ECE 219 Project 2 — Part 2

Part 2: Deep Learning & Image Clustering (Q13–Q19)

Part 1 Task 3: Held-Out Steam Game Profiling (Q9–Q12)

```
import importlib
import subprocess
import sys

def ensure_package(import_name, pip_name=None):
    """Install a package into the active kernel env only if import is
missing."""
    try:
        importlib.import_module(import_name)
        return
    except ModuleNotFoundError:
        pkg = pip_name or import_name
        print(f'Installing missing dependency: {pkg} ...')
        subprocess.check_call([sys.executable, '-m', 'pip', 'install',
'-q', pkg])

# Notebook dependencies used later in Q9-Q12 and Q18.
ensure_package('umap', 'umap-learn')
ensure_package('sentence_transformers', 'sentence-transformers')
ensure_package('transformers', 'transformers')

import os
import tarfile
import requests
import warnings
warnings.filterwarnings('ignore')

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib
matplotlib.rcParams['figure.dpi'] = 120

import torch
import torch.nn as nn
from torch.utils.data import DataLoader, TensorDataset
from torchvision import transforms, datasets
from tqdm import tqdm

from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.decomposition import PCA, TruncatedSVD
```

```
from sklearn.manifold import TSNE
from sklearn.cluster import KMeans, AgglomerativeClustering
try:
    # sklearn >= 1.3
    from sklearn.cluster import HDBSCAN
except Exception:
    # fallback for older sklearn environments
    ensure_package('hdbscan')
    from hdbscan import HDBSCAN
from sklearn.neighbors import kneighbors_graph
from sklearn.metrics import (
    homogeneity_score, completeness_score, v_measure_score,
    adjusted_rand_score, adjusted_mutual_info_score
)
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer

import umap
if torch.cuda.is_available():
    DEVICE = torch.device('cuda')
elif torch.backends.mps.is_available():
    DEVICE = torch.device('mps')
else:
    DEVICE = torch.device('cpu')

print(f'Using device: {DEVICE}')


Using device: cuda
```

PART 2 — Deep Learning and Clustering of Image Data

Q13 — Transfer Learning: Why VGG Features Generalize

Although VGG-16 is trained on ImageNet's 1,000 categories, its internal layers learn broadly useful visual patterns rather than only class-specific rules. In the earlier layers, the network captures simple cues like edges, corners, gradients, and textures; in the middle layers, it combines these into larger structures such as curves and repeated patterns; and in deeper layers, it represents more complex part combinations that remain informative even for unseen categories. Because natural images share this underlying structure, these learned features transfer well across datasets. When flower images are passed through VGG-16 and represented by the 4,096-dimensional activations from the first fully connected layer, each image is encoded in a compact space that reflects visual properties like shape, petal arrangement, texture, and

color distribution. These are exactly the cues that separate flower types, so the features remain discriminative even without flower-specific training. This is the key idea of transfer learning: a model trained on a large and diverse source task can provide reusable feature representations for a new task, reducing the need to train a deep model from scratch.

