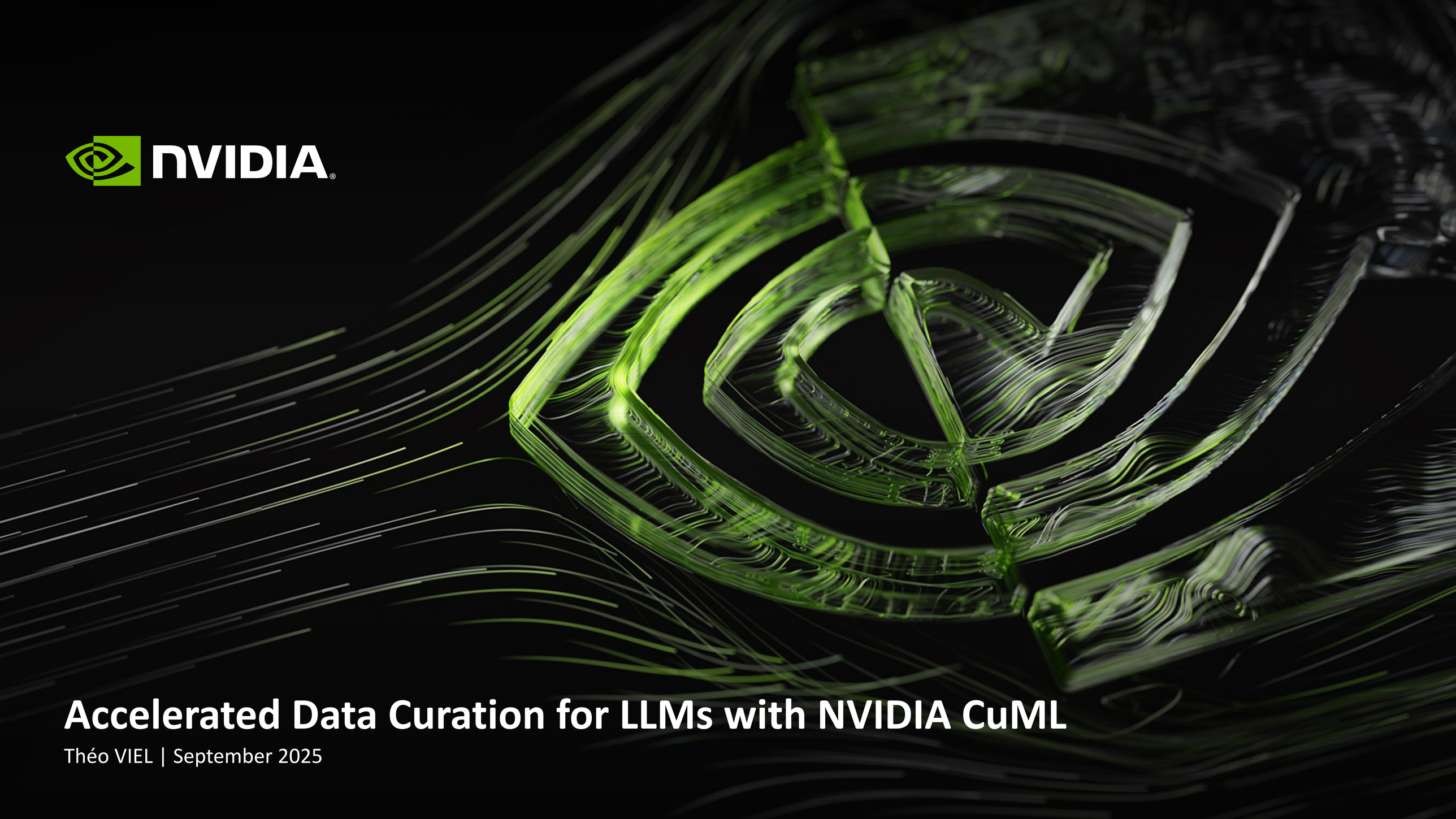


Accelerated Data Curation for LLMs with NVIDIA CuML

Théo VIEL | September 2025

Agenda

- Introduction
- Initialization: NVIDIA CuML DBSCAN & t-SNE
- Coreset
- Results

Kaggle GrandMasters Of NVIDIA (KGMON)

What We Do

- **Competitive Machine Learning**
  - Use and showcase NVIDIA products
    - GPUs
    - Software : NVIDIA CUDA-X DS (RAPIDS), XGBoost, MONAI, & more !
  - Kaggle, but not only !
    - MICCAI, CVPR, NeurIPS, …
    - ARC Prize


ONODERA


CPMP


Chris Deotte


Giba


Dieter


Theo Viel


David Austin

- **Modeling expertise for NVIDIA internal and external customers**
  - Competitive ML makes us up to date with “applied SOTA”
    - We know what works well in practice and how to squeeze the most out of models
  - Examples
    - The Kaggle Grandmasters Playbook: 7 Battle-Tested Modeling Techniques for Tabular Data
    - Open-source Nemotron RAG models family

