P2

February 20, 2026

0.1 ECE 219 Project 2

0.1.1 Part 1 Task 1

[3]:
```
#read dataset main.csv
%pip install -q gdown
import gdown
import pandas as pd

file_id = "1peRSVgEbhGvLQZIsnQq93IW9_nwmqlOT"
url = f"https://drive.google.com/uc?id={file_id}"
gdown.download(url, "main.csv", quiet=False)


df = pd.read_csv("main.csv")
```

```
Downloading…
From: https://drive.google.com/uc?id=1peRSVgEbhGvLQZIsnQq93IW9_nwmqlOT
To: /content/main.csv
100%|      | 40.5M/40.5M [00:00<00:00, 162MB/s]
```

Task 1.1 Defining Pseudo Labels

[4]:
```
reviews = df["review_text"]
lengths = reviews.apply(lambda x: len(x.split()))

# Create mask to select the short and long reviews
short_mask = lengths <= lengths.quantile(0.25)
long_mask = lengths >= lengths.quantile(0.75)

df_short = df[short_mask]
df_long = df[long_mask]

df_short["length_label"] = 0   # Short
df_long["length_label"]  = 1   # Long

df_filtered = pd.concat([df_short, df_long], axis=0)
display(df_filtered)

num_retained = len(df_short) + len(df_long)
```

```
print(f"Retained Long and Short after filtering: {num_retained}; Discarded:␣
↪{len(df) - num_retained}")

avg_len_short = lengths[short_mask].mean()
avg_len_long = lengths[long_mask].mean()
overall_avg = lengths[short_mask | long_mask].mean()
print(f"Avg short length: {avg_len_short}; Avg long length: {avg_len_long}; Avg␣
↪overall length: {overall_avg}")
```

```
/tmp/ipython-input-1705718629.py:11: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_short["length_label"] = 0   # Short
/tmp/ipython-input-1705718629.py:12: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_long["length_label"]  = 1   # Long
```

```
                   user  playtime          post_date  helpfulness  \
0      Pakistan warrior      47.8   November 3, 2023         3911
1                  Zuvi    1969.8   November 2, 2022         3154
3              Lunación     482.2  November 10, 2023         2530
5                 ORION    1165.6  December 10, 2022         2245
6                 Bandy    1723.2   October 29, 2022         2001
…                  …       …                 …          …
39945          Alisdair       0.5     March 25, 2024            7
39948        Smutrifice       0.6  November 11, 2023            6
39952       Propetar100       0.3   February 4, 2024            5
39958              Zezd       2.9  February 13, 2020            5
39966                PV       0.4     March 30, 2024            4

                                          review_text  recommend  \
0                ALT + F4 best feature in the game 10/10       True
1        SPAWN DIE, SPAWN DIE, SPAWN DIE, SPAWN DIE.-Jev       True
3      Nobody will read my review, so I'll just say I…       True
5                Crash Of Duty : Update Requires Restart       True
6                                update requires restart       True
…                                                   …        …
39945  I bought this game thought it will be a fun ro…      False
39948  Once again, wishing Steam offered a neutral ra…      False
39952  Dear Reddit, today I bought a game that was on…      False
39958  The game's descriptions says that this game wi…      False
```

```
39966  1.Similar to other reviewers, I don't consider…      False

       early_access_review    appid      game_name  release_date  \
0                      NaN  1938090  Call of Duty®  Oct 27, 2022
1                      NaN  1938090  Call of Duty®  Oct 27, 2022
3                      NaN  1938090  Call of Duty®  Oct 27, 2022
5                      NaN  1938090  Call of Duty®  Oct 27, 2022
6                      NaN  1938090  Call of Duty®  Oct 27, 2022
…                    …      …            …           …
39945                  NaN   939400     LoveChoice  Nov 15, 2021
39948                  NaN   939400     LoveChoice  Nov 15, 2021
39952                  NaN   939400     LoveChoice  Nov 15, 2021
39958  Early Access Review   939400     LoveChoice  Nov 15, 2021
39966                  NaN   939400     LoveChoice  Nov 15, 2021

                                 genres  length_label
0                                Action             0
1                                Action             0
3                                Action             0
5                                Action             0
6                                Action             0
…                                 …           …
39945  Adventure, Indie, RPG, Simulation             1
39948  Adventure, Indie, RPG, Simulation             1
39952  Adventure, Indie, RPG, Simulation             1
39958  Adventure, Indie, RPG, Simulation             1
39966  Adventure, Indie, RPG, Simulation             1

[20497 rows x 12 columns]
```

```
Retained Long and Short after filtering: 20497; Discarded: 19503
Avg short length: 6.391474720443467; Avg long length: 493.00209288419376; Avg
overall length: 244.60442991657317
```

Answer Q1 - Retained(Short + Long): 20497 - Avg short length: 6.391474720443467 - Avg long length: 493.00209288419376

Task 1.2 Representations

[5]:
```python
from sklearn.feature_extraction.text import TfidfVectorizer
from sentence_transformers import SentenceTransformer

vectorizer = TfidfVectorizer(min_df=3, stop_words="english", ngram_range=(1,␣
↪1)) # only retain unigram
X_tfidf = vectorizer.fit_transform(df_filtered["review_text"])

model = SentenceTransformer("sentence-transformers/all-MiniLM-L6-v2")
X_minilm = model.encode(df_filtered["review_text"].to_list(),␣
↪show_progress_bar=True)
```

```
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:104:
UserWarning:
Error while fetching `HF_TOKEN` secret value from your vault: 'Requesting secret
HF_TOKEN timed out. Secrets can only be fetched when running from the Colab
UI.'.
You are not authenticated with the Hugging Face Hub in this notebook.
If the error persists, please let us know by opening an issue on GitHub
(https://github.com/huggingface/huggingface_hub/issues/new).
  warnings.warn(
Warning: You are sending unauthenticated requests to the HF Hub. Please set a
HF_TOKEN to enable higher rate limits and faster downloads.
WARNING:huggingface_hub.utils._http:Warning: You are sending unauthenticated
requests to the HF Hub. Please set a HF_TOKEN to enable higher rate limits and
faster downloads.
```

```
Loading weights:   0%|          | 0/103 [00:00<?, ?it/s]
```

```
BertModel LOAD REPORT from: sentence-transformers/all-MiniLM-L6-v2
Key                     | Status     |  |
------------------------+------------+--+-
embeddings.position_ids | UNEXPECTED |  |

Notes:
- UNEXPECTED    :can be ignored when loading from different task/architecture;
not ok if you expect identical arch.
```

```
Batches:   0%|          | 0/641 [00:00<?, ?it/s]
```

[6]:
```
# Report shapes:
print(f"X_tfidf shape: {X_tfidf.shape}; X_minilm shape: {X_minilm.shape}")
```

```
X_tfidf shape: (20497, 25085); X_minilm shape: (20497, 384)
```

Answer Q2: - X_tfidf shape: (20497, 25085); X_minilm shape: (20497, 384) - TF-IDF matrix is sparse because this representation needs to contain all of the words from the full vocabulary even though each review only contains a small subset. This leads to a large number of zero entries. In contrast, MiniLM use deep learning and encode each review into a fixed size space based on their semantical meaning.

Task 1.3 Clustering Pipeline

[ ]:
```
import numpy as np
from sklearn.decomposition import TruncatedSVD
from sklearn.cluster import KMeans, AgglomerativeClustering
from sklearn.metrics import homogeneity_score, completeness_score,␣
↪v_measure_score, adjusted_rand_score, adjusted_mutual_info_score
from sklearn.neighbors import kneighbors_graph
import umap

y_true = df_filtered["length_label"].astype(int).to_numpy()
```

```python
# TF-IDF:
Z_tfidf_none = X_tfidf
Z_tfidf_svd = TruncatedSVD(n_components=50, random_state=0).
 ↪fit_transform(X_tfidf)
# Here we are reusing the SVD(50) instead of SVD(200) to avoid unreasonably␣
 ↪long runtime
Z_tfidf_umap = umap.UMAP(n_components=50, random_state=0).
 ↪fit_transform(Z_tfidf_svd)

# MiniLM:
Z_minilm_none = X_minilm
Z_minilm_svd = TruncatedSVD(n_components=50, random_state=0).
 ↪fit_transform(X_minilm)
Z_minilm_umap = umap.UMAP(n_components=50, random_state=0).
 ↪fit_transform(X_minilm)

assert X_tfidf.shape[0] == len(df_filtered) == len(y_true)
assert X_minilm.shape[0] == len(df_filtered)


# Helper functions for different clustering models and metrics computation:
def kmeans(Z, n_clusters=2):
    return KMeans(n_clusters=n_clusters, n_init=10, random_state=0).
 ↪fit_predict(Z)

def agglo(Z, k=10, n_clusters=2, use_connectivity=True):
    if hasattr(Z, "toarray"):
        Z = Z.toarray()  # make dense

    conn = None
    if use_connectivity:
        conn = kneighbors_graph(
            Z,
            n_neighbors=k,
            mode="connectivity",
            include_self=False
        )

    return AgglomerativeClustering(
        n_clusters=n_clusters,
        linkage="ward",
        connectivity=conn
    ).fit_predict(Z)

def compute_metrics(y_true, y_pred):
    return dict(
```

```
        homogeneity=homogeneity_score(y_true, y_pred),
        completeness=completeness_score(y_true, y_pred),
        v_measure=v_measure_score(y_true, y_pred),
        ARI=adjusted_rand_score(y_true, y_pred),
        AMI=adjusted_mutual_info_score(y_true, y_pred),
    )

# Compute and summarize:
representations = {
    "TF-IDF": {"None": Z_tfidf_none, "SVD": Z_tfidf_svd, "UMAP": Z_tfidf_umap},
    "MiniLM": {"None": Z_minilm_none, "SVD": Z_minilm_svd, "UMAP":␣
 ↪Z_minilm_umap}
}

clustering_algos = {
    "K-Means": kmeans,
    "Agglomerative": agglo
}

rows = []
for rep_name, variants in representations.items():
    for var_name, Z in variants.items():
        for algo_name, algo_func in clustering_algos.items():

            # Required skip: TF-IDF + None + Agglo
            if rep_name == "TF-IDF" and var_name == "None" and algo_name ==␣
 ↪"Agglomerative":
                rows.append({
                    "Representation": rep_name,
                    "DimRed": var_name,
                    "Clustering Algorithm": algo_name,
                    "homogeneity": np.nan,
                    "completeness": np.nan,
                    "v_measure": np.nan,
                    "ARI": np.nan,
                    "AMI": np.nan,
                    "note": "Skipped: Agglomerative (ward) requires dense input;
 ↪ raw TF-IDF is sparse & high-dim"
                })
                continue

            try:
                if algo_name == "Agglomerative":
                    y_pred = agglo(Z, k=10, n_clusters=2, use_connectivity=True)
                else:
                    y_pred = kmeans(Z, n_clusters=2)
```

```
                m = compute_metrics(y_true=y_true, y_pred=y_pred)
                rows.append({
                    "Representation": rep_name,
                    "DimRed": var_name,
                    "Clustering Algorithm": algo_name,
                    **m,
                    "note": ""
                })
            except Exception as e:
                rows.append({
                    "Representation": rep_name,
                    "DimRed": var_name,
                    "Clustering Algorithm": algo_name,
                    "homogeneity": np.nan,
                    "completeness": np.nan,
                    "v_measure": np.nan,
                    "ARI": np.nan,
                    "AMI": np.nan,
                    "note": f"Failed: {type(e).__name__}: {e}"
                })

df_q3 = pd.DataFrame(rows)
df_q3_sorted = df_q3.sort_values(["ARI", "AMI", "v_measure"], ascending=False).
 ↪reset_index(drop=True)

display(df_q3)
display(df_q3_sorted.head(5))
```

```
/usr/local/lib/python3.12/dist-packages/umap/umap_.py:1952: UserWarning: n_jobs
value 1 overridden to 1 by setting random_state. Use no seed for parallelism.
  warn(
/usr/local/lib/python3.12/dist-packages/umap/umap_.py:1952: UserWarning: n_jobs
value 1 overridden to 1 by setting random_state. Use no seed for parallelism.
  warn(
/usr/local/lib/python3.12/dist-packages/sklearn/cluster/_agglomerative.py:325:
UserWarning: the number of connected components of the connectivity matrix is 3
> 1. Completing it to avoid stopping the tree early.
  connectivity, n_connected_components = _fix_connectivity(
/usr/local/lib/python3.12/dist-packages/sklearn/cluster/_agglomerative.py:325:
UserWarning: the number of connected components of the connectivity matrix is
136 > 1. Completing it to avoid stopping the tree early.
  connectivity, n_connected_components = _fix_connectivity(
/usr/local/lib/python3.12/dist-packages/sklearn/cluster/_agglomerative.py:325:
UserWarning: the number of connected components of the connectivity matrix is 2
> 1. Completing it to avoid stopping the tree early.
  connectivity, n_connected_components = _fix_connectivity(
/usr/local/lib/python3.12/dist-packages/sklearn/cluster/_agglomerative.py:325:
UserWarning: the number of connected components of the connectivity matrix is 53
```

```
> 1. Completing it to avoid stopping the tree early.
  connectivity, n_connected_components = _fix_connectivity(
```

	Representation	DimRed	Clustering Algorithm	homogeneity	completeness	\
0	TF-IDF	None	K-Means	0.621066	0.622417	
1	TF-IDF	None	Agglomerative	NaN	NaN	
2	TF-IDF	SVD	K-Means	0.030210	0.160841	
3	TF-IDF	SVD	Agglomerative	0.030682	0.161443	
4	TF-IDF	UMAP	K-Means	0.216986	0.282128	
5	TF-IDF	UMAP	Agglomerative	0.170389	0.247026	
6	MiniLM	None	K-Means	0.508931	0.509284	
7	MiniLM	None	Agglomerative	0.531876	0.534285	
8	MiniLM	SVD	K-Means	0.508523	0.508884	
9	MiniLM	SVD	Agglomerative	0.707697	0.712751	
10	MiniLM	UMAP	K-Means	0.034294	0.159779	
11	MiniLM	UMAP	Agglomerative	0.034294	0.159779	

	v_measure	ARI	AMI	\
0	0.621741	0.725521	0.621727	
1	NaN	NaN	NaN	
2	0.050866	0.005739	0.050809	
3	0.051565	0.005878	0.051509	
4	0.245306	0.162376	0.245276	
5	0.201672	0.106571	0.201639	
6	0.509108	0.612753	0.509090	
7	0.533078	0.636972	0.533061	
8	0.508703	0.612295	0.508686	
9	0.710215	0.789024	0.710205	
10	0.056468	0.007423	0.056413	
11	0.056468	0.007423	0.056413	

	note
0	
1	Skipped: Agglomerative (ward) requires dense i…
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	

	Representation	DimRed	Clustering Algorithm	homogeneity	completeness	\
0	MiniLM	SVD	Agglomerative	0.707697	0.712751	
1	TF-IDF	None	K-Means	0.621066	0.622417	
2	MiniLM	None	Agglomerative	0.531876	0.534285	

```
3         MiniLM   None              K-Means     0.508931      0.509284
4         MiniLM    SVD              K-Means     0.508523      0.508884

   v_measure       ARI       AMI note
0   0.710215  0.789024  0.710205
1   0.621741  0.725521  0.621727
2   0.533078  0.636972  0.533061
3   0.509108  0.612753  0.509090
4   0.508703  0.612295  0.508686
```

