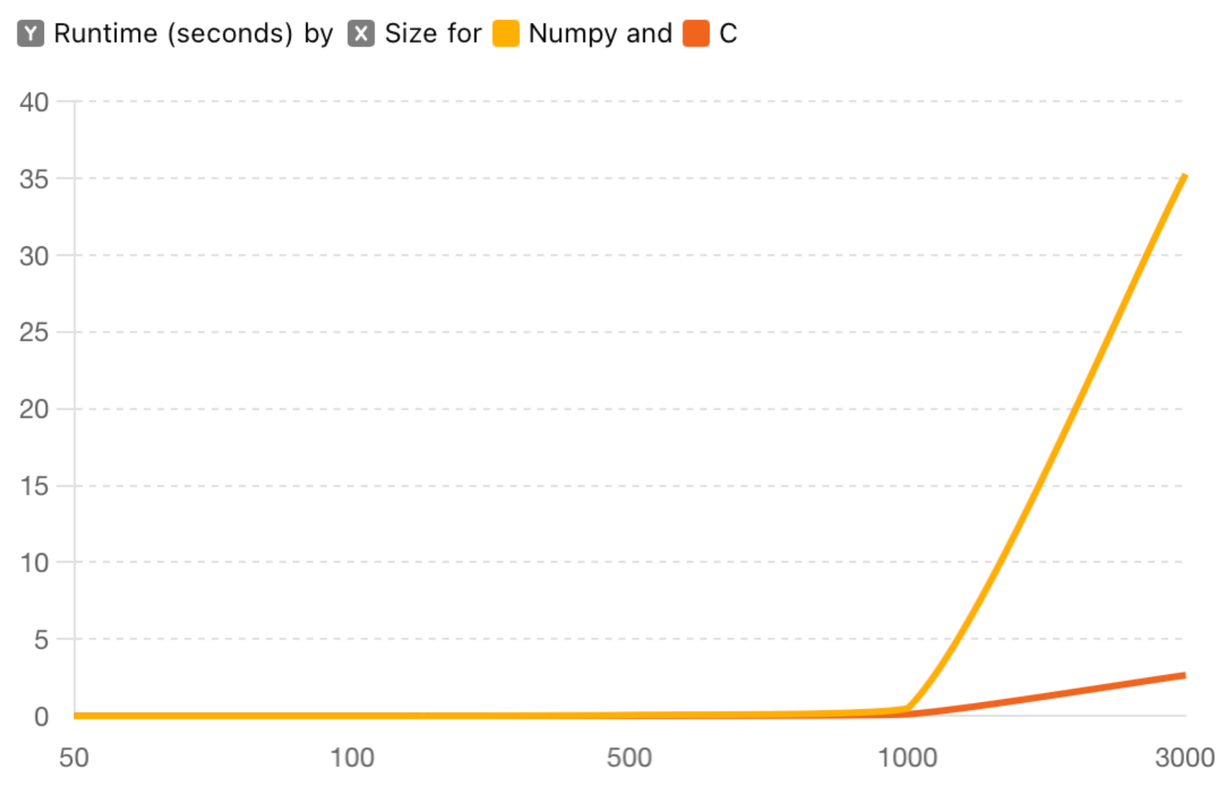
All simple taken from M2 8-core cpu

MATRIX MULTIPLICATION

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix} \cdot \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

$$= \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \\ a_{31}b_{11} + a_{32}b_{21} & a_{31}b_{12} + a_{32}b_{22} \end{bmatrix}$$