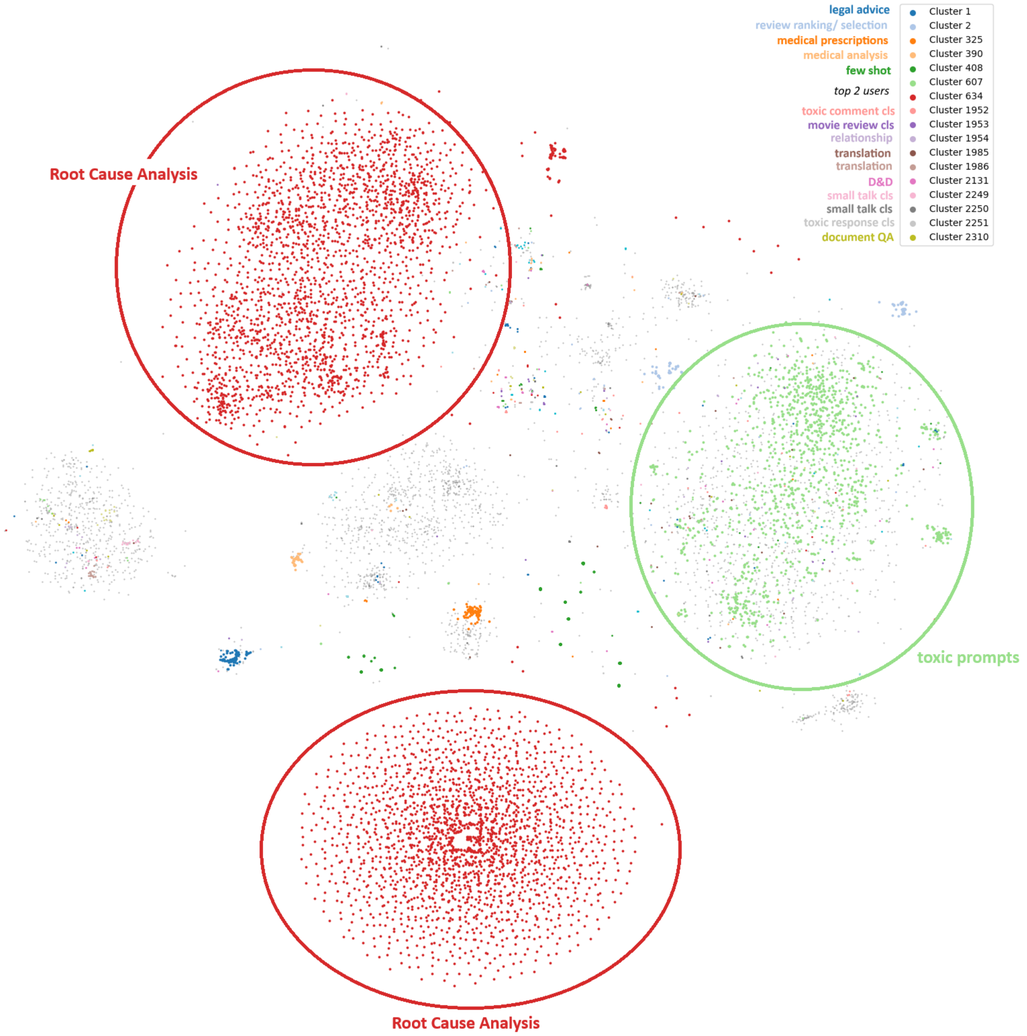
Introduction

Motivation & Idea

- **Motivation :** Subsample datasets
  - For training / finetuning models
    - More efficient training
    - Do not hinder performance with data biases
  - To build a validation set
    - Diverse and complex
  - Maximize use of manual annotation
    - And reduce time and costs

Data curation plays a significant role in the training of LLMs / VLMs. More data is not enough anymore.

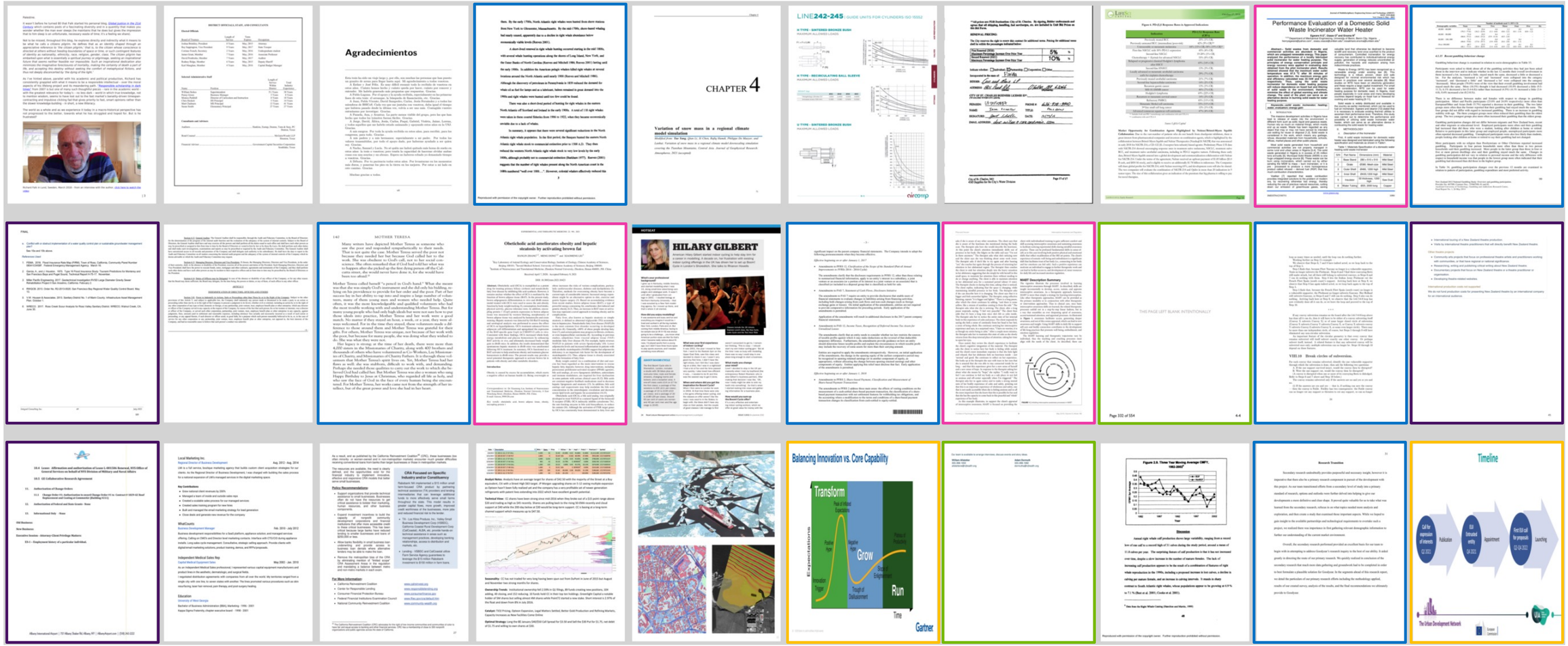
- **Idea :** Find a subset of the data that best covers the distribution
  - **Core-set** problem -> NP-Hard
  - Without losing modes in the data distribution
- **Github :** https://github.com/NVIDIA/Diversity-Sampling



