Massive-Scale Out-of-Core UMAP on the GPU

Jinsol Park*, Corey Nolet*, Akira Naruse, Edward Raff, Tim Oates

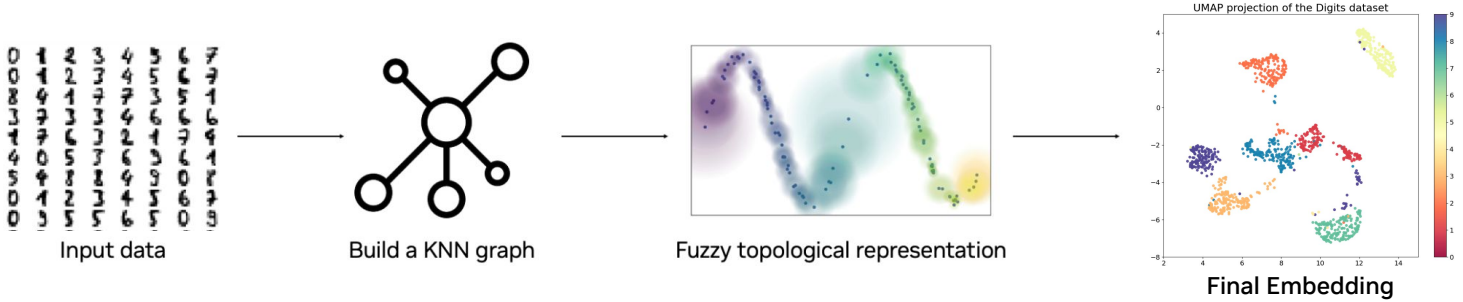
May 21st, 2026

Speaker: Jinsol Park

* Equal Contribution

UMAP

- **Dimensionality reduction algorithm**
  for visualization / feature extraction



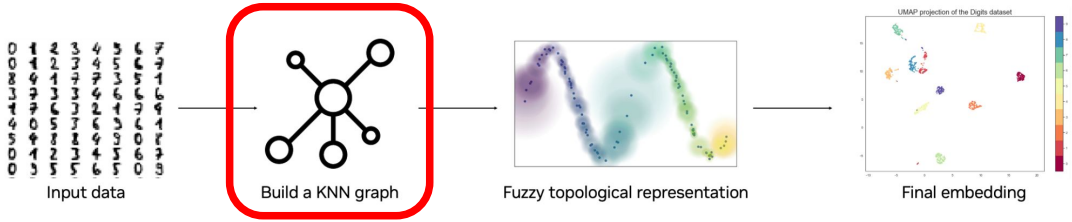
- **Fast on GPU** (prior work)
  We optimized this full pipeline on GPU

Prior Work

Bringing UMAP Closer to the Speed of Light with GPU Acceleration
arXiv: 2008.00325

