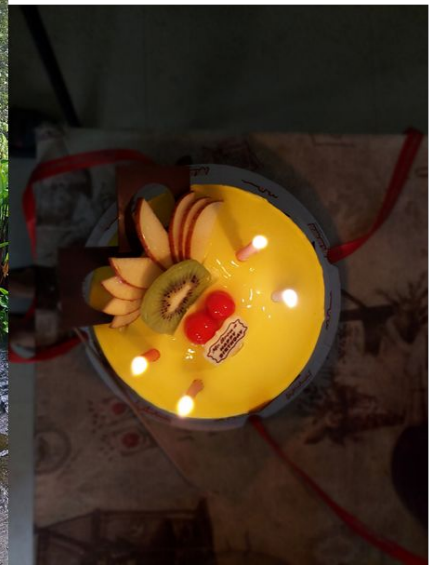
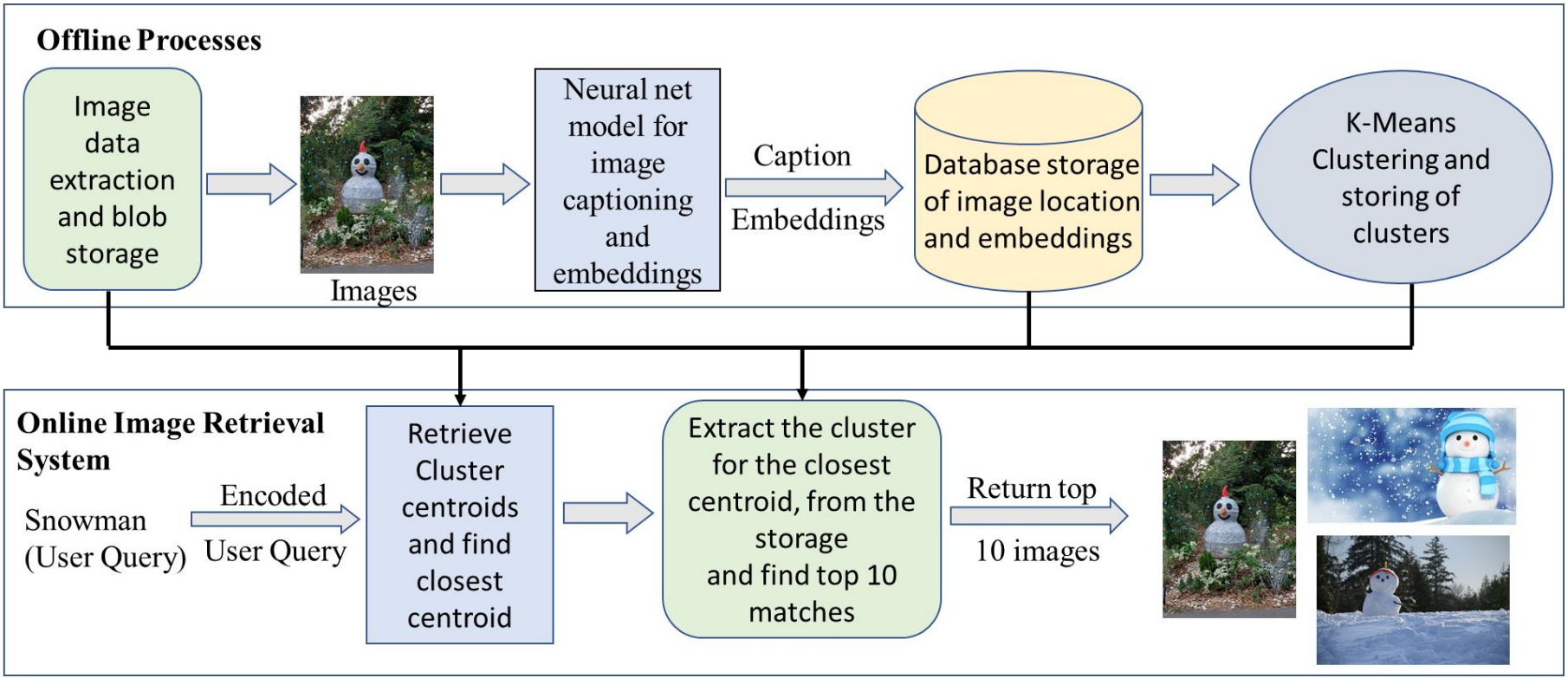
Image Retrieval System