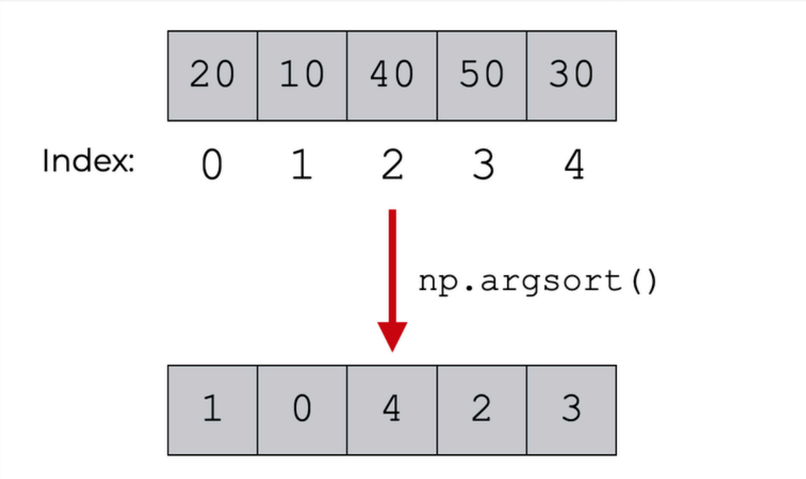
Implemented by C using OpenMP and blocking



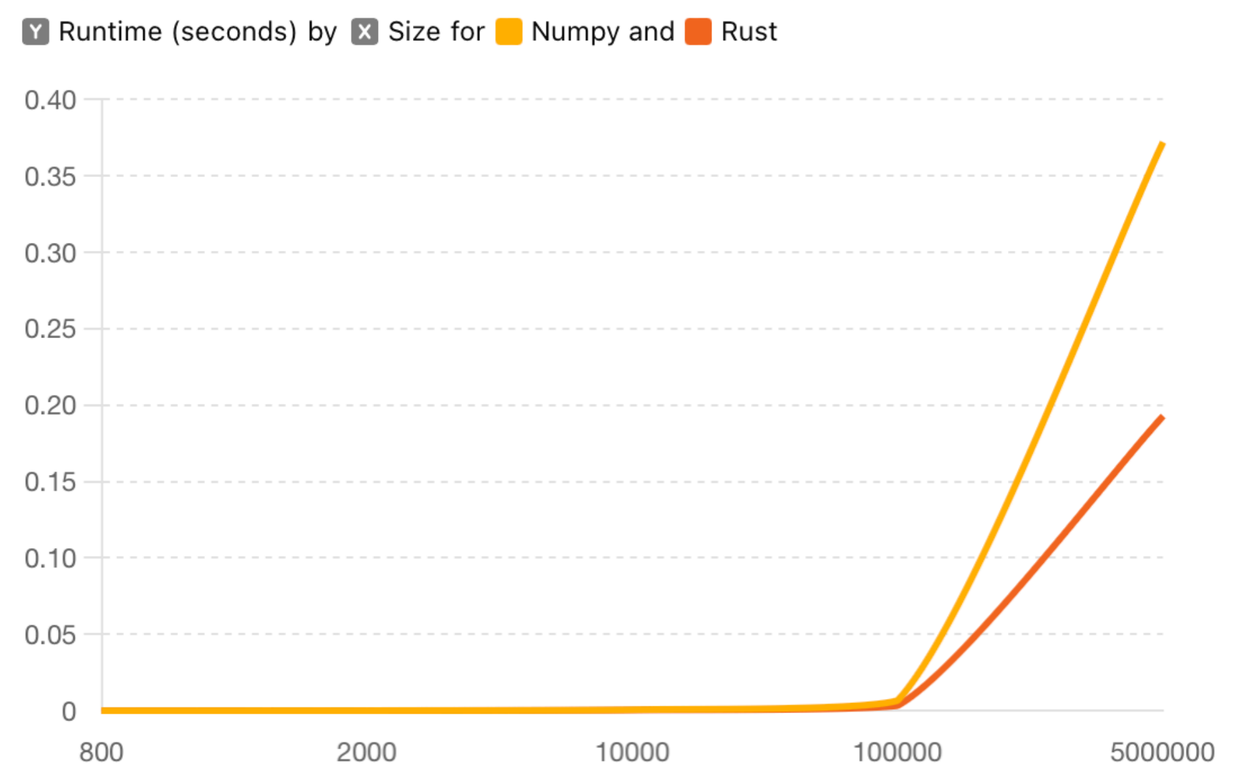
	Size	Numpy Time (s)	Rust Time (s)	Speedup Factor
1	50	0.000207	0.000226	0.92
2	100	0.000982	0.000366	2.68
3	500	0.060273	0.015104	3.99
4	1000	0.491882	0.114127	4.31
5	3000	35.257888	2.655063	13.28

