Large-Scale Data Mining: Models and Algorithms **ECE 219 Winter 2026**
Prof. Vwani Roychowdhury
UCLA, Department of ECE

Project 2: Clustering & Unsupervised Learning and Intro to Multi-modal models

Due February 20, 2026 by 11:59 pm

Introduction

Machine learning algorithms are applied to a wide variety of data, including text and images. Before applying these algorithms, one needs to convert the raw data into feature representations that are suitable for downstream algorithms. In project 1, we studied feature extraction from text data, and the downstream task of classification. We also learned that reducing the dimension of the extracted features often helps with a downstream task.

In this project, we explore the concepts of feature extraction and clustering together. In an ideal world, all we need are data points – encoded using certain features– and AI should be able to find what is important to learn, or more specifically, determine what are the underlying modes or categories in the dataset. This is the ultimate goal of General AI: the machine is able to bootstrap a knowledge base, acting as its own teacher and interacting with the outside world to explore to be able to operate autonomously in an environment.

We first explore this field of unsupervised learning using textual data, which is a continuation of concepts learned in Project 1. We ask if a combination of feature engineering and clustering techniques can automatically separate a document set into groups that match known labels.

Next we focus on a new type of data, i.e. images. Specifically, we first explore how to use "deep learning" or "deep neural networks (DNNs)" to obtain image features. Large neural networks have been trained on huge labeled image datasets to recognize objects of different types from images. For example, networks trained on the Imagenet dataset can classify more than one thousand different categories of objects. Such networks can be viewed as comprising two parts: the first part maps a given RGB image into a feature vector using convolutional filters, and the second part then classifies this feature vector into an appropriate category, using a fully-connected multi-layered neural network (we will study such NNs in a later lecture). Such pre-trained networks could be considered as experienced agents that have learned to discover features that are salient for image understanding. Can one use the experience of such pre-trained agents in understanding new images that the machine has never seen before? It is akin to asking a human expert on forensics to explore a new crime scene. One would expect such an expert to be able to *transfer* their domain knowledge into a new scenario. In a similar vein, can a pre-trained network for image understanding be used for **transfer learning**? One could use the output of the network in the last few layers as expert features. Then, given a multi-modal dataset –consisting of images from categories that the DNN was not trained for– one can use feature engineering (such as dimensionality reduction) and clustering algorithms to automatically extract unlabeled categories from such expert features.

For both the text and image data, one can use a common set of multiple evaluation metrics to compare the groups extracted by the unsupervised learning algorithms to the corresponding ground truth human labels.

Clustering Methods

Clustering is the task of grouping a dataset in such a way that data points in the same group (called a cluster) are more similar (in some sense) to each other than to those in other groups (clusters). Thus, there is an inherent notion of a metric that is used to compute similarity among data points, and different clustering algorithms differ in the type of similarity measure they use, e.g., Euclidean vs Riemannian geometry. Clustering algorithms are considered "unsupervised learning", i.e. they do not require labels during training. In principle, if two categories of objects or concepts are distinct from some perspective (e.g. visual or functional), then data points from these two categories – when properly coded in a feature space and augmented with an associated distance metric – should form distinct clusters. Thus, if one can perform perfect clustering then one can discover and obtain computational characterizations of categories without any labeling. In practice, however, finding such optimal choices of features and metrics has proven to be a computationally intractable task, and any clustering result needs to be validated against tasks for which one can measure performance. Thus, we use labeled datasets in this project, which allows us to evaluate the learned clusters by comparing them with ground truth labels.

Below, we summarize several clustering algorithms:

K-means: K-means clustering is a simple and popular clustering algorithm. Given a set of data points $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ in multidimensional space, and a hyperparameter K denoting the number of clusters, the algorithm finds the K cluster centers such that each data point belongs to exactly one cluster. This cluster membership is found by minimizing the sum of the squares of the distances between each data point and the center of the cluster it belongs to. If we define $\boldsymbol{\mu}_k$ to be the "center" of the kth cluster, and

$$r_{nk} = \begin{cases} 1, & \text{if } \mathbf{x}_n \text{ is assigned to cluster } k \\ 0, & \text{otherwise} \end{cases}, \quad n = 1, \ldots, N \quad k = 1, \ldots, K$$

Then our goal is to find r_{nk}'s and $\boldsymbol{\mu}_k$'s that minimize $J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \|\mathbf{x}_n - \boldsymbol{\mu}_k\|^2$. The approach of K-means algorithm is to repeatedly perform the following two steps until convergence:

1. (Re)assign each data point to the cluster whose center is nearest to the data point.
2. (Re)calculate the position of the centers of the clusters: setting the center of the cluster to the mean of the data points that are currently within the cluster.