UMAP doesn't scale

- kNN construction is a bottleneck

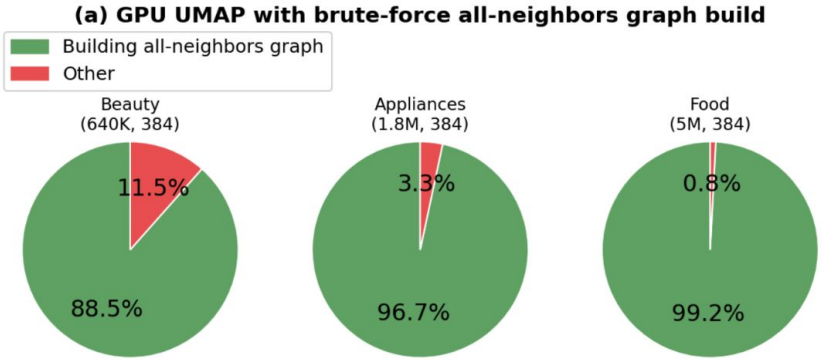


Compute

- Quadratic cost $O(N^2)$

Memory

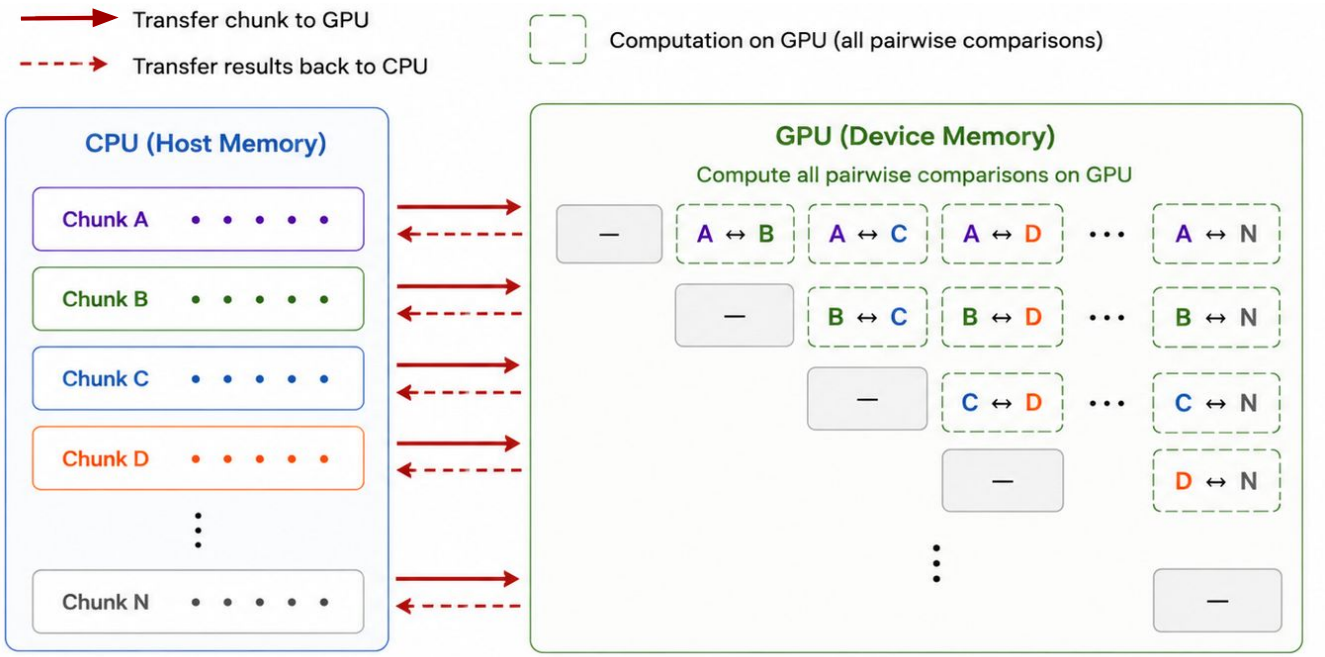
- Requires full dataset on GPU



We tackle **compute** with GPU-accelerated approximate algorithms (ANN)
memory with out-of-core chunking

Naive Chunking of the Dataset

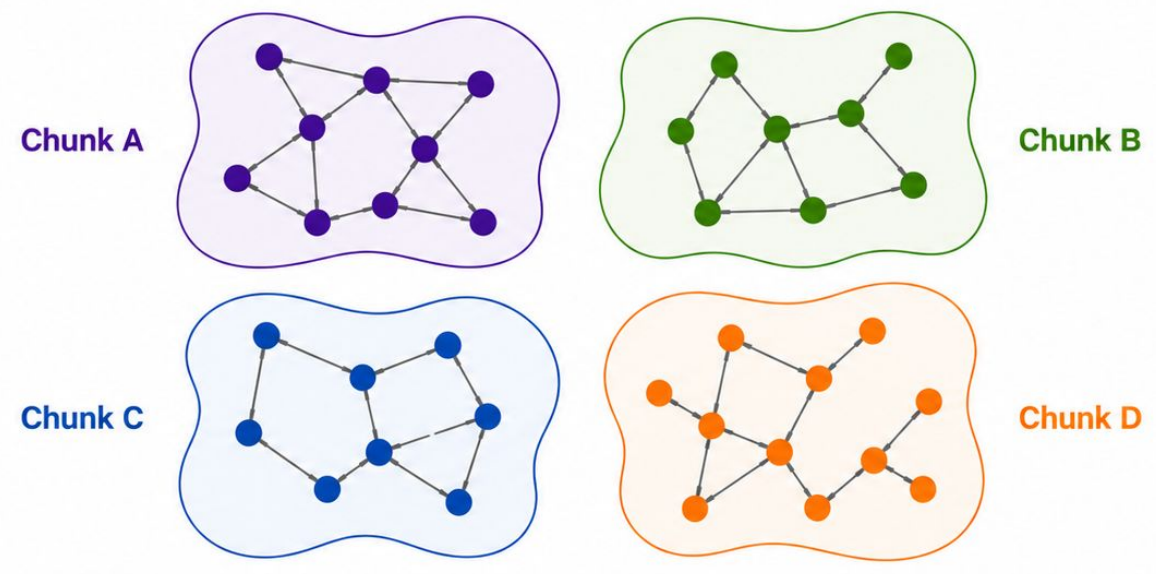
- Every chunk needs to be compared with every other chunk
- Massive **compute** and **memory transfers**



Out-of-core computation

Our Solution: Make the Problem Local

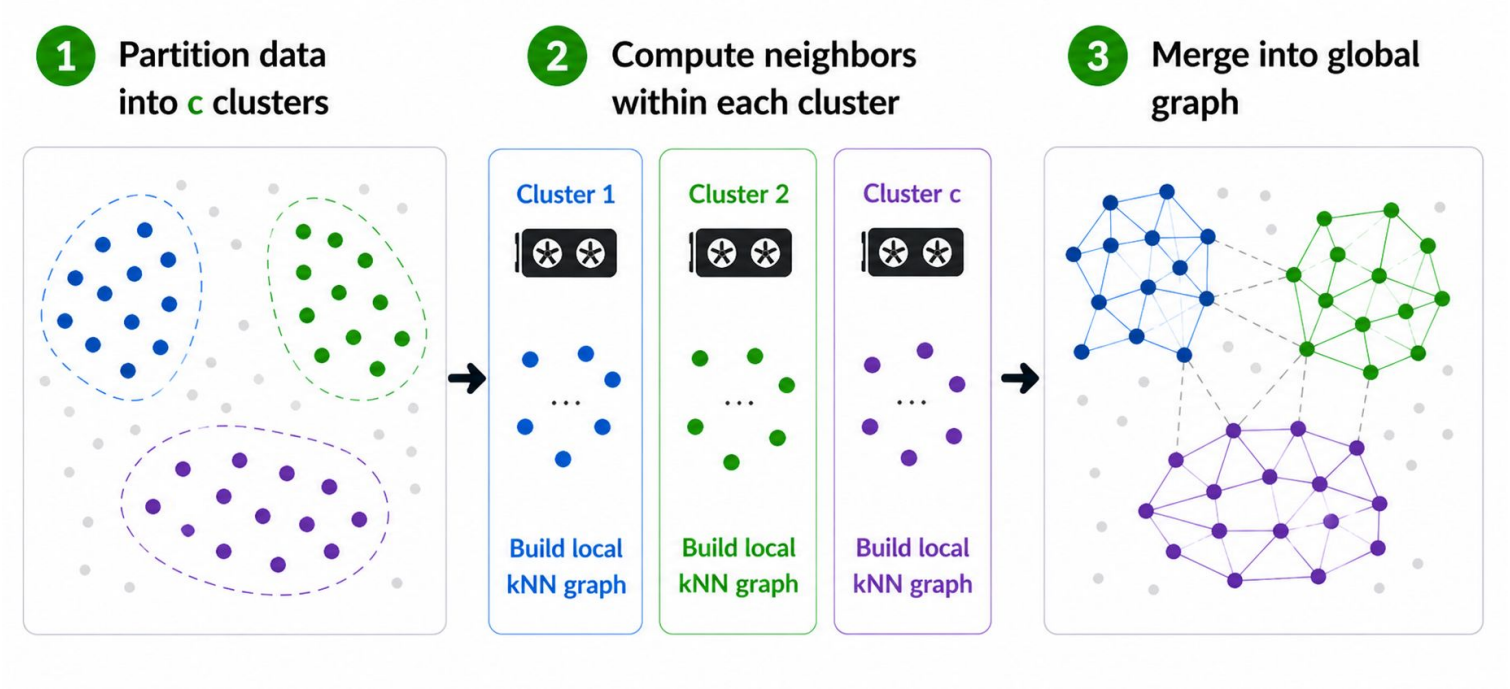
- **Locality-aware** chunks follow data distribution so most nearest neighbors stay **within the same chunk**