All simple taken from M2 8-core cpu

ARGSORT



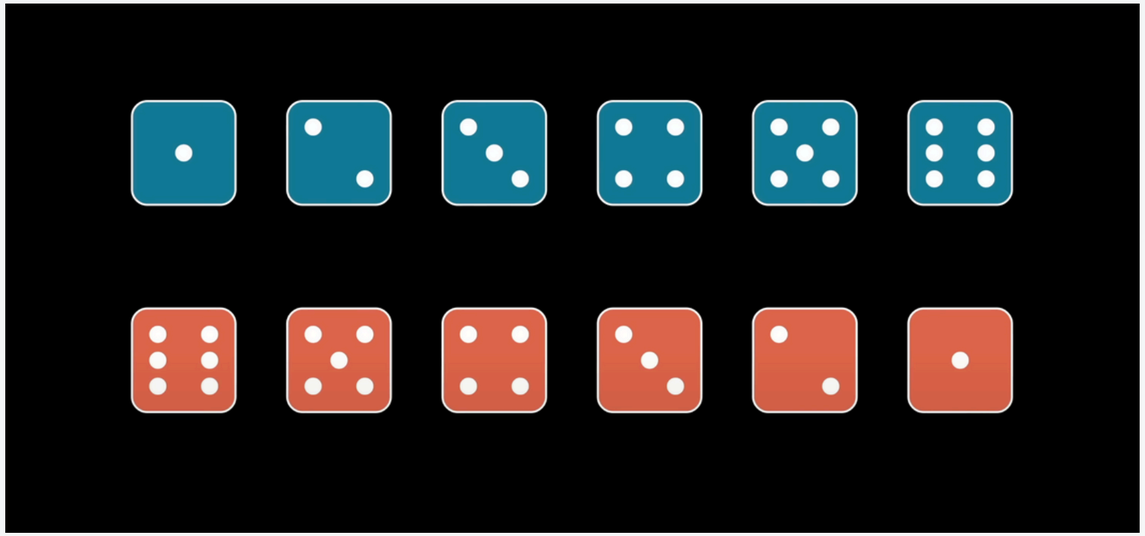
Implemented by
Rust using Rayon

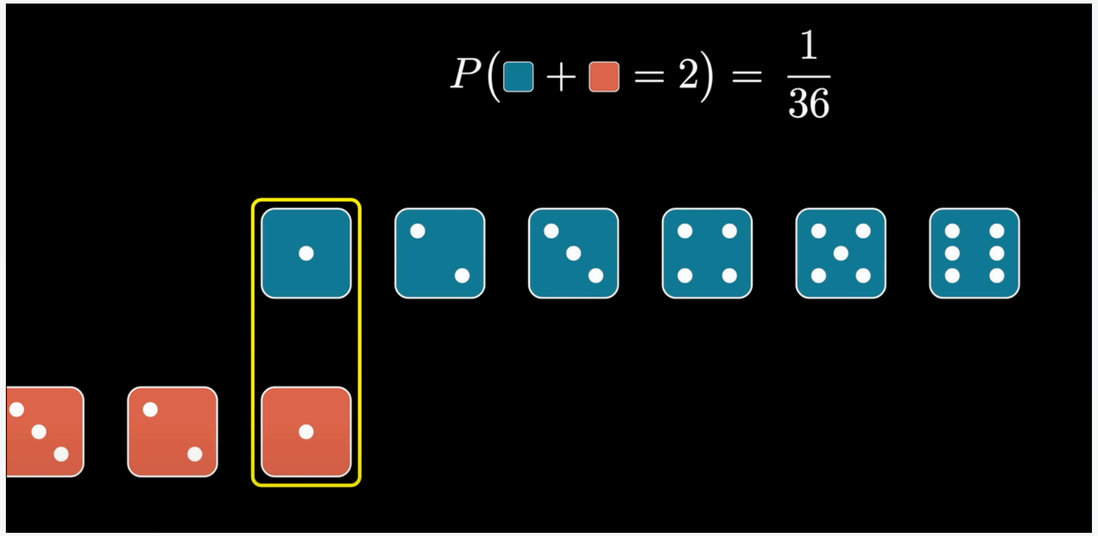


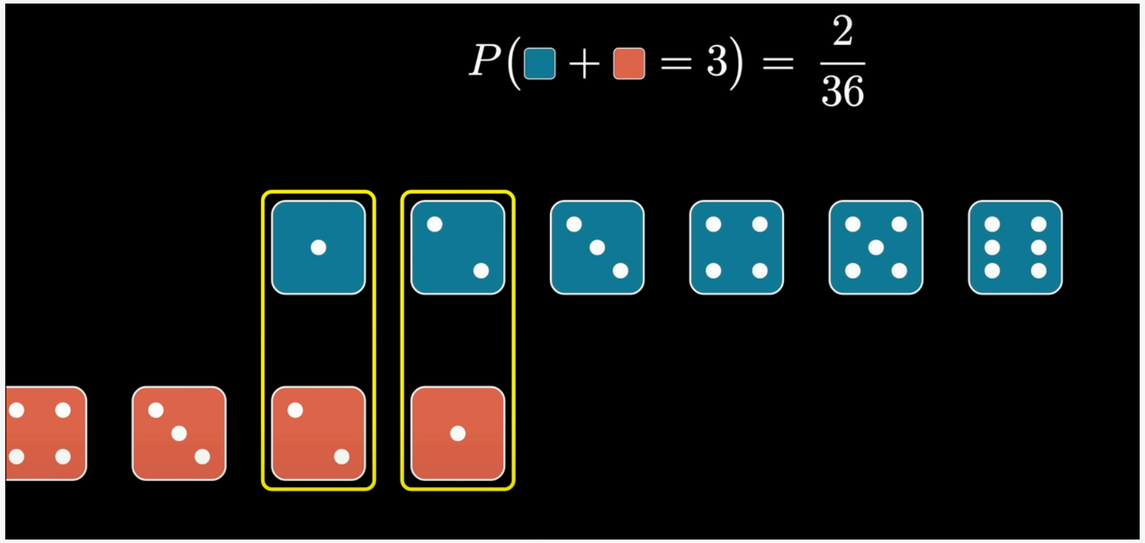
	Size	Numpy Time (s)	Rust Time (s)	Speedup Factor
1	800	5.6e-05	5.2e-05	1.08
2	2000	0.000136	0.000139	0.98
3	10000	0.000709	0.000525	1.35
4	100000	0.00675	0.003369	2.0
5	5 million	0.37189	0.192737	1.93