Feature Extraction — VGG-16 on Flowers Dataset

```
class FeatureExtractor(nn.Module):
    """Extracts the 4096-dim activation from VGG-16's first FC
layer."""

    def __init__(self):
        super().__init__()
        vgg = torch.hub.load('pytorch/vision:v0.10.0', 'vgg16',
pretrained=True)
        self.features = vgg.features          # conv
        self.pooling  = vgg.avgpool            # adaptive avg pool 7×7
        self.flatten  = nn.Flatten()
        self.fc       = vgg.classifier[0]      # Linear(25088 -> 4096)

    def forward(self, x):
        out = self.features(x)
        out = self.pooling(out)
        out = self.flatten(out)
        out = self.fc(out)
        return out

FLOWERS_NPZ = './flowers_features_and_labels.npz'

if os.path.exists(FLOWERS_NPZ):
    print('Loading cached VGG features…')
    file  = np.load(FLOWERS_NPZ)
    f_all = file['f_all']
    y_all = file['y_all'].astype(int)
    class_names = list(file['class_names']) if 'class_names' in file
else\
                  ['daisy', 'dandelion', 'roses', 'sunflowers',
'tulips']
else:
    if not os.path.exists('./flower_photos'):
        url =
'http://download.tensorflow.org/example_images/flower_photos.tgz'
        print('Downloading flowers dataset…')
        with open('./flower_photos.tgz', 'wb') as fh:
            fh.write(requests.get(url).content)
        with tarfile.open('./flower_photos.tgz') as fh:
            fh.extractall('./')
        os.remove('./flower_photos.tgz')
        print('Download complete.')
```

```
    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]),
    ])
    dataset     = datasets.ImageFolder(root='./flower_photos',
transform=transform)
    class_names = dataset.classes
    dataloader  = DataLoader(dataset, batch_size=64, shuffle=False)

    print(f'Extracting VGG-16 features on {DEVICE}…')
    extractor = FeatureExtractor().to(DEVICE).eval()
    f_all, y_all = np.zeros((0, 4096)), np.zeros((0,), dtype=int)
    for x, y in tqdm(dataloader):
        with torch.no_grad():
            feats = extractor(x.to(DEVICE)).cpu().numpy()
        f_all = np.vstack([f_all, feats])
        y_all = np.concatenate([y_all, y.numpy()])

    np.savez(FLOWERS_NPZ, f_all=f_all, y_all=y_all,
             class_names=np.array(class_names))
    print('Features saved.')

NUM_FEATURES = f_all.shape[1]
N_CLASSES    = len(np.unique(y_all))
print(f'Feature matrix: {f_all.shape}  |  Classes: {N_CLASSES}  |
Labels: {class_names}')
```

```
Loading cached VGG features…
Feature matrix: (3670, 4096)  |  Classes: 5  |  Labels:
[np.str_('daisy'), np.str_('dandelion'), np.str_('roses'),
np.str_('sunflowers'), np.str_('tulips')]
```

Q14 — How the Helper Code Performs Feature Extraction

The helper code extracts features by first loading a pre-trained VGG-16 model with torch.hub.load, which includes convolutional blocks (vgg.features), average pooling (vgg.avgpool), and a classifier head (vgg.classifier). It then truncates the network by keeping only features, avgpool, Flatten, and the first fully connected layer (classifier[0], Linear(25088 → 4096)), while removing the remaining classifier layers and softmax so the output is a transferable feature representation rather than ImageNet class scores. Each flower image is preprocessed to match VGG's expected input by resizing to 224×224, converting to tensor format, and normalizing channels with ImageNet mean and standard deviation values. Finally, images are passed through the truncated model in batches of 64 under torch.no_grad(), and the resulting 4,096-dimensional vectors are concatenated into f_all with shape (N_images, 4096) and saved for reuse.

Q15 — Pixel Count vs. Feature Dimension

```
PROCESSED_H, PROCESSED_W, CHANNELS = 224, 224, 3
pixels_processed = PROCESSED_H * PROCESSED_W * CHANNELS

if os.path.exists('./flower_photos'):
    from PIL import Image
    sizes = []
    for root, dirs, files in os.walk('./flower_photos'):
        for fname in files:
            if fname.lower().endswith(('.jpg', '.jpeg')):
                img = Image.open(os.path.join(root, fname))
                sizes.append(img.size)  # (width, height)
                if len(sizes) >= 50:
                    break
        if len(sizes) >= 50:
            break
    widths  = [s[0] for s in sizes]
    heights = [s[1] for s in sizes]
    print(f'Sample of original image sizes (first 50):')
    print(f'  Width  — min: {min(widths)}, max: {max(widths)}, mean:
{np.mean(widths):.0f}')
    print(f'  Height — min: {min(heights)}, max: {max(heights)}, mean:
{np.mean(heights):.0f}')
    avg_orig_pixels = int(np.mean(widths) * np.mean(heights) * 3)
    print(f'  Approximate avg pixels (RGB) in originals:
{avg_orig_pixels:,}')

print(f'\nAfter pre-processing (224×224×3):')
print(f'  Pixels per image fed to VGG : {pixels_processed:,}
({PROCESSED_H}×{PROCESSED_W}×{CHANNELS})')
print(f'  VGG feature dimension       : {NUM_FEATURES:,}')
print(f'  Compression ratio           : {pixels_processed /
NUM_FEATURES:.1f}×')
```

```
Sample of original image sizes (first 50):
  Width  — min: 159, max: 500, mean: 361
  Height — min: 192, max: 375, mean: 265
  Approximate avg pixels (RGB) in originals: 287,471

After pre-processing (224×224×3):
  Pixels per image fed to VGG : 150,528  (224×224×3)
  VGG feature dimension       : 4,096
  Compression ratio           : 36.8×
```