Our Approach

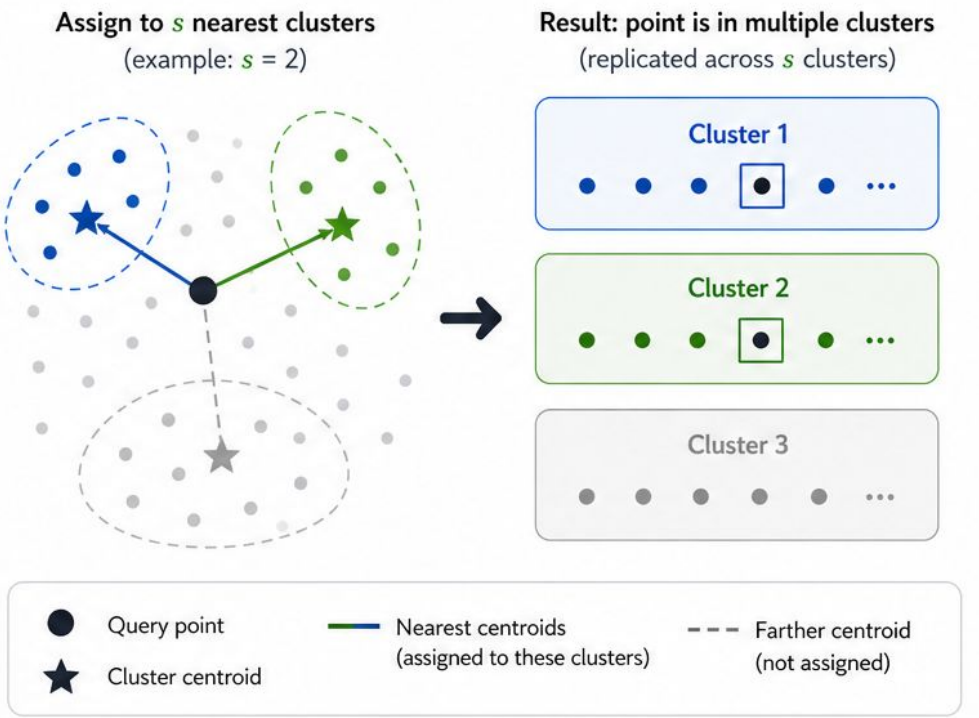
How we make it local

- Break the global problem into local problems, then merge
- Eliminates redundant broadcasting



Handling cross-cluster vectors

- Assign each point to its s closest clusters
- c total clusters, and each point belongs to s of them.



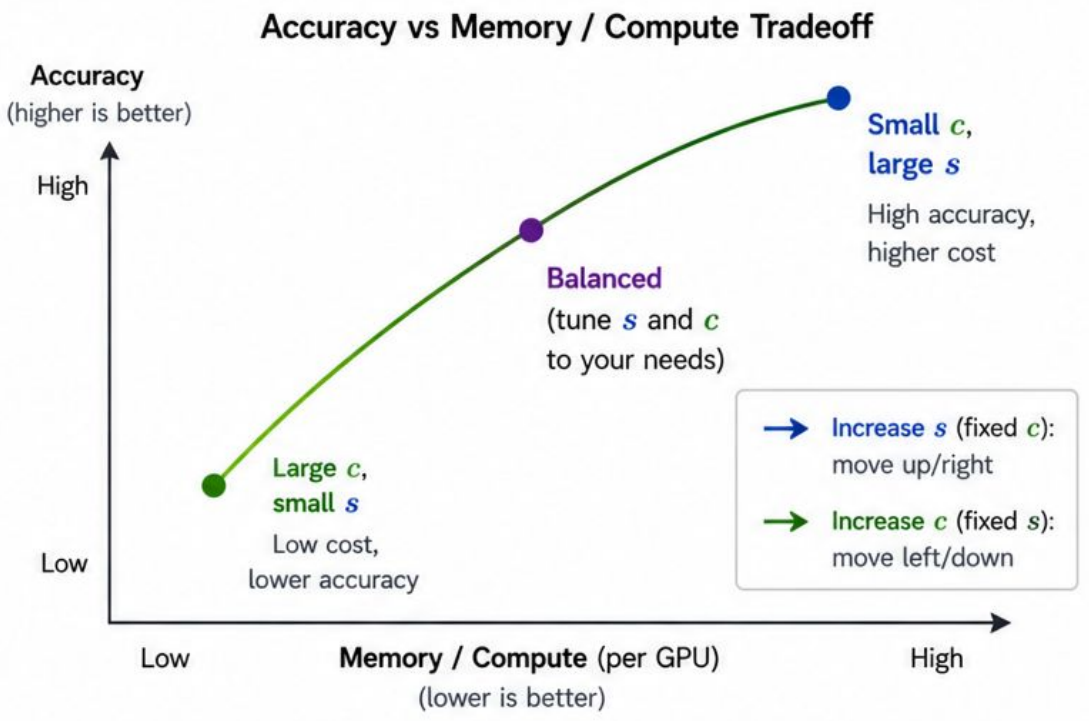
Tradeoff: Accuracy **VS** Memory and Compute