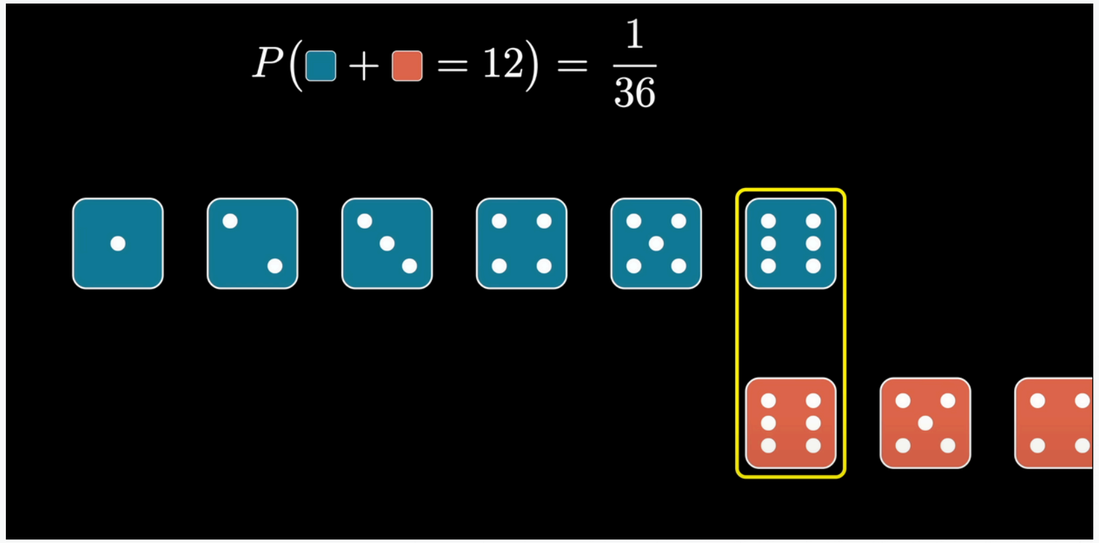
All simple taken
from M2 8-core cpu

CONVOLVE




$P(\square + \square = 2) = \frac{1}{36}$


$P(\square + \square = 3) = \frac{2}{36}$


$P(\square + \square = 12) = \frac{1}{36}$

CONVOLVE

$$P(\square + \square = 2) = a_1 \cdot b_1$$

$$P(\square + \square = 3) = a_1 \cdot b_2 + a_2 \cdot b_1$$

$$P(\square + \square = 4) = a_1 \cdot b_3 + a_2 \cdot b_2 + a_3 \cdot b_1$$

$$P(\square + \square = 5) = a_1 \cdot b_4 + a_2 \cdot b_3 + a_3 \cdot b_2 + a_4 \cdot b_1$$

$$P(\square + \square = 6) = a_1 \cdot b_5 + a_2 \cdot b_4 + a_3 \cdot b_3 + a_4 \cdot b_2 + a_5 \cdot b_1$$

$$P(\square + \square = 7) = a_1 \cdot b_6 + a_2 \cdot b_5 + a_3 \cdot b_4 + a_4 \cdot b_3 + a_5 \cdot b_2 + a_6 \cdot b_1$$

$$P(\square + \square = 8) = a_2 \cdot b_6 + a_3 \cdot b_5 + a_4 \cdot b_4 + a_5 \cdot b_3 + a_6 \cdot b_2$$

$$P(\square + \square = 9) = a_3 \cdot b_6 + a_4 \cdot b_5 + a_5 \cdot b_4 + a_6 \cdot b_3$$

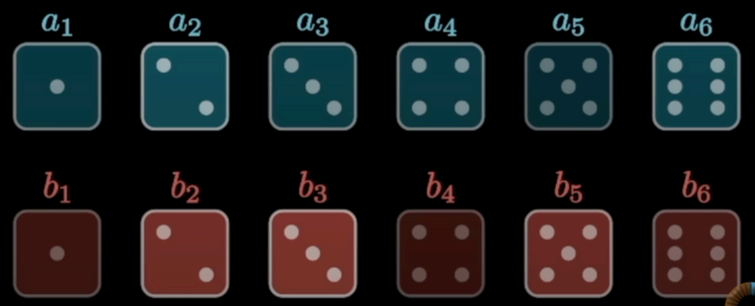
$$P(\square + \square = 10) = a_4 \cdot b_6 + a_5 \cdot b_5 + a_6 \cdot b_4$$

