```
!wget https://developer.download.nvidia.com/compute/cuda/11.8.0/local_installers/cuda_11.8.0
!chmod +x cuda_11.8.0_520.61.05_linux.run
!./cuda_11.8.0_520.61.05_linux.run --silent --toolkit
```

```
--2025-05-04 16:10:33--  https://developer.download.nvidia.com/compute/cuda/11.8.0/local
Resolving developer.download.nvidia.com (developer.download.nvidia.com)... 23.59.88.207,
Connecting to developer.download.nvidia.com (developer.download.nvidia.com)|23.59.88.207
HTTP request sent, awaiting response... 200 OK
Length: 4336730777 (4.0G) [application/octet-stream]
Saving to: ‘cuda_11.8.0_520.61.05_linux.run’

cuda_11.8.0_520.61. 100%[===================>]   4.04G   173MB/s    in 33s

2025-05-04 16:11:06 (127 MB/s) - ‘cuda_11.8.0_520.61.05_linux.run’ saved [4336730777/433
```

```
%%writefile cuda_fixed.cu
#include <stdio.h>
#include <stdlib.h>
#include <cuda_runtime.h>

#define N 4
#define MATRIX_SIZE 4

__global__ void add(int* A, int* B, int* C, int size) {
    int tid = blockIdx.x * blockDim.x + threadIdx.x;
    if (tid < size) {
        C[tid] = A[tid] + B[tid];
    }
}

__global__ void multiply(int* A, int* B, int* C, int size) {
    int row = blockIdx.y * blockDim.y + threadIdx.y;
    int col = blockIdx.x * blockDim.x + threadIdx.x;
    int sum = 0;
    if (row < size && col < size) {
        for (int i = 0; i < size; ++i) {
            sum += A[row * size + i] * B[i * size + col];
        }
        C[row * size + col] = sum;
    }
}

void printVector(const char* name, int* vec, int size) {
    printf("%s: ", name);
    for (int i = 0; i < size; ++i) {
        printf("%d ", vec[i]);
    }
```

```
        printf("\n");
    }

    void printMatrix(const char* name, int* mat, int size) {
        printf("%s:\n", name);
        for (int i = 0; i < size; ++i) {
            for (int j = 0; j < size; ++j) {
                printf("%d ", mat[i * size + j]);
            }
            printf("\n");
        }
    }

    int main() {
        int A[N], B[N], C[N];
        for (int i = 0; i < N; ++i) {
            A[i] = rand() % 10;
            B[i] = rand() % 10;
        }

        int *d_A, *d_B, *d_C;
        cudaMalloc(&d_A, N * sizeof(int));
        cudaMalloc(&d_B, N * sizeof(int));
        cudaMalloc(&d_C, N * sizeof(int));

        cudaMemcpy(d_A, A, N * sizeof(int), cudaMemcpyHostToDevice);
        cudaMemcpy(d_B, B, N * sizeof(int), cudaMemcpyHostToDevice);

        add<<<1, N>>>(d_A, d_B, d_C, N);
        cudaDeviceSynchronize();

        cudaError_t err = cudaGetLastError();
        if (err != cudaSuccess) printf("Add Kernel Error: %s\n", cudaGetErrorString(err));

        cudaMemcpy(C, d_C, N * sizeof(int), cudaMemcpyDeviceToHost);

        printVector("Vector A", A, N);
        printVector("Vector B", B, N);
        printVector("Addition", C, N);

        cudaFree(d_A); cudaFree(d_B); cudaFree(d_C);

        int size = MATRIX_SIZE;
        int elements = size * size;
        int D[elements], E[elements], F[elements];
        for (int i = 0; i < elements; ++i) {
