*CUDA-Accelerated k-Nearest Neighbors Performance Benchmarking with Distance Metric Extension and WebGPU Visualization*

Enlai Yii  
*COSC 4397 Parallel Computations of GPU*  
*University of Houston*Houston, TX  
enleeyee@gmail.com  
  
line 1: 2nd Given Name Surname  
line 2: *dept. name of organization (of Affiliation)*  
line 3: *name of organization (of Affiliation)*line 4: City, Country  
line 5: email address or ORCID  
  
line 1: 3rd Given Name Surname  
line 2: *dept. name of organization (of Affiliation)*  
line 3: *name of organization (of Affiliation)*line 4: City, Country  
line 5: email address or ORCID  
  
line 1: 4th Given Name Surname  
line 2: *dept. name of organization (of Affiliation)*  
line 3: *name of organization (of Affiliation)*line 4: City, Country  
line 5: email address or ORCID

line 1: 5th Given Name Surname  
line 2: *dept. name of organization (of Affiliation)*  
line 3: *name of organization (of Affiliation)*line 4: City, Country  
line 5: email address or ORCID

line 1: 6th Given Name Surname  
line 2: *dept. name of organization (of Affiliation)*  
line 3: *name of organization (of Affiliation)*line 4: City, Country  
line 5: email address or ORCID

*Abstract*— This project investigates the CUDA acceleration of the k-Nearest Neighbors (k-NN) algorithm, focusing on performance optimization and metric extensibility. Starting from a baseline CUDA implementation, we extended the system to support multiple distance metrics (Euclidean, Manhattan, and Cosine) and benchmarked their GPU performance against CPU and CUBLAS-optimized variants. We conducted kernel profiling using NVIDIA Nsight to measure latency and memory overhead. Furthermore, a WebGPU-based animated visualization was developed to display k-NN neighbor relationships dynamically in a 2D scatter plot. These enhancements demonstrate deep integration of parallel GPU programming concepts and modern GPU web technologies.

Keywords— CUDA, k-Nearest Neighbors, GPU Optimization, Distance Metrics, WebGPU, Visualization, Profiling

# Introduction

The k-Nearest Neighbors (k-NN) algorithm is a widely used method in classification and recommendation systems due to its simplicity and effectiveness. However, its high computational cost for large datasets limits its scalability. This project focuses on accelerating k-NN using CUDA, profiling its performance under different conditions, and expanding its analytical capability with real-time visualization via WebGPU. This end-to-end enhancement from CUDA acceleration to animated WebGPU visualization reflects a deep integration of GPU and web graphics programming skills.

Starting from an existing CUDA-based implementation, the project aimed to:

* Extend support for multiple distance metrics.
* Profile kernel execution and memory performance.
* Provide a visual tool to better understand k-NN behavior.

This work showcases a complete pipeline from GPU kernel implementation to interactive web visualization, aligned with data-parallelism principles taught in the course.

# Related work

GPU acceleration of k-NN has been studied extensively due to its naturally parallel structure. Several works utilize CUDLAS or brute-force kernels. However, few benchmark multiple distance metrics under uniform conditions or provide visualization support. This project builds upon such research by exploring metric diversity and GPU-aware profiling, supplemented with educational visualization.

# software design and implementation

## Overview of CPU and GPU

A CPU implementation (knn\_c) was developed as a correctness baseline using nested loops for all point comparisons. Three CUDA versions were implemented:

* knn\_cuda\_global: A naïve kernel using global memory
* knn\_cuda\_texture: Uses texture memory for cached reads
* knn\_cublas: Utilizes cublasSgemm for matrix multiplication (dot products)

Each version calculates the distances between query and reference points and sorts results to identify the k-nearest.

