Why Everyone is Talking About GPUs

Dhruv Nigam

October 11, 2025

MumPy Meetup

About Me

- Lead MLE at Dream 11
- Built massive ML systems at Philips, CLSA, Protium and Dream11
- Spent a lot of time and money(not mine) on GPUs

HOME > NVDA · NASDAQ

NVIDIA Corp

+ Follow Share

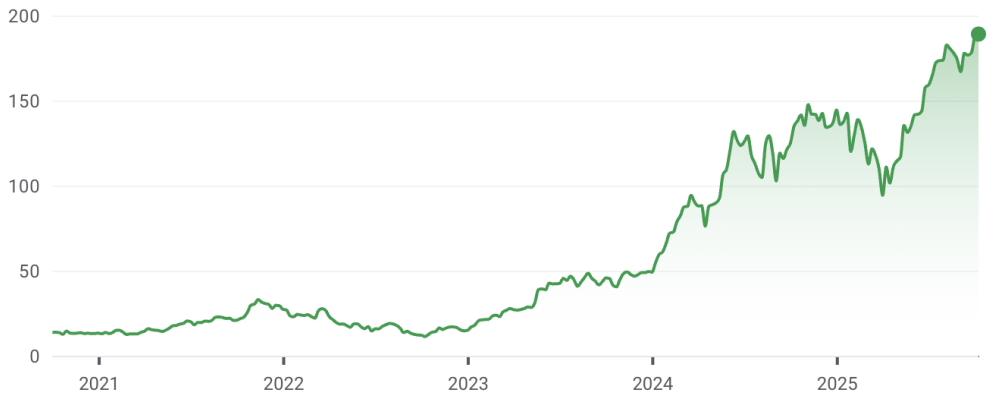
$189.11

↑1,274.35% +175.35 5Y

After Hours: $189.25 (↑0.074%) +0.14

Closed: Oct 8, 7:59:58 PM UTC-4 · USD · NASDAQ · Disclaimer

1D 5D 1M 6M YTD 1Y 5Y MAX



Compare to

Stock US listed security US headquartered

PREVIOUS CLOSE	$185.04
DAY RANGE	$186.54 - $189.60
YEAR RANGE	$86.63 - $191.05
MARKET CAP	4.60T USD
AVG VOLUME	185.53M
P/E RATIO	53.82
DIVIDEND YIELD	0.02%
PRIMARY EXCHANGE	NASDAQ