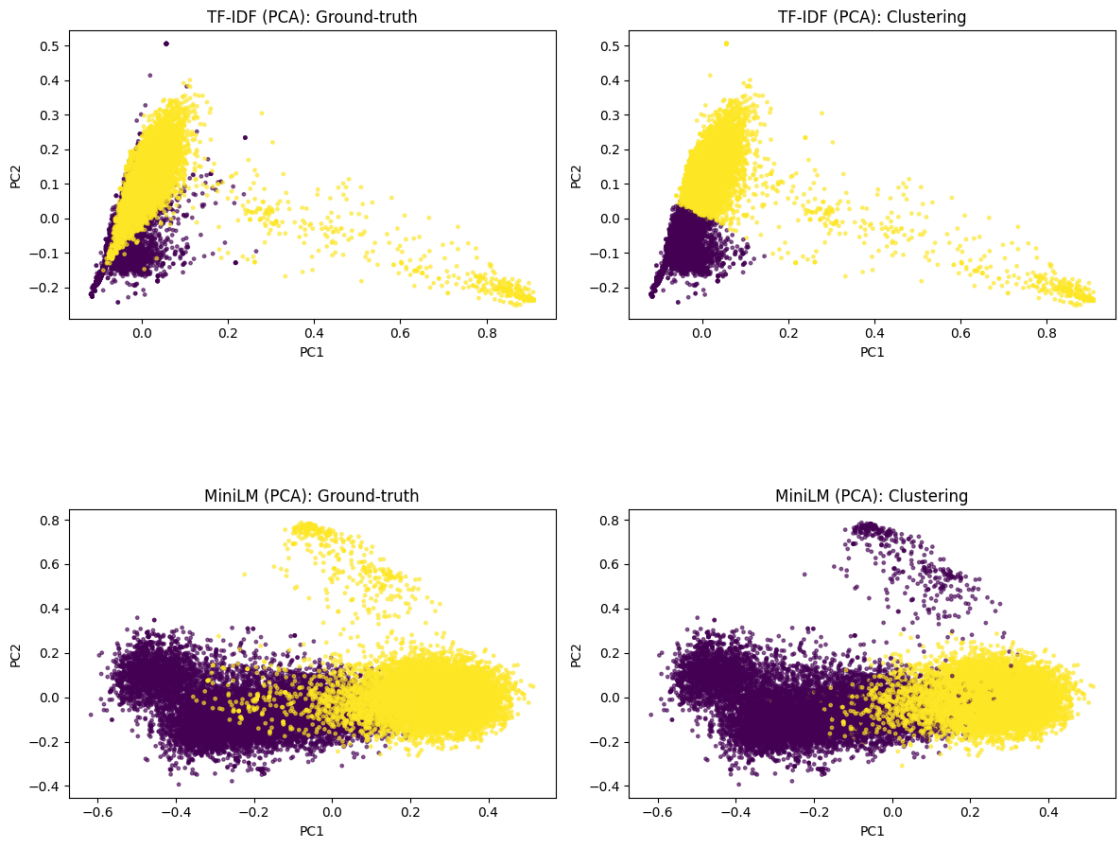
Answer Q3: - Best performing pipeline: **MiniLM + SVD(50) + Agglomerative Clustering**, homogeneity: 0.707697, completeness: 0.712751, v_measure: 0.710215, ARI: 0.789024, AMI: 0.710205 (Top 5 see table) - Agglomerative clustering on raw TF-IDF representations was skipped because the Ward linkage requires dense input, and converting high-dimensional sparse TF-IDF features to dense form is not feasible due to memory constraints.

Task 1.4 Interpretation **Answer Q4**: - According to the best 5 results shown in the previous part, MiniLM can seperate long vs. short reviews more clearly. This is because MiniLM can produce dense and low-dimensional semantic embeddings, which make it easier for clustering algorithms to capture the structural patterns related to review length (long reviews tend to produce embeddings with great textual variabilities/complexity and distinguishable patterns). However, TF-IDF features are sparse and high-dimensional, and length differences are mainly reflected in vector magnitude. After SVD, this kind of variation may not be translated into geometric seperation.

[8]:
```
# Q5 visualization
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

Z_tfidf_best = Z_tfidf_none
Z_minilm_best = Z_minilm_svd

# Clustering
y_pred_tfidf = kmeans(Z_tfidf_best, n_clusters=2)
y_pred_minilm = agglo(Z_minilm_best, k=10, n_clusters=2)

# PCA for visualization
Z_tfidf_2d = PCA(n_components=2, random_state=0).fit_transform(
    Z_tfidf_best.toarray() if hasattr(Z_tfidf_best, "toarray") else␣
 ↪Z_tfidf_best)

Z_minilm_2d = PCA(n_components=2, random_state=0).fit_transform(Z_minilm_best)

# Plot
plt.figure(figsize=(12, 4))

# ground truth
plt.subplot(1, 2, 1)
```

```
plt.scatter(Z_tfidf_2d[:, 0], Z_tfidf_2d[:, 1],
            c=y_true, s=6, alpha=0.6)
plt.title("TF-IDF (PCA): Ground-truth")
plt.xlabel("PC1")
plt.ylabel("PC2")

# clustering result
plt.subplot(1, 2, 2)
plt.scatter(Z_tfidf_2d[:, 0], Z_tfidf_2d[:, 1],
            c=y_pred_tfidf, s=6, alpha=0.6)
plt.title("TF-IDF (PCA): Clustering")
plt.xlabel("PC1")
plt.ylabel("PC2")

plt.tight_layout()
plt.show()


plt.figure(figsize=(12, 4))

# ground truth
plt.subplot(1, 2, 1)
plt.scatter(Z_minilm_2d[:, 0], Z_minilm_2d[:, 1],
            c=y_true, s=6, alpha=0.6)
plt.title("MiniLM (PCA): Ground-truth")
plt.xlabel("PC1")
plt.ylabel("PC2")

# clustering result
plt.subplot(1, 2, 2)
plt.scatter(Z_minilm_2d[:, 0], Z_minilm_2d[:, 1], #0: PC1 as x axis, 1: PC2 as␣
 ↪y axis
            c=y_pred_minilm, s=6, alpha=0.6)
plt.title("MiniLM (PCA): Clustering")
plt.xlabel("PC1")
plt.ylabel("PC2")

plt.tight_layout()
plt.show()
```

```
/usr/local/lib/python3.12/dist-packages/sklearn/cluster/_agglomerative.py:325:
UserWarning: the number of connected components of the connectivity matrix is 2
> 1. Completing it to avoid stopping the tree early.
  connectivity, n_connected_components = _fix_connectivity(
```



Answer Q5: - From the plots, we observe that for **TF-IDF**, short and long reviews cannot be separated clearly (strongly overlapped based on the ground truth visualization), and the clustering result only capture some of the structure, leading to relatively worse clustering performance. - On the other hand, **MiniLM** provides a clearer seperation between short and long reviews in the PCA space. The clustering assignments closely match the ground truth label, which aligns with our results in Q3.

0.1.2 Part 1 Task 2

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

[9]:
```
import pandas as pd

# 1) Positive reviews only
df_pos = df[df["recommend"] == True].copy()

# 2) Build one doc per game by concatenating all positive reviews
# Use appid as the game key (unique + stable)
game_docs = (
    df_pos.groupby("appid")["review_text"]
    .apply(lambda s: " ".join(s.astype(str).tolist()))
    .reset_index(name="doc")
```

```
)

# 3) Keep metadata (game_name, genres) for later (Task 2.2 / Q7-Q8)
game_meta = (
    df_pos.groupby("appid")[["game_name", "genres"]]
    .first()
    .reset_index()
)

games = game_docs.merge(game_meta, on="appid", how="left")

print("num games:", len(games))
games.head()
```

```
num games: 200
```

[9]:

```
    appid                                                  doc  \
0   12210  Guys, please do yourself a favor and download …
1  239140  Glad to see the misinformed reviews have been …
2  251570  One of my favorite games that I have played in…
3  261550  I didn't expect to play this game in my lifeti…
4  304390  ---{Graphics}---  Masterpiece  Beautiful  Good…

                                game_name  \
0  Grand Theft Auto IV: The Complete Edition
1                              Dying Light
2                            7 Days to Die
3              Mount & Blade II: Bannerlord
4                                FOR HONOR

                                           genres
0                                Action, Adventure
1                                      Action, RPG
2  Action, Adventure, Indie, RPG, Simulation, Str…
3          Action, Indie, RPG, Simulation, Strategy
4                                           Action
```

[10]:

```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_vec = TfidfVectorizer(min_df=3, stop_words="english", ngram_range=(1,1))
X_game_tfidf = tfidf_vec.fit_transform(games["doc"].values)

print("TF-IDF game matrix shape:", X_game_tfidf.shape)
```

```
TF-IDF game matrix shape: (200, 16554)
```

[11]:

```
!pip -q install sentence-transformers
```

```python
import numpy as np
from collections import defaultdict
from sentence_transformers import SentenceTransformer

model = SentenceTransformer("sentence-transformers/all-MiniLM-L6-v2")

# Encode each positive review
texts = df_pos["review_text"].astype(str).values
appids = df_pos["appid"].values

E = model.encode(texts, batch_size=64, show_progress_bar=True,␣
↪normalize_embeddings=True)

# Average embeddings per appid
sum_vec = defaultdict(lambda: np.zeros(E.shape[1], dtype=np.float32))
cnt_vec = defaultdict(int)

for a, e in zip(appids, E):
    sum_vec[a] += e.astype(np.float32)
    cnt_vec[a] += 1

# Align to the same order as `games`
X_game_minilm = np.vstack([sum_vec[a] / cnt_vec[a] for a in games["appid"].
↪values])

print("MiniLM game embedding matrix shape:", X_game_minilm.shape)
```

```
Loading weights:   0%|          | 0/103 [00:00<?, ?it/s]

BertModel LOAD REPORT from: sentence-transformers/all-MiniLM-L6-v2
Key                      | Status     |  |
-------------------------+------------+--+-
embeddings.position_ids | UNEXPECTED |  |

Notes:
- UNEXPECTED    :can be ignored when loading from different task/architecture;
not ok if you expect identical arch.

Batches:   0%|          | 0/313 [00:00<?, ?it/s]

MiniLM game embedding matrix shape: (200, 384)
```

Answer Q6: Using only positive reviews, we constructed one vector per game.

- The TF-IDF game matrix has shape (200, 16,554), corresponding to 200 games and a vocabulary of 16,554 terms.
- The MiniLM game embedding matrix has shape (200, 384), corresponding to 200 games and a 384-dimensional embedding per game.

Task 2.2 Cluster games with default pipelines

[12]:
```python
from collections import Counter
import numpy as np

# Convert "Action, RPG" → ["Action", "RPG"]
def parse_genres(genres_str):
    if pd.isna(genres_str):
        return []
    return [g.strip() for g in str(genres_str).split(",") if g.strip()]

games["genre_list"] = games["genres"].apply(parse_genres)

def top_genres_for_cluster(indices, topk=3):
    counter = Counter()
    for i in indices:
        counter.update(games.loc[i, "genre_list"])
    total = len(indices)
    top = counter.most_common(topk)
    return [(g, f, f/total) for g, f in top]

def summarize_clusters(labels, pipeline_name):
    labels = np.array(labels)
    unique = sorted(set(labels))

    # handle noise
    clusters = [c for c in unique if c != -1]
    noise_frac = np.mean(labels == -1) if -1 in unique else 0.0

    rows = []
    for c in clusters:
        idxs = np.where(labels == c)[0]
        size = len(idxs)
        top3 = top_genres_for_cluster(idxs)

        top3_str = ", ".join([f"{g} ({p:.0%})" for g, f, p in top3])

        rows.append({
            "pipeline": pipeline_name,
            "cluster": c,
            "size": size,
            "top_genres": top3_str
        })

    return {
        "pipeline": pipeline_name,
        "num_clusters": len(clusters),
        "noise_fraction": noise_frac,
        "cluster_sizes": sorted(
```

```
            [np.sum(labels == c) for c in clusters],
            reverse=True
        ),
        "rows": rows
    }
```

[13]:
```
from sklearn.cluster import KMeans, AgglomerativeClustering

def run_kmeans(X):
    return KMeans(n_clusters=5, random_state=42, n_init="auto").fit_predict(X)

def run_agglo(X):
    if hasattr(X, "toarray"):
        X = X.toarray()
    return AgglomerativeClustering(n_clusters=5).fit_predict(X)

def run_hdbscan(X):
    !pip -q install hdbscan
    import hdbscan
    if hasattr(X, "toarray"):
        X = X.toarray()
    return hdbscan.HDBSCAN(min_cluster_size=5).fit_predict(X)
```

[14]:
```
from sklearn.decomposition import TruncatedSVD

def dr_none(X):
    return X

def dr_svd50(X):
    return TruncatedSVD(n_components=50, random_state=42).fit_transform(X)
```

[15]:
```
pipelines = []
summary_rows = []

representations = {
    "tfidf": X_game_tfidf,
    "minilm": X_game_minilm
}

reducers = {
    "none": dr_none,
    "svd50": dr_svd50
}

clusterers = {
    "kmeans5": run_kmeans,
    "agglo5": run_agglo,
```

```
    "hdbscan": run_hdbscan
}

for r_name, X in representations.items():
    for d_name, dr in reducers.items():
        Z = dr(X)

        for c_name, cl in clusterers.items():
            labels = cl(Z)
            pname = f"{r_name}_{d_name}_{c_name}"

            summary = summarize_clusters(labels, pname)
            pipelines.append((pname, labels, summary))

            summary_rows.append({
                "pipeline": pname,
                "num_clusters": summary["num_clusters"],
                "noise_fraction": summary["noise_fraction"],
                "cluster_sizes": summary["cluster_sizes"]
            })

q7_overview = pd.DataFrame(summary_rows)
q7_overview.sort_values("pipeline")
```

[15]:

	pipeline	num_clusters	noise_fraction	cluster_sizes
7	minilm_none_agglo5	5	0.000	[70, 44, 41, 34, 11]
8	minilm_none_hdbscan	2	0.905	[13, 6]
6	minilm_none_kmeans5	5	0.000	[84, 57, 40, 11, 8]
10	minilm_svd50_agglo5	5	0.000	[86, 62, 33, 13, 6]
11	minilm_svd50_hdbscan	2	0.875	[20, 5]
9	minilm_svd50_kmeans5	5	0.000	[79, 62, 28, 24, 7]
1	tfidf_none_agglo5	5	0.000	[113, 49, 29, 5, 4]
2	tfidf_none_hdbscan	0	1.000	[]
0	tfidf_none_kmeans5	5	0.000	[77, 62, 51, 6, 4]
4	tfidf_svd50_agglo5	5	0.000	[141, 38, 12, 5, 4]
5	tfidf_svd50_hdbscan	0	1.000	[]
3	tfidf_svd50_kmeans5	5	0.000	[94, 62, 35, 5, 4]

[16]:

```
def show_pipeline(pipeline_name):
    for name, labels, summary in pipelines:
        if name == pipeline_name:
            return pd.DataFrame(summary["rows"]).sort_values("size",␣
 ↪ascending=False)

show_pipeline("minilm_svd50_kmeans5")
```

[16]:
```
                pipeline  cluster  size  \
3  minilm_svd50_kmeans5        3    79
0  minilm_svd50_kmeans5        0    62
4  minilm_svd50_kmeans5        4    28
2  minilm_svd50_kmeans5        2    24
1  minilm_svd50_kmeans5        1     7

                                      top_genres
3      Action (71%), Indie (59%), Adventure (49%)
0        Action (77%), Adventure (47%), RPG (31%)
4        Action (79%), Adventure (50%), RPG (46%)
2  Adventure (75%), Simulation (54%), Indie (50%)
1    Simulation (86%), Sports (86%), Action (14%)
```

Answer Q7:

We evaluated clustering pipelines using TF-IDF and MiniLM game representations, with dimensionality reduction (None, SVD-50) and clustering methods (K-Means, Agglomerative, HDBSCAN).

Across all pipelines:

- K-Means and Agglomerative consistently produced 5 clusters with no noise.
- HDBSCAN performed poorly:
  - For TF-IDF: all games were labeled as noise (noise fraction = 1.0).
  - For MiniLM: only 2 clusters were found, with very high noise fractions (0.875–0.905).