The center positions may be initialized randomly.

Hierarchical Clustering Hierarchical clustering is a general family of clustering algorithms that builds nested clusters by merging or splitting them successively. This hierarchy of clusters is represented as a tree (or dendrogram). A flat clustering result is obtained by cutting the dendrogram at a level that yields a desired number of clusters.

DBSCAN DBSCAN or Density-Based Spatial Clustering of Applications with Noise finds core samples of high density and expands clusters from them. It is a density-based clustering non-parametric algorithm: Given a set of points, the algorithm groups together points that are closely packed together (points with many nearby neighbors), marking as outliers points that lie alone in low-density regions (whose nearest neighbors are too far away).

HDBSCAN HDBSCAN extends DBSCAN by converting it into a hierarchical clustering algorithm, and then using an empirical technique to extract a flat clustering based on the

stability of clusters (similar to the elbow method in k-Means). The resulting algorithm gets rid of the hyperparameter "epsilon", which is necessary in DBSCAN (see here for more on that).

Common Clustering Evaluation Metrics

In order to evaluate a clustering pipeline, one can use the ground-truth class labels and compare them with the cluster labels. This analysis determines the quality of the clustering algorithm in recovering the ground-truth underlying labels. It also indicates if the adopted feature extraction and dimensionality reduction methods retain enough information about the ground-truth classes. Below we provide several evaluation metrics available in `sklearn.metrics`. *Note that for the clustering sub-tasks, you do not need to separate your data to training and test sets.*

Homogeneity is a measure of how "pure" the clusters are. If each cluster contains only data points from a single class, the homogeneity is satisfied.

Completeness indicates how much of the data points of a class are assigned to the same cluster.

V-measure is the harmonic average of homogeneity score and completeness score.

Adjusted Rand Index is similar to accuracy, which computes similarity between the clustering labels and ground truth labels. This method counts all pairs of points that both fall either in the same cluster and the same class or in different clusters and different classes.

Adjusted mutual information score measures the mutual information between the cluster label distribution and the ground truth label distributions.

Dimensionality Reduction Methods

In project 1, we studied SVD/PCA and NMF as linear dimensionality reduction techniques. Here, we consider some additional non-linear methods.

Uniform Manifold Approximation and Projection (UMAP) The UMAP algorithm constructs a graph-based representation of the high-dimensional data manifold, and learns a low-dimensional representation space based on the relative inter-point distances. UMAP allows more choices of distance metrics besides Euclidean distance. In particular, we are interested in "cosine distance" for text data, because as we shall see it bypasses the magnitude of the vectors, meaning that the length of the documents does not affect the distance metric.

Autoencoders An autoencoder[1] is a special type of neural network that is trained to copy its input to its output. For example, given an image of a handwritten digit, an autoencoder first encodes the image into a lower dimensional latent representation, then decodes the latent representation back to an image. An autoencoder learns to compress the data while minimizing the reconstruction error. Further details can be found in chapter 14 of [4].

[1] also known as "auto-associative networks" in older jargon

Part 1 - Steam Reviews: Product Analytics with Representations and Clustering

Imagine you are a **product engineer** (or data scientist) on a game platform team. Your job is to help the company answer questions like:

- **Are players actually enjoying this game?** How does sentiment compare across games?
- **What are players complaining about most?** (crashes? balance? monetization? controls?)
- **What makes a game feel like a particular genre?** Can we infer genre from what players say?
- **For a new or held-out game, can we profile it quickly?** (positivity ratio, likely genre, top issues, top praises)

We will the following toolkit to solve these tasks:

- text representations (sparse vs dense),
- dimensionality reduction (to make patterns easier to discover),
- clustering (to discover themes without manual labeling),
- and evaluation using known signals (ratings, thumbs up/down) when available.

In this first part of the project, You will build a system that can:

1. discover clusters in the review lengths (Task 1),
2. infer game genres from **positive player feedback** (Task 2),
3. and generate a concise **product report** for a held-out game (Task 3).