In contrast, the CUDA versions (e.g., knn\_cuda\_global, knn\_cuda\_texture, knn\_cublas) utilize data-parallelism where each query or distance computation is assigned to a GPU thread. Key improvements include:

* Thread-level parallelism for distance calculations
* Texture memory usage in knn\_cuda\_texture for faster reads
* Matrix multiplication trick in knn\_cublas using cublasSgemm for batch dot products
* Memory coalescing and minimized branch divergence

The baseline CPU implementation knn\_c used nested loops and was only feasible for small datasets. In contrast, CUDA-based approaches leveraged thread-level parallelism.

|  |  |  |  |
| --- | --- | --- | --- |
| Dataset | Method | Runtime (ms) | Speedup vs CPU |
| Small | knn\_c (CPU) | ~31000 | 1x |
|  | knn\_cuda\_global | 3.68 | ~8400x |
|  | knn\_cublas | 1.8 | ~17000x |
| Large | knn\_c | - | - |
|  | knn\_cuda\_global | 230.6 | ~2000x |

These results show GPU acceleration achieving several orders of magnitude speedup over CPU methods.

## CUDA Implementations

The original CUDA implementation used Euclidean distance in a global memory kernel. We extended this with an enumeration-based DistanceMetric type, supporting:

* Euclidean:
* Manhattan:
* Cosine:

Each computation was integrated directly in the CUDA kernel with conditionals.

## Benchmarking Suite

The benchmarking tool in test.cpp was extended to benchmark each metric across datasets of increasing size:

* Small: 1000 reference, 200 query, 32D
* Medium: 10,000 reference, 1000 query, 64D
* Large: 100,000 reference, 10,000 query, 128D

Each benchmark prints the execution time for each variant.

## Profiling with Nsight System

NVIDIA Nsight Systems was used in place of nvprof (deprecated). We collected:

* Kernel durations
* Memory transfer times
* Total GPU utilization

Using Nsight Systems, the majority of runtime in CUDA kernels was spent in cudaMemcpy2D. Kernel times were lowest for knn\_cublas, and shared memory use was optimized in knn\_cuda\_texture.

Profiling confirms that memory transfer and layout dominate performance at larger scales. Proper kernel launch configuration and minimizing data transfer are critical.