- Choosing the spilling factor s and the number of clusters c balances **accuracy** and **efficiency**.

Memory per GPU
(approximate scaling)

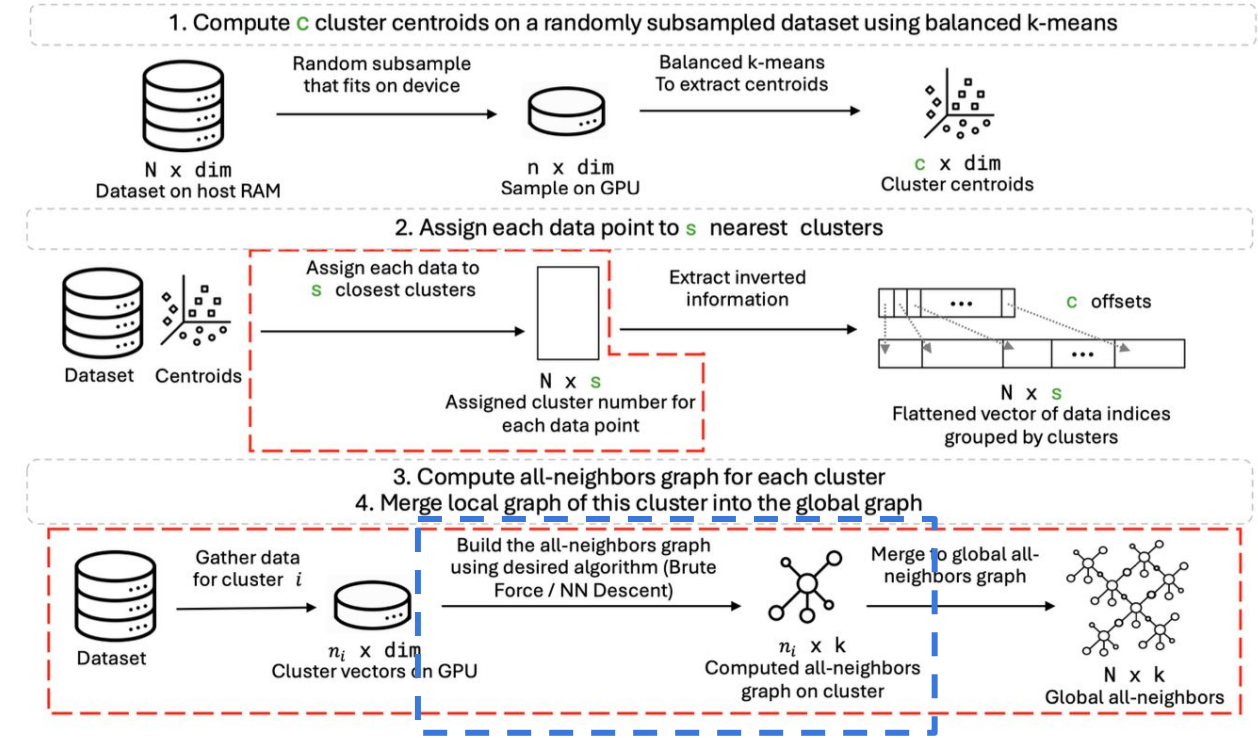
$$M_{\text{gpu}} \propto N \cdot \frac{s}{c}$$

N : number of data points
s : spilling factor (clusters per point)
c : number of clusters