Large number of images are captured each day by people all over the world. Most of these images are uncaptioned. The images in this slide being example of few such images taken by us. The problem we have tried to address in our project is how to use such uncaptioned large database of images for a scalable image retrieval systems which returns images based on human queries.

Overall Architecture



Dataset

- Training DL model -
  - MS COCO Training Images (~83K)

- Image Search corpus -
  - MS COCO Test images (~21K)
  - Google Open Images V6 (~41K)

The man at bat readies to swing at the pitch while the umpire looks on.

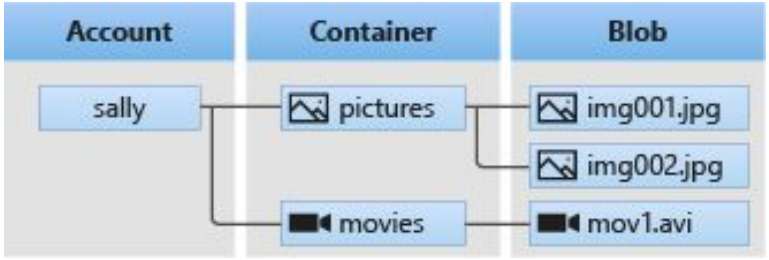
A large bus sitting next to a very tall building.

A horse carrying a large load of hay and two people sitting on it.

Bunk bed with a narrow shelf sitting underneath it.

Image Reference: Example images and captions from the Microsoft COCO Caption dataset.

Data Storage - Images



- Images are stored in the cloud using **Azure Blob Storage**.

- Features -
  - Optimized for storing massive amounts of unstructured data.
  - Flexible scale-up for ML workloads
  - Optimised for Data Lakes
  - Data retrievable using unique URLs