The MiniLM + SVD(50) + K-Means pipeline produced balanced clusters and interpretable genre structure:

- Cluster sizes: [79, 62, 28, 24, 7]
- Top genres per cluster:
  - C3: Action (71%), Indie (59%), Adventure (49%)
  - C0: Action (77%), Adventure (47%), RPG (31%)
  - C4: Action (79%), Adventure (50%), RPG (46%)
  - C2: Adventure (75%), Simulation (54%), Indie (50%)
  - C1: Simulation (86%), Sports (86%), Action (14%)

This pipeline yields coherent genre groupings with no noise, making it the most interpretable.

Task 2.3 Multi-genre interpretation

[17]:
```python
import numpy as np

# compute purity for a given cluster
def cluster_purity(labels, cluster_id):
    idxs = np.where(labels == cluster_id)[0]
    total = len(idxs)

    genre_counter = Counter()
    for i in idxs:
        genre_counter.update(games.loc[i, "genre_list"])
```

```
    if total == 0:
        return 0.0, []

    top_genre, top_count = genre_counter.most_common(1)[0]
    purity = top_count / total

    top3 = genre_counter.most_common(3)
    top3 = [(g, f/total) for g, f in top3]

    return purity, top3

# find the chosen pipeline
for name, labels, summary in pipelines:
    if name == "minilm_svd50_kmeans5":
        best_labels = labels
        break

# compute purity for each cluster
clusters = sorted(set(best_labels))
purity_rows = []

for c in clusters:
    purity, top3 = cluster_purity(best_labels, c)
    purity_rows.append({
        "cluster": c,
        "purity": purity
    })

purity_df = pd.DataFrame(purity_rows).sort_values("purity", ascending=False)
purity_df
```

[17]:

	cluster	purity
1	1	0.857143
4	4	0.785714
0	0	0.774194
2	2	0.750000
3	3	0.708861

[18]:

```
def show_cluster_examples(cluster_id, n=5):
    idxs = np.where(best_labels == cluster_id)[0]
    return games.loc[idxs, ["game_name", "genres"]].head(n)

show_cluster_examples(1)
show_cluster_examples(4)
```

[18]:
```
                                game_name  \
8          Mafia III: Definitive Edition
14               Wolcen: Lords of Mayhem
19  State of Decay 2: Juggernaut Edition
24             KARDS - The WW2 Card Game
31                       Quake Champions

                                   genres
8                       Action, Adventure
14          Action, Adventure, Indie, RPG
19                Action, RPG, Simulation
24  Casual, Free To Play, Indie, Strategy
31                   Action, Free To Play
```

Answer Q8:

We select MiniLM + SVD(50) + K-Means (k=5) as the best pipeline because it produces balanced clusters, no noise, and clear, interpretable genre groupings.

Cluster 1 - Top genres: Simulation (86%), Sports (86%), Action (14%) - Genre purity: 0.86 - Representative games: - Mafia III: Definitive Edition — Action, Adventure - Wolcen: Lords of Mayhem — Action, Adventure, Indie, RPG - State of Decay 2: Juggernaut Edition — Action, RPG, Simulation - KARDS – The WW2 Card Game — Casual, Free To Play, Indie, Strategy - Quake Champions — Action, Free To Play

Interpretation: This cluster mainly represents simulation- and sports-oriented games, often with action elements.

Cluster 4 - Top genres: Action (79%), Adventure (50%), RPG (46%) - Genre purity: 0.79

Interpretation: This cluster represents action-adventure games, often with RPG mechanics and story-driven gameplay.

0.1.3 Part 1 Task 3

Task 3.1 Genre estimation via nearest game cluster

[19]:
```
# Identify the held-out game
heldout_game = df["game_name"].value_counts().index[-1]
print("Held-out game:", heldout_game)

# Get its positive reviews
heldout_pos = df[
    (df["game_name"] == heldout_game) &
    (df["recommend"] == True)
]["review_text"].astype(str).values

# Compute MiniLM embedding (average)
heldout_embeds = model.encode(
    heldout_pos,
    batch_size=32,
```

```
    show_progress_bar=True,
    normalize_embeddings=True
)

heldout_vec = heldout_embeds.mean(axis=0).reshape(1, -1)

print("Held-out vector shape:", heldout_vec.shape)
```

```
Held-out game: LoveChoice

Batches:   0%|          | 0/4 [00:00<?, ?it/s]

Held-out vector shape: (1, 384)
```

[20]:
```
# Refit SVD on the MiniLM game matrix (same as pipeline)
from sklearn.decomposition import TruncatedSVD

svd = TruncatedSVD(n_components=50, random_state=42)
Z_games = svd.fit_transform(X_game_minilm)

# Transform held-out vector
Z_heldout = svd.transform(heldout_vec)
```

[21]:
```
from sklearn.cluster import KMeans
from sklearn.metrics.pairwise import cosine_distances

kmeans = KMeans(n_clusters=5, random_state=42, n_init="auto")
game_labels = kmeans.fit_predict(Z_games)

centroids = kmeans.cluster_centers_

# find closest centroid
dists = cosine_distances(Z_heldout, centroids)[0]
assigned_cluster = dists.argmin()

print("Assigned cluster:", assigned_cluster)
```

```
Assigned cluster: 3
```

[22]:
```
def get_cluster_info(cluster_id, n=3):
    idxs = np.where(game_labels == cluster_id)[0]

    # top genres
    genre_counter = Counter()
    for i in idxs:
        genre_counter.update(games.loc[i, "genre_list"])

    total = len(idxs)
    top3 = genre_counter.most_common(3)
```

```
    top3 = [(g, f/total) for g, f in top3]

    # representative games
    reps = games.loc[idxs, ["game_name", "genres"]].head(n)

    return top3, reps

top3_genres, reps = get_cluster_info(assigned_cluster)

print("Top genres:", top3_genres)
reps
```

```
Top genres: [('Action', 0.7088607594936709), ('Indie', 0.5949367088607594),
('Adventure', 0.4936708860759494)]
```

[22]:

```
                        game_name  \
3   Mount & Blade II: Bannerlord
4                     FOR HONOR
11                         Squad

                                          genres
3         Action, Indie, RPG, Simulation, Strategy
4                                          Action
11  Action, Indie, Massively Multiplayer, Strategy
```

Answer Q9:

Using the MiniLM + SVD(50) + K-Means pipeline, the held-out game LoveChoice is assigned to cluster 3 based on nearest-centroid similarity.

- Assigned cluster: 3
- Top 3 genres of the cluster:
  - Action (71%)
  - Indie (59%)
  - Adventure (49%)
- Representative games:
  - Mount & Blade II: Bannerlord — Action, Indie, RPG, Simulation, Strategy
  - FOR HONOR™ — Action
  - Squad — Action, Indie, Massively Multiplayer, Strategy

Justification: The held-out game is assigned to the nearest cluster in the learned embedding space, and the cluster's dominant genres reflect the shared themes across similar games. In a multi-genre setting, this cluster-level genre distribution provides a more realistic estimate than forcing a single-label prediction, capturing the mixture of action, indie, and adventure elements common to these titles.

Task 3.2 Theme clustering (positive and negative)

[23]:

```
df_neg = df[df["recommend"] == False].copy()
df_pos = df[df["recommend"] == True].copy()
```

```
print("Negative reviews:", len(df_neg))
print("Positive reviews:", len(df_pos))
```

```
Negative reviews: 20000
Positive reviews: 20000
```

[24]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import KMeans
import numpy as np

def cluster_reviews(df_subset, k=5):
    texts = df_subset["review_text"].astype(str).values

    tfidf = TfidfVectorizer(min_df=5, stop_words="english")
    X = tfidf.fit_transform(texts)

    kmeans = KMeans(n_clusters=k, random_state=42, n_init="auto")
    labels = kmeans.fit_predict(X)

    return tfidf, X, labels, kmeans
```

[25]:
```
def summarize_review_clusters(tfidf, X, labels, kmeans, df_subset, topk=10):
    terms = np.array(tfidf.get_feature_names_out())
    results = []

    for c in sorted(set(labels)):
        idxs = np.where(labels == c)[0]

        # centroid
        centroid = kmeans.cluster_centers_[c]
        top_terms = terms[np.argsort(centroid)[-topk:][::-1]]

        # exemplar review (closest to centroid)
        from sklearn.metrics.pairwise import cosine_distances
        dists = cosine_distances(X[idxs], centroid.reshape(1, -1)).flatten()
        best_idx = idxs[dists.argmin()]

        exemplar = df_subset.iloc[best_idx]["review_text"]

        results.append({
            "cluster": c,
            "top_terms": top_terms[:5],
            "exemplar": exemplar
        })

    return results
```

[26]:
```
tfidf_neg, X_neg, labels_neg, kmeans_neg = cluster_reviews(df_neg, k=5)
neg_clusters = summarize_review_clusters(
    tfidf_neg, X_neg, labels_neg, kmeans_neg, df_neg
)

for c in neg_clusters:
    print("\nCluster", c["cluster"])
    print("Top terms:", c["top_terms"])
    print("Example:", c["exemplar"][:200])
```

```
Cluster 0
Top terms: ['game' 'like' 'just' 'really' 'fun']
Example: This game is good for the first 50% or so but the game-play loop gets
old and the story isn't strong enough to make up for that. Its not a bad game
but its not great or even good it is just ok. I woul

Cluster 1
Top terms: ['game' 'ea' 'play' 'games' 'buy']
Example: Don't buy    from EA

Cluster 2
Top terms: ['game' 'just' 'like' 'time' 'really']
Example: So I won't lie… It's not a terrible game, but I am genuinely
surprised this game is getting "Very Positive" Reviews right now. Because
      there are some serious issues in the game that need t

Cluster 3
Top terms: ['game' 'play' 'just' 'buy' 'good']
Example: haram game

Cluster 4
Top terms: ['sex' 'robot' 'gay' 'kelvin' 'add']
Example: cannot sex the maiden
```

Answer Q10:

Cluster 0 — Repetitive or weak gameplay - Top terms: game, like, just, really, fun - Exemplar: "This game is good for the first 50% or so but the gameplay loop gets old and the story isn't strong enough to make up for that."

Cluster 1 — Publisher or monetization complaints - Top terms: game, ea, play, games, buy - Exemplar: "Don't buy **** from EA."

Cluster 2 — Mixed or disappointing experience - Top terms: game, just, like, time, really - Exemplar: "It's not a terrible game, but I'm surprised it's getting very positive reviews right now because there are some serious issues."

Cluster 3 — Low-effort or short complaints - Top terms: game, play, just, buy, good - Exemplar: "haram game"

Cluster 4 — Missing adult or feature content - Top terms: sex, robot, gay, kelvin, add - Exemplar: "cannot sex the maiden"

[27]:
```
tfidf_pos, X_pos, labels_pos, kmeans_pos = cluster_reviews(df_pos, k=5)
pos_clusters = summarize_review_clusters(
    tfidf_pos, X_pos, labels_pos, kmeans_pos, df_pos
)

for c in pos_clusters:
    print("\nCluster", c["cluster"])
    print("Top terms:", c["top_terms"])
    print("Example:", c["exemplar"][:200])
```

```
Cluster 0
Top terms: ['good' 'grind' 'average' 'bugs' 'bad']
Example: ---{ Graphics }--- You forget what reality is Beautiful Good
Decent Bad Don‘t look too long at it MS-DOS---{ Gameplay }--- Very good
Good It's just gameplay Mehh Watch paint dry instead J

Cluster 1
Top terms: ['10' 'game' 'like' 'better' 'just']
Example: A.B.A. IS BACK 10/10 GAME

Cluster 2
Top terms: ['sex' 'add' 'better' 'update' 'game']
Example: cant sex the bracken

Cluster 3
Top terms: ['pet' 'cat' '__' 'x ' ' ']
Example: I will leave the cat here, so that everybody who passes by can pet it
and give it a thumbs up and awards.                          |   _   _ l       ␣
↪   `  x
/          |       /                   |  |  |   |     |  |  |  | (

Cluster 4
Top terms: ['game' 'like' 'good' 'play' 'just']
Example: i would impregnate this game if i could
```

Answer Q11:

Cluster 0 — Generally enjoyable but flawed - Top terms: good, grind, average, bugs, bad - Exemplar: "Graphics are beautiful… Gameplay is very good… It's just gameplay meh."

Cluster 1 — Strong positive recommendation - Top terms: 10, game, like, better, just - Exemplar: "A.B.A. IS BACK 10/10 GAME"

Cluster 2 — Update or feature appreciation - Top terms: sex, add, better, update, game - Exemplar: "cant sex the bracken"

Cluster 3 — Community or meme-style praise - Top terms: pet, cat, x/, = - Exemplar: "I will leave the cat here so that everybody who passes by can pet it and give it a thumbs up."

Cluster 4 — Strong emotional praise - Top terms: game, like, good, play, just - Exemplar: "i would impregnate this game if i could"

[28]:
```
!pip install -q transformers accelerate

from transformers import AutoTokenizer, AutoModelForCausalLM
import torch

MODEL_NAME = "Qwen/Qwen2.5-3B-Instruct"

tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)
model = AutoModelForCausalLM.from_pretrained(
    MODEL_NAME,
    torch_dtype=torch.float16 if torch.cuda.is_available() else torch.float32,
    device_map="auto"
)

def generate(prompt, max_new_tokens=120):
    inputs = tokenizer(prompt, return_tensors="pt").to(model.device)
    prompt_len = inputs["input_ids"].shape[1]
    with torch.no_grad():
        out = model.generate(
            **inputs,
            max_new_tokens=max_new_tokens,
            do_sample=False
        )
    # only decode newly generated tokens (not the prompt)
    return tokenizer.decode(out[0][prompt_len:], skip_special_tokens=True).
 ↪strip()
```

```
`torch_dtype` is deprecated! Use `dtype` instead!

Loading weights:   0%|          | 0/434 [00:00<?, ?it/s]

WARNING:accelerate.big_modeling:Some parameters are on the meta device because
they were offloaded to the disk and cpu.
```

[ ]:
```
# print("\n--- LLM Output ---\n")
# print(generate(prompt))
```

```
The following generation flags are not valid and may be ignored: ['temperature',
'top_p', 'top_k']. Set `TRANSFORMERS_VERBOSITY=info` for more details.


--- LLM Output ---

You are helping label clusters of Steam reviews.
```

```
Task: Given (A) whether these are POSITIVE or NEGATIVE reviews, (B) top TF-IDF
terms,
and (C) 1-2 exemplar reviews, produce:
1) A short cluster label (3-6 words).
2) One-sentence justification referencing the evidence.

Constraints:
- Keep the label generic (not game-specific).
- Do NOT include profanity or slurs in the label.
- If exemplars are meme-like or low-effort, reflect that.

INPUT:
Sentiment: NEGATIVE
Cluster ID: 0
Top terms: game, like, just, really, fun

Exemplar 1: This game is good for the first 50% or so but the game-play loop
gets old and the story isn't strong enough to make up for that. Its not a bad
game but its not great or even good it is just ok. I would say if you get it on
a sale it is worth it but full price it is not worth

OUTPUT FORMAT:
Label: …
Justification: … captures the overall sentiment and key themes from the
exemplar review, which express disappointment with the game's longevity and
storytelling while acknowledging it's not entirely bad but merely average.