T-SNE plot of a prompt dataset

Our Dataset

PDF Pages from Digital Corpora



Initialization

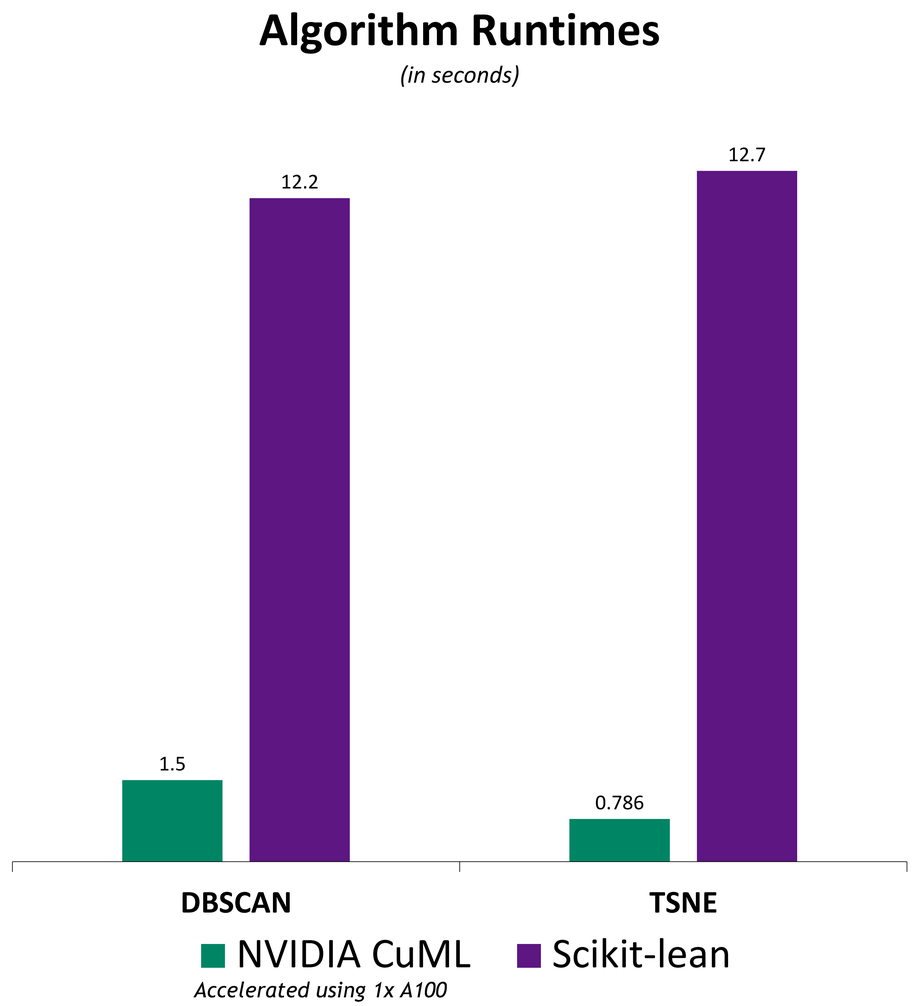
DBSCAN & t-SNE

Requirements :

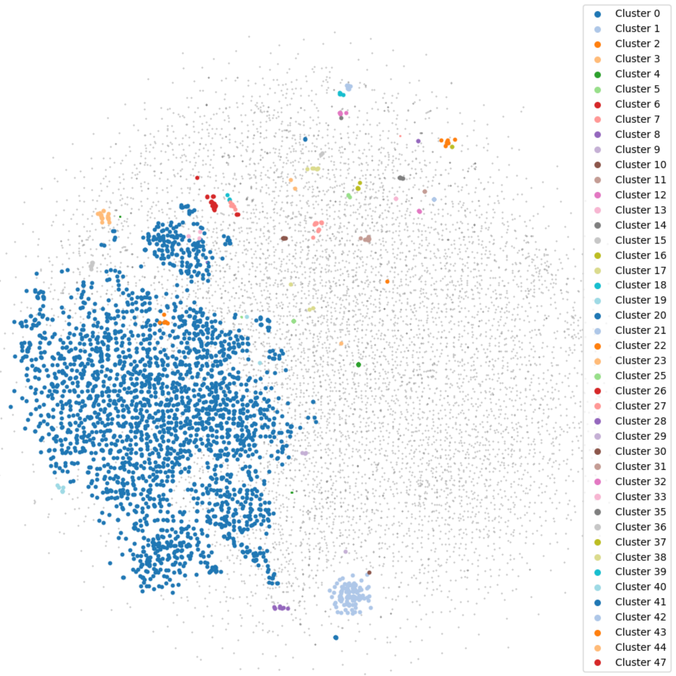
- A GPU
- Your data, embedded in a vector space
  - Using a text embedding model - e.g., `nvidia/NV-Embed-v2`
  - Using a Vision Transformer - e.g., `facebook/dinov2-small`, `openai/clip`

Objective : Find the clusters we want to keep in the sampling

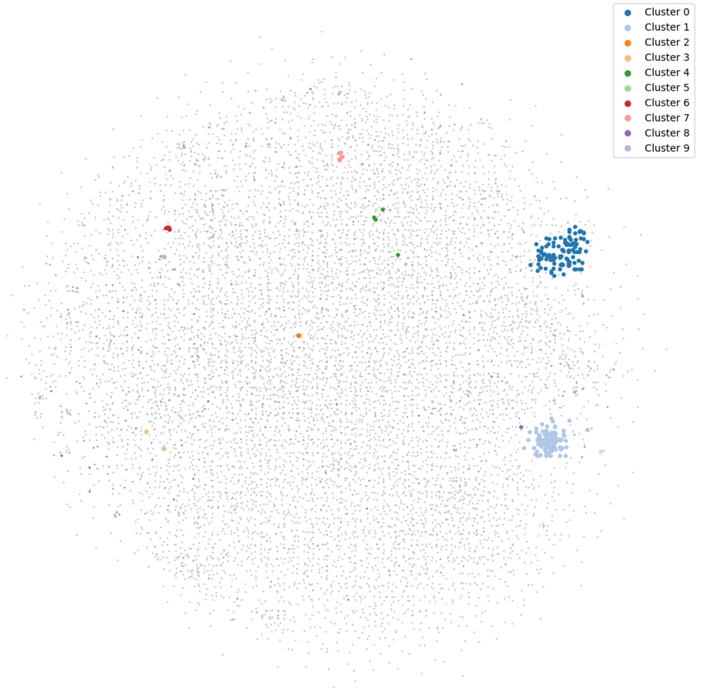
- Tweak DBSCAN parameters
  - `epsilon`: Controls distance used to group samples into clusters
  - `min_sample`: Minimum size needed to consider a cluster
- Use T-SNE for visualization