Full Pipeline

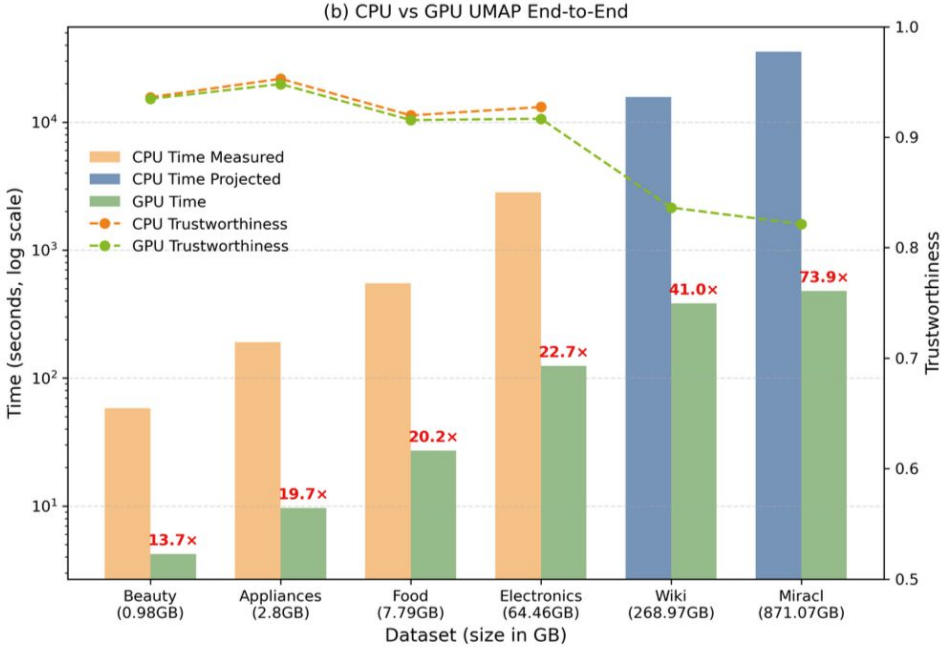
- Parallel, and ANN-agnostic



Results

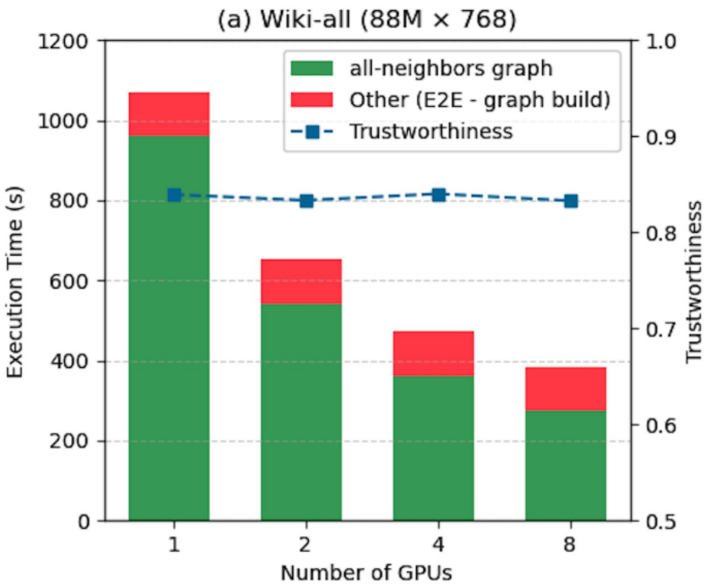
Results: End-to-End

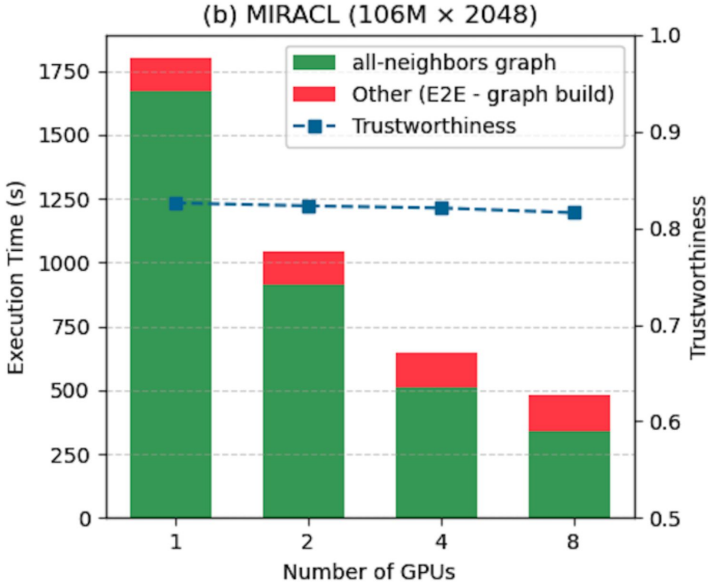
- Projected speedup of **74x** over CPU UMAP
- CPU cannot run to completion for large-scale