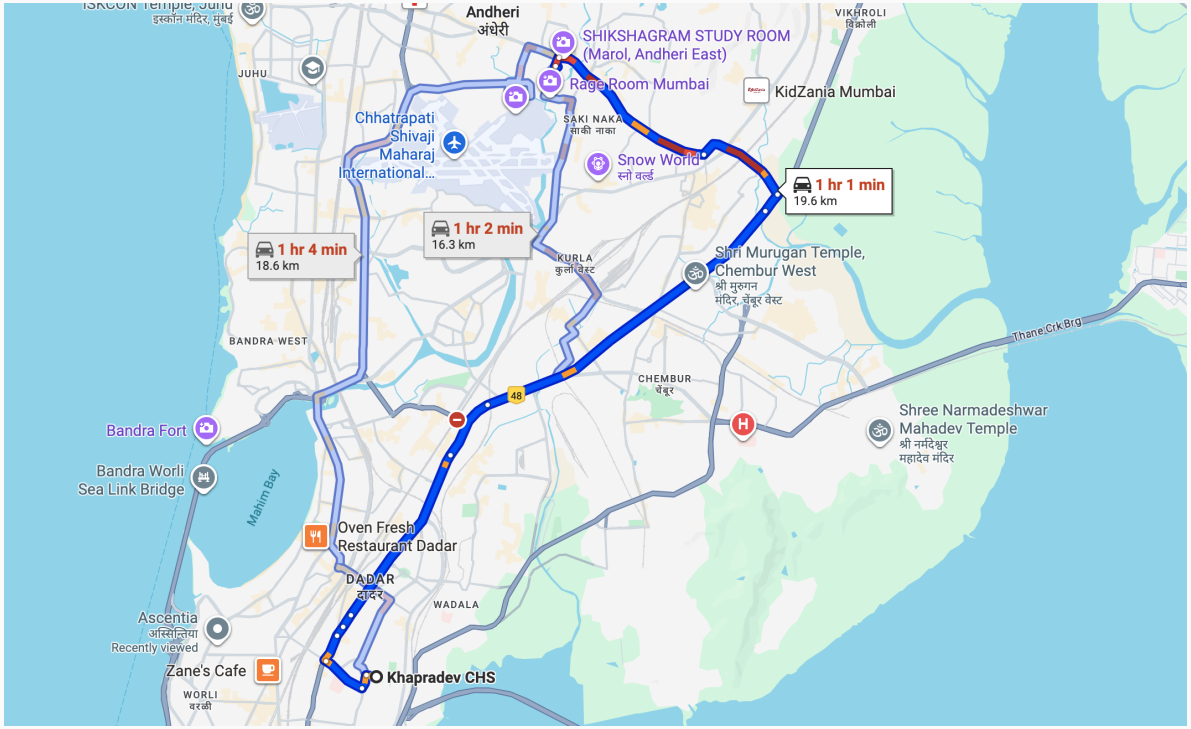
GPUs are Everywhere Now

- ChatGPT uses **over 30,000 NVIDIA GPUs** to handle **2.5+ billion queries daily**
- Each ChatGPT prompt uses 0.34 watt-hours of electricity
- Self-driving cars use GPUs for real-time processing
- Gaming has always used GPUs (the "G" is for Graphics!)

But why?

A Relatable Problem

Question: How long will it take me to reach Andheri?



What Affects Travel Time?

- Distance
- Time of day
- Is it raining?
- Weekday/Weekend
- Traffic conditions

Two Approaches

Traditional: Rule-Based

- IF distance $>$ 15km THEN add 20 min
- IF is_raining AND traffic $>$ 50% THEN multiply by 1.5
- IF is_weekend THEN subtract 10 min
- IF time_of_day between 8-10am OR 5-8pm THEN add 25 min
- IF construction_zones $>$ 0 THEN add 5 min per zone
- ... (hundreds of nested rules!)

Machine Learning

Convert everything to numbers, learn a function from data

We won't cover the learning part today, just the prediction part!

Making a Prediction

For one location, we do:

$$\begin{aligned}\text{predicted\_time} = \text{distance} \times w_1 \\ + \text{time\_of\_day} \times w_2 \\ + \text{is\_raining} \times w_3 \\ + \text{is\_weekend} \times w_4\end{aligned}$$