Q16 — Are VGG Features Dense or Sparse?

```
zero_frac  = (f_all == 0).mean()
near0_frac = (np.abs(f_all) < 1e-3).mean()
```

```
print('=== VGG Feature Density Analysis ===')
print(f'Feature matrix shape : {f_all.shape}')
print(f'Exact zeros          : {zero_frac*100:.2f}%')
print(f'Near-zero (<1e-3)    : {near0_frac*100:.2f}%')
print()
```

```
=== VGG Feature Density Analysis ===
Feature matrix shape : (3670, 4096)
Exact zeros          : 0.00%
Near-zero (<1e-3)    : 0.03%
```

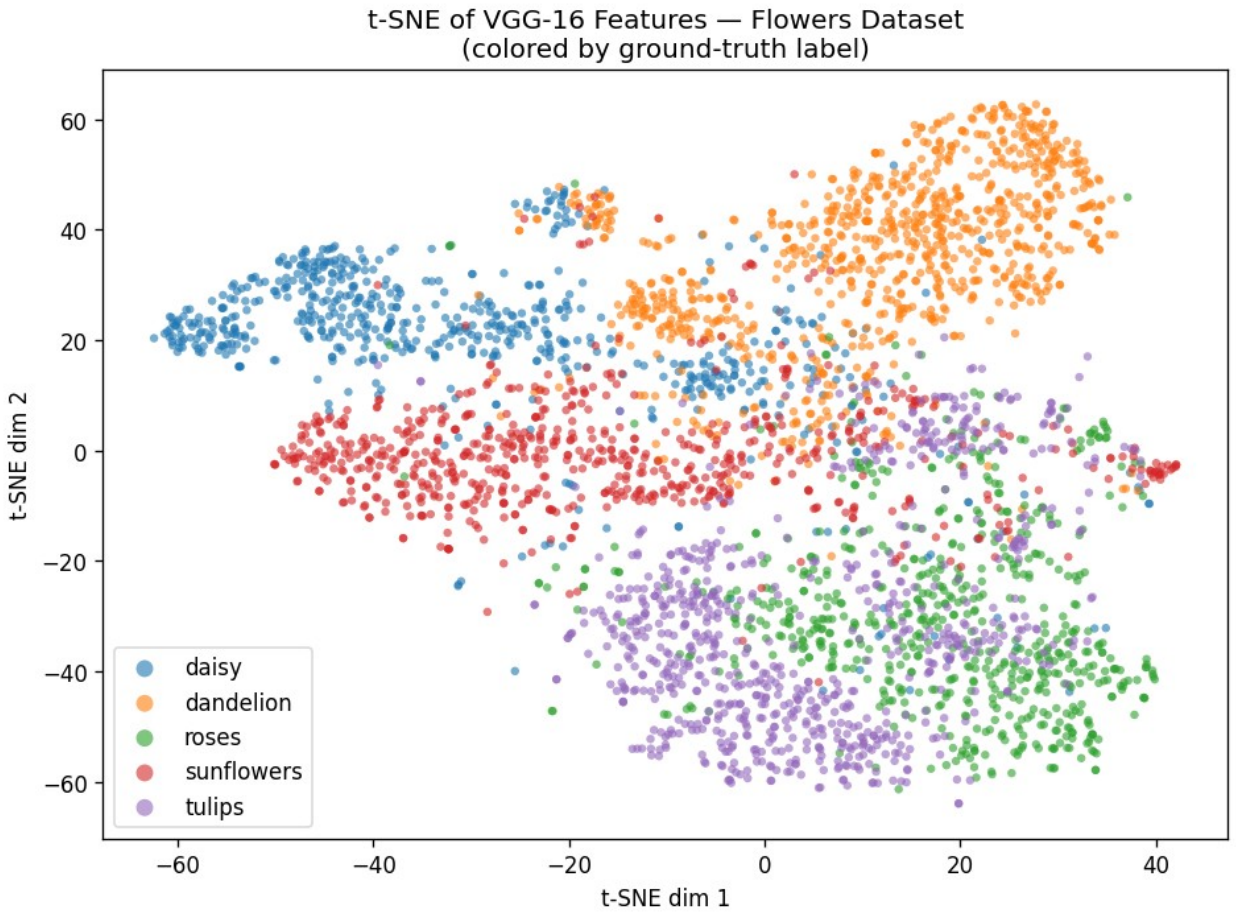
The VGG features are DENSE. Almost every entry is a non-zero real-valued activation produced by ReLU + fully-connected layers — unlike TF-IDF which is structurally sparse because most words simply do not appear in a given document.