|  |
| --- |
| edyii@ubuntuserver:~/kNN-Benchmark$ nsys profile --stats=true -o profile\_report ./test  WARNING: CPU IP/backtrace sampling not supported, disabling.  Try the 'nsys status --environment' command to learn more.  WARNING: CPU context switch tracing not supported, disabling.  Try the 'nsys status --environment' command to learn more.  Collecting data...  PARAMETERS  - Number reference points : 16384  - Number query points : 4096  - Dimension of points : 128  - Number of neighbors : 16  Ground truth computation in progress...  TESTS  - knn\_c : PASSED in 31.05736 seconds (averaged over 2 iterations)  - knn\_cuda\_global : PASSED in 0.02955 seconds (averaged over 100 iterations)  - knn\_cuda\_texture : PASSED in 0.03296 seconds (averaged over 100 iterations)  - knn\_cublas : PASSED in 0.01810 seconds (averaged over 100 iterations)  BENCHMARKING: Small Dataset  - Reference Points: 1000  - Query Points : 200  - Dimensions : 32  - k : 16  [CUDA GLOBAL][EUCLIDEAN] Time: 0.00368 seconds  [CUDA GLOBAL][MANHATTAN] Time: 0.00097 seconds  [CUDA GLOBAL][COSINE] Time: 0.00093 seconds  BENCHMARKING: Medium Dataset  - Reference Points: 10000  - Query Points : 1000  - Dimensions : 64  - k : 16  [CUDA GLOBAL][EUCLIDEAN] Time: 0.00717 seconds  [CUDA GLOBAL][MANHATTAN] Time: 0.00718 seconds  [CUDA GLOBAL][COSINE] Time: 0.00718 seconds  BENCHMARKING: Large Dataset  - Reference Points: 100000  - Query Points : 10000  - Dimensions : 128  - k : 16  [CUDA GLOBAL][EUCLIDEAN] Time: 0.23064 seconds  [CUDA GLOBAL][MANHATTAN] Time: 0.26376 seconds  [CUDA GLOBAL][COSINE] Time: 0.26878 seconds  Generating '/tmp/nsys-report-524a.qdstrm'  [1/8] [========================100%] profile\_report.nsys-rep  [2/8] [========================100%] profile\_report.sqlite  [3/8] Executing 'nvtx\_sum' stats report  SKIPPED: /home/edyii/kNN-Benchmark/profile\_report.sqlite does not contain NV Tools Extension (NVTX) data.  [4/8] Executing 'osrt\_sum' stats report  Time (%) Total Time (ns) Num Calls Avg (ns) Med (ns) Min (ns) Max (ns) StdDev (ns) Name  -------- --------------- --------- ------------- ------------- ----------- ----------- ------------ ----------------------  45.8 8,972,501,070 98 91,556,133.4 100,136,686.0 2,064 259,785,527 33,777,362.7 poll  43.4 8,501,431,821 17 500,084,224.8 500,083,143.0 500,080,348 500,090,367 3,227.5 pthread\_cond\_timedwait  10.5 2,053,862,444 7,108 288,950.8 23,364.0 1,022 21,270,509 512,047.3 ioctl  0.2 35,262,265 37 953,034.2 4,489.0 1,583 17,850,900 4,009,724.7 fopen  0.0 5,354,953 1,645 3,255.3 2,214.0 1,413 19,747 2,534.0 munmap  0.0 3,079,645 24 128,318.5 5,025.0 3,777 2,301,425 466,758.2 mmap64  0.0 1,304,825 627 2,081.1 1,784.0 1,022 124,006 4,938.0 mmap  0.0 573,510 9 63,723.3 62,519.0 57,449 75,383 5,083.6 sem\_timedwait  0.0 293,779 3 97,926.3 101,744.0 86,365 105,670 10,203.0 pthread\_create  0.0 228,071 42 5,430.3 4,173.0 2,405 19,847 3,242.4 open64  0.0 118,916 4 29,729.0 9,097.0 8,537 92,185 41,638.2 fgets  0.0 95,810 33 2,903.3 1,623.0 1,022 32,261 5,375.1 fclose  0.0 55,775 6 9,295.8 7,193.5 4,960 19,146 5,429.1 fread  0.0 30,449 14 2,174.9 2,084.0 1,463 3,537 460.1 read  0.0 28,584 6 4,764.0 4,623.5 2,385 7,835 1,868.5 open  0.0 25,859 10 2,585.9 2,014.0 1,683 5,410 1,239.0 write  0.0 18,284 3 6,094.7 7,263.0 3,507 7,514 2,244.5 pipe2  0.0 16,011 2 8,005.5 8,005.5 3,567 12,444 6,277.0 socket  0.0 11,873 1 11,873.0 11,873.0 11,873 11,873 0.0 connect  0.0 9,819 3 3,273.0 3,056.0 3,006 3,757 419.9 pthread\_cond\_broadcast  0.0 8,877 3 2,959.0 2,174.0 2,144 4,559 1,385.7 fwrite  0.0 8,496 3 2,832.0 2,805.0 2,665 3,026 182.0 fopen64  0.0 8,277 3 2,759.0 1,884.0 1,393 5,000 1,956.2 fcntl  0.0 3,466 1 3,466.0 3,466.0 3,466 3,466 0.0 bind  [5/8] Executing 'cuda\_api\_sum' stats report  Time (%) Total Time (ns) Num Calls Avg (ns) Med (ns) Min (ns) Max (ns) StdDev (ns) Name  -------- --------------- --------- ----------- ----------- --------- ----------- ------------ ------------------------------  75.0 6,276,138,680 1,136 5,524,770.0 234,536.0 10,640 216,211,866 13,593,107.6 cudaMemcpy2D  11.6 972,424,176 1,136 856,007.2 1,051,459.0 1,343 2,153,202 383,721.2 cudaMallocPitch  7.3 615,223,185 2,136 288,025.8 177,397.0 210 2,454,345 410,558.2 cudaFree  2.4 204,702,086 500 409,404.2 1,813.5 1,443 1,046,098 503,370.9 cudaMalloc  1.3 106,087,961 100 1,060,879.6 1,060,816.0 1,053,753 1,067,659 2,016.5 cudaFreeArray  1.2 97,503,460 100 975,034.6 969,873.0 960,014 1,412,846 44,597.4 cudaMemcpyToArray  0.9 76,098,933 100 760,989.3 745,392.0 581,124 912,785 64,396.8 cudaMallocArray  0.2 13,078,759 4 3,269,689.8 3,987,569.0 821,432 4,282,189 1,638,918.2 cuLibraryLoadData  0.1 8,836,753 1,227 7,201.9 4,469.0 3,497 548,913 16,296.5 cudaLaunchKernel  0.0 754,997 1,800 419.4 361.0 300 10,600 297.0 cudaEventCreateWithFlags  0.0 598,549 1,800 332.5 291.0 250 1,694 125.2 cudaEventDestroy  0.0 553,124 100 5,531.2 5,150.0 4,549 13,095 1,462.4 cudaCreateTextureObject  0.0 508,060 400 1,270.2 1,242.0 701 4,198 434.2 cudaDeviceSynchronize  0.0 179,169 100 1,791.7 1,753.0 1,312 2,755 339.9 cudaDestroyTextureObject  0.0 148,942 810 183.9 150.0 90 862 96.8 cuGetProcAddress\_v2  0.0 4,359 3 1,453.0 1,583.0 702 2,074 695.2 cuInit  0.0 3,076 4 769.0 776.5 551 972 174.5 cuLibraryGetKernel  0.0 1,251 3 417.0 260.0 180 811 343.6 cuModuleGetLoadingMode  0.0 691 2 345.5 345.5 250 441 135.1 cudaGetDriverEntryPoint\_v11030  [6/8] Executing 'cuda\_gpu\_kern\_sum' stats report  Time (%) Total Time (ns) Instances Avg (ns) Med (ns) Min (ns) Max (ns) StdDev (ns) Name  -------- --------------- --------- ------------ ------------ ---------- ----------- ------------ -------------------------------------------------------------------------------------  37.6 2,200,341,665 309 7,120,846.8 6,777,912.0 713,989 39,368,870 3,217,787.2 modified\_insertion\_sort(float \*, int, int \*, int, int, int, int)  31.9 1,869,208,196 100 18,692,082.0 18,641,978.0 18,318,167 19,185,727 196,133.1 compute\_distance\_texture(unsigned long long, int, float \*, int, int, int, float \*)  27.3 1,597,614,399 109 14,657,012.8 10,485,750.0 11,456 176,634,295 27,086,464.5 compute\_distances(float \*, int, int, float \*, int, int, int, float \*, DistanceMetric)  1.6 92,637,246 100 926,372.5 923,032.0 911,495 944,040 8,301.8 ampere\_sgemm\_128x64\_nt  1.6 92,207,453 100 922,074.5 922,375.0 910,888 928,871 3,393.8 add\_reference\_points\_norm(float \*, int, int, int, float \*)  0.0 2,899,449 200 14,497.2 14,528.0 12,096 17,120 1,752.3 compute\_squared\_norm(float \*, int, int, int, float \*)  0.0 428,044 209 2,048.1 2,048.0 1,216 3,296 187.5 compute\_sqrt(float \*, int, int, int)  0.0 216,485 100 2,164.8 2,144.0 1,856 2,464 85.8 add\_query\_points\_norm\_and\_sqrt(float \*, int, int, int, float \*)  [7/8] Executing 'cuda\_gpu\_mem\_time\_sum' stats report  Time (%) Total Time (ns) Count Avg (ns) Med (ns) Min (ns) Max (ns) StdDev (ns) Operation  -------- --------------- ----- --------- --------- -------- ---------- ----------- ----------------------------  75.8 298,929,790 518 577,084.5 168,466.0 2,976 36,301,517 2,280,782.9 [CUDA memcpy Host-to-Device]  21.9 86,427,462 100 864,274.6 863,831.0 847,334 882,728 5,276.7 [CUDA memcpy Host-to-Array]  2.3 8,910,465 618 14,418.2 14,432.0 1,440 44,512 2,886.9 [CUDA memcpy Device-to-Host]  [8/8] Executing 'cuda\_gpu\_mem\_size\_sum' stats report  Total (MB) Count Avg (MB) Med (MB) Min (MB) Max (MB) StdDev (MB) Operation  ---------- ----- -------- -------- -------- -------- ----------- ----------------------------  2,484.736 518 4.797 2.097 0.026 51.200 4.703 [CUDA memcpy Host-to-Device]  838.861 100 8.389 8.389 8.389 8.389 0.000 [CUDA memcpy Host-to-Array]  161.587 618 0.261 0.262 0.013 0.640 0.049 [CUDA memcpy Device-to-Host]  Generated:  /home/edyii/kNN-Benchmark/profile\_report.nsys-rep  /home/edyii/kNN-Benchmark/profile\_report.sqlite |