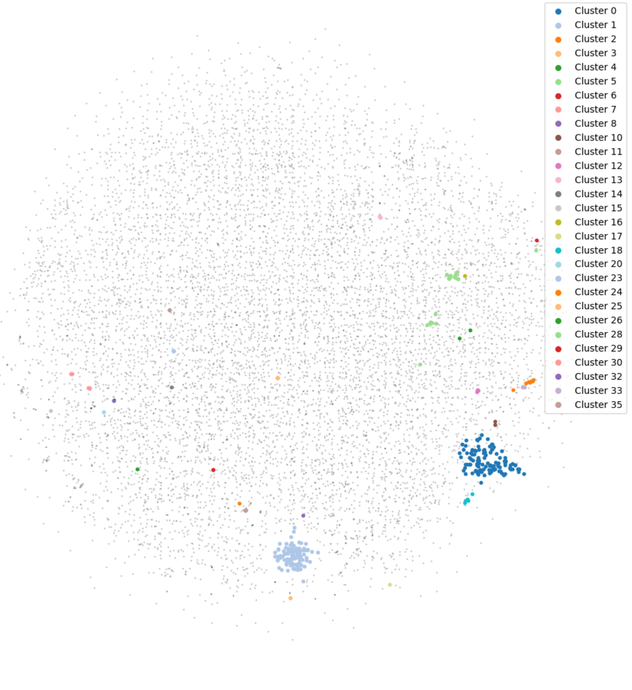
Tweaking DBScan



`eps=10 , min_samples=20`



`eps=5, min_samples=20`

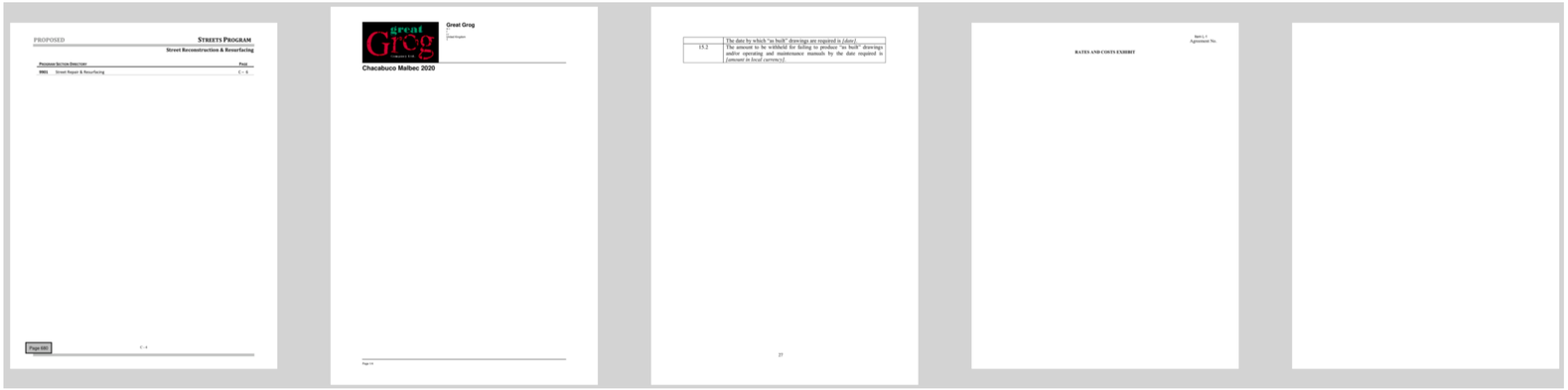


`eps=5, min_samples=12`

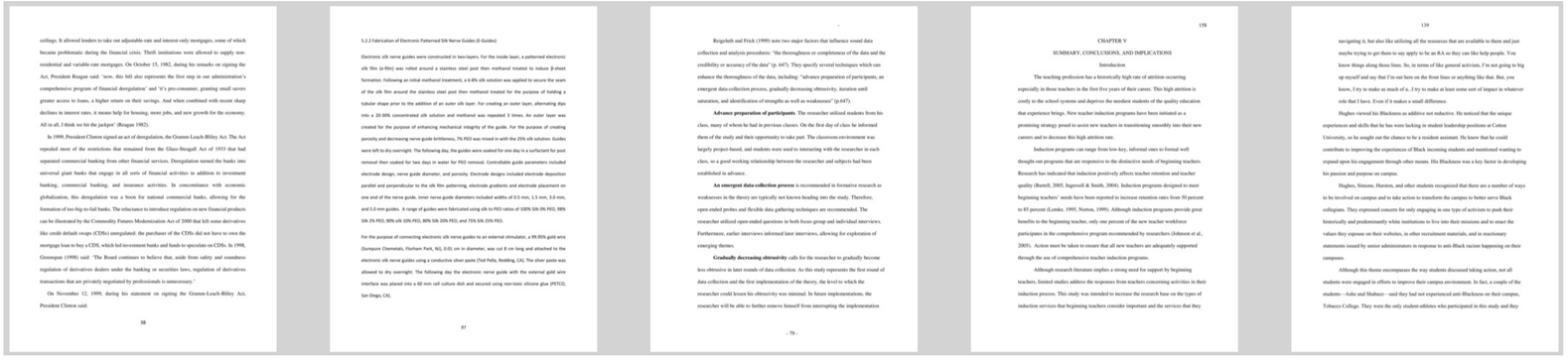
Clusters Preview

eps=5, min_samples=12

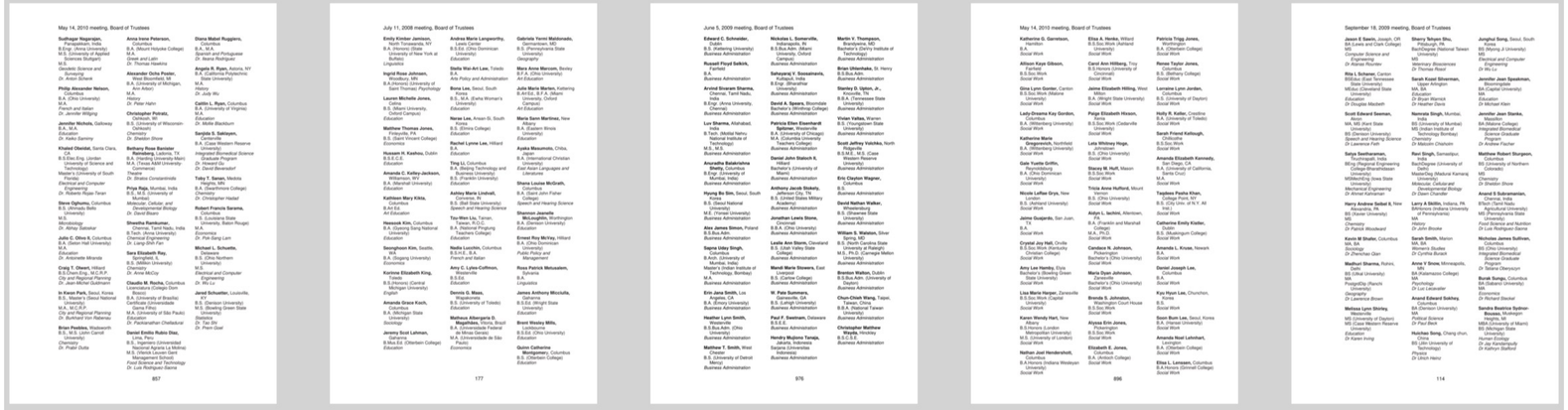
Cluster #1: (almost) empty pages



Cluster #0: Simple text



Cluster #11: Boards



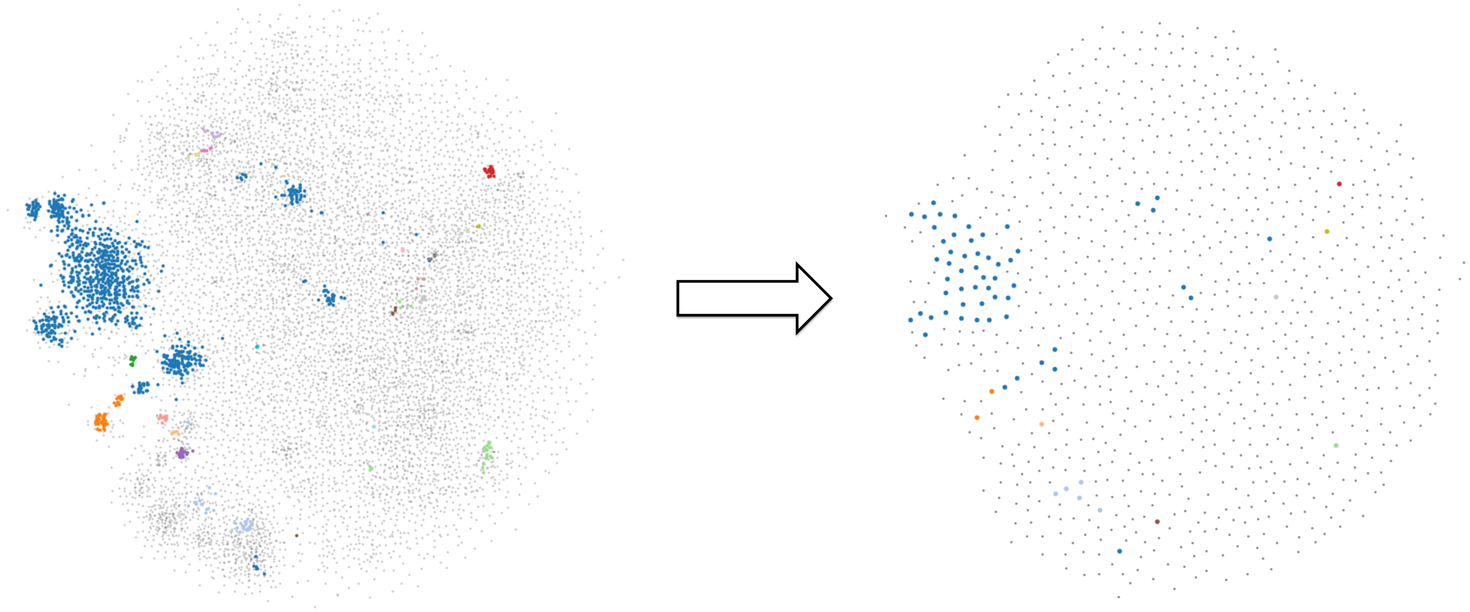