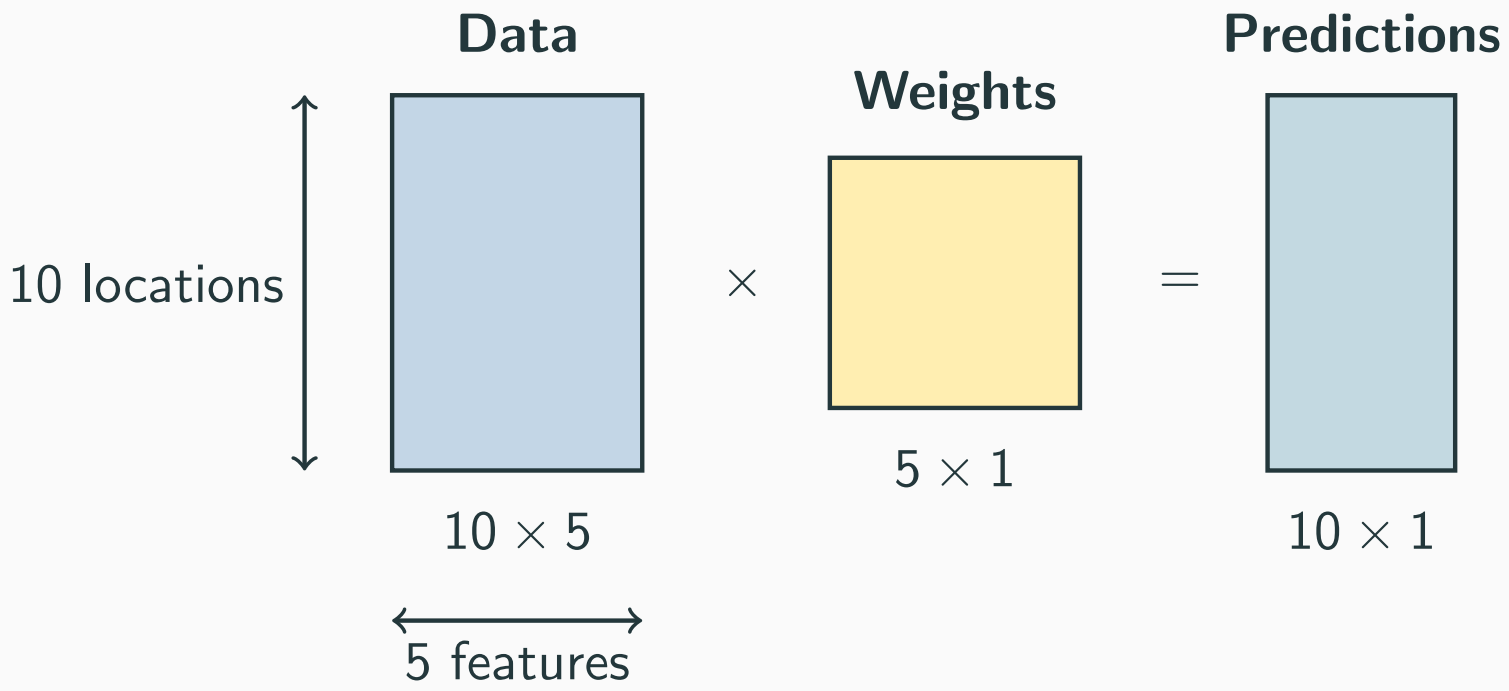
- **Features**: distance, time_of_day, etc. (your input data)
- **Weights** (w_1, w_2, w_3, w_4): learned from data
- Weights are also called **model parameters**

But What If...

- We have **1000 features** instead of 4?
- We want to predict for **10,000 locations** at once?
- We have **multiple layers** of transformations? (deep learning)

This becomes **matrix multiplication**

Matrix Multiplication



Interactive demo: `http://matrixmultiplication.xyz/`

The Real Scale

Modern ML models:

- GPT-3: 175 **billion** parameters
- Billions of multiply-add operations per prediction
- Training involves trillions of operations

How do we compute this efficiently?

Why Matrix Multiplication is Special

Perfect for parallel computing:

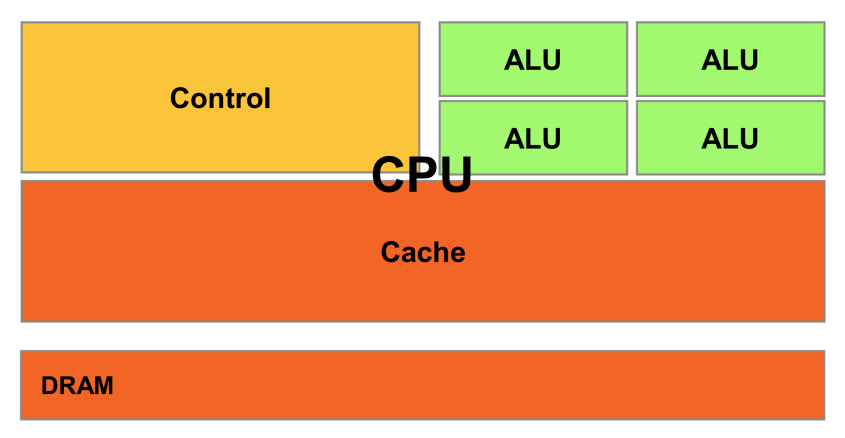
1. **Embarrassingly Parallel**
   - Each element can be computed independently
   - No coordination needed between computations
2. **Simple Operations**
   - Just multiply and add: $a \times b + c \times d + \ldots$
   - Same operation repeated millions of times