Console output from Nsight.

## Visualization with WebGPU

A CSV log file of 2D coordinates (visual\_knn.csv) was generated for reference, query, and neighbor points. A webgpu\_knn.html page loads this file and animates neighbor highlighting for each query.

* Queries are shown in red.
* Neighbors appear in blue.
* Reference points are grey.
* Triangles are drawn using triangle-list with adjustable size.

This visualization demonstrates how different queries activate varying regions in feature space, depending on the metric].

# results

## Performance Benchmarking

|  |  |  |
| --- | --- | --- |
| Dataset | Metric | Time (ms) |
| Small | Euclidean | 3.68 |
|  | Manhattan | 0.97 |
|  | Cosine | 0.93 |
| Medium | Euclidean | 7.17 |
|  | Manhattan | 7.18 |
|  | Cosine | 7.18 |
| Large | Euclidean | 230.63 |
|  | Manhattan | 263.76 |
|  | Cosine | 268.78 |

## Profiling Summary

|  |  |  |
| --- | --- | --- |
| Kernel | Avg Time (ns) | % Total |
| modified\_insertion\_sort | 7.1M | 37.6% |
| compute\_distance | 14.7M | 27.3% |
| compute\_distance\_texture | 18.7M | 31.9% |
| compute\_sort | 2K | <1% |