The Coreset Problem

- For a given dataset $D \in \mathbb{R}^{n \times d}$, build a subsample S of m data points that best represent D

Algorithm – Greedy approximation to the Coreset solution

- Initialize S with at least one data point
- While $\text{card}(S) < m$, append to S the data point $\hat{x} \in D$ that is the most diverse to it

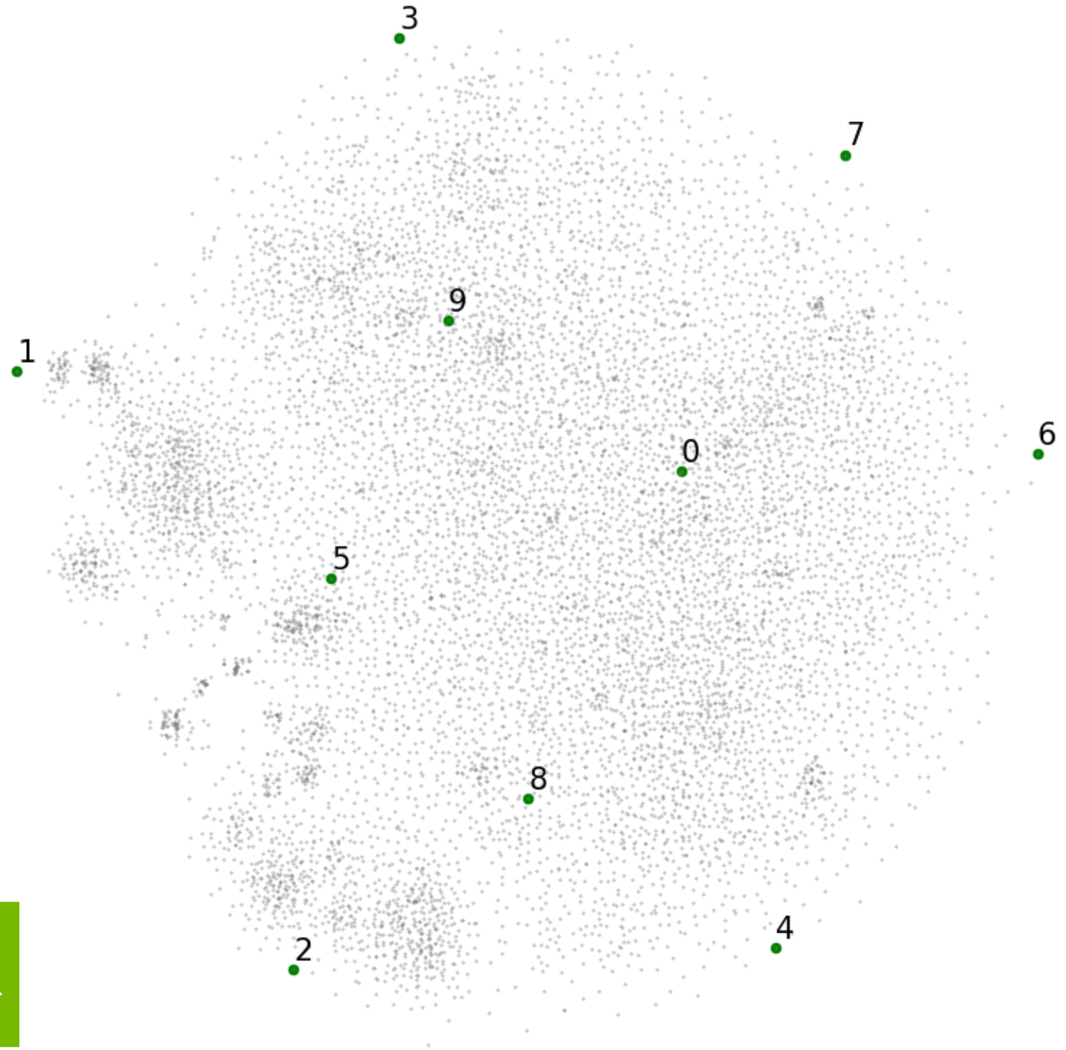
$$\hat{x} = \text{argmax}_{x \in D}(\|x - S\|)$$