Results: Multi-GPU Scaling

- Observed scaling with more GPUs

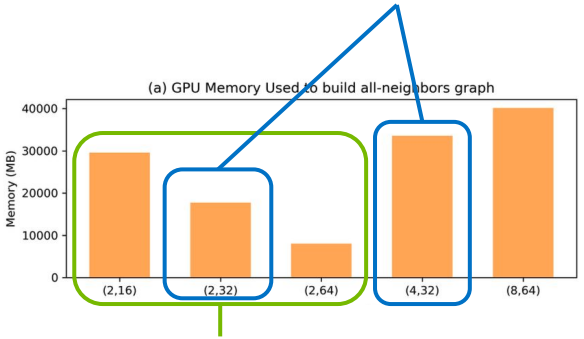




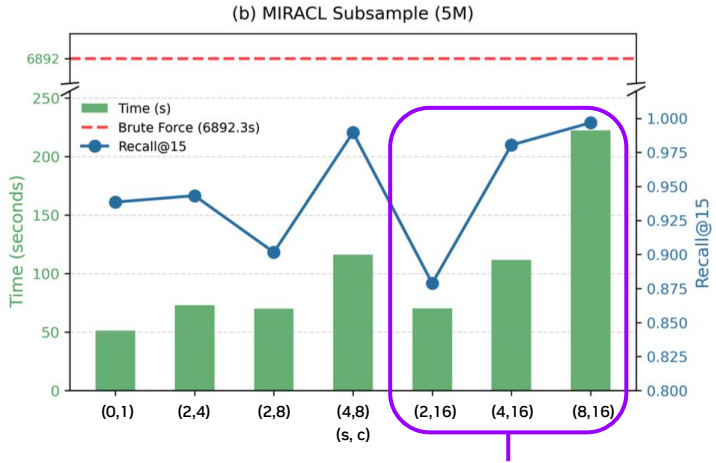
Results: Impact of s and c

- s and c enable controllable tradeoff

Larger s results in more points per cluster



More clusters c reduces number of points per cluster

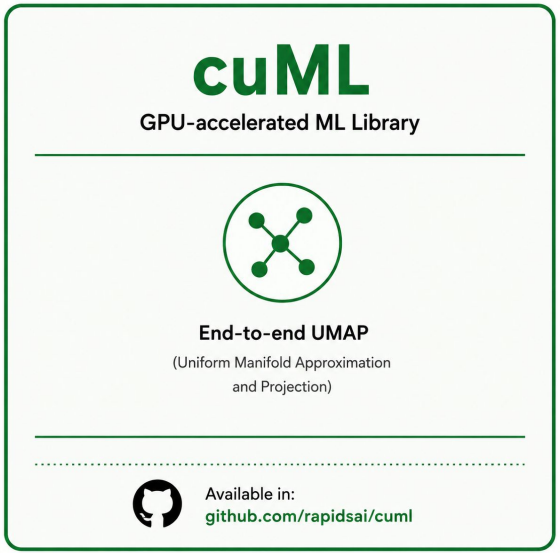


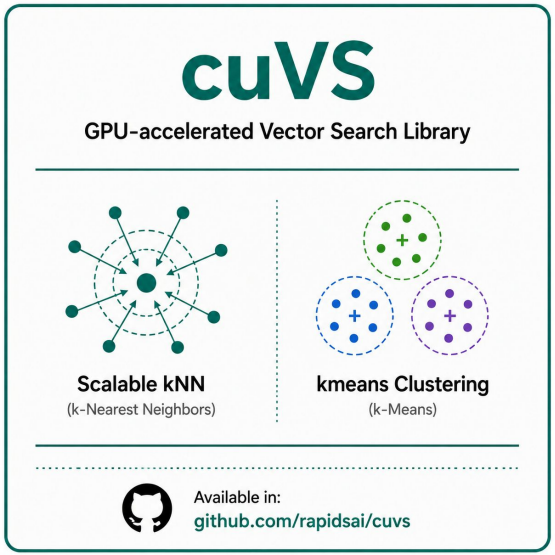
More compute but better recall with larger s

Takeaways

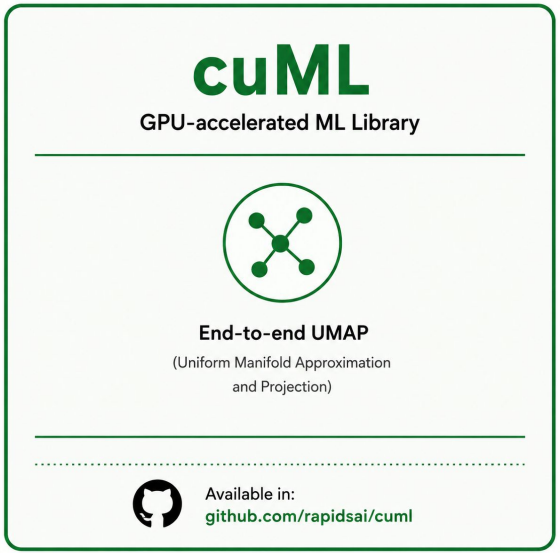
Scalable, Parallel, and Ready-to-Use

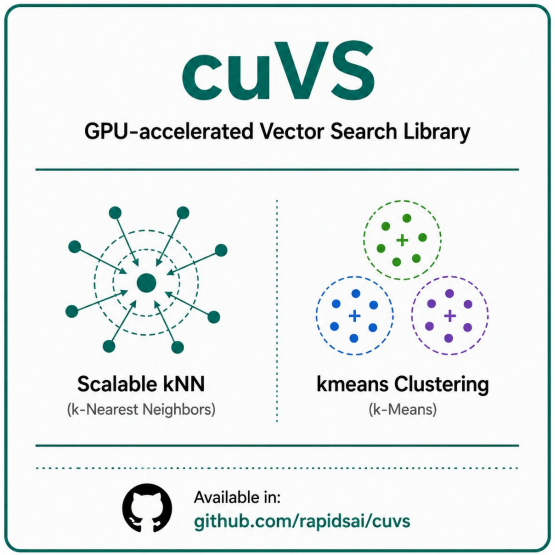
- Enables **out-of-core** UMAP beyond single GPU memory limits
- Our scalable kNN graph computation algorithm is applicable to **any downstream task**
- Everything is **open-sourced** and ready to use.