Q17 — t-SNE Visualization of VGG Features

```
print('Pre-reducing to 50 dims...')
f_pca50 = PCA(n_components=50, random_state=42).fit_transform(
    StandardScaler().fit_transform(f_all)
)

print('Running t-SNE (this may take ~1-2 minutes)...')
tsne   = TSNE(n_components=2, perplexity=30, max_iter=1000,
              learning_rate='auto', init='pca', random_state=42)
f_tsne = tsne.fit_transform(f_pca50)

CMAP   = plt.get_cmap('tab10')
COLORS = [CMAP(i) for i in range(N_CLASSES)]

fig, ax = plt.subplots(figsize=(8, 6))
for cls_idx, cls_name in enumerate(class_names):
    mask = y_all == cls_idx
    ax.scatter(f_tsne[mask, 0], f_tsne[mask, 1],
               c=[COLORS[cls_idx]], label=cls_name,
               alpha=0.6, s=15, edgecolors='none')
ax.set_title('t-SNE of VGG-16 Features — Flowers Dataset\n(colored by
ground-truth label)')
ax.set_xlabel('t-SNE dim 1')
ax.set_ylabel('t-SNE dim 2')
ax.legend(markerscale=2, framealpha=0.7)
plt.tight_layout()
plt.savefig('q17_tsne_vgg_flowers.png', bbox_inches='tight')
plt.show()

centroids = np.vstack([f_tsne[y_all == i].mean(axis=0) for i in
range(N_CLASSES)])
within = np.mean(np.linalg.norm(f_tsne - centroids[y_all], axis=1))
pairwise = []
```

```
for i in range(N_CLASSES):
    for j in range(i + 1, N_CLASSES):
        pairwise.append((i, j, float(np.linalg.norm(centroids[i] -
centroids[j]))))
mean_between = float(np.mean([d for _, _, d in pairwise]))
closest_i, closest_j, closest_d = min(pairwise, key=lambda x: x[2])
sep_ratio = mean_between / max(within, 1e-12)

print('\nQ17 — Observation Summary')
print(f'  Mean within-class spread        : {within:.3f}')
print(f'  Mean between-centroid distance  : {mean_between:.3f}')
print(f'  Separation ratio (between/within): {sep_ratio:.3f}')
print(f'  Closest class-centroid pair     : {class_names[closest_i]}
vs {class_names[closest_j]} (dist={closest_d:.3f})')
if sep_ratio > 1.8:
    print('  Interpretation: clusters are generally well separated
with limited overlap.')
elif sep_ratio > 1.2:
    print('  Interpretation: moderate class separation with some
overlapping regions.')
else:
    print('  Interpretation: substantial overlap; labels are not
strongly separated in 2D t-SNE.')
```

```
Pre-reducing to 50 dims...
Running t-SNE (this may take ~1-2 minutes)...
```




```
Q17 — Observation Summary
  Mean within-class spread        : 20.666
  Mean between-centroid distance  : 46.885
  Separation ratio (between/within): 2.269
  Closest class-centroid pair     : roses vs tulips (dist=12.903)
  Interpretation: clusters are generally well separated with limited
overlap.
```