Implementing & Optimizing

+ 2D example

- Initialize S_0 with a (or several) random point(s)
- At each iteration $i < m$:
  - Compute the distance between S_i and $D \in \mathbb{R}^{n \times d}$
    - $\|x - S_i\| = \min_{s \in S_i} \|x - s\|_2$ for all $x \in D$
  - Append the furthest point
    - $S_{i+1} = S_i + \{\hat{x}_i\}$ $\hat{x}_i = \text{argmax}_{x \in D}(\|x - S_i\|)$
- This is **slow** and uses a lot of memory
  - $\|x - S_i\|$ complexity is linear in S_i size plus D can be big
  - S_i gets bigger and bigger !
- Trick : use $\|x - S_{i-1}\|$ to compute $\|x - S_i\|$
  - $\min(\{a, b, c\}) = \min(\{c\}, \min(\{a, b\}))$
  - $\min_{s \in S_i} \|x - s\|_2 = \min\left(\underbrace{\min_{s \in S_{i-1}} \|x - s\|_2}_{\text{Computed at step } i-1 \text{ !}}, \|x - \hat{x}_i\|\right)$



NVIDIA
GPU accelerated !

Sampling speed: 2000 points per second !

Implementation

With PyTorch

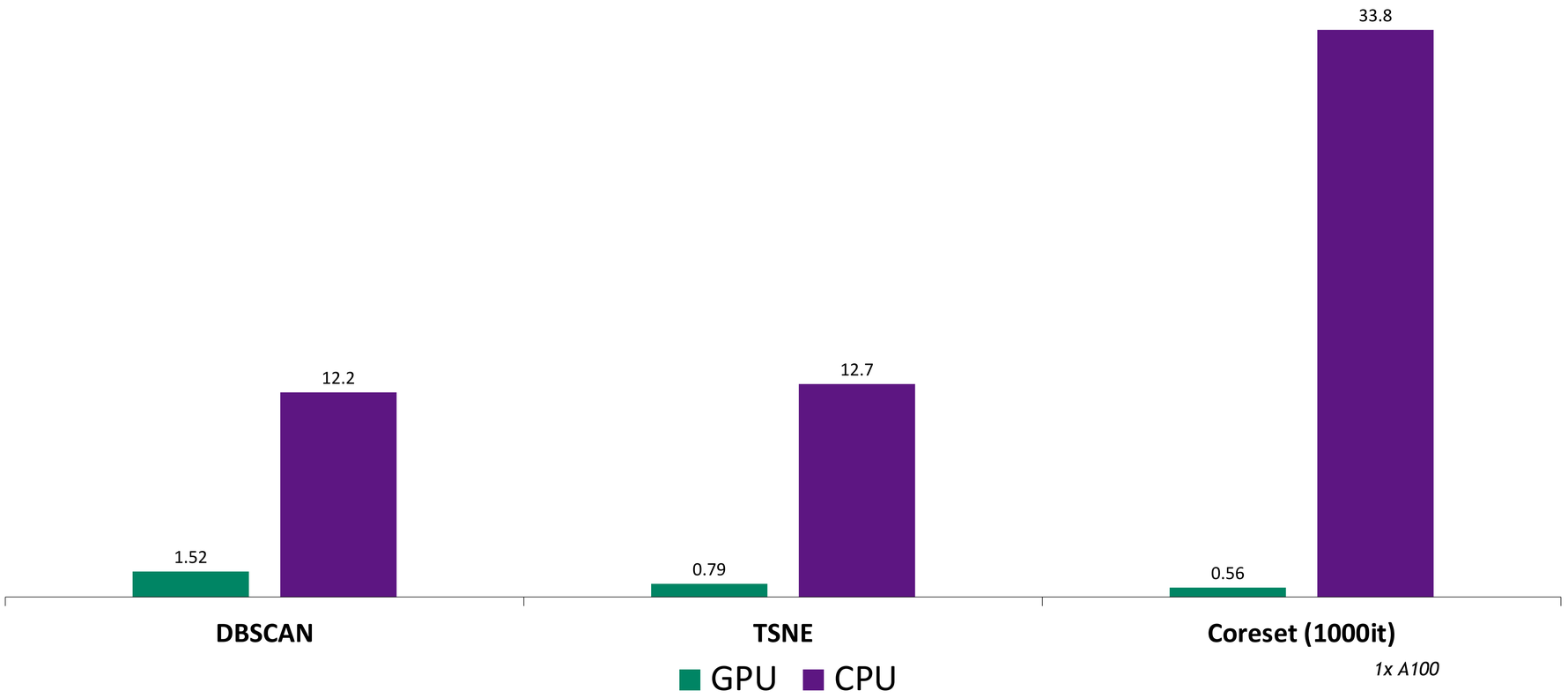
```
# Prerequisites
n_samples   # Number of points you want to sample
embeddings  # Your embedded data

# Initialization - DBScan
ids = initialize(embeddings)

# Move to GPU
embeddings = torch.from_numpy(embeddings).cuda()

# Compute minimum distances to the initialization ids of all embeddings
min_distances = [((embeddings[id] - embeddings) ** 2).sum(-1) for id in ids]  # [N x len(ids)]
min_distances = torch.minimum(min_distances, 1)  # [N]

# Sample new points
for _ in tqdm(range(len(ids), n_samples):
    # Sample the farthest point and update the dataset
    farthest = min_distances.argmax()
    ids.append(farthest)

    # Compute distances to the last sampled point
    new_dist = ((embeddings[farthest].unsqueeze(0) - embeddings) ** 2).sum(-1, keepdims=True)

    # Update the minimum distance using(min(all_previous, current))
    min_distances = torch.minimum(min_distances, new_dist)

# Done !
```

Use GPUs !