Label: Game Disappointment
Justification: This label encapsulates the overall negative sentiment expressed
in the exemplar review, where the reviewer expresses disappointment with the
game's lack of engaging gameplay and weak storyline, despite acknowledging it's
not terrible but merely satisfactory. The term "game" in the top terms
reinforces this theme. The exemplar review also mentions the game becoming old
after the initial period, further supporting the "disappointment"
```

[29]:
```python
# build negative clusters
tfidf_neg, X_neg, labels_neg, kmeans_neg = cluster_reviews(df_neg, k=5)
neg_clusters = summarize_review_clusters(
    tfidf_neg, X_neg, labels_neg, kmeans_neg, df_neg
)

# build positive clusters
tfidf_pos, X_pos, labels_pos, kmeans_pos = cluster_reviews(df_pos, k=5)
pos_clusters = summarize_review_clusters(
    tfidf_pos, X_pos, labels_pos, kmeans_pos, df_pos
)
```

[33]:
```python
import re

# Q12: Run LLM labeling on 3 example clusters (2 negative + 1 positive)

assert callable(globals().get("generate", None)), "Run the helper code so␣
↪generate(prompt) exists."

def build_prompt(sentiment, cluster_id, top_terms, exemplar):
    terms_str = ", ".join([str(t) for t in top_terms])
    ex = str(exemplar).replace("\n", " ").strip()
    ex = ex[:350] + ("..." if len(ex) > 350 else "")

    return f"""You are helping label clusters of Steam reviews.

Task: Given (A) whether these are POSITIVE or NEGATIVE reviews, (B) top TF-IDF␣
↪terms,
and (C) 1 exemplar review, produce:
1) A short cluster label (3-6 words).
2) One-sentence justification referencing the evidence.

Constraints:
- Keep the label generic (not game-specific or company-specific).
- Do NOT mention game titles, companies, or proper nouns.
- Do NOT include profanity or slurs in the label.
- If the exemplar is meme-like or low-effort, reflect that.
- Do NOT include profanity or slurs in the justification.

INPUT:
Sentiment: {sentiment}
Cluster ID: {cluster_id}
Top terms: {terms_str}

Exemplar: {ex}

OUTPUT FORMAT:
Label: ...
Justification: ...

Output ONLY one label and one justification.
Do not provide multiple alternatives.
"""

def list_to_dict(cluster_list):
    # cluster_list is like neg_clusters / pos_clusters: list of dicts with keys
    # "cluster", "top_terms", "exemplar"
    return {int(c["cluster"]): c for c in cluster_list}
```

```python
def extract_first_label_and_justification(text: str) -> str:
    m = re.search(r"Label\s*:\s*(.*?)\nJustification\s*:\s*(.*)", text,␣
 ↪flags=re.DOTALL)
    if not m:
        return text.strip()
    label = m.group(1).strip()
    just = m.group(2).strip()
    # cut justification if another "Label:" appears later
    just = re.split(r"\nLabel\s*:", just)[0].strip()
    return f"Label: {label}\nJustification: {just}"

def keep_one_sentence_justification(block: str) -> str:
    # expects "Label: ...\nJustification: ..."
    m = re.search(r"Label\s*:\s*(.*?)\nJustification\s*:\s*(.*)", block,␣
 ↪flags=re.DOTALL)
    if not m:
        return block
    label = m.group(1).strip()
    just = m.group(2).strip()

    # keep only first sentence (up to first . ! ?)
    m2 = re.search(r"^(.+?[.!?])(\s|$)", just)
    if m2:
        just = m2.group(1).strip()
    else:
        just = just.split("\n")[0].strip()

    return f"Label: {label}\nJustification: {just}"

neg_map = list_to_dict(neg_clusters)
pos_map = list_to_dict(pos_clusters)

examples = [
    ("NEGATIVE", 0, neg_map[0]),
    ("NEGATIVE", 2, neg_map[2]),
    ("POSITIVE", 1, pos_map[1]),
]

print("=== Q12 Prompting Strategy ===")
print("Evidence provided to LLM: top TF-IDF terms + 1 exemplar review␣
 ↪(closest-to-centroid).\n")

for sentiment, cid, c in examples:
    prompt = build_prompt(sentiment, cid, c["top_terms"], c["exemplar"])
    print("=" * 70)
    print(f"DEMO CLUSTER: {sentiment} / Cluster {cid}")
    print(f"- Evidence (top terms): {c['top_terms']}")
```

```
    print(f"- Evidence (exemplar): {str(c['exemplar'])[:200]}{'...' if␣
↪len(str(c['exemplar']))>200 else ''}\n")
    print("--- LLM Output ---")
    out = extract_first_label_and_justification(generate(prompt))
    out = keep_one_sentence_justification(out)
    print(out)
```

```
=== Q12 Prompting Strategy ===
Evidence provided to LLM: top TF-IDF terms + 1 exemplar review (closest-to-
centroid).

======================================================================
DEMO CLUSTER: NEGATIVE / Cluster 0
- Evidence (top terms): ['game' 'like' 'just' 'really' 'fun']
- Evidence (exemplar): This game is good for the first 50% or so but the game-
play loop gets old and the story isn't strong enough to make up for that. Its
not a bad game but its not great or even good it is just ok. I woul…

--- LLM Output ---
Label: Generic Gameplay Dissatisfaction
Justification: The exemplar review expresses disappointment with the game's
gameplay loop, stating it becomes repetitive ("gets old") and lacking in a
compelling story, despite initial enjoyment ("good for the first 50%").
======================================================================
DEMO CLUSTER: NEGATIVE / Cluster 2
- Evidence (top terms): ['game' 'just' 'like' 'time' 'really']
- Evidence (exemplar): So I won't lie… It's not a terrible game, but I am
genuinely surprised this game is getting "Very Positive" Reviews right now.
Because       there are some serious issues in the game that need t…

--- LLM Output ---
Label: Generic Negative Feedback
Justification: The exemplar review expresses dissatisfaction with the game,
mentioning serious issues and noting that many positive reviews seem insincere,
indicating a cluster of negative feedback.
======================================================================
DEMO CLUSTER: POSITIVE / Cluster 1
- Evidence (top terms): ['10' 'game' 'like' 'better' 'just']
- Evidence (exemplar): A.B.A. IS BACK 10/10 GAME

--- LLM Output ---
Label: Fun Game Experience
Justification: The positive sentiment and use of phrases like "like" and
"better" indicate a favorable experience with the game, as evidenced by the
exemplar review praising it highly.
```

Answer Q12:

Prompting strategy:

We used a structured prompt that provides the LLM with three pieces of information for each cluster:

1. The review sentiment (positive or negative).
2. The top TF-IDF terms for the cluster.
3. One exemplar review selected as the item closest to the cluster centroid.

The model is asked to produce: - A short cluster label (3–6 words). - A one-sentence justification referencing the provided evidence.

We include constraints to ensure: - The label is generic and not game-specific. - No proper nouns, profanity, or slurs appear in the output. - Only one label and justification are produced.

Evidence provided to the LLM:

For each cluster, we supply: - Top TF-IDF terms to capture the dominant vocabulary. - A centroid-nearest exemplar review to provide concrete context.

This combination helps the LLM infer both the topic and the tone of the cluster.

Example LLM-generated labels:

Negative Cluster 0 - Label: Generic Gameplay Dissatisfaction - Justification: The exemplar review describes a repetitive gameplay loop and weak story, matching the negative sentiment reflected in the top terms.

Negative Cluster 2 - Label: Generic Negative Feedback - Justification: The exemplar expresses dissatisfaction and mentions serious issues despite positive ratings, indicating a cluster of negative reviews.

Positive Cluster 1 - Label: Fun Game Experience - Justification: The exemplar gives a "10/10" rating and the top terms reflect positive sentiment, indicating strong approval of the game.

219pset2part2

February 20, 2026

0.1 Q13

Even if a VGG network is trained on a dataset with different classes such as ImageNet, it mainly learns general visual features rather than class-specific information. The early layers capture basic patterns like edges and textures, while deeper layers learn more complex shapes and object parts. Since these features are common across most natural images, they are still useful for distinguishing new classes when VGG is used as a feature extractor.

[1]:
```
import torch
import torch.nn as nn
from torchvision import transforms, datasets
from torch.utils.data import DataLoader, TensorDataset
import numpy as np
import matplotlib.pyplot as plt

from tqdm import tqdm
import requests
import os
import tarfile

from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.metrics import confusion_matrix, adjusted_rand_score,␣
↪adjusted_mutual_info_score
from sklearn.pipeline import Pipeline
from sklearn.base import TransformerMixin
```

0.2 Q14

The helper code extracts features using a pretrained VGG-16 model. Each flower image is resized and normalized, then passed through the VGG-16 network. Instead of using the final classification layer, the code takes the output from the first fully connected layer, which produces a 4096-dimensional feature vector for each image. These vectors represent high-level visual patterns in the images. The features and their labels are saved to disk so they can be reused later without running the network again.

[2]:
```
filename = './flowers_features_and_labels.npz'

if os.path.exists(filename):
    file = np.load(filename)
    f_all, y_all = file['f_all'], file['y_all']

else:
    if not os.path.exists('./flower_photos'):
        # download the flowers dataset and extract its images
        url = 'http://download.tensorflow.org/example_images/flower_photos.tgz'
        with open('./flower_photos.tgz', 'wb') as file:
            file.write(requests.get(url).content)
        with tarfile.open('./flower_photos.tgz') as file:
            file.extractall('./')
        os.remove('./flower_photos.tgz')

    class FeatureExtractor(nn.Module):
        def __init__(self):
            super().__init__()

            vgg = torch.hub.load('pytorch/vision:v0.10.0', 'vgg16',␣
 ↪pretrained=True)

            # Extract VGG-16 Feature Layers
            self.features = list(vgg.features)
            self.features = nn.Sequential(*self.features)
            # Extract VGG-16 Average Pooling Layer
            self.pooling = vgg.avgpool
            # Convert the image into one-dimensional vector
            self.flatten = nn.Flatten()
            # Extract the first part of fully-connected layer from VGG16
            self.fc = vgg.classifier[0]

        def forward(self, x):
            # It will take the input 'x' until it returns the feature vector␣
 ↪called 'out'
            out = self.features(x)
            out = self.pooling(out)
            out = self.flatten(out)
            out = self.fc(out)
            return out

    # Initialize the model
    assert torch.cuda.is_available()
    feature_extractor = FeatureExtractor().cuda().eval()

    dataset = datasets.ImageFolder(root='./flower_photos',
```

```
                                    transform=transforms.Compose([transforms.
↪Resize(224),
                                                                  transforms.
↪CenterCrop(224),
                                                                  transforms.
↪ToTensor(),
                                                                  transforms.
↪Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])]))
    dataloader = DataLoader(dataset, batch_size=64, shuffle=True)

    # Extract features and store them on disk
    f_all, y_all = np.zeros((0, 4096)), np.zeros((0,))
    for x, y in tqdm(dataloader):
        with torch.no_grad():
            f_all = np.vstack([f_all, feature_extractor(x.cuda()).cpu()])
            y_all = np.concatenate([y_all, y])
    np.savez(filename, f_all=f_all, y_all=y_all)
```

```
/tmp/ipython-input-2130011012.py:14: DeprecationWarning: Python 3.14 will, by
default, filter extracted tar archives and reject files or modify their
metadata. Use the filter argument to control this behavior.
  file.extractall('./')
```

```
Downloading: "https://github.com/pytorch/vision/zipball/v0.10.0" to
/root/.cache/torch/hub/v0.10.0.zip
```

```
/usr/local/lib/python3.12/dist-packages/torchvision/models/_utils.py:208:
UserWarning: The parameter 'pretrained' is deprecated since 0.13 and may be
removed in the future, please use 'weights' instead.
  warnings.warn(
/usr/local/lib/python3.12/dist-packages/torchvision/models/_utils.py:223:
UserWarning: Arguments other than a weight enum or `None` for 'weights' are
deprecated since 0.13 and may be removed in the future. The current behavior is
equivalent to passing `weights=VGG16_Weights.IMAGENET1K_V1`. You can also use
`weights=VGG16_Weights.DEFAULT` to get the most up-to-date weights.
  warnings.warn(msg)
```

```
Downloading: "https://download.pytorch.org/models/vgg16-397923af.pth" to
/root/.cache/torch/hub/checkpoints/vgg16-397923af.pth
```

```
100%|      | 528M/528M [00:03<00:00, 148MB/s]
100%|      | 58/58 [00:34<00:00,  1.68it/s]
```

0.3 Q15

224 * 224 *3 =150528 pixel values. 4096-dimensional feature vector.

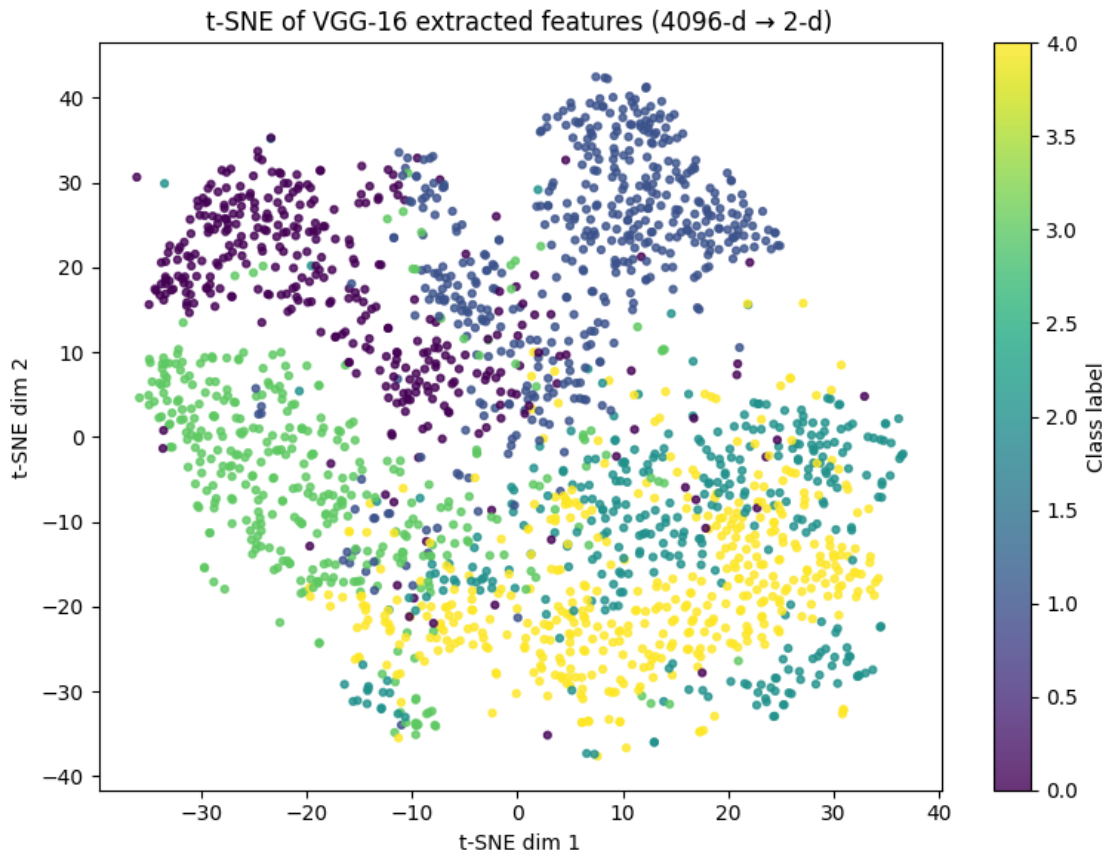
0.4 Q16

The VGG features are dense, meaning most values in the 4096-dimensional vector are non-zero because they come from neural network activations. This is different from TF-IDF features, where most values are zero since each document only contains a small number of words. In short, VGG features capture visual patterns with many active values, while TF-IDF represents text by counting which words appear.

0.5 Q17

[3]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.manifold import TSNE
from sklearn.preprocessing import StandardScaler

file = np.load("./flowers_features_and_labels.npz")
f_all = file["f_all"]
y_all = file["y_all"].astype(int)

print("f_all:", f_all.shape, "y_all:", y_all.shape)

np.random.seed(42)
N = f_all.shape[0]
max_points = 2000
idx = np.random.choice(N, size=min(max_points, N), replace=False)
X = f_all[idx]
y = y_all[idx]

X = StandardScaler().fit_transform(X)

tsne = TSNE(
    n_components=2,
    perplexity=30,
    learning_rate="auto",
    init="pca",
    random_state=42
)
Z = tsne.fit_transform(X)

plt.figure(figsize=(8, 6))
sc = plt.scatter(Z[:, 0], Z[:, 1], c=y, s=12, alpha=0.8)
plt.xlabel("t-SNE dim 1")
plt.ylabel("t-SNE dim 2")
plt.title("t-SNE of VGG-16 extracted features (4096-d → 2-d)")
cbar = plt.colorbar(sc)
cbar.set_label("Class label")
plt.tight_layout()
plt.show()
```

```
f_all: (3670, 4096) y_all: (3670,)
```