Thank You

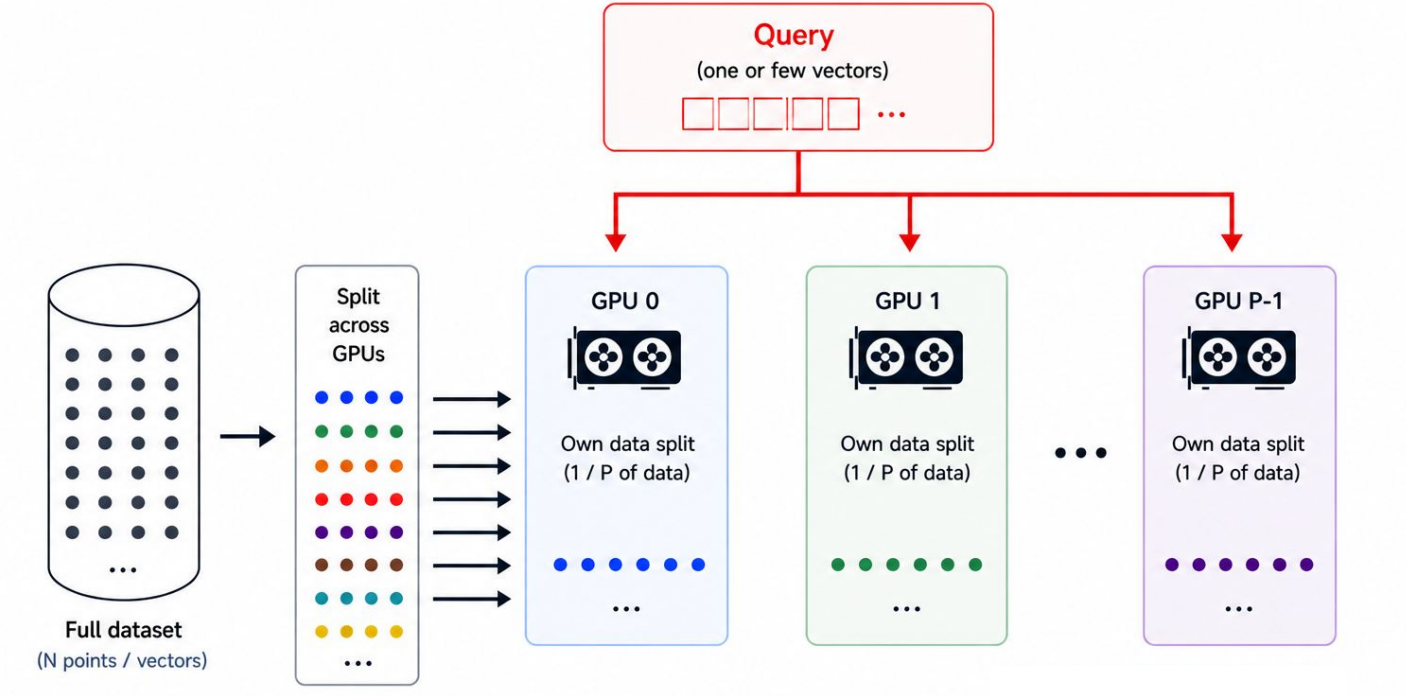

cuML
GPU-accelerated ML Library
End-to-end UMAP
(Uniform Manifold Approximation
and Projection)
Available in:
github.com/rapidsai/cuml


cuVS
GPU-accelerated Vector Search Library
Scalable kNN
(k-Nearest Neighbors)
kmeans Clustering
(k-Means)
Available in:
github.com/rapidsai/cuvs