Dataset

You are provided with a curated subset of Steam reviews (Download here) :

- **Main dataset (CSV):** reviews from the **top 200 games** ranked by total reviews (`positive+negative`) in the metadata. For each game, we keep:
  - 100 English **Recommended** reviews, and
  - 100 English **Not Recommended** reviews,

  selected by highest helpfulness (upvotes). This ensures the reviews are typically informative rather than one-word memes.
- **Held-out dataset (CSV):** *A secret* game's reviews are provided separately and will be used only in Task 3.

Each review row includes:

- **user**: User Name
- **playtime**: Number of hours this Uswer plays this game.
- **post_date**: When is the review posted.
- **helpfulness**: Number of upvotes this review receieved.
- **review_text**: The review itself.
- **recommend**: Whether the user recommended the game or not. True or False
- **early_access_review**: Whether this review is for a game during early access stage, either empty or early_access.
- **appid**: Unique id of the game.
- **game_name**: Game name.
- **release_date**: Relase date of the game.
- **genres**: Genres of the game, either single a string separated by comma.

Important note on genres: Games often have **multiple genres**. In this project, genres should be treated as **multi-label metadata**.

Methods and Modules

Your system will be built from modular choices:

- **Representations:** TF-IDF, MiniLM embeddings (`sentence-transformers/all-MiniLM-L6-v2`).
- **Dimensionality Reduction:** None, SVD, UMAP, Autoencoder.
- **Clustering:** K-Means, Agglomerative, HDBSCAN.

Default setting policy: To keep the project lightweight and focus on interpretation, you only need to run **one default hyperparameter choice per method** (unless a question explicitly requests a sweep).

Module	Alternatives	Default Hyperparameters
Dimensionality Reduction	None	N/A
	SVD	$r = 50$
	UMAP	n_components = 50
	Autoencoder	latent dim = 50
Clustering	K-Means	$k = 2$ (Task 1), $k = 5$ (Task 3 themes)
	Agglomerative	$n\_clusters = 2$ (Task 1), $n\_clusters = 5$ (Task 3 themes)
	HDBSCAN	min_cluster_size = 2/5 (can experiment here a bit if noise dominates)

Compute note (Colab): Dense embeddings + UMAP/Autoencoder can be memory intensive.

Note on TF-IDF: For TF-IDF, due to its large and sparse representations, running certain methods can be really slow. Thus, you can skip the **None**, **UMAP** and **Autoencoder** for it.

Note on HDBSCAN: You cannot specify the designed number of cluster you want for HDBSCAN, instead, you will specify the minimum cluster size for each cluster it finds. For tasks

that we want a specific number of cluster, consider finding the largest two clusters, getting it's centroids, and assign the rest points to them)

Note on UMAP: UMAP might not work efficiently on high-dimensional data. Consider first using SVD to reduce raw data to a dimension of 200, then use UMAP.

Note on Agglomerative: If Agglomerative method runs really slow, consider using the following setups:

```
conn = kneighbors_graph(
    Z,
    n_neighbors=k,
    mode="connectivity",
    include_self=False
)

model = AgglomerativeClustering(
    n_clusters=2,
    linkage="ward",
    connectivity=conn
)
```

Task 1 - Unsupervised Review Length Discovery

We begin with a **warm-up unsupervised task** designed to build intuition about **representations, geometry, and clustering** before tackling semantic concepts.

In this task, the goal is to answer the question:

> *"Can we discover review length structure from our textual representations"*

Although review length is not a semantic label, it induces strong and interpretable structure in common text representations. This makes it an ideal first unsupervised learning problem.

Task 1.1 Defining pseudo-labels (for evaluation only)

For each review, define its **length** as the number of tokens (or words) in the review text.

To simplify the task and create a clear separation:

- Reviews in the **top 25%** ($\geq$ q75) by length are labeled as **Long**,
- Reviews in the **bottom 25%** ($\leq$ q25) by length are labeled as **Short**,
- Reviews in the middle 50% are **discarded** for this task.

These labels are used **only for evaluation**. During clustering, review length labels must be treated as **unknown**.

QUESTION 1: Report the number of reviews retained after filtering, and the average length (in tokens) of Short and Long reviews.

Task 1.2 Representations

1. **TF-IDF representation.** Construct TF-IDF features using `min_df=3` and English stopwords. Use unigrams only.
2. **MiniLM embeddings.** Compute dense sentence embeddings using `sentence-transformers/all-MiniLM-L6-v2`.