$$P(\square + \square = 11) = a_5 \cdot b_6 + a_6 \cdot b_5$$

$$P(\square + \square = 12) = a_6 \cdot b_6$$

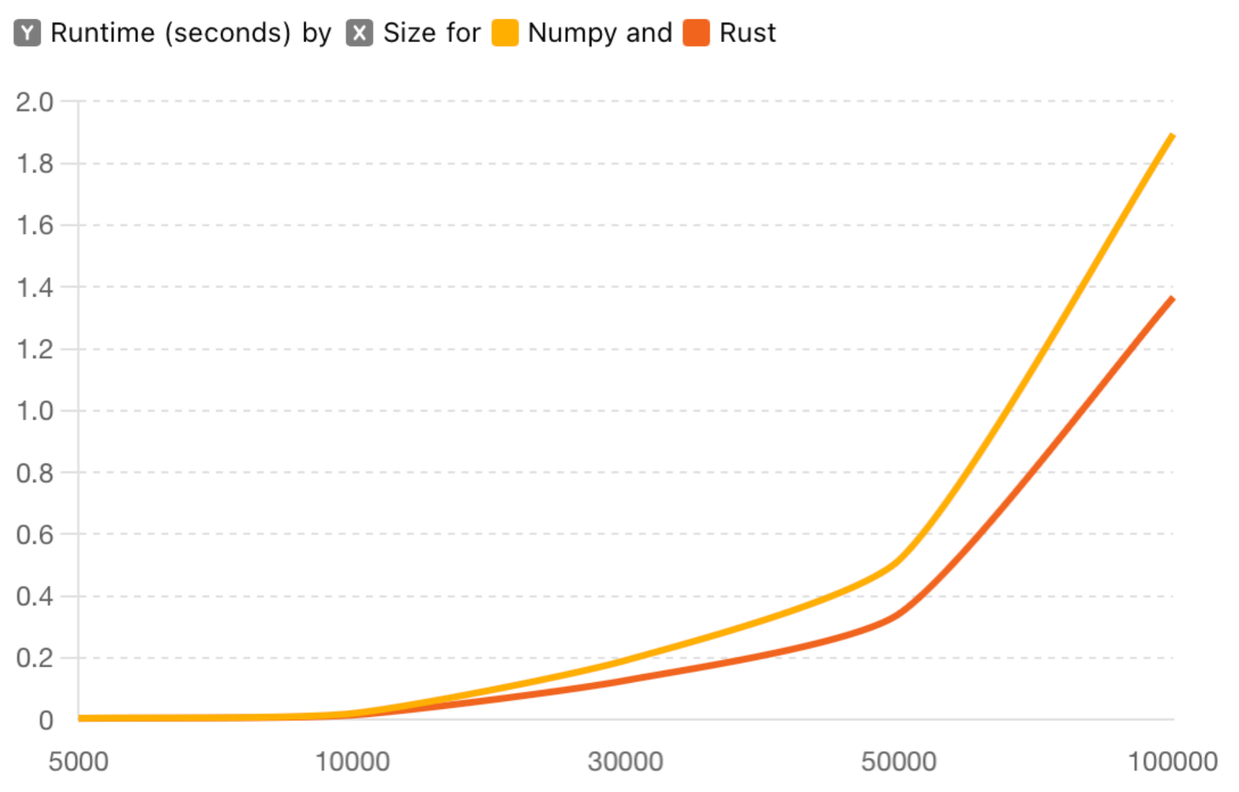
Convolution of (a_i) and (b_i)

$$(a * b)_n = \sum_{\substack{i,j \\ i+j=n}} a_i \cdot b_j$$



CONVOLVE

	Size	Numpy Time (s)	Rust Time (s)	Speedup Factor
1	5000	0.004614	0.003895	1.18
2	10000	0.020123	0.014725	1.37
3	30000	0.192098	0.127037	1.51
4	50000	0.517631	0.343414	1.51
5	100000	1.89345	1.365702	1.39



Implemented by Rust using Rayon and thread pool

All simple taken from M2 8-core cpu