Backup

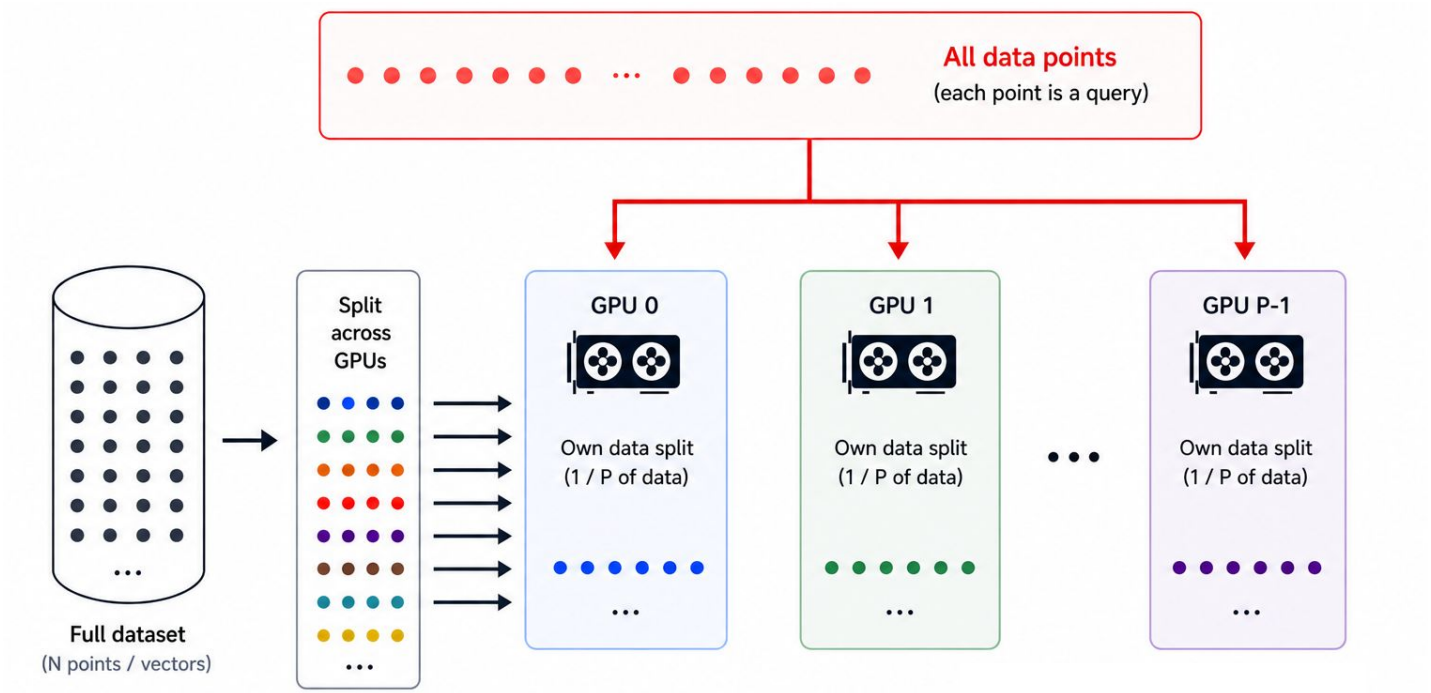
Online Vector Search

- Each query is sent to every GPU



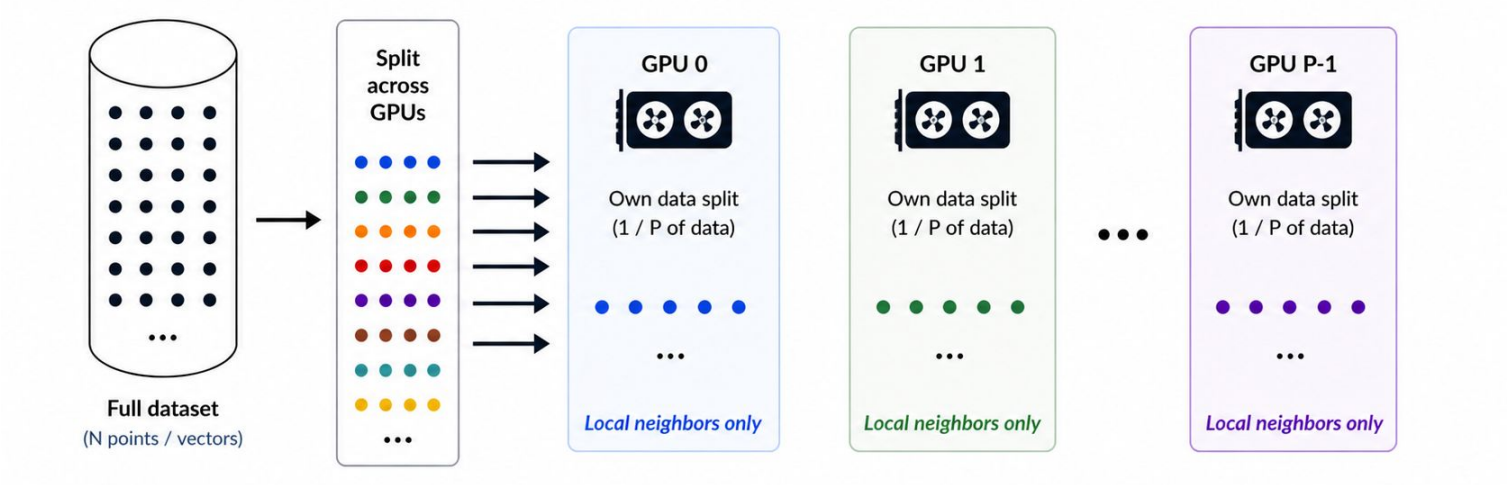
Finding Nearest Neighbors at Scale

- In UMAP, every point becomes a query (this is called an **all-neighbors** problem)



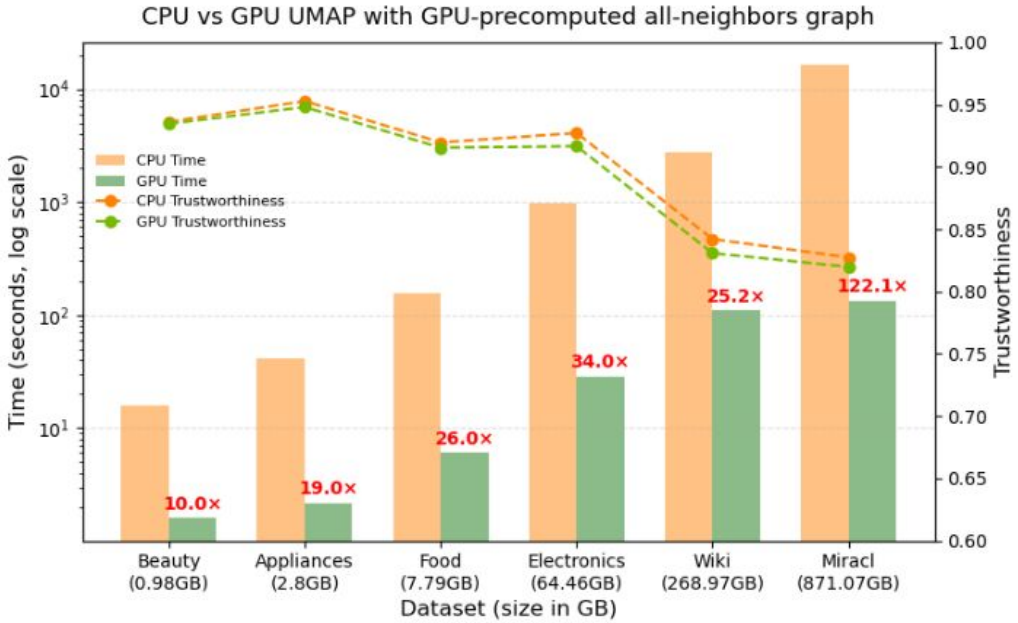
Our solution: Make the problem local

- Avoid redundant broadcasts by computing all-neighbors locally on each chunk



Results: Precomputed kNN

- With precomputed kNN, **122x** speedup over CPU UMAP
- Even without the kNN computation, GPU UMAP is significantly faster



Results: Impact of s and c

- Results of kNN