QUESTION 2: Report the dimensions of the TF-IDF matrix and the MiniLM embedding matrix. Briefly explain why TF-IDF is sparse while MiniLM embeddings are dense.

Task 1.3 Clustering pipelines

For each representation, run clustering pipelines with:

- **Dimensionality reduction:** None, SVD(50), UMAP(50),
- **Clustering:** K-Means ($k = 2$), Agglomerative (n_clusters = 2), optionally HDBSCAN

Note: You might notice when no dimensionality reduction method is used for TF-IDF embeddings, some of the clustering methods might crash; if it does, mention the reason in the report and skip them. Agglomerative requires dense input. If using TF-IDF, consider applying SVD first or convert to dense only if feasible. Applying UMAP directly on TF-IDF might be very slow, consider using SVD before.

QUESTION 3: For each pipeline, report the following clustering agreement metrics with respect to the ground-truth length labels: **homogeneity, completeness, v-measure, ARI, AMI**.

Summarize results in a table and identify the best-performing pipeline.

Task 1.4 Interpretation

QUESTION 4: Compare TF-IDF and MiniLM performance on this task. Which representation separates Short vs Long reviews more cleanly, and why?

QUESTION 5: Plots and Visualization: Select the best-performing configuration for **TF–IDF** and **MiniLM** based on clustering performance. For each representation:

- Reduce the embeddings to two dimensions using PCA (`sklearn.decomposition.PCA`).
- Create a split visualization with:
  - One plot colored by the ground-truth `length_label`.
  - One plot colored by the cluster assignments obtained using your best clustering method.

The resulting plots should enable a direct visual comparison between the true labels and the discovered clusters for both TF–IDF and MiniLM representations, highlighting how well each representation supports unsupervised separation.

Note. You should observe that review length can be separated *somehow well* by certain settings of unsupervised clustering. This is intentional.

Review length induces strong, global structure in representation space, making it a good first sanity-check problem for unsupervised learning pipelines.

Another intuitive tasks, as you may think of, is the sentiment analysis (positive comments vs. negative comments). If you are interested, feel free to try on it and report what you find (it is definitely much more difficult!)

Task 2 - Unsupervised Game Similarity & Genre Structure

In Task 2, we are now trying to answer a different question: *"Can we group games into meaningful clusters based on what players praise?"* In other words, instead of clustering *reviews*, we will cluster *games*.

Task 2.1 Construct one representation per game (positive reviews only)

Use only **positive reviews** (`recommend=True`). For each game, construct a single vector representation:

- **TF-IDF game vector (baseline):** concatenate all positive reviews for the game into one document, then compute TF-IDF for all games.
- **MiniLM game vector:** compute MiniLM embeddings for each positive review, then **average** them to obtain one embedding per game.

QUESTION 6: Report the dimensions of the TF-IDF game matrix and the MiniLM game embedding matrix.

Task 2.2 Cluster games with default pipelines

Run the same module table (one default hyperparameter each), but now clustering is performed on **game vectors** rather than independent review vectors:

- Dimensionality reduction: None, SVD(50), UMAP(50), Autoencoder(50).
- Clustering: K-Means ($k = 5$), Agglomerative (n_clusters $= 5$), HDBSCAN.

Note on UMAP: UMAP might not work well on high-dimensional inputs. Thus, consider first applying SVD to reduce dimension to 200, then use UMAP on it.

QUESTION 7: For each pipeline, report a summary table that includes:

- number of clusters found (for HDBSCAN, also report the fraction of games labeled as noise −1),
- cluster sizes,
- for each cluster: top 3 most common genres (by frequency across games in that cluster).

Task 2.3 Multi-genre interpretation

Because games may have multiple genres, do **not** treat this as a single-label problem. Instead, you will evaluate clusters using **genre-overlap analysis**:

- For each cluster, compute the distribution over genre labels (multi-label frequency).
- Define **cluster genre purity** as: the fraction of games in the cluster that contain the cluster's most common genre labels.
- Define **cluster genre entropy** over the genre labels distribution (optional).

QUESTION 8: Pick one best pipeline (justify your choice), then report two cluster with high purity:

- top 3 genres with percentages,
- cluster genre purity (as defined above),
- Representative games in the cluster (game name + genres).

Provide a short interpretation: what type of games does these clusters represent?