From the t-SNE plot, the VGG features form noticeable clusters that roughly correspond to different flower classes, showing that the features capture meaningful visual differences. Some clusters are clearly separated in the 2D space, indicating that certain classes are easy to distinguish. However, there is also overlap in the middle area where points from different classes mix together, suggesting that some flowers look similar in color or shape. Overall, the plot shows that VGG features contain useful class information but are not perfectly separable, so a supervised model is still helpful for achieving the best classification performance

0.6 Q18

[5]:
```
class Autoencoder(torch.nn.Module, TransformerMixin):
    def __init__(self, n_components):
        super().__init__()
        self.n_components = n_components
        self.n_features = None  # to be determined with data
        self.encoder = None
        self.decoder = None
```

```python
    def _create_encoder(self):
        return nn.Sequential(
            nn.Linear(4096, 1280),
            nn.ReLU(True),
            nn.Linear(1280, 640),
            nn.ReLU(True), nn.Linear(640, 120), nn.ReLU(True), nn.Linear(120,␣
↪self.n_components))

    def _create_decoder(self):
        return nn.Sequential(
            nn.Linear(self.n_components, 120),
            nn.ReLU(True),
            nn.Linear(120, 640),
            nn.ReLU(True),
            nn.Linear(640, 1280),
            nn.ReLU(True), nn.Linear(1280, 4096))

    def forward(self, X):
        encoded = self.encoder(X)
        decoded = self.decoder(encoded)
        return decoded

    def fit(self, X):
        X = torch.tensor(X, dtype=torch.float32, device='cuda')
        self.n_features = X.shape[1]
        self.encoder = self._create_encoder()
        self.decoder = self._create_decoder()
        self.cuda()
        self.train()

        criterion = nn.MSELoss()
        optimizer = torch.optim.Adam(self.parameters(), lr=1e-3,␣
↪weight_decay=1e-5)

        dataset = TensorDataset(X)
        dataloader = DataLoader(dataset, batch_size=128, shuffle=True)

        for epoch in tqdm(range(100)):
            for (X_,) in dataloader:
                X_ = X_.cuda()

                output = self(X_)
                loss = criterion(output, X_)
                optimizer.zero_grad()
                loss.backward()
                optimizer.step()
```

```
        return self

    def transform(self, X):
        X = torch.tensor(X, dtype=torch.float32, device='cuda')
        self.eval()
        with torch.no_grad():
            return self.encoder(X).cpu().numpy()
```

[6]:
```
import numpy as np
import pandas as pd

from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.cluster import KMeans, AgglomerativeClustering
from sklearn.metrics import adjusted_rand_score

import umap
import hdbscan


HDBSCAN_MIN_CLUSTER_SIZE_GRID = [5, 10, 20]
HDBSCAN_MIN_SAMPLES_GRID = [None, 5, 10]

RANDOM_STATE = 42
np.random.seed(RANDOM_STATE)

def run_kmeans(Z, k=5):
    model = KMeans(n_clusters=k, n_init=20, random_state=RANDOM_STATE)
    pred = model.fit_predict(Z)
    return pred

def run_agglo(Z, n_clusters=5):
    model = AgglomerativeClustering(n_clusters=n_clusters)
    pred = model.fit_predict(Z)
    return pred

def run_hdbscan_grid(Z, y_true):

    best = {
        "ari": -1.0,
        "min_cluster_size": None,
        "min_samples": None,
        "n_clusters_found": None,
        "noise_frac": None,
        "labels": None
    }
```

```
    for mcs in HDBSCAN_MIN_CLUSTER_SIZE_GRID:
        for ms in HDBSCAN_MIN_SAMPLES_GRID:
            clusterer = hdbscan.HDBSCAN(
                min_cluster_size=mcs,
                min_samples=ms
            )
            labels = clusterer.fit_predict(Z)

            ari = adjusted_rand_score(y_true, labels)
            n_clusters = len(set(labels)) - (1 if -1 in labels else 0)
            noise_frac = float(np.mean(labels == -1))

            if ari > best["ari"]:
                best.update({
                    "ari": float(ari),
                    "min_cluster_size": mcs,
                    "min_samples": ms,
                    "n_clusters_found": int(n_clusters),
                    "noise_frac": noise_frac,
                    "labels": labels
                })

    return best

def ensure_numpy(X):
    if hasattr(X, "toarray"):
        return X.toarray()
    return np.asarray(X)

reducers = {}

# None
reducers["None"] = ensure_numpy(f_all)

svd = make_pipeline(
    StandardScaler(with_mean=True, with_std=True),
    TruncatedSVD(n_components=50, random_state=RANDOM_STATE)
)
reducers["SVD(r=50)"] = svd.fit_transform(ensure_numpy(f_all))

um = make_pipeline(
    StandardScaler(with_mean=True, with_std=True),
    umap.UMAP(
        n_components=50,
        random_state=RANDOM_STATE,
        n_neighbors=15,
```

```
        min_dist=0.1,
        metric="euclidean"
    )
)
reducers["UMAP(n=50)"] = um.fit_transform(ensure_numpy(f_all))


try:
    Z_ae = Autoencoder(50).fit_transform(ensure_numpy(f_all))
    reducers["Autoencoder(f=50)"] = Z_ae
except Exception as e:
    print("[WARN] Autoencoder failed (likely input dim != 4096 or no CUDA).␣
 ↪Error:")
    print(e)

rows = []

for red_name, Z in reducers.items():
    Z = ensure_numpy(Z)

    # KMeans
    pred_km = run_kmeans(Z, k=5)
    ari_km = adjusted_rand_score(y_all, pred_km)
    rows.append({
        "DimRed": red_name,
        "Clustering": "KMeans",
        "Hyperparameters": "k=5",
        "ARI": float(ari_km)
    })

    # Agglomerative
    pred_ag = run_agglo(Z, n_clusters=5)
    ari_ag = adjusted_rand_score(y_all, pred_ag)
    rows.append({
        "DimRed": red_name,
        "Clustering": "Agglomerative",
        "Hyperparameters": "n_clusters=5",
        "ARI": float(ari_ag)
    })

    # HDBSCAN grid search
    best_hdb = run_hdbscan_grid(Z, y_all)
    rows.append({
        "DimRed": red_name,
        "Clustering": "HDBSCAN",
        "Hyperparameters": f"min_cluster_size={best_hdb['min_cluster_size']},␣
 ↪min_samples={best_hdb['min_samples']}, "
```

```
                            f"clusters_found={best_hdb['n_clusters_found']},␣
↪noise_frac={best_hdb['noise_frac']:.3f}",
        "ARI": float(best_hdb["ari"])
    })

df = pd.DataFrame(rows).sort_values("ARI", ascending=False).
↪reset_index(drop=True)

print("=== All results (sorted by ARI desc) ===")
display(df)

best_row = df.iloc[0]
print("\n=== BEST RESULT ===")
print(best_row.to_dict())
```

```
/usr/local/lib/python3.12/dist-packages/hdbscan/robust_single_linkage_.py:175:
SyntaxWarning: invalid escape sequence '\{'
  $max \{ core_k(a), core_k(b), 1/\alpha d(a,b) \}$.
/usr/local/lib/python3.12/dist-packages/umap/umap_.py:1952: UserWarning: n_jobs
value 1 overridden to 1 by setting random_state. Use no seed for parallelism.
  warn(
100%|      | 100/100 [00:19<00:00,  5.11it/s]

=== All results (sorted by ARI desc) ===

               DimRed     Clustering  \
0          UMAP(n=50)         KMeans
1          UMAP(n=50)  Agglomerative
2   Autoencoder(f=50)         KMeans
3          UMAP(n=50)        HDBSCAN
4           SVD(r=50)  Agglomerative
5   Autoencoder(f=50)  Agglomerative
6                 None  Agglomerative
7           SVD(r=50)         KMeans
8                 None         KMeans
9                 None        HDBSCAN
10          SVD(r=50)        HDBSCAN
11  Autoencoder(f=50)        HDBSCAN

                                      Hyperparameters       ARI
0                                                 k=5  0.418576
1                                        n_clusters=5  0.366950
2                                                 k=5  0.242802
3   min_cluster_size=20, min_samples=None, cluster…  0.227520
4                                        n_clusters=5  0.222730
5                                        n_clusters=5  0.220893
6                                        n_clusters=5  0.218450
7                                                 k=5  0.191255
```

```
8                                                   k=5  0.189198
9   min_cluster_size=5, min_samples=None, clusters…  0.006676
10  min_cluster_size=10, min_samples=5, clusters_f…  0.005775
11  min_cluster_size=10, min_samples=5, clusters_f…  0.004659

=== BEST RESULT ===
{'DimRed': 'UMAP(n=50)', 'Clustering': 'KMeans', 'Hyperparameters': 'k=5',
'ARI': 0.41857617946991876}
```

0.7 Q19

[7]:
```
import numpy as np
import torch
import torch.nn as nn
from torch.utils.data import TensorDataset, DataLoader
from tqdm import tqdm

class MLP(torch.nn.Module):
    def __init__(self, num_features):
        super().__init__()
        self.model = nn.Sequential(
            nn.Linear(num_features, 1280),
            nn.ReLU(True),
            nn.Linear(1280, 640),
            nn.ReLU(True),
            nn.Linear(640, 5),
            nn.LogSoftmax(dim=1)
        )
        self.cuda()

    def forward(self, X):
        return self.model(X)

    def train(self, X, y):
        X = torch.tensor(X, dtype=torch.float32, device='cuda')
        y = torch.tensor(y, dtype=torch.int64, device='cuda')

        self.model.train()

        criterion = nn.NLLLoss()
        optimizer = torch.optim.Adam(self.parameters(), lr=1e-3,␣
 ↪weight_decay=1e-5)

        dataset = TensorDataset(X, y)
        dataloader = DataLoader(dataset, batch_size=128, shuffle=True)

        for epoch in tqdm(range(100)):
```

```
            for (X_, y_) in dataloader:
                # forward
                log_probs = self.forward(X_)
                loss = criterion(log_probs, y_)

                # backward
                optimizer.zero_grad()
                loss.backward()
                optimizer.step()

        return self

    def eval(self, X_test, y_test):
        X_test = torch.tensor(X_test, dtype=torch.float32, device='cuda')
        y_test = torch.tensor(y_test, dtype=torch.int64, device='cuda')

        self.model.eval()
        with torch.no_grad():
            log_probs = self.forward(X_test)
            pred = torch.argmax(log_probs, dim=1)
            acc = (pred == y_test).float().mean().item()
        return acc, pred.cpu().numpy()
```

[8]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
import umap

X_train, X_test, y_train, y_test = train_test_split(
    f_all, y_all, test_size=0.2, stratify=y_all, random_state=42
)

mlp_orig = MLP(num_features=X_train.shape[1]).train(X_train, y_train)
acc_orig, pred_orig = mlp_orig.eval(X_test, y_test)
print("Test accuracy (Original VGG 4096D):", acc_orig)

reducer = make_pipeline(
    StandardScaler(with_mean=True, with_std=True),
    umap.UMAP(n_components=50, random_state=42, n_neighbors=15, min_dist=0.1)
)

Z_train = reducer.fit_transform(X_train)
Z_test  = reducer.transform(X_test)

mlp_red = MLP(num_features=Z_train.shape[1]).train(Z_train, y_train)
acc_red, pred_red = mlp_red.eval(Z_test, y_test)
print("Test accuracy (Reduced features 50D):", acc_red)
```

```
100%|      | 100/100 [00:07<00:00, 14.12it/s]

Test accuracy (Original VGG 4096D): 0.9032697677612305

/usr/local/lib/python3.12/dist-packages/umap/umap_.py:1952: UserWarning: n_jobs
value 1 overridden to 1 by setting random_state. Use no seed for parallelism.
  warn(
100%|      | 100/100 [00:06<00:00, 16.48it/s]

Test accuracy (Reduced features 50D): 0.8174386620521545
```

Test accuracy (Original VGG 4096D): 0.9032697677612305

Test accuracy (Reduced features 50D): 0.8174386620521545

This indicates that the model's performance does suffer after dimensionality reduction. However, the drop is moderate rather than drastic, suggesting that the lower-dimensional representation still retains a substantial amount of class-relevant information.

The classification results are consistent with the clustering findings in Question 18. Clustering may not perfectly separate the classes because it does not use labels. However, the relatively high classification accuracy shows that the VGG features even after dimensionality reduction still contain useful information for distinguishing classes. This makes sense because supervised models use labels to learn clearer boundaries, while clustering does not.

P2_part3

February 20, 2026

0.0.1 Part 3

Preparation

[1]:
```
# Helper code session (Copied from pokedex_helper):
!pip install datasets transformers numpy pandas Pillow matplotlib
!pip install torch tqdm scipy
!pip install git+https://github.com/openai/CLIP.git
!pip install plotly umap-learn
```

```
Requirement already satisfied: datasets in /usr/local/lib/python3.12/dist-
packages (4.0.0)
Requirement already satisfied: transformers in /usr/local/lib/python3.12/dist-
packages (5.2.0.dev0)
Requirement already satisfied: numpy in /usr/local/lib/python3.12/dist-packages
(2.0.2)
Requirement already satisfied: pandas in /usr/local/lib/python3.12/dist-packages
(2.2.2)
Requirement already satisfied: Pillow in /usr/local/lib/python3.12/dist-packages
(12.1.1)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.12/dist-
packages (3.10.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.12/dist-
packages (from datasets) (3.20.3)
Requirement already satisfied: pyarrow>=15.0.0 in
/usr/local/lib/python3.12/dist-packages (from datasets) (18.1.0)
Requirement already satisfied: dill<0.3.9,>=0.3.0 in
/usr/local/lib/python3.12/dist-packages (from datasets) (0.3.8)
Requirement already satisfied: requests>=2.32.2 in
/usr/local/lib/python3.12/dist-packages (from datasets) (2.32.4)
Requirement already satisfied: tqdm>=4.66.3 in /usr/local/lib/python3.12/dist-
packages (from datasets) (4.67.3)
Requirement already satisfied: xxhash in /usr/local/lib/python3.12/dist-packages
(from datasets) (3.6.0)
Requirement already satisfied: multiprocess<0.70.17 in
/usr/local/lib/python3.12/dist-packages (from datasets) (0.70.16)
Requirement already satisfied: fsspec<=2025.3.0,>=2023.1.0 in
/usr/local/lib/python3.12/dist-packages (from
fsspec[http]<=2025.3.0,>=2023.1.0->datasets) (2025.3.0)
Requirement already satisfied: huggingface-hub>=0.24.0 in
```