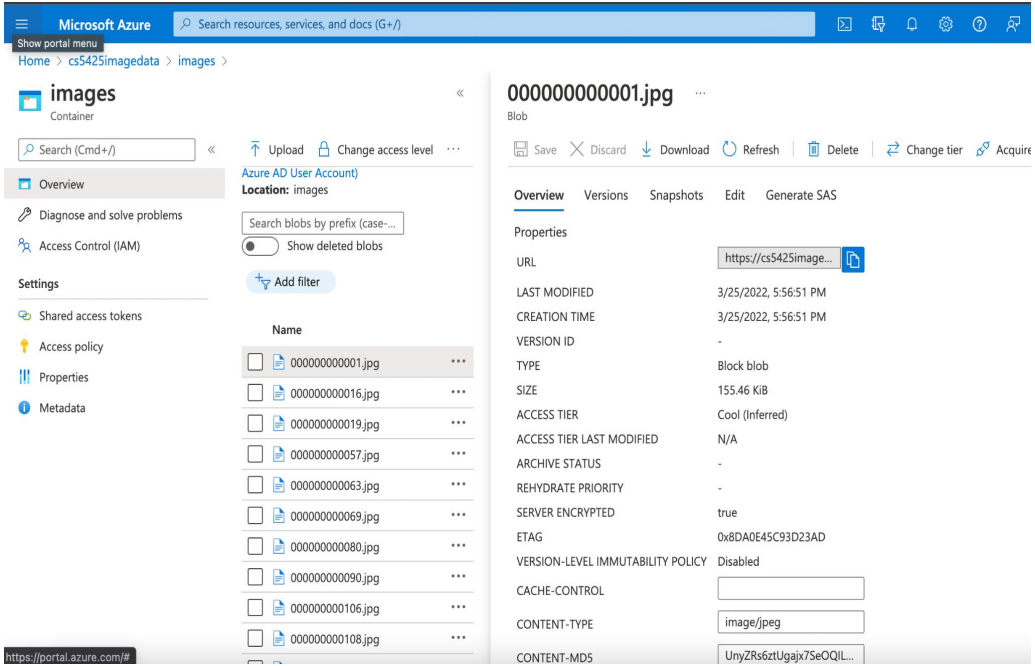
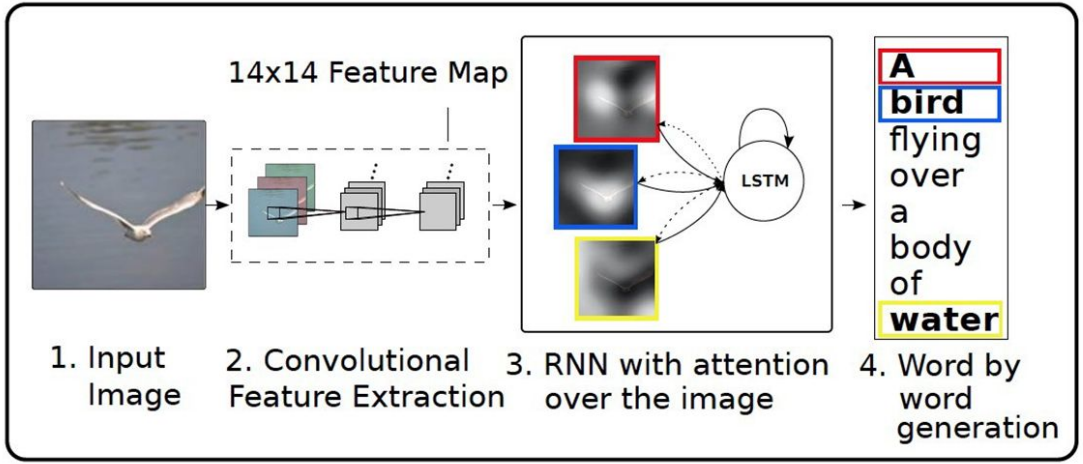


Image Reference: Relationship between blob resources

Deep Learning Model for Image Captioning

For captioning images we train the deep learning model based upon the paper Show, Attend and Tell: Neural Image Caption Generation with Visual Attention.

The model was trained using the COCO dataset and used a vocabulary of 7500 words. The final loss for the model was approximately 0.45. The captions were encoded using BERT.

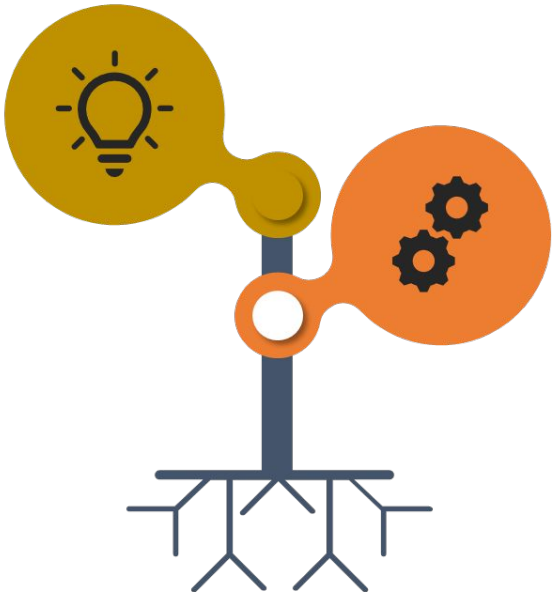


Generating Encodings

- For inferencing 2 methods were tried:

Inferencing without Spark

Normal inferencing had shorter runtime than the distributed but it cannot scale in case of billions of images. In case of a billion images using a multiple GPU system with spark or dividing the inferencing part such that it benefits both from CPUs and GPUs for distribution is preferrable. For current case generating and encoding captions for 60000 images took 3 hours.