Task 3 - Held-Out Game Profiling and Theme Discovery (Clustering + LLM)

Finally, you will produce a **product report** for a held-out game. Your goal is to quickly answer:

- What genre does it most resemble (estimated)?
- What are the main complaint themes and praise themes?

Task 3.1 Genre estimation via nearest game cluster

Use your best Task 2 pipeline to estimate the held-out game's genre profile:

1. Compute the held-out game's **game vector** using its **positive** reviews:
   - TF-IDF: concatenate positive reviews, transform using your fitted TF-IDF vectorizer.
   - MiniLM: average MiniLM embeddings of positive reviews.
2. Assign the held-out game to the **closest cluster** (e.g., nearest centroid for K-Means, nearest cluster medoid, or nearest cluster by average cosine distance).
3. Report the cluster's top genres as the held-out game's estimated genre distribution.

QUESTION 9: Report: (i) the assigned cluster ID, (ii) the top 3 genres of that cluster, and (iii) 3 representative games from that cluster. Briefly justify why this constitutes a genre estimate in a multi-genre world.

Task 3.2 Theme clustering (positive and negative)

Now perform theme discovery separately on:

- negative reviews (complaints),
- positive reviews (praises).

For each subset, run the module table again with default hyperparameters, but now use:

- K-Means($k = 5$) and Agglomerative (n_clusters $= 5$) for a fixed number of themes,
- HDBSCAN for a variable number of themes (explain handling of noise points).

For each discovered cluster, provide:

- top TF-IDF terms (cluster-level),
- 1–2 exemplar reviews (closest to cluster centroid / medoid),
- a short cluster label (3–6 words).

QUESTION 10: For negative reviews: report 3–5 clusters with (i) top terms and (ii) exemplar reviews, and assign a short label to each complaint cluster.

QUESTION 11: For positive reviews: repeat the same analysis and label 3–5 praise clusters.

LLM labeling

You are now using an LLM to help assign cluster labels (For each cluster, you feed the LLM with examples and top terms, and let LLM think what is cluster of praise / complaints is about). You need to include:

- your prompt template,
- what evidence you provided (top terms + exemplars),

QUESTION 12: Include your prompting strategy and 3 example of LLM-generated labels for clusters. You can use the same helper code from Project 1 for Qwen model (*Qwen/Qwen3-4B-Instruct-2507*.

Part 2 - Deep Learning and Clustering of Image Data

Note: If you are using Google Colab, make sure to use GPU runtimes.

In this part, we aim to cluster the images of the `tf_flowers` dataset. This dataset consists of images of five types of flowers. Explore this link to see actual samples of the data.

Extracting meaningful features from images has a long history in computer vision. Instead of considering the raw pixel values as features, researchers have explored various hand-engineered

feature extraction methods, e.g. [5]. With the recent rise of "deep learning", these methods are replaced with using appropriate neural networks. Particularly, one can adopt a neural network already trained to classify another large dataset of images[2]. These pre-trained networks have been trained to morph the highly non-smooth scatter of images in the higher dimension, into smooth lower-dimensional manifolds.

In this project, we use a VGG network [6] pre-trained on the ImageNet dataset [7]. We provide **a helper codebase** (check Week 4 in BruinLearn), which guides you through the necessary steps for loading the VGG network and for using it for feature extraction.

QUESTION 13: In a brief paragraph discuss: If the VGG network is trained on a dataset with perhaps totally different classes as targets, why would one expect the features derived from such a network to have discriminative power for a custom dataset?

Use the helper code to load the flowers dataset and extract their features. To perform computations on deep neural networks fast enough, GPU resources are often required. GPU resources can be freely accessed through "Google Colab".

QUESTION 14: In a brief paragraph explain how the helper code base is performing feature extraction.

QUESTION 15: How many pixels are there in the original images? How many features does the VGG network extract per image; i.e what is the dimension of each feature vector for an image sample?

QUESTION 16: Are the extracted features dense or sparse? (Compare with sparse TF-IDF features in text.)

QUESTION 17: In order to inspect the high-dimensional features, t-SNE is a popular off-the-shelf choice for visualizing Vision features. Map the features you have extracted onto 2 dimensions with t-SNE. Then plot the mapped feature vectors along x and y axes. Color-code the data points with ground-truth labels. Describe your observation.