```
/usr/local/lib/python3.12/dist-packages (from datasets) (1.4.1)
Requirement already satisfied: packaging in /usr/local/lib/python3.12/dist-
packages (from datasets) (26.0)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.12/dist-
packages (from datasets) (6.0.3)
Requirement already satisfied: regex!=2019.12.17 in
/usr/local/lib/python3.12/dist-packages (from transformers) (2025.11.3)
Requirement already satisfied: tokenizers<=0.23.0,>=0.22.0 in
/usr/local/lib/python3.12/dist-packages (from transformers) (0.22.2)
Requirement already satisfied: typer-slim in /usr/local/lib/python3.12/dist-
packages (from transformers) (0.21.1)
Requirement already satisfied: safetensors>=0.4.3 in
/usr/local/lib/python3.12/dist-packages (from transformers) (0.7.0)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.12/dist-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.12/dist-
packages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dist-
packages (from pandas) (2025.3)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.12/dist-packages (from matplotlib) (1.3.3)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.12/dist-
packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.12/dist-packages (from matplotlib) (4.61.1)
Requirement already satisfied: kiwisolver>=1.3.1 in
/usr/local/lib/python3.12/dist-packages (from matplotlib) (1.4.9)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.12/dist-packages (from matplotlib) (3.3.2)
Requirement already satisfied: aiohttp!=4.0.0a0,!=4.0.0a1 in
/usr/local/lib/python3.12/dist-packages (from
fsspec[http]<=2025.3.0,>=2023.1.0->datasets) (3.13.3)
Requirement already satisfied: hf-xet<2.0.0,>=1.2.0 in
/usr/local/lib/python3.12/dist-packages (from huggingface-hub>=0.24.0->datasets)
(1.2.0)
Requirement already satisfied: httpx<1,>=0.23.0 in
/usr/local/lib/python3.12/dist-packages (from huggingface-hub>=0.24.0->datasets)
(0.28.1)
Requirement already satisfied: shellingham in /usr/local/lib/python3.12/dist-
packages (from huggingface-hub>=0.24.0->datasets) (1.5.4)
Requirement already satisfied: typing-extensions>=4.1.0 in
/usr/local/lib/python3.12/dist-packages (from huggingface-hub>=0.24.0->datasets)
(4.15.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-
packages (from python-dateutil>=2.8.2->pandas) (1.17.0)
Requirement already satisfied: charset_normalizer<4,>=2 in
/usr/local/lib/python3.12/dist-packages (from requests>=2.32.2->datasets)
(3.4.4)
```

```
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-
packages (from requests>=2.32.2->datasets) (3.11)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.12/dist-packages (from requests>=2.32.2->datasets)
(2.5.0)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.12/dist-packages (from requests>=2.32.2->datasets)
(2026.1.4)
Requirement already satisfied: click>=8.0.0 in /usr/local/lib/python3.12/dist-
packages (from typer-slim->transformers) (8.3.1)
Requirement already satisfied: aiohappyeyeballs>=2.5.0 in
/usr/local/lib/python3.12/dist-packages (from
aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]<=2025.3.0,>=2023.1.0->datasets) (2.6.1)
Requirement already satisfied: aiosignal>=1.4.0 in
/usr/local/lib/python3.12/dist-packages (from
aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]<=2025.3.0,>=2023.1.0->datasets) (1.4.0)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.12/dist-
packages (from
aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]<=2025.3.0,>=2023.1.0->datasets)
(25.4.0)
Requirement already satisfied: frozenlist>=1.1.1 in
/usr/local/lib/python3.12/dist-packages (from
aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]<=2025.3.0,>=2023.1.0->datasets) (1.8.0)
Requirement already satisfied: multidict<7.0,>=4.5 in
/usr/local/lib/python3.12/dist-packages (from
aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]<=2025.3.0,>=2023.1.0->datasets) (6.7.1)
Requirement already satisfied: propcache>=0.2.0 in
/usr/local/lib/python3.12/dist-packages (from
aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]<=2025.3.0,>=2023.1.0->datasets) (0.4.1)
Requirement already satisfied: yarl<2.0,>=1.17.0 in
/usr/local/lib/python3.12/dist-packages (from
aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]<=2025.3.0,>=2023.1.0->datasets)
(1.22.0)
Requirement already satisfied: anyio in /usr/local/lib/python3.12/dist-packages
(from httpx<1,>=0.23.0->huggingface-hub>=0.24.0->datasets) (4.12.1)
Requirement already satisfied: httpcore==1.* in /usr/local/lib/python3.12/dist-
packages (from httpx<1,>=0.23.0->huggingface-hub>=0.24.0->datasets) (1.0.9)
Requirement already satisfied: h11>=0.16 in /usr/local/lib/python3.12/dist-
packages (from httpcore==1.*->httpx<1,>=0.23.0->huggingface-
hub>=0.24.0->datasets) (0.16.0)
Requirement already satisfied: torch in /usr/local/lib/python3.12/dist-packages
(2.10.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.12/dist-packages
(4.67.3)
Requirement already satisfied: scipy in /usr/local/lib/python3.12/dist-packages
(1.16.3)
Requirement already satisfied: filelock in /usr/local/lib/python3.12/dist-
packages (from torch) (3.20.3)
```

```
Requirement already satisfied: typing-extensions>=4.10.0 in
/usr/local/lib/python3.12/dist-packages (from torch) (4.15.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.12/dist-
packages (from torch) (75.2.0)
Requirement already satisfied: sympy>=1.13.3 in /usr/local/lib/python3.12/dist-
packages (from torch) (1.14.0)
Requirement already satisfied: networkx>=2.5.1 in
/usr/local/lib/python3.12/dist-packages (from torch) (3.6.1)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.12/dist-packages
(from torch) (3.1.6)
Requirement already satisfied: fsspec>=0.8.5 in /usr/local/lib/python3.12/dist-
packages (from torch) (2025.3.0)
Requirement already satisfied: cuda-bindings==12.9.4 in
/usr/local/lib/python3.12/dist-packages (from torch) (12.9.4)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.8.93 in
/usr/local/lib/python3.12/dist-packages (from torch) (12.8.93)
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.8.90 in
/usr/local/lib/python3.12/dist-packages (from torch) (12.8.90)
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.8.90 in
/usr/local/lib/python3.12/dist-packages (from torch) (12.8.90)
Requirement already satisfied: nvidia-cudnn-cu12==9.10.2.21 in
/usr/local/lib/python3.12/dist-packages (from torch) (9.10.2.21)
Requirement already satisfied: nvidia-cublas-cu12==12.8.4.1 in
/usr/local/lib/python3.12/dist-packages (from torch) (12.8.4.1)
Requirement already satisfied: nvidia-cufft-cu12==11.3.3.83 in
/usr/local/lib/python3.12/dist-packages (from torch) (11.3.3.83)
Requirement already satisfied: nvidia-curand-cu12==10.3.9.90 in
/usr/local/lib/python3.12/dist-packages (from torch) (10.3.9.90)
Requirement already satisfied: nvidia-cusolver-cu12==11.7.3.90 in
/usr/local/lib/python3.12/dist-packages (from torch) (11.7.3.90)
Requirement already satisfied: nvidia-cusparse-cu12==12.5.8.93 in
/usr/local/lib/python3.12/dist-packages (from torch) (12.5.8.93)
Requirement already satisfied: nvidia-cusparselt-cu12==0.7.1 in
/usr/local/lib/python3.12/dist-packages (from torch) (0.7.1)
Requirement already satisfied: nvidia-nccl-cu12==2.27.5 in
/usr/local/lib/python3.12/dist-packages (from torch) (2.27.5)
Requirement already satisfied: nvidia-nvshmem-cu12==3.4.5 in
/usr/local/lib/python3.12/dist-packages (from torch) (3.4.5)
Requirement already satisfied: nvidia-nvtx-cu12==12.8.90 in
/usr/local/lib/python3.12/dist-packages (from torch) (12.8.90)
Requirement already satisfied: nvidia-nvjitlink-cu12==12.8.93 in
/usr/local/lib/python3.12/dist-packages (from torch) (12.8.93)
Requirement already satisfied: nvidia-cufile-cu12==1.13.1.3 in
/usr/local/lib/python3.12/dist-packages (from torch) (1.13.1.3)
Requirement already satisfied: triton==3.6.0 in /usr/local/lib/python3.12/dist-
packages (from torch) (3.6.0)
Requirement already satisfied: cuda-pathfinder~=1.1 in
/usr/local/lib/python3.12/dist-packages (from cuda-bindings==12.9.4->torch)
```

```
(1.3.3)
Requirement already satisfied: numpy<2.6,>=1.25.2 in
/usr/local/lib/python3.12/dist-packages (from scipy) (2.0.2)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in
/usr/local/lib/python3.12/dist-packages (from sympy>=1.13.3->torch) (1.3.0)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.12/dist-packages (from jinja2->torch) (3.0.3)
Collecting git+https://github.com/openai/CLIP.git
  Cloning https://github.com/openai/CLIP.git to /tmp/pip-req-build-7e6ue96z
  Running command git clone --filter=blob:none --quiet
https://github.com/openai/CLIP.git /tmp/pip-req-build-7e6ue96z
  Resolved https://github.com/openai/CLIP.git to commit
dcba3cb2e2827b402d2701e7e1c7d9fed8a20ef1
  Preparing metadata (setup.py) … done
Requirement already satisfied: ftfy in /usr/local/lib/python3.12/dist-packages
(from clip==1.0) (6.3.1)
Requirement already satisfied: packaging in /usr/local/lib/python3.12/dist-
packages (from clip==1.0) (26.0)
Requirement already satisfied: regex in /usr/local/lib/python3.12/dist-packages
(from clip==1.0) (2025.11.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.12/dist-packages
(from clip==1.0) (4.67.3)
Requirement already satisfied: torch in /usr/local/lib/python3.12/dist-packages
(from clip==1.0) (2.10.0)
Requirement already satisfied: torchvision in /usr/local/lib/python3.12/dist-
packages (from clip==1.0) (0.25.0)
Requirement already satisfied: wcwidth in /usr/local/lib/python3.12/dist-
packages (from ftfy->clip==1.0) (0.5.3)
Requirement already satisfied: filelock in /usr/local/lib/python3.12/dist-
packages (from torch->clip==1.0) (3.20.3)
Requirement already satisfied: typing-extensions>=4.10.0 in
/usr/local/lib/python3.12/dist-packages (from torch->clip==1.0) (4.15.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.12/dist-
packages (from torch->clip==1.0) (75.2.0)
Requirement already satisfied: sympy>=1.13.3 in /usr/local/lib/python3.12/dist-
packages (from torch->clip==1.0) (1.14.0)
Requirement already satisfied: networkx>=2.5.1 in
/usr/local/lib/python3.12/dist-packages (from torch->clip==1.0) (3.6.1)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.12/dist-packages
(from torch->clip==1.0) (3.1.6)
Requirement already satisfied: fsspec>=0.8.5 in /usr/local/lib/python3.12/dist-
packages (from torch->clip==1.0) (2025.3.0)
Requirement already satisfied: cuda-bindings==12.9.4 in
/usr/local/lib/python3.12/dist-packages (from torch->clip==1.0) (12.9.4)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.8.93 in
/usr/local/lib/python3.12/dist-packages (from torch->clip==1.0) (12.8.93)
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.8.90 in
/usr/local/lib/python3.12/dist-packages (from torch->clip==1.0) (12.8.90)
```

```
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.8.90 in
/usr/local/lib/python3.12/dist-packages (from torch->clip==1.0) (12.8.90)
Requirement already satisfied: nvidia-cudnn-cu12==9.10.2.21 in
/usr/local/lib/python3.12/dist-packages (from torch->clip==1.0) (9.10.2.21)
Requirement already satisfied: nvidia-cublas-cu12==12.8.4.1 in
/usr/local/lib/python3.12/dist-packages (from torch->clip==1.0) (12.8.4.1)
Requirement already satisfied: nvidia-cufft-cu12==11.3.3.83 in
/usr/local/lib/python3.12/dist-packages (from torch->clip==1.0) (11.3.3.83)
Requirement already satisfied: nvidia-curand-cu12==10.3.9.90 in
/usr/local/lib/python3.12/dist-packages (from torch->clip==1.0) (10.3.9.90)
Requirement already satisfied: nvidia-cusolver-cu12==11.7.3.90 in
/usr/local/lib/python3.12/dist-packages (from torch->clip==1.0) (11.7.3.90)
Requirement already satisfied: nvidia-cusparse-cu12==12.5.8.93 in
/usr/local/lib/python3.12/dist-packages (from torch->clip==1.0) (12.5.8.93)
Requirement already satisfied: nvidia-cusparselt-cu12==0.7.1 in
/usr/local/lib/python3.12/dist-packages (from torch->clip==1.0) (0.7.1)
Requirement already satisfied: nvidia-nccl-cu12==2.27.5 in
/usr/local/lib/python3.12/dist-packages (from torch->clip==1.0) (2.27.5)
Requirement already satisfied: nvidia-nvshmem-cu12==3.4.5 in
/usr/local/lib/python3.12/dist-packages (from torch->clip==1.0) (3.4.5)
Requirement already satisfied: nvidia-nvtx-cu12==12.8.90 in
/usr/local/lib/python3.12/dist-packages (from torch->clip==1.0) (12.8.90)
Requirement already satisfied: nvidia-nvjitlink-cu12==12.8.93 in
/usr/local/lib/python3.12/dist-packages (from torch->clip==1.0) (12.8.93)
Requirement already satisfied: nvidia-cufile-cu12==1.13.1.3 in
/usr/local/lib/python3.12/dist-packages (from torch->clip==1.0) (1.13.1.3)
Requirement already satisfied: triton==3.6.0 in /usr/local/lib/python3.12/dist-
packages (from torch->clip==1.0) (3.6.0)
Requirement already satisfied: cuda-pathfinder~=1.1 in
/usr/local/lib/python3.12/dist-packages (from cuda-
bindings==12.9.4->torch->clip==1.0) (1.3.3)
Requirement already satisfied: numpy in /usr/local/lib/python3.12/dist-packages
(from torchvision->clip==1.0) (2.0.2)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in
/usr/local/lib/python3.12/dist-packages (from torchvision->clip==1.0) (12.1.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in
/usr/local/lib/python3.12/dist-packages (from sympy>=1.13.3->torch->clip==1.0)
(1.3.0)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.12/dist-packages (from jinja2->torch->clip==1.0) (3.0.3)
Requirement already satisfied: plotly in /usr/local/lib/python3.12/dist-packages
(5.24.1)
Requirement already satisfied: umap-learn in /usr/local/lib/python3.12/dist-
packages (0.5.11)
Requirement already satisfied: tenacity>=6.2.0 in
/usr/local/lib/python3.12/dist-packages (from plotly) (9.1.3)
Requirement already satisfied: packaging in /usr/local/lib/python3.12/dist-
packages (from plotly) (26.0)
```

```
Requirement already satisfied: numpy>=1.23 in /usr/local/lib/python3.12/dist-
packages (from umap-learn) (2.0.2)
Requirement already satisfied: scipy>=1.3.1 in /usr/local/lib/python3.12/dist-
packages (from umap-learn) (1.16.3)
Requirement already satisfied: scikit-learn>=1.6 in
/usr/local/lib/python3.12/dist-packages (from umap-learn) (1.6.1)
Requirement already satisfied: numba>=0.51.2 in /usr/local/lib/python3.12/dist-
packages (from umap-learn) (0.60.0)
Requirement already satisfied: pynndescent>=0.5 in
/usr/local/lib/python3.12/dist-packages (from umap-learn) (0.6.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.12/dist-packages
(from umap-learn) (4.67.3)
Requirement already satisfied: llvmlite<0.44,>=0.43.0dev0 in
/usr/local/lib/python3.12/dist-packages (from numba>=0.51.2->umap-learn)
(0.43.0)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.12/dist-
packages (from pynndescent>=0.5->umap-learn) (1.5.3)
Requirement already satisfied: threadpoolctl>=3.1.0 in
/usr/local/lib/python3.12/dist-packages (from scikit-learn>=1.6->umap-learn)
(3.6.0)
```

[2]:
```python
from datasets import load_dataset
from transformers import CLIPProcessor, CLIPModel
import numpy as np
import pandas as pd
from glob import glob
from PIL import Image
import matplotlib.pyplot as plt
import clip
import torch
from tqdm import tqdm
from scipy.special import softmax
import plotly.express as px
import plotly.graph_objects as go
from sklearn.manifold import TSNE
```

[3]:
```python
# load csv file and image paths to construct pokedex, use type_to_load=None to␣
↪load all types, else use a list of types 1 to load
def construct_pokedex(csv_path='Pokemon.csv', image_dir='./images/',␣
↪type_to_load=None):
    pokedex = pd.read_csv(csv_path)
    image_paths = []

    for pokemon_name in pokedex["Name"]:
        imgs = glob(f"{image_dir}/{pokemon_name}/0.jpg")
        if len(imgs) > 0:
            image_paths.append(imgs[0])
```