For a >10x speed-up

Algorithm Runtimes

(in seconds)

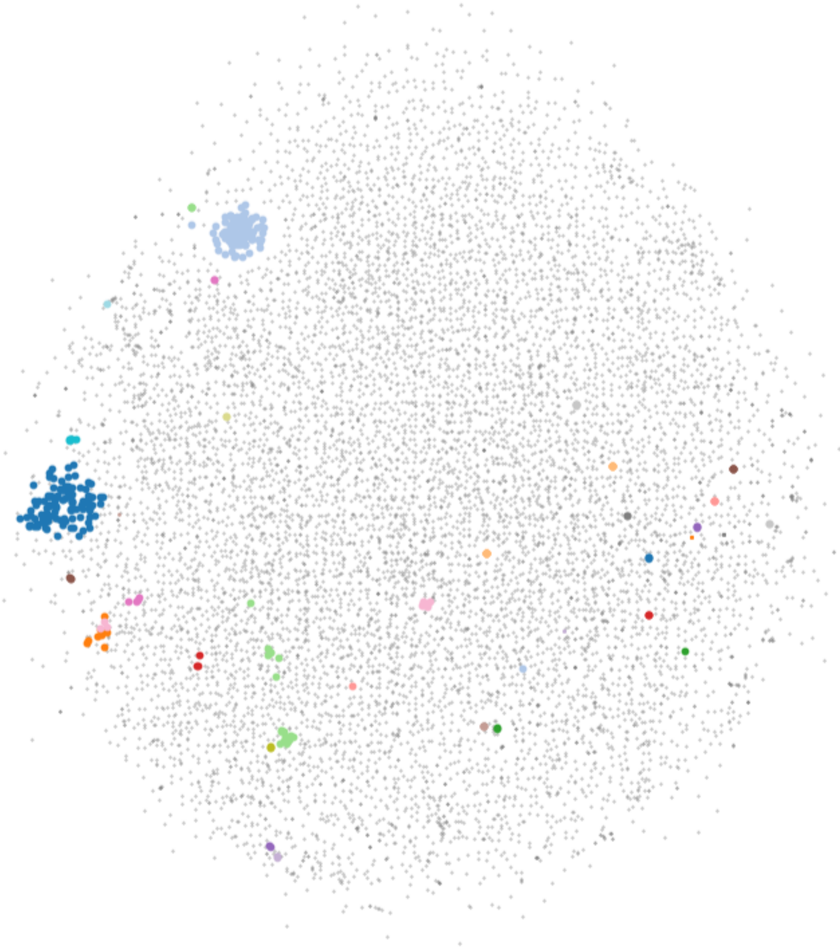


1x A100

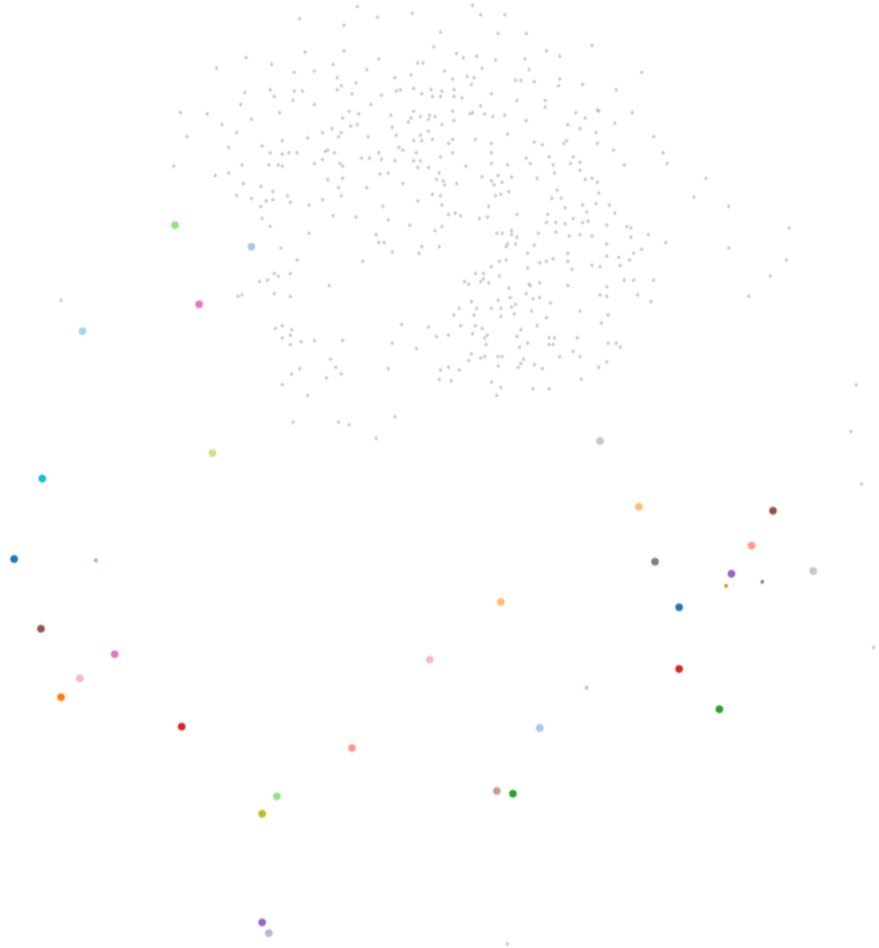
Results

T-SNE Plot

Before Coreset

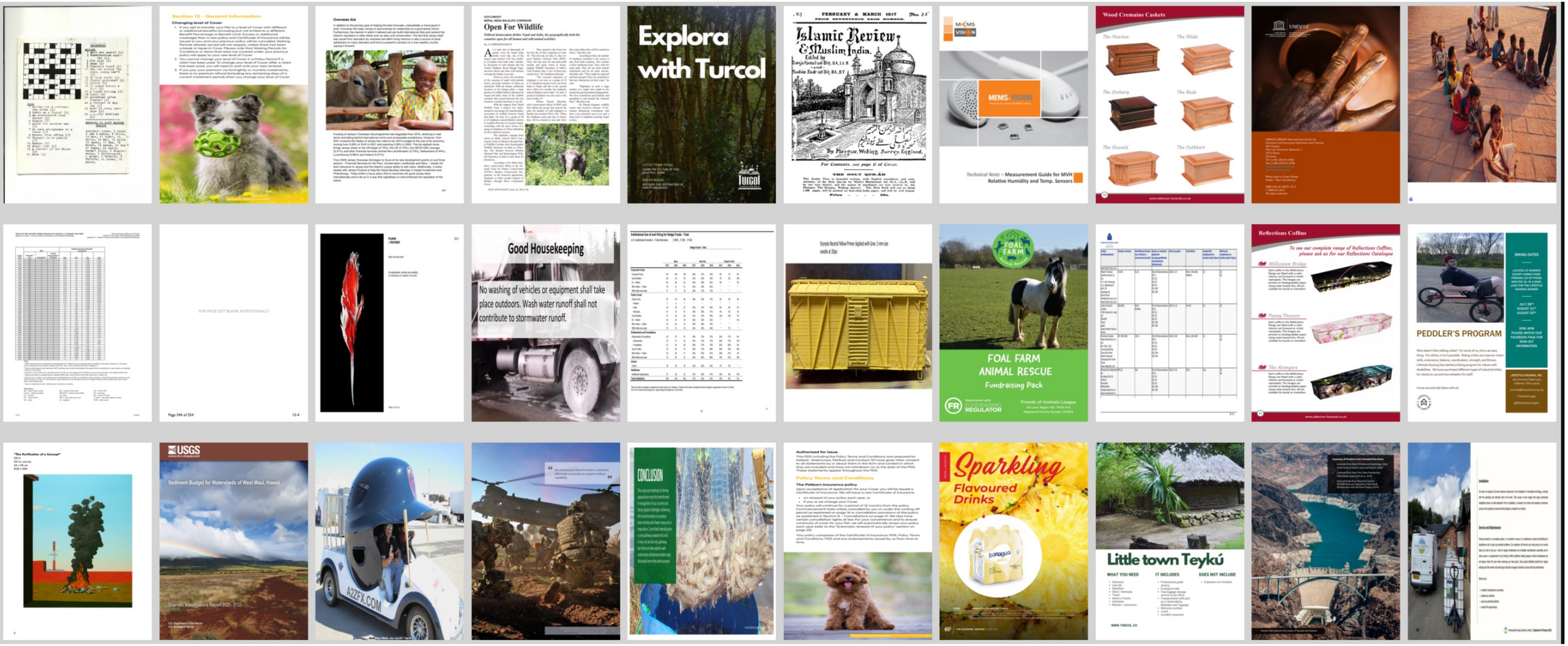


After Coreset



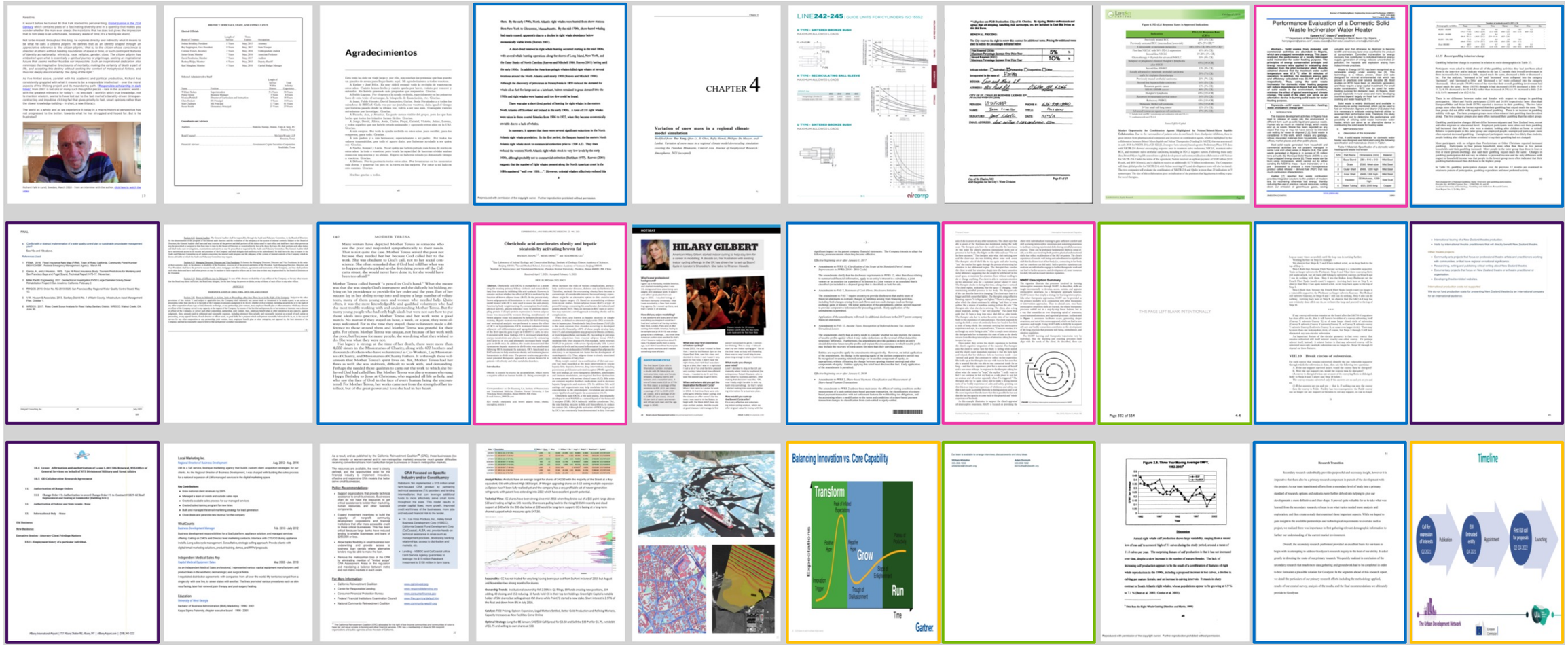
Results

Samples pages



Our Dataset

PDF Pages from Digital Corpora



NVIDIA

Try it yourself !

NVIDIA CuML: https://developer.nvidia.com/topics/ai/data-science/cuda-x-data-science-libraries/cuml

Diversity Sampling: https://github.com/NVIDIA/Diversity-Sampling