Q18 — Clustering Pipeline Table (best ARI)

```python
class Autoencoder(nn.Module):
    """Trains on 4096-dim VGG features and reduces to n_components
dims."""

    def __init__(self, n_components=50):
        super().__init__()
        self.n_components = n_components
        self.encoder = 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, n_components),
        )
        self.decoder = nn.Sequential(
            nn.Linear(n_components, 120), nn.ReLU(True),
            nn.Linear( 120,  640), nn.ReLU(True),
            nn.Linear( 640, 1280), nn.ReLU(True),
            nn.Linear(1280, 4096),
        )
        self.to(DEVICE)

    def forward(self, x):
        return self.decoder(self.encoder(x))

    def fit(self, X, epochs=100, batch_size=128, lr=1e-3):
        X_t   = torch.tensor(X, dtype=torch.float32)
        loader = DataLoader(TensorDataset(X_t), batch_size=batch_size,
shuffle=True)
        crit  = nn.MSELoss()
        optim = torch.optim.Adam(self.parameters(), lr=lr,
weight_decay=1e-5)
        self.train()
        for _ in tqdm(range(epochs), desc='Autoencoder'):
            for (xb,) in loader:
                xb = xb.to(DEVICE)
                optim.zero_grad()
                crit(self(xb), xb).backward()
                optim.step()
        return self

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

    def fit_transform(self, X, **kw):
        return self.fit(X, **kw).transform(X)

scaler = StandardScaler()
f_scaled = scaler.fit_transform(f_all)

DR_CACHE = {'None': f_scaled}
print('[1/4] No reduction (4096 dims, scaled)…')

print('[2/4] SVD r=50…')
DR_CACHE['SVD(50)'] = TruncatedSVD(n_components=50,
random_state=42).fit_transform(f_scaled)
```

```
print('[3/4] SVD(200) → UMAP(50)…')
f_svd200 = TruncatedSVD(n_components=200,
random_state=42).fit_transform(f_scaled)
DR_CACHE['UMAP(50)'] = umap.UMAP(
    n_components=50, n_neighbors=15, min_dist=0.1, random_state=42
).fit_transform(f_svd200)

print('[4/4] Autoencoder(50)…')
DR_CACHE['AE(50)'] =
Autoencoder(n_components=50).fit_transform(f_scaled)

print('All reductions done.')
```

```
[1/4] No reduction (4096 dims, scaled)…
[2/4] SVD r=50…
[3/4] SVD(200) → UMAP(50)…
[4/4] Autoencoder(50)…

Autoencoder: 100%|██████████| 100/100 [00:13<00:00,  7.45it/s]

All reductions done.
```

```
def eval_clustering(labels_true, labels_pred):
    """Returns clustering metrics; excludes HDBSCAN noise points (-
1)."""
    mask = labels_pred != -1
    lt, lp = labels_true[mask], labels_pred[mask]
    if len(np.unique(lp)) < 2:
        return dict(homogeneity=0, completeness=0, v_measure=0, ARI=0,
AMI=0,
                    n_clusters=1, noise_frac=1 - mask.mean())
    return dict(
        homogeneity=homogeneity_score(lt, lp),
        completeness=completeness_score(lt, lp),
        v_measure=v_measure_score(lt, lp),
        ARI=adjusted_rand_score(lt, lp),
        AMI=adjusted_mutual_info_score(lt, lp),
        n_clusters=len(np.unique(lp)),
        noise_frac=1 - mask.mean(),
    )


def run_kmeans(Z, k=5):
    return KMeans(n_clusters=k, n_init=10,
random_state=42).fit_predict(Z)


def run_agglomerative(Z, k=5, n_neighbors=10):
    conn = kneighbors_graph(Z, n_neighbors=n_neighbors,
mode='connectivity', include_self=False)
    return AgglomerativeClustering(n_clusters=k, linkage='ward',
```

```
connectivity=conn).fit_predict(Z)


HDBSCAN_FAST_MODE = False
HDBSCAN_GRID = (
    [(10, 5), (15, 5), (20, 10)]
    if HDBSCAN_FAST_MODE else
    [(mcs, ms) for mcs in [5, 10, 15, 20] for ms in [3, 5, 10]]
)


def run_best_hdbscan(Z, y_true):
    best, best_labels = {'ARI': -1}, None
    for mcs, ms in HDBSCAN_GRID:
        labels = HDBSCAN(min_cluster_size=mcs,
min_samples=ms).fit_predict(Z)
        m = eval_clustering(y_true, labels)
        if m['ARI'] > best['ARI']:
            best = m | {'min_cluster_size': mcs, 'min_samples': ms}
            best_labels = labels
        if best['ARI'] >= 0.98:
            break
    return best, best_labels