            D[i] = rand() % 10;
            E[i] = rand() % 10;
        }

        int *d_D, *d_E, *d_F;
```

```
    cudaMalloc(&d_D, elements * sizeof(int));
    cudaMalloc(&d_E, elements * sizeof(int));
    cudaMalloc(&d_F, elements * sizeof(int));

    cudaMemcpy(d_D, D, elements * sizeof(int), cudaMemcpyHostToDevice);
    cudaMemcpy(d_E, E, elements * sizeof(int), cudaMemcpyHostToDevice);

    dim3 threadsPerBlock(2, 2);
    dim3 blocksPerGrid((size + 1)/2, (size + 1)/2);

    multiply<<<blocksPerGrid, threadsPerBlock>>>(d_D, d_E, d_F, size);
    cudaDeviceSynchronize();

    err = cudaGetLastError();
    if (err != cudaSuccess) printf("Multiply Kernel Error: %s\n", cudaGetErrorString(err));

    cudaMemcpy(F, d_F, elements * sizeof(int), cudaMemcpyDeviceToHost);

    printMatrix("Matrix D", D, size);
    printMatrix("Matrix E", E, size);
    printMatrix("Multiplication", F, size);

    cudaFree(d_D); cudaFree(d_E); cudaFree(d_F);

    return 0;
}
```

```
Overwriting cuda_fixed.cu
```

```
!nvcc cuda_fixed.cu -o cuda_fixed
!./cuda_fixed
```

```
Vector A: 3 7 3 6
Vector B: 6 5 5 2
Addition: 9 12 8 8
Matrix D:
9 2 0 3
0 2 1 7
2 2 7 9
2 9 3 1
Matrix E:
1 7 9 6
6 6 8 9
0 3 5 2
8 7 6 2
Multiplication:
45 96 115 78
68 64 63 34
86 110 123 62
64 84 111 101
```

```
🔁 Step 1: Enable GPU
Go to:

Runtime > Change Runtime Type > Select GPU > C

📦 Step 2: Install CUDA (with nvcc)
Paste this into a Colab cell and run it:

bash
Copy
Edit
# Download and install CUDA 11.8
!wget https://developer.download.nvidia.com/co
local_installers/cuda_11.8.0_520.61.05_linux.r
!chmod +x cuda_11.8.0_520.61.05_linux.run
!./cuda_11.8.0_520.61.05_linux.run --silent --

# Set environment variables
import os
os.environ['PATH'] = '/usr/local/cuda-11.8/bin
os.environ['LD_LIBRARY_PATH'] = '/usr/local/cu
('LD_LIBRARY_PATH', '')

# Confirm installation
!nvcc --version
🧠 Step 3: Paste and Compile Your CUDA Program
Save code:
bash
Copy
Edit
%%writefile cuda_fixed.cu
# Paste the fixed CUDA code I gave you earlier
Compile and run:
bash
Copy
Edit
!nvcc cuda_fixed.cu -o cuda_fixed
!./cuda_fixed
```

🔁 Step 1: Enable GPU Go to:

Runtime > Change Runtime Type > Select GPU > Click Save

📦 Step 2: Install CUDA (with nvcc) Paste this into a Colab cell and run it:

bash Copy Edit

Download and install CUDA 11.8

!wget https://developer.download.nvidia.com/compute/cuda/11.8.0/local_installers/cuda_11.8.0_520.61.05_linux.run !chmod +x cuda_11.8.0_520.61.05_linux.run !./cuda_11.8.0_520.61.05_linux.run --silent --toolkit

Set environment variables

import os os.environ['PATH'] = '/usr/local/cuda-11.8/bin:' + os.environ['PATH'] os.environ['LD_LIBRARY_PATH'] = '/usr/local/cuda-11.8/lib64:' + os.environ.get('LD_LIBRARY_PATH', '')

Confirm installation

!nvcc --version 🧠 Step 3: Paste and Compile Your CUDA Program Save code: bash Copy Edit %%writefile cuda_fixed.cu

Paste the fixed CUDA code I gave you earlier

> Compile and run: bash Copy Edit !nvcc cuda_fixed.cu -o cuda_fixed !./cuda_fixed