While PCA is a powerful method for dimensionality reduction, it is limited to "linear" transformations. This might not be particularly good if a dataset is distributed non-linearly. An alternative approach is use of an "autoencoder" or UMAP. The helper has implemented an autoencoder which is ready to use.

QUESTION 18: Report the best result (in terms of adjusted rand imdex) within the table below. For HDBSCAN, introduce a conservative parameter grid over `min_cluster_size` and `min_samples`.

Lastly, we can conduct an experiment to ensure that VGG features are rich enough in information about the data classes. In particular, we can train a fully-connected neural network classifier to predict the labels of data. For this task, you may use the `MLP`[3] module provided in the helper code base.

QUESTION 19: Report the test accuracy of the MLP classifier on the original VGG features. Report the same when using the reduced-dimension features (you have freedom in choosing the dimensionality reduction algorithm and its parameters). Does the performance of the model suffer with the reduced-dimension representations? Is it significant? Does the success in classification make sense in the context of the clustering results obtained for the same features in Question 18.

[2]Such an approach, in which the knowledge gained to solve one problem, is applied to a different but related problem, is often referred to as "transfer learning". We visited another instance of transfer learning when we used GLoVe vectors for text classification in project 1.

[3]Multi-Layer Perceptron

Module	Alternatives	Hyperparameters
Dimensionality Reduction	None	N/A
	SVD	r = 50
	UMAP	n_components = 50
	Autoencoder	num_features = 50
Clustering	K-Means	k = 5
	Agglomerative Clustering	n_clusters = 5
	HDBSCAN	min_cluster_size & min_samples

Part 3 - Clustering using both image and text

In part 1 and part 2, we have practived the art of clustering text and images separately. However, can we map image and text to the same space? In the Pokemon world, Pokedex catalogs Pokemon's appearances and various metadata. We will build our Pokedex from image dataset link and meta metadata link. Fortunately, ECE 219 Gym kindly provides new Pokemon trainers with the helper code for data preprocessing and inferencing. Please find the code on Bruinlearn modules Week 4.

Each Pokémon may be represented by multiple images and up to two types (for example, Bulbasaur is categorized as both Grass and Poison types). In this section, we will focus on the first image (named 0.jpg) in each folder for our analysis.

We will use the pre-trained CLIP [**?**] to illustrate the idea of multimodal clustering. CLIP (Contrastive Language–Image Pretraining) is an innovative model developed by OpenAI, designed to understand and connect concepts from both text and images. CLIP is trained on a vast array of internet-sourced text-image pairs. This extensive training enables the model to understand a broad spectrum of visual concepts and their textual descriptions.

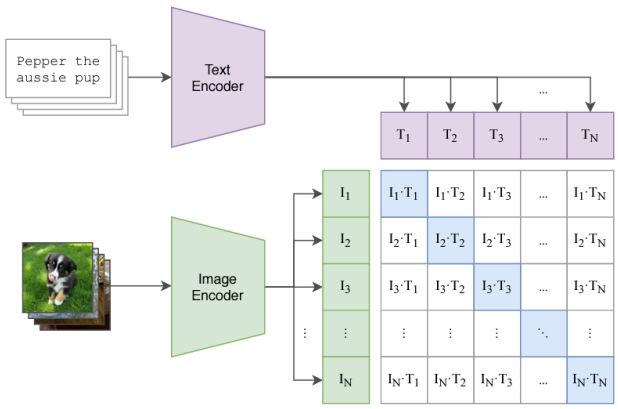

Figure 1: CLIP training summary

CLIP consists of two primary components: a text encoder and an image encoder. The text encoder processes textual data, converting sentences and phrases into numerical representations. Simultaneously, the image encoder transforms visual inputs into a corresponding set of numerical values. These encoders are trained to map both text and images into a shared embedding space, allowing the model to compare and relate the two different types of data directly. The training employs a contrastive learning approach, where the model learns to match corresponding text and image pairs against numerous non-matching pairs. This approach helps the model in accurately associating images with their relevant textual descriptions and vice versa.

QUESTION 20: Try to construct various text queries regarding types of Pokemon (such as "type: Bug", "electric type Pokémon" or "Pokémon with fire abilities") to find the relevant images from the dataset. Once you have found the most suitable template for queries, please find the top five most relevant Pokemon for type Bug, Fire and Grass. For each of the constructed query, please plot the five most relevant Pokemon horizontally in one figure with following specifications:

- the title of the figure should be the query you used;
- the title of each Pokemon should be the name of the Pokemon and its first and second type.