Inferencing using Spark

Distributed inferencing by the model was tried using Spark. However as the inferencing was being done on a single GPU system this method took too long and was discarded.

Model Analysis

01

New Objects

The model did not perform so well on objects it had not seen during training such as rifles. In such case the model gave random captions.

Inconsistent Captions

The model is not consistent with captions and gives different captions for same image on running multiple times

02

03

Long range Dependency

GRUs are unable to capture long range dependencies in sentences, resulting in meaningless sentences.

Clustering embeddings

- Objective:
  - Cluster embeddings of existing images offline
  - The query embedding will only be compared within the cluster it's assigned to

- Principle Component Analysis (PCA) for dimension reduction:
  - The embeddings from deep learning model is high
  - Curse of dimensionality – difficult to cluster vectors in high dimension
  - Comparing high-dimensional vectors is more costly

- Algorithms experimented:
  - K-means, DBSCAN, and Gaussian Mixture Model (GMM)
  - K-means works the best because:
    - Parameters in DBSCAN are sensitive and difficult to tune
    - GMM requires storing covariance matrices in database, and soft-clustering is not needed here
  - Cluster centroids and images' memberships are stored in database

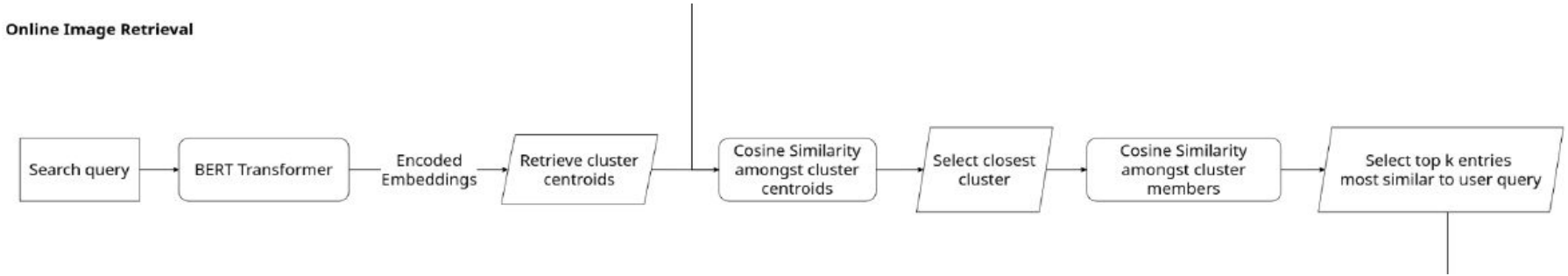
Image retrieval

- Objective:
  - Find the most similar images to the query

- Assign cluster:
  - The query embedding is passed to PCA and K-means model
  - K-means will assign the query embedding a cluster

- Within-cluster comparison:
  - Query embedding is only compared against image embeddings within the cluster assigned
  - Cosine similarity to measure the similarities
  - Returns the top K most similar image ids to front end for display

Request Scaling



Azure Function Apps (Serverless) - Analysis

Elasticity	Can deal with surges and drops in request numbers effectively with little overhead.
Scalability	Can scale up or down indefinitely. No need to worry about provisioning more resources manually, handled automatically by Azure.
Memory	As the Bert model is large in size, we need to minimize the network overhead of accessing it or redownloading it every function call. Azure offers mounted drive architecture such that each worker can access the pre-downloaded model as if it were stored in a local drive.
Parallelizability	1 function call handles 1 request. Function is atomic and thus there is lots of repeated work. For example, each function call will initiate a DB connection, load models etc.
Cost	Cheaper compared to running our own dedicated server, pay-per-use. Periods of lower demand will have correspondingly lower cost, resources will not be left idle and wasted.
Maintainability	Easy to integrate into a CI/CD pipeline such that updates to the function can easily be applied.