CPU: The Generalist

Few powerful cores

- 4-16 cores typically
- Sequential processing
- Great for general tasks
- **Latency-oriented design**

Analogy: 8 very smart people solving 10,000 problems one by one

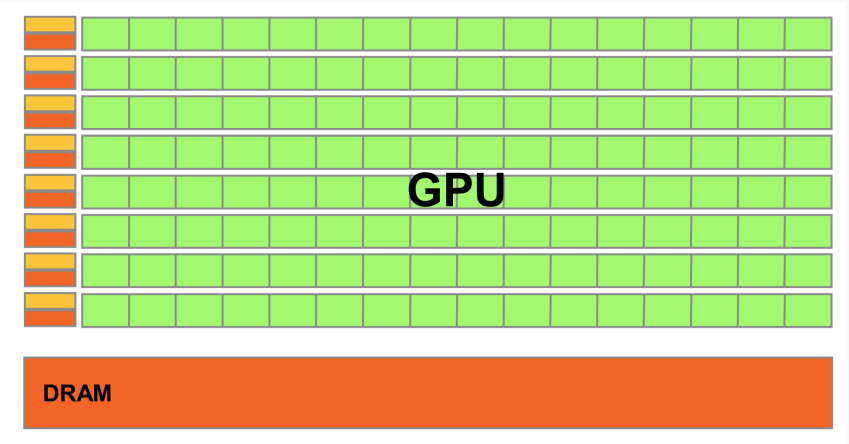


GPU: The Parallel Powerhouse

Thousands of smaller cores

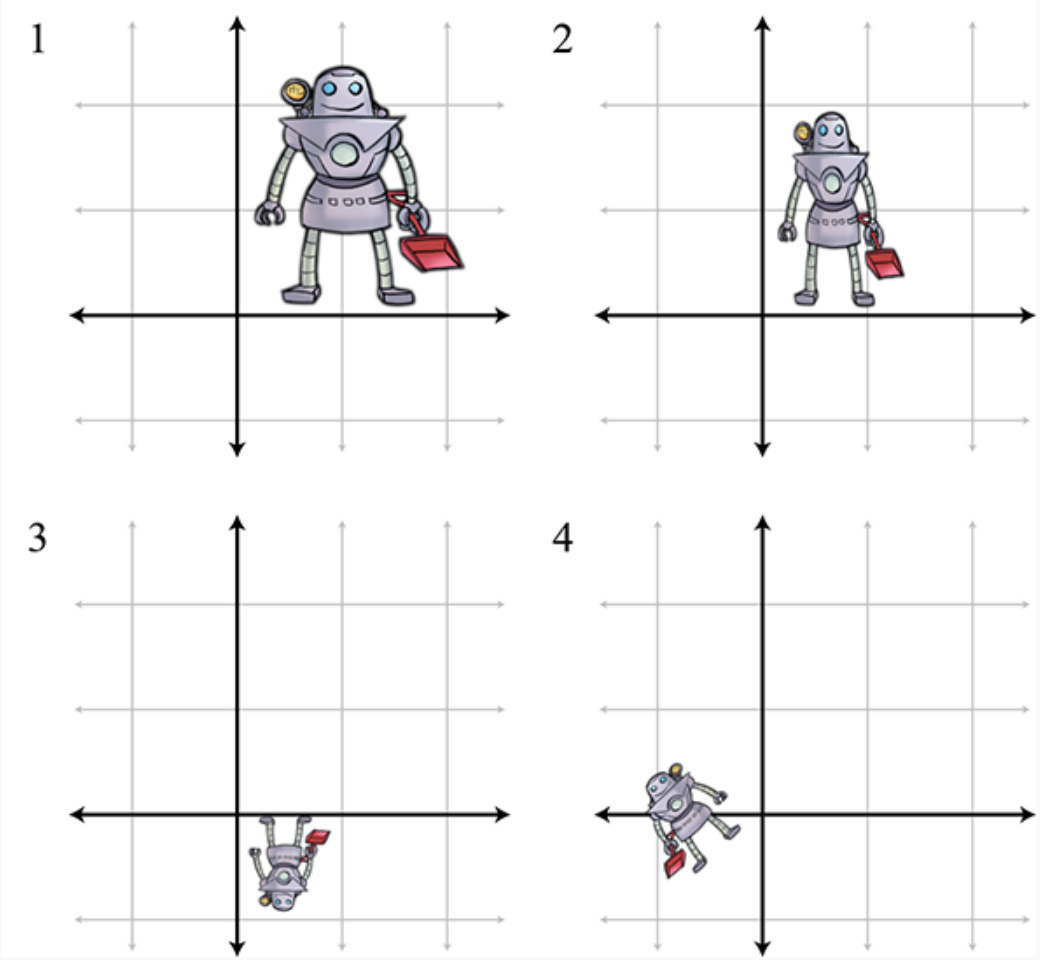
- 10,000+ CUDA cores
- Parallel processing
- Specialized for repetitive tasks
- **Throughput-oriented design**