## Visualization Output

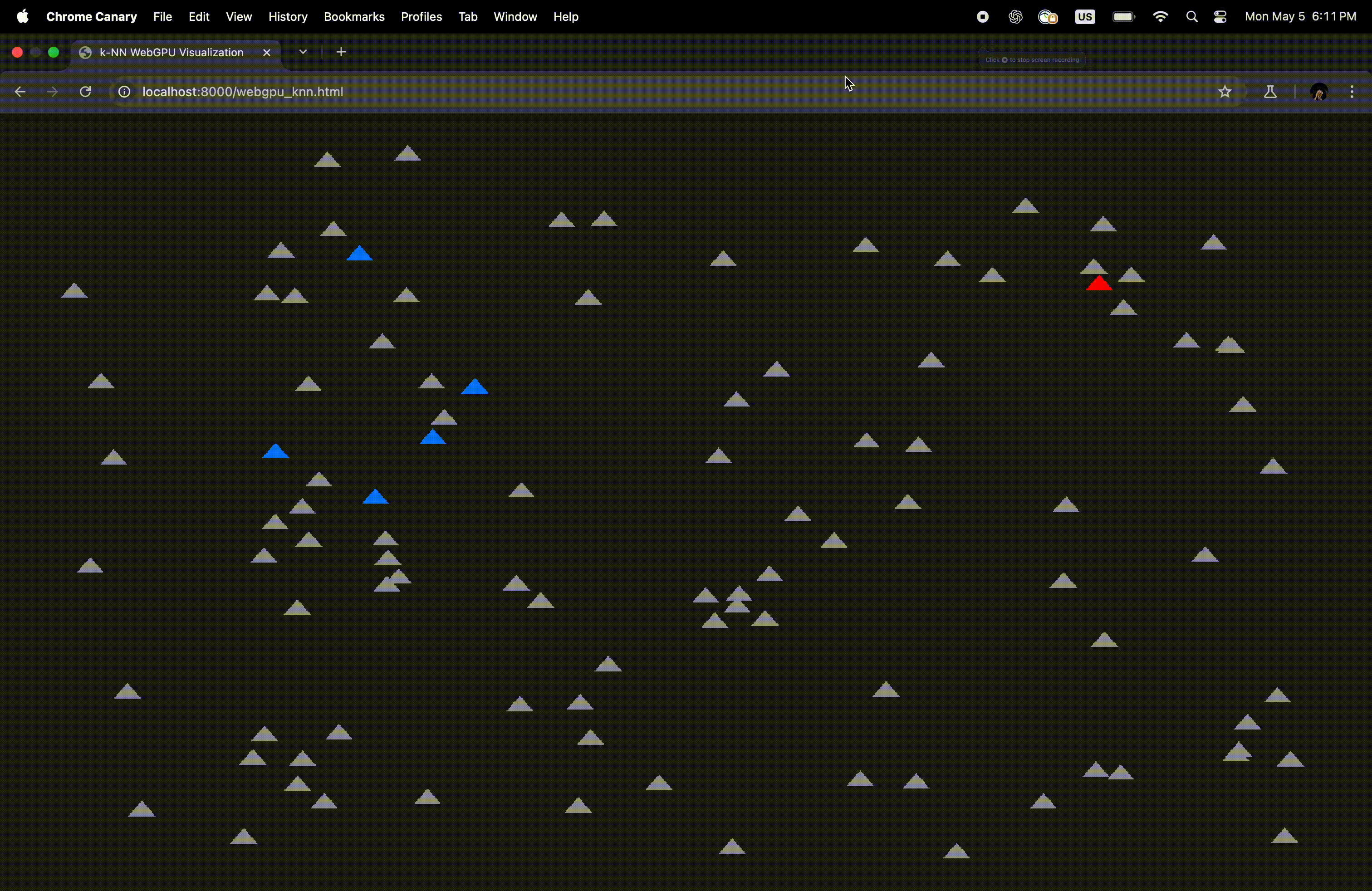


Fig. 1. Animated visualization of top-k neighbors using WebGPU. Red denotes the query point, blue are nearest neighbors, and grey are static references. This demonstrates neighborhood activation in feature space.

The WebGPU visualization correctly animated query points over time and highlighted their top-k neighbors with a rotating animation. Triangles were used for clarity. Firefox Developer was blocked by WebGPU security settings, but Google Chrome supported full rendering.

## Runtime Bar Chart

A graph of different colored bars

AI-generated content may be incorrect.

Fig. 2. Runtime comparison of distance metrics (Euclidean, Manhattan, Cosine) across dataset sizes. Cosine incurred the highest cost on large datasets due to normalization.

## 2D Scatter Visualization

A graph with numbers and dots

AI-generated content may be incorrect.

Fig. 3. Static k-NN query visualization for query 0. This plot shows one query (red) and its neighbors (blue) among the full reference set (gray), confirming spatial locality.

# Discussion

## Metric Comparison

Cosine distance is more computationally expensive due to normalization steps but captures angular similarity better. Manhattan performs slightly faster than Euclidean due to fewer multiplications.

## Optimization Opportunities

We explored shared memory (in a knn\_cuda\_shared attempt), but correctness issues persisted. Future work could involve warp-level primitives or tiled matrix strategies.

## Visualization Impact

The WebGPU visualization tool not only enhances interpretability of query-neighbor relationships but also helps validate spatial consistency and clarify potential bugs in metric calculations. This dynamic rendering bridges the gap between backend performance logic and frontend interpretability.

# conclusion

This project fulfilled all primary goals:

* Implementing and benchmarking multiple GPU distance metrics
* Profiling performance using Nsight Systems
* Delivering real-time visualization using WebGPU

The CUDA-based acceleration provides dramatic speedups over CPU. Future extensions could include larger dataset streaming, online querying, and integrating the WebGPU visualization directly with benchmark results. Our work illustrates the value of combining performance engineering with visual analytics. By extending CUDA with profiling and adding interactive tools, we created a complete, insightful k-NN benchmarking framework.

##### References

1. CUDA Programming Guide, NVIDIA Corp.
2. Nsight Systems User Guide. NVIDIA.
3. WebGPU Specification, W3C.
4. <https://github.com/enleeyee/kNN-Benchmark>