Repeat this process for Pokemon of Dark and Dragon types. Assess the effectiveness of your queries in these cases as well and try to explain any differences.

QUESTION 21: Randomly select 10 Pokemon images from the dataset and use CLIP to find the most relevant types (use your preferred template, e.g "type: Bug"). For each selected Pokemon, please plot it and indicate:

- its name and first and second type;
- the five most relevant types predicted by CLIP and their predicted similarities.

QUESTION 22: In this question, reuse the exact same CLIP setup from the previous question on all the Pokemon images (same model, same type prompt template, and the same set of candidate types), but instead of returning only the most relevant type, return the **top-5** most relevant types for each Pokemon image. Using the ground-truth primary type label `Type1`, report:

- **Accuracy@1 (Acc@1):** the fraction of images whose top-1 predicted type matches the ground-truth primary type (`Type1`).
- **Hit@5 (a.k.a. Recall@5):** the fraction of images whose ground-truth primary type appears anywhere in the top-5 predicted types.

In practice, you will likely observe that CLIP's **Acc@1** for predicting the primary type (`Type1`) is relatively low, while **Hit@5** is often reasonably good. This gap suggests that CLIP frequently retrieves the correct type *somewhere* in a short candidate list, but does not reliably rank it as the top-1 prediction. A key reason is that CLIP is a *dual-encoder* model trained with a contrastive objective: it independently embeds images and text prompts into a shared vector space, and predictions are made purely by *embedding similarity* (e.g., dot-product/cosine similarity). This is powerful for coarse semantic alignment, but it lacks an explicit reasoning step and can be sensitive to prompt wording and visually similar concepts.

Modern vision-language models (VLMs) address this limitation by *tokenizing* the image: a vision encoder converts the image into a sequence of visual tokens, which are then fed alongside text tokens into a large language model [8]. The LLM attends jointly over image and text tokens and generates outputs autoregressively, enabling it to use additional context (such as a candidate list of types) and make a more discrete, instruction-following decision rather than relying solely on nearest-neighbor similarity. Figure 2 illustrates this architecture (we will use a lightweight VLM in the next question to rerank CLIP's top-5 candidate types and re-evaluate Acc@1).

QUESTION 23: VLM Reranking of CLIP Top-5 Candidates. Reuse the exact same CLIP setup and evaluation protocol from the previous question. For each Pokemon image, first use CLIP to obtain the **top-5** predicted types (and their probabilities/similarities). Then, use a modern