Analogy: 10,000 people each solving one problem simultaneously



Matrix Operations in Graphics

How matrices rotate and move objects in 3D space:



Why GPUs Exist

GPUs were originally designed for **graphics and gaming**

- 3D transformations: matrix operations
- Rendering millions of pixels: parallel
- Real-time requirements: fast

Turns out: AI has the same needs!

Seeing is Believing

Python Timing Examples

Demo 1: NumPy vs CuPy

Setup: Large matrix multiplication (5000×5000)

CPU (NumPy)

```
import numpy as np
import time

size = 5000
A = np.random.rand(size, size)
B = np.random.rand(size, size)

start = time.time()
C = A @ B
end = time.time()

print(f"Time: {end-start:.3f}s")
# Result: 1.026s
```

GPU (CuPy)

```
import cupy as cp
import time

size = 5000
A = cp.random.rand(size, size)
B = cp.random.rand(size, size)

start = time.time()
C = A @ B
cp.cuda.Stream.null.synchronize()
end = time.time()

print(f"Time: {end-start:.3f}s")
# Typical result: ~0.015s
```

Demo 1: Results

Matrix Multiplication: 5000 × 5000

NumPy (CPU): 1.026s

CuPy (GPU): ~0.015s

↓

68x faster!

NumPy result measured on this system. GPU time is typical for NVIDIA RTX GPUs.

When Should You Care About GPUs?

Use GPUs when you have:

- Large matrices (thousands of rows/columns)
- Repeated operations (training loops)
- Real-time requirements
- Deep learning models

Skip GPUs for:

- Small data (¡ 1000 elements)
- One-off calculations
- Overhead of data transfer ¿ computation time

How to Get Started

Free options:

- **Google Colab** - Free T4 GPU access
- **Kaggle Notebooks** - Free GPU hours

Cloud providers:

- AWS, Google Cloud, Azure
- Pay per hour of GPU usage

Local GPU:

- NVIDIA GPUs (required for CUDA)
- For serious/regular ML work

Python Tools for GPU Computing

PyTorch	Most popular for deep learning, easy GPU support
TensorFlow	Google's framework, mature ecosystem
CuPy	NumPy, but on GPU
JAX	Google's new framework, automatic differentiation
RAPIDS	GPU-accelerated data science (pandas-like)

Most just need `.to('cuda')` or similar!

Putting It All Together

1. Modern AI = **lots of matrix multiplication**
2. Matrix multiplication = **embarrassingly parallel**
3. CPUs = sequential, GPUs = **parallel powerhouses**
4. Result: **50-200x speedup** for ML workloads

That's why every AI breakthrough mentions "GPU hours"!

Thank You!

Questions?