```
        else:
            image_paths.append(None)

    pokedex["image_path"] = image_paths
    pokedex = pokedex[pokedex["image_path"].notna()].reset_index(drop=True)

    # only keep pokemon with distinct id
    ids, id_counts = np.unique(pokedex["ID"], return_counts=True)
    ids, id_counts = np.array(ids), np.array(id_counts)
    keep_ids = ids[id_counts == 1]

    pokedex = pokedex[pokedex["ID"].isin(keep_ids)].reset_index(drop=True)
    pokedex["Type2"] = pokedex["Type2"].str.strip()
    if type_to_load is not None:
        pokedex = pokedex[pokedex["Type1"].isin(type_to_load)].
 ↪reset_index(drop=True)
    return pokedex

# load clip model
def load_clip_model():
    device = "cuda" if torch.cuda.is_available() else "cpu"
    model, preprocess = clip.load("ViT-L/14", device=device)
    return model, preprocess, device

# inference clip model on a list of image path
def clip_inference_image(model, preprocess, image_paths, device):
    image_embeddings = []
    with torch.no_grad():
        for img_path in tqdm(image_paths):
            img = Image.open(img_path)
            img_preprocessed = preprocess(img).unsqueeze(0).to(device)
            image_embedding = model.encode_image(img_preprocessed).detach().
 ↪cpu().numpy()
            image_embeddings += [image_embedding]

    image_embeddings = np.concatenate(image_embeddings, axis=0)
    image_embeddings /= np.linalg.norm(image_embeddings, axis=-1, keepdims=True)
    return image_embeddings

# inference clip model on a list of texts
def clip_inference_text(model, preprocess, texts, device):
    with torch.no_grad():
        text_embeddings = model.encode_text(clip.tokenize(texts).to(device)).
 ↪detach().cpu().numpy()
    text_embeddings /= np.linalg.norm(text_embeddings, axis=-1, keepdims=True)
    return text_embeddings
```

```
# compute similarity of texts to each image
def compute_similarity_text_to_image(image_embeddings, text_embeddings):
    similarity = softmax((100.0 * image_embeddings @ text_embeddings.T),␣
 ↪axis=-1)
    return similarity

# compute similarity of iamges to each text
def compute_similarity_image_to_text(image_embeddings, text_embeddings):
    similarity = softmax((100.0 * image_embeddings @ text_embeddings.T), axis=0)
    return similarity

# Use TSNE to project CLIP embeddings to 2D space
def umap_projection(image_embeddings, n_neâighbors=15, min_dist=0.1,␣
 ↪metric='cosine'):
    distance_matrix = np.zeros((image_embeddings.shape[0], image_embeddings.
 ↪shape[0]))
    for i in range(image_embeddings.shape[0]):
        for j in range(image_embeddings.shape[0]):
            if i == j:
                distance_matrix[i, j] = 1
            else:
                distance_matrix[i, j] = np.dot(image_embeddings[i],␣
 ↪image_embeddings[j])
    distance_matrix = 1 - distance_matrix
    reducer = TSNE(n_components=2, metric="precomputed", init="random",␣
 ↪random_state=42)
    visualization_data = reducer.fit_transform(distance_matrix)
    return visualization_data
```

[4]:
```
# Part 3 dataset download

! pip install -q kaggle
import kagglehub
path = kagglehub.dataset_download("hlrhegemony/pokemon-image-dataset")
print("Path to dataset files:", path)
!wget -O Pokemon.csv https://raw.githubusercontent.com/lgreski/pokemonData/
 ↪master/Pokemon.csv
```

```
Using Colab cache for faster access to the 'pokemon-image-dataset' dataset.
Path to dataset files: /kaggle/input/pokemon-image-dataset
--2026-02-11 23:11:01--
https://raw.githubusercontent.com/lgreski/pokemonData/master/Pokemon.csv
Resolving raw.githubusercontent.com (raw.githubusercontent.com)…
185.199.108.133, 185.199.109.133, 185.199.110.133, …
Connecting to raw.githubusercontent.com
(raw.githubusercontent.com)|185.199.108.133|:443… connected.
HTTP request sent, awaiting response… 200 OK
```

```
Length: 74060 (72K) [text/plain]
Saving to: ‘Pokemon.csv’

Pokemon.csv         100%[===================>]  72.32K  --.-KB/s    in 0.001s

2026-02-11 23:11:02 (51.3 MB/s) - ‘Pokemon.csv’ saved [74060/74060]
```

[5]:
```python
# Part 3 Dataset read
IMAGE_ROOT = "/kaggle/input/pokemon-image-dataset/images"

pokedex = construct_pokedex(
    csv_path="Pokemon.csv",
    image_dir=IMAGE_ROOT, # path of download, should be modified, see output of␣
 ↪Path to dataset files
    type_to_load=None
)

# ensure the structure of the datasets
print("pokedex shape:", pokedex.shape)
print(pokedex[["Name","Type1","Type2","image_path"]].head())
print("Type1 counts top10:\n", pokedex["Type1"].value_counts().head(10))
```

```
pokedex shape: (754, 14)
         Name  Type1   Type2  \
0   Bulbasaur  Grass  Poison
1     Ivysaur  Grass  Poison
2  Charmander   Fire
3  Charmeleon   Fire
4    Squirtle  Water

                                           image_path
0  /kaggle/input/pokemon-image-dataset/images/Bul…
1  /kaggle/input/pokemon-image-dataset/images/Ivy…
2  /kaggle/input/pokemon-image-dataset/images/Cha…
3  /kaggle/input/pokemon-image-dataset/images/Cha…
4  /kaggle/input/pokemon-image-dataset/images/Squ…
Type1 counts top10:
 Type1
Water       106
Normal       92
Grass        79
Bug          69
Psychic      46
Fire         44
Rock         41
Electric     38
Fighting     32
```

```
Dark         30
Name: count, dtype: int64
```

[6]:
```python
import os
# load model and retrieve image embeds
model, preprocess, device = load_clip_model()

CACHE_PATH = "clip_image_embeds_vitl14.npy"
image_paths = pokedex["image_path"].tolist()

if os.path.exists(CACHE_PATH):
    image_embeds = np.load(CACHE_PATH)
    print("Loaded cached embeddings:", image_embeds.shape)
else:
    image_embeds = clip_inference_image(model, preprocess, image_paths, device)
    np.save(CACHE_PATH, image_embeds)
    print("Saved embeddings:", image_embeds.shape)
```

```
Loaded cached embeddings: (754, 768)
```

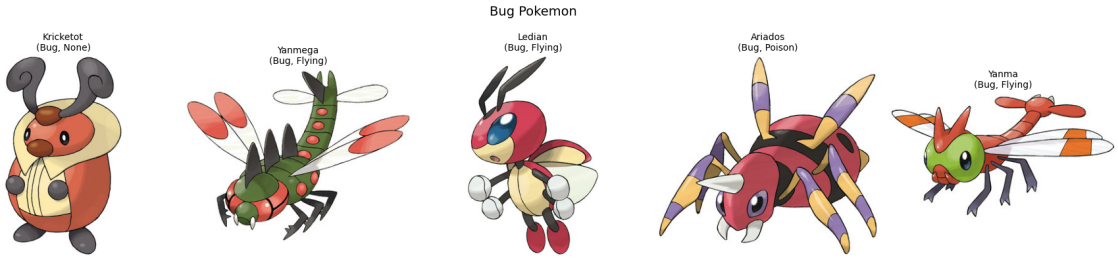
Question 20

[7]:
```python
# By default retrieve the top 5 images according to the text embedding
def retrieve(image_embeds, text_embed, k=5):
    text_embed = np.atleast_2d(text_embed)
    sims = compute_similarity_image_to_text(image_embeddings=image_embeds,␣
 ↪text_embeddings=text_embed)
    scores = sims[:, 0]
    top_idx = np.argsort(-scores)[:k] # score descending order, top k
    return top_idx, scores[top_idx]


def plot_top5(pokedex, top_idx, query):
    fig, axes = plt.subplots(1, 5, figsize=(18, 4))
    fig.suptitle(query, fontsize=14)

    for ax, idx in zip(axes, top_idx):
        row = pokedex.iloc[idx]
        img = Image.open(row["image_path"]).convert("RGB")
        ax.imshow(img)
        t2 = row["Type2"] if row["Type2"] != "" else "None"
        ax.set_title(f'{row["Name"]}\n({row["Type1"]}, {t2})', fontsize=10)
        ax.axis("off")

    plt.tight_layout()
    plt.show()


def hit_rate_for_type(pokedex, idx_list, poke_type):
```

```
    sub = pokedex.iloc[idx_list]
    return ((sub["Type1"] == poke_type) | (sub["Type2"] == poke_type)).mean()

def evaluate_templates(pokedex, image_embeds, poke_types=("Bug", "Fire",␣
 ↪"Grass"), topn=20):
    templates = [
        "A photo of {t} type Pokemon",
        "{t} type Pokemon",
        "A Pokemon of type {t}",
        "{t}",
        "{t} Pokemon"
    ]

    # template -> list of hit rate at k over types
    template_scores = {temp: [] for temp in templates}
    per_type_detail = {temp: {} for temp in templates}

    for t in poke_types:
        for temp in templates:
            query = temp.format(t=t)

            text_embed = clip_inference_text(
                model=model,
                preprocess=preprocess,
                texts=[query],
                device=device,
            )[0]  # (D,)

            top_idx, scores = retrieve(
                image_embeds=image_embeds,
                text_embed=text_embed,
                k=topn,
            )

            hit = hit_rate_for_type(pokedex, top_idx, t)
            template_scores[temp].append(hit)
            per_type_detail[temp][t] = hit

    # select best template with best rit rate
    best_template = max(
        templates,
        key=lambda temp: np.mean(template_scores[temp]),
    )
    best_avg_hit = float(np.mean(template_scores[best_template]))
    per_type_hits = per_type_detail[best_template]

    return best_template, best_avg_hit, per_type_hits, template_scores
```

```python
def retrieve_and_plot(best_template, poke_types):
    results = {}
    for t in poke_types:
        query = best_template.format(t=t)
        text_embed = clip_inference_text(model=model, preprocess=preprocess,␣
 ↪texts=[query], device=device)[0]
        top_idx, scores = retrieve(image_embeds=image_embeds,␣
 ↪text_embed=text_embed, k=5)
        plot_top5(pokedex=pokedex, top_idx=top_idx, query=query)
        hit = hit_rate_for_type(pokedex=pokedex, idx_list=top_idx, poke_type=t)
        print(f"Type {t} hit rate at 5: {hit}")
        results[t] = {
            "query": query,
            "top_idx": top_idx,
            "scores": scores,
            "hit@k": float(hit),
        }
    return results

# Template evaluation
best_template, best_avg, per_type_hits, _ = evaluate_templates(
    pokedex,
    image_embeds,
    poke_types=("Bug", "Fire", "Grass"),
    topn=20,
)

print(f"Selected template: {best_template}")
print(f"Average hit rate: {best_avg}")
print(f"Per-type hit rate: {per_type_hits}")

# Retrieve and analyze for the required types
retrieve_and_plot(best_template=best_template, poke_types=("Bug", "Fire",␣
 ↪"Grass"))
retrieve_and_plot(best_template=best_template, poke_types=("Dark", "Dragon"))
```

```
Selected template: {t} Pokemon
Average hit rate: 0.9666666666666667
Per-type hit rate: {'Bug': np.float64(0.95), 'Fire': np.float64(1.0), 'Grass':
np.float64(0.95)}
```



Type Bug hit rate at 5: 1.0

Type Fire hit rate at 5: 1.0

Type Grass hit rate at 5: 1.0

Type Dark hit rate at 5: 0.2

```
Type Dragon hit rate at 5: 0.6
```

[7]:
```
{'Dark': {'query': 'Dark Pokemon',
  'top_idx': array([666,  59, 499, 391, 687]),
  'scores': array([0.07745, 0.047  , 0.03897, 0.03836, 0.0349 ], dtype=float16),
  'hit@k': 0.2},
 'Dragon': {'query': 'Dragon Pokemon',
  'top_idx': array([523, 191, 100, 167, 705]),
  'scores': array([0.0761 , 0.0651 , 0.0641 , 0.06116, 0.05835], dtype=float16),
  'hit@k': 0.6}}
```

Answer Q20: - Compared with Fire, Grass and Bug, Dark and Dragon have significantly lower top 5 hit rate. This is probably because of the mismatch between appearnace and type definition: Many Pokémon has dragon-like or dark-looking visual characteristics but are not classified as Dragon or Dark type officially (Grovyle, Chandelure). CLIP tends to retrieve Pokemon based on their visual and semantic resemblance instead of the type labels, leading to the lower top 5 hit rate than Bug, Fire and Grass.

Question 21

[ ]:
```python
def retreive_types(text_embeds, image_embed, k=5):
    image_embed = np.atleast_2d(image_embed)
    sims = compute_similarity_text_to_image(image_embeddings=image_embed,␣
 ↪text_embeddings=text_embeds)
    scores = sims[0, :]
    top_idx = np.argsort(-scores)[:k]
    return top_idx, scores[top_idx]

def plot_top5_type(pokedex, idx, pred_types, pred_scores):
    row = pokedex.iloc[idx]
    img = Image.open(row["image_path"]).convert("RGB")

    t2 = row["Type2"]
    if pd.isna(t2) or t2 == "" or t2 == " ":
        t2 = "None"

    title = f'{row["Name"]} ({row["Type1"]}, {t2})'
```

```
    plt.figure(figsize=(4,4))
    plt.imshow(img)
    plt.axis("off")
    plt.title(title, fontsize=11)
    plt.show()

    print("Top5 predicted types:")
    for t, s in zip(pred_types, pred_scores):
        print(f"{t} sim={float(s)}")

# Randomly choose 10 Pokemons
np.random.seed(0)
rand_idx = np.random.choice(len(pokedex), size=10, replace=False) #choose ten␣
 ↪different Pokemon pics

# Prepare the queries, template is xxx Pokemon
types = (pd.concat([pokedex["Type1"], pokedex["Type2"]]).replace("", np.nan).
 ↪dropna().unique()) # Need to rule out the impact of None
type_queries = [best_template.format(t=t) for t in types]
type_text_embeds = clip_inference_text(model=model, preprocess=preprocess,␣
 ↪texts=type_queries, device=device) # (num_types, D)

for i in rand_idx:
    top_idx, scores = retreive_types(text_embeds=type_text_embeds,␣
 ↪image_embed=image_embeds[i], k=5)
    pred_types = [types[j] for j in top_idx]
    plot_top5_type(pokedex=pokedex, idx=i, pred_types=pred_types,␣
 ↪pred_scores=scores)
```

Hakamo-o (Dragon, Fighting)

```
Top5 predicted types:
Normal sim=0.271728515625
Rock sim=0.21484375
Dragon sim=0.208251953125
Ice sim=0.10308837890625
Steel sim=0.0968017578125
```

Barraskewda (Water, None)