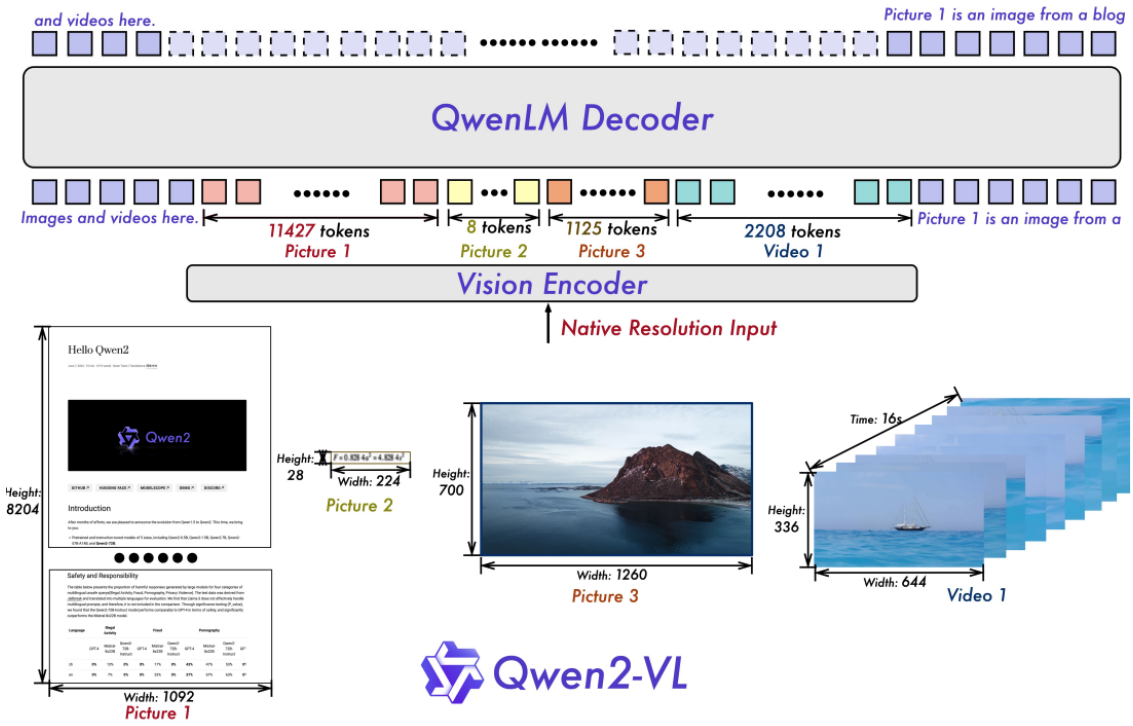

Figure 2: Qwen2-VL Architecture Demo

vision-language model (VLM), e.g., `Qwen/Qwen3-VL-2B-Instruct`, to select the single most likely *primary* type from *only* these five candidates. You can check the helper code for how to use Qwen vision language models (please use Google-colab or if you have your local pc with GPU)

Concretely, for each image i, let CLIP return a candidate set C_i containing the top-5 types. Prompt the VLM with the image and the candidate list C_i, and force the VLM to output exactly one type from C_i (e.g., as a JSON field `{"type1": "..."}`). If the VLM output is invalid (not in C_i), fall back to CLIP's top-1 type.

Report:

- **Reranked Accuracy@1:** the fraction of images whose final predicted type (after VLM selection) matches the ground-truth `Type1`.
- A comparison table of **CLIP Acc@1**, **CLIP Hit@5**, and **VLM-reranked Acc@1**.
- Briefly discuss: Does VLM reranking help?

FAQ

- **Help! My UMAP function works with the Euclidean distance but not cosine?**: Please follow the following library versions to get UMAP to work.

```
annoy==1.17.0
cython==0.29.21
fuzzywuzzy==0.18.0
hdbscan==0.8.26
joblib==1.0.0
kiwisolver==1.3.1
llvmlite==0.35.0
```

```
matplotlib==3.3.2
numba==0.52.0
numpy==1.20.0
pandas==1.1.2
pillow==8.1.0
pyarrow==1.0.1
python-levenshtein==0.12.1
pytz==2021.1
scikit-learn==0.24.1
scipy==1.6.0
six==1.15.0
threadpoolctl==2.1.0
tqdm==4.50.0
umap-learn==0.5.0
```

Submission

Please submit a PDF report to Gradescope via BruinLearn. You can find a link to Gradescope in the left panel on BruinLearn.

In addition, please submit a zip file containing your **report**, and your **codes** with a **readme file** on how to run your code to BruinLearn. The zip file should be named as "Project2_UID1_UID2_..._UIDn.zip" where UIDx's are student ID numbers of the team members. Only one submission per team is required. If you have any questions, please ask on Piazza or through email.

References

[1] Why is Euclidean distance not a good metric in high dimensions? [online]. (https://stats.stackexchange.com/questions/99171/why-is-euclidean-distance-not-a-good-metric-in-high-dimensions).

[2] https://en.wikipedia.org/wiki/DBSCAN

[3] https://hdbscan.readthedocs.io/en/latest/how_hdbscan_works.html

[4] Heaton, J. (2018). Ian goodfellow, yoshua bengio, and aaron courville: Deep learning.

[5] Rybski, P. E., Huber, D., Morris, D. D., & Hoffman, R. (2010, June). Visual classification of coarse vehicle orientation using histogram of oriented gradients features. In 2010 IEEE Intelligent vehicles symposium (pp. 921-928). IEEE.

[6] Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.

[7] Deng, J., Dong, W., Socher, R., Li, L. J., Li, K., & Fei-Fei, L. (2009, June). Imagenet: A large-scale hierarchical image database. In 2009 IEEE conference on computer vision and pattern recognition (pp. 248-255). Ieee.

[8] Wang, P., Bai, S., Tan, S., Wang, S., Fan, Z., Bai, J., ... Lin, J. (2024). Qwen2-VL: Enhancing Vision-Language Model's Perception of the World at Any Resolution. arXiv [Cs.CV]. http://arxiv.org/abs/2409.12191