```
Top5 predicted types:
Normal sim=0.474609375
Water sim=0.311279296875
Dark sim=0.05938720703125
Rock sim=0.0462646484375
Steel sim=0.027618408203125
```

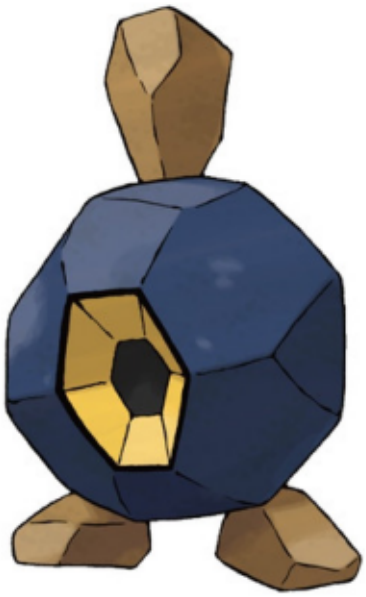
Lillipup (Normal, None)

```
Top5 predicted types:
Normal sim=0.423095703125
Dark sim=0.1331787109375
Fairy sim=0.1290283203125
Rock sim=0.08868408203125
Ice sim=0.06292724609375
```

Seaking (Water, None)

```
Top5 predicted types:
Normal sim=0.4775390625
Water sim=0.313232421875
Fairy sim=0.08428955078125
Rock sim=0.03515625
Bug sim=0.0203399658203125
```

Roggenrola (Rock, None)



```
Top5 predicted types:
Rock sim=0.6669921875
Normal sim=0.1253662109375
Dark sim=0.10723876953125
Ice sim=0.0293121337890625
Steel sim=0.0189208984375
```

Poliwrath (Water, Fighting)

```
Top5 predicted types:
Poison sim=0.40185546875
Water sim=0.2052001953125
Normal sim=0.13671875
Psychic sim=0.06561279296875
Rock sim=0.045074462890625
```

Krokorok (Ground, Dark)

```
Top5 predicted types:
Rock sim=0.65966796875
Normal sim=0.15185546875
```

```
Dark sim=0.054168701171875
Dragon sim=0.04150390625
Ice sim=0.01503753662109375
```

Oddish (Grass, Poison)

```
Top5 predicted types:
Poison sim=0.3779296875
Grass sim=0.3720703125
Normal sim=0.1170654296875
Fairy sim=0.052764892578125
Water sim=0.0226898193359375
```

Yamper (Electric, None)

```
Top5 predicted types:
Normal sim=0.56689453125
Electric sim=0.259521484375
Dark sim=0.037384033203125
Ice sim=0.032989501953125
Fairy sim=0.02490234375
```

Phanpy (Ground, None)

```
Top5 predicted types:
Normal sim=0.66015625
Ice sim=0.1749267578125
Water sim=0.0202484130859375
Bug sim=0.0178680419921875
Grass sim=0.0175933837890625
```

Question 22

[9]:
```python
def get_acc_hit(pokedex, type_text_embeds, image_embeds, k=5):
    correct_top1 = 0
    correct_top5 = 0
    all_top5 = {}
    num_pokes = len(pokedex)
    for i in range(num_pokes):
        top_idx, scores = retreive_types(text_embeds=type_text_embeds,␣
 ↪image_embed=image_embeds[i], k=k)
        pred_types = [types[j] for j in top_idx]
        all_top5[i] = pred_types
        type1 = pokedex.iloc[i]["Type1"]

        if type1 == pred_types[0]:
            correct_top1 += 1
        if type1 in pred_types:
            correct_top5 += 1

    acc1 = correct_top1 / num_pokes
    hit5 = correct_top5 / num_pokes

    print(f"Acc@1: {acc1}")
    print(f"Hit@5: {hit5}")

    return acc1, hit5, all_top5

acc1, hit5, _ = get_acc_hit(pokedex, type_text_embeds, image_embeds, k=5)
```

```
Acc@1: 0.3753315649867374
Hit@5: 0.7360742705570292
```

Answer 22 - Acc@1: 0.3753315649867374 - Hit@5: 0.7360742705570292

Question 23

[ ]:
```python
from huggingface_hub import login
HF_TOKEN = "ENTER YOUR TOKEN HERE"
login(HF_TOKEN) #Please enter the hugging face token above here to ensure the␣
 ↪subsequent model usage
```

[ ]:
```
# VLM setup
!pip install -U git+https://github.com/huggingface/transformers
!pip install -q transformers accelerate qwen-vl-utils


import torch
from transformers import AutoProcessor, Qwen3VLForConditionalGeneration
from PIL import Image


VLM_ID = "Qwen/Qwen3-VL-2B-Instruct"

vlm = Qwen3VLForConditionalGeneration.from_pretrained(
    VLM_ID,
    dtype="auto",
    device_map="auto",
)

processor = AutoProcessor.from_pretrained(VLM_ID)
```

```
Collecting git+https://github.com/huggingface/transformers
  Cloning https://github.com/huggingface/transformers to /tmp/pip-req-build-
gka96_x4
  Running command git clone --filter=blob:none --quiet
https://github.com/huggingface/transformers /tmp/pip-req-build-gka96_x4
  Resolved https://github.com/huggingface/transformers to commit
ae05b2ae619aa28fdfdcb8244009d585b7e1fed7
  Installing build dependencies … done
  Getting requirements to build wheel … done
  Preparing metadata (pyproject.toml) … done
Requirement already satisfied: huggingface-hub<2.0,>=1.3.0 in
/usr/local/lib/python3.12/dist-packages (from transformers==5.2.0.dev0) (1.4.1)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.12/dist-
packages (from transformers==5.2.0.dev0) (2.0.2)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.12/dist-packages (from transformers==5.2.0.dev0) (26.0)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.12/dist-
packages (from transformers==5.2.0.dev0) (6.0.3)
Requirement already satisfied: regex!=2019.12.17 in
/usr/local/lib/python3.12/dist-packages (from transformers==5.2.0.dev0)
(2025.11.3)
Requirement already satisfied: tokenizers<=0.23.0,>=0.22.0 in
/usr/local/lib/python3.12/dist-packages (from transformers==5.2.0.dev0) (0.22.2)
Requirement already satisfied: typer-slim in /usr/local/lib/python3.12/dist-
packages (from transformers==5.2.0.dev0) (0.21.1)
Requirement already satisfied: safetensors>=0.4.3 in
/usr/local/lib/python3.12/dist-packages (from transformers==5.2.0.dev0) (0.7.0)
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.12/dist-
```

```
packages (from transformers==5.2.0.dev0) (4.67.3)
Requirement already satisfied: filelock in /usr/local/lib/python3.12/dist-
packages (from huggingface-hub<2.0,>=1.3.0->transformers==5.2.0.dev0) (3.20.3)
Requirement already satisfied: fsspec>=2023.5.0 in
/usr/local/lib/python3.12/dist-packages (from huggingface-
hub<2.0,>=1.3.0->transformers==5.2.0.dev0) (2025.3.0)
Requirement already satisfied: hf-xet<2.0.0,>=1.2.0 in
/usr/local/lib/python3.12/dist-packages (from huggingface-
hub<2.0,>=1.3.0->transformers==5.2.0.dev0) (1.2.0)
Requirement already satisfied: httpx<1,>=0.23.0 in
/usr/local/lib/python3.12/dist-packages (from huggingface-
hub<2.0,>=1.3.0->transformers==5.2.0.dev0) (0.28.1)
Requirement already satisfied: shellingham in /usr/local/lib/python3.12/dist-
packages (from huggingface-hub<2.0,>=1.3.0->transformers==5.2.0.dev0) (1.5.4)
Requirement already satisfied: typing-extensions>=4.1.0 in
/usr/local/lib/python3.12/dist-packages (from huggingface-
hub<2.0,>=1.3.0->transformers==5.2.0.dev0) (4.15.0)
Requirement already satisfied: click>=8.0.0 in /usr/local/lib/python3.12/dist-
packages (from typer-slim->transformers==5.2.0.dev0) (8.3.1)
Requirement already satisfied: anyio in /usr/local/lib/python3.12/dist-packages
(from httpx<1,>=0.23.0->huggingface-hub<2.0,>=1.3.0->transformers==5.2.0.dev0)
(4.12.1)
Requirement already satisfied: certifi in /usr/local/lib/python3.12/dist-
packages (from httpx<1,>=0.23.0->huggingface-
hub<2.0,>=1.3.0->transformers==5.2.0.dev0) (2026.1.4)
Requirement already satisfied: httpcore==1.* in /usr/local/lib/python3.12/dist-
packages (from httpx<1,>=0.23.0->huggingface-
hub<2.0,>=1.3.0->transformers==5.2.0.dev0) (1.0.9)
Requirement already satisfied: idna in /usr/local/lib/python3.12/dist-packages
(from httpx<1,>=0.23.0->huggingface-hub<2.0,>=1.3.0->transformers==5.2.0.dev0)
(3.11)
Requirement already satisfied: h11>=0.16 in /usr/local/lib/python3.12/dist-
packages (from httpcore==1.*->httpx<1,>=0.23.0->huggingface-
hub<2.0,>=1.3.0->transformers==5.2.0.dev0) (0.16.0)
```

```
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:104:
UserWarning:
Error while fetching `HF_TOKEN` secret value from your vault: 'Requesting secret
HF_TOKEN timed out. Secrets can only be fetched when running from the Colab
UI.'.
You are not authenticated with the Hugging Face Hub in this notebook.
If the error persists, please let us know by opening an issue on GitHub
(https://github.com/huggingface/huggingface_hub/issues/new).
  warnings.warn(
```

```
Loading weights:   0%|          | 0/625 [00:00<?, ?it/s]
```

```
generation_config.json:   0%|          | 0.00/269 [00:00<?, ?B/s]
```

```
preprocessor_config.json:   0%|          | 0.00/390 [00:00<?, ?B/s]
```

```
chat_template.json: 0.00B [00:00, ?B/s]
tokenizer_config.json: 0.00B [00:00, ?B/s]
vocab.json: 0.00B [00:00, ?B/s]
merges.txt: 0.00B [00:00, ?B/s]
tokenizer.json: 0.00B [00:00, ?B/s]
video_preprocessor_config.json:   0%|          | 0.00/385 [00:00<?, ?B/s]
```

[ ]:
```python
import json, re

def build_prompt(candidates):
    return (
        "This is the image of a Pokemon. \n"
        "You are responsible for choosing single most likely PRIMARY Pokemon␣
 ↪type from the candidate list. \n"
        f"Candidates: {candidates}\n\n"
        "You should only return your choice as JSON with no additional␣
 ↪explanation, following the format provides subsequently. \n"
        "{\"type1\": \"<one of the candidates>\"}"
    )


@torch.no_grad()
def vlm_infer_func(image_path, prompt, max_new_tokens=64):
    image = Image.open(image_path).convert("RGB")

    messages = [{
        "role": "user",
        "content": [
            {"type": "image", "image": image},
            {"type": "text", "text": prompt},
        ],
    }]

    inputs = processor.apply_chat_template(
        messages,
        tokenize=True,
        add_generation_prompt=True,
        return_dict=True,
        return_tensors="pt"
    )
    inputs = {k: v.to(vlm.device) for k, v in inputs.items()}

    output_ids = vlm.generate(
        **inputs,
        max_new_tokens=max_new_tokens,
```

```
        do_sample=False,
        num_beams=1
    )

    in_ids_batch = inputs["input_ids"]
    output_ids_trimmed = [
        out_ids[len(in_ids):] for in_ids, out_ids in zip(in_ids_batch,␣
 ↪output_ids)
    ]

    text = processor.batch_decode(
        output_ids_trimmed,
        skip_special_tokens=True,
        clean_up_tokenization_spaces=False
    )[0]
    return text


def extract_type1(text, debug=False):
    obj = safe_json(text)
    if isinstance(obj, dict) and "type1" in obj:
        return obj["type1"]
    if debug:
        print("Returned non-json object:", text)
    return None


def safe_json(text): # prevent wrong return type from VLM
    if not isinstance(text, str):
        return {"raw": text}

    t = text.strip()
    start = t.find("{")
    end = t.rfind("}")

    if start == -1 or end == -1:
        return {"raw": text}

    t = t[start:end+1]
    # fix trailing commas
    t = re.sub(r",\s*}", "}", t)
    t = re.sub(r",\s*]", "]", t)

    try:
        return json.loads(t)
    except:
        return {"raw": text}
```

```python
from tqdm import tqdm

def run_q23_rerank(pokedex, clip_top5, limit=None, max_new_tokens=64,␣
↪verbose=0):
    num_pokes = len(pokedex) if limit is None else min(limit, len(pokedex))
    correct = 0
    invalid = 0
    preds = []

    for i in tqdm(range(num_pokes), desc="Q23 VLM reranking"):
        candidates = clip_top5[i]
        clip_top1 = candidates[0]
        img_path = pokedex.iloc[i]["image_path"]

        prompt = build_prompt(candidates)
        raw = vlm_infer_func(img_path, prompt, max_new_tokens=max_new_tokens)
        chosen = extract_type1(raw)

        # must choose from candidates, else fallback
        if chosen not in candidates:
            invalid += 1
            final = clip_top1 #fall back
        else:
            final = chosen

        preds.append(final)

        type1 = pokedex.iloc[i]["Type1"]
        if final == type1:
            correct += 1

        if verbose and i < verbose:
            print("----")
            print("i:", i)
            print("Name:", os.path.basename(img_path))
            print("Ground Truth (Type1):", type1)
            print("Candidates:", candidates)
            print("Raw:", raw)
            print("Chosen:", chosen)
            print("Final:", final)

    rerank_acc1 = correct / num_pokes
    return rerank_acc1, preds, invalid

clip_acc1, clip_hit5, clip_top5 = get_acc_hit(pokedex, type_text_embeds,␣
↪image_embeds, k=5)
```

```
rerank_acc1, preds, invalid_cnt = run_q23_rerank(
    pokedex, clip_top5, limit=None, max_new_tokens=64, verbose=0
)

print("CLIP Acc@1:", clip_acc1)
print("CLIP Hit@5:", clip_hit5)
print("VLM reranked Acc@1:", rerank_acc1)
print("Invalid / fallback count:", invalid_cnt, "out of", len(pokedex))
```

```
Acc@1: 0.3753315649867374
Hit@5: 0.7360742705570292

Q23 VLM reranking: 100%|      | 754/754 [53:33<00:00,  4.26s/it]

CLIP Acc@1: 0.3753315649867374
CLIP Hit@5: 0.7360742705570292
VLM reranked Acc@1: 0.45225464190981435
Invalid / fallback count: 0 out of 754
```

[ ]:
```
results = pd.DataFrame({
    "Metric": ["CLIP Acc@1", "CLIP Hit@5", "VLM Acc@1"],
    "Value": [clip_acc1, clip_hit5, rerank_acc1]
})
results
```

[ ]:

	Metric	Value
0	CLIP Acc@1	0.375332
1	CLIP Hit@5	0.736074
2	VLM Acc@1	0.452255

Answer Q23: - **Metrics report**: - CLIP Acc@1: 0.3753315649867374 - CLIP Hit@5: 0.7360742705570292 - VLM reranked Acc@1: 0.45225464190981435

- **Analysis:**
  - Reranking by VLM does help in this setting: VLM reranking improves Acc@1 from 0.3753 to 0.4523, which has a relative improvement of approximately 20.5% over CLIP's top-1 accuracy. CLIP is a dual-encoder model trained with a contrastive objective and relies solely on embedding similarity between independently encoded image and text features. As a result, it can often retrieve the correct primary type within top 5 candidates but lacks the reasoning required to reliably rank it as top 1. In contrast, a VLM tokenizes the image into visual tokens and jointly attends over image and text tokens within an LLM, enabling a reasoning and instruction-based decision. Therefore, VLM can better resolve ambiguities among visually similar types.