def add_result(rows, reduction, clustering, metrics, notes=''):
    rows.append({'Reduction': reduction, 'Clustering': clustering,
**metrics, 'notes': notes})


results = []
y_true = y_all

for dr_name, Z in DR_CACHE.items():
    print(f'\n--- {dr_name} ---')

    m = eval_clustering(y_true, run_kmeans(Z))
    add_result(results, dr_name, 'K-Means(5)', m)
    print(f'  K-Means   ARI={m["ARI"]:.3f}')

    m = eval_clustering(y_true, run_agglomerative(Z))
    add_result(results, dr_name, 'Agglom.(5)', m)
    print(f'  Agglom.   ARI={m["ARI"]:.3f}')

    m, _ = run_best_hdbscan(Z, y_true)
    note = f'mcs={m.get("min_cluster_size",
"?")},ms={m.get("min_samples", "?")}'
    add_result(results, dr_name, 'HDBSCAN', m, notes=note)
    print(f'  HDBSCAN   ARI={m["ARI"]:.3f}
(mcs={m.get("min_cluster_size")},ms={m.get("min_samples")})')
```

```
print('\nAll pipelines complete.')


--- None ---
  K-Means   ARI=0.191
  Agglom.   ARI=0.201
  HDBSCAN   ARI=0.015  (mcs=5,ms=5)

--- SVD(50) ---
  K-Means   ARI=0.192
  Agglom.   ARI=0.317
  HDBSCAN   ARI=0.018  (mcs=5,ms=5)

--- UMAP(50) ---
  K-Means   ARI=0.430
  Agglom.   ARI=0.356
  HDBSCAN   ARI=0.631  (mcs=15,ms=10)

--- AE(50) ---
  K-Means   ARI=0.306
  Agglom.   ARI=0.282
  HDBSCAN   ARI=0.180  (mcs=15,ms=3)

All pipelines complete.
```

```
results_df = pd.DataFrame(results)
display_cols = ['Reduction', 'Clustering', 'n_clusters', 'noise_frac',
                'homogeneity', 'completeness', 'v_measure', 'ARI',
'AMI', 'notes']
tbl = results_df[display_cols].copy()
float_cols = ['noise_frac', 'homogeneity', 'completeness',
'v_measure', 'ARI', 'AMI']
for c in float_cols:
    tbl[c] = pd.to_numeric(tbl[c], errors='coerce')
tbl = tbl.sort_values('ARI', ascending=False, na_position='last')

print('\nQ18 — Clustering Results (sorted by ARI descending)\n')
max_rows = 30
print(tbl.head(max_rows).to_string(index=False, float_format=lambda x:
f'{x:.3f}'))
if len(tbl) > max_rows:
    print(f'\n... {len(tbl) - max_rows} more rows omitted for speed.')

# Best pipeline
best_row = tbl.iloc[0]
print(f'\n★ Best pipeline: {best_row["Reduction"]} +
{best_row["Clustering"]}')
print(f'  ARI = {best_row["ARI"]:.4f}  |  V-measure =
{best_row["v_measure"]:.4f}')
```

```
Q18 — Clustering Results (sorted by ARI descending)

Reduction Clustering  n_clusters  noise_frac  homogeneity
completeness  v_measure   ARI   AMI        notes
 UMAP(50)    HDBSCAN          17       0.378        0.740
0.523      0.613 0.631 0.610 mcs=15,ms=10
 UMAP(50) K-Means(5)           5       0.000        0.494
0.507      0.500 0.430 0.500
 UMAP(50) Agglom.(5)           5       0.000        0.451
0.489      0.469 0.356 0.468
  SVD(50) Agglom.(5)           5       0.000        0.336
0.361      0.348 0.317 0.347
   AE(50) K-Means(5)           5       0.000        0.322
0.348      0.335 0.306 0.334
   AE(50) Agglom.(5)           5       0.000        0.317
0.342      0.329 0.282 0.328
     None Agglom.(5)           5       0.000        0.360
0.409      0.383 0.201 0.382
  SVD(50) K-Means(5)           5       0.000        0.331
0.367      0.348 0.192 0.347
     None K-Means(5)           5       0.000        0.331
0.367      0.348 0.191 0.347
   AE(50)    HDBSCAN           2       0.951        0.153
0.626      0.245 0.180 0.233  mcs=15,ms=3
  SVD(50)    HDBSCAN           3       0.762        0.037
0.361      0.067 0.018 0.061   mcs=5,ms=5
     None    HDBSCAN           3       0.829        0.027
0.342      0.050 0.015 0.041   mcs=5,ms=5

★ Best pipeline: UMAP(50) + HDBSCAN
  ARI = 0.6307  |  V-measure = 0.6130
```

Q19 — MLP Classifier on Original and Reduced VGG Features

```python
class MLP(nn.Module):
    def __init__(self, num_features, num_classes=5):
        super().__init__()
        self.model = nn.Sequential(
            nn.Linear(num_features, 1280),
            nn.ReLU(True),
            nn.Linear(1280, 640),
            nn.ReLU(True),
            nn.Linear(640, num_classes),
            nn.LogSoftmax(dim=1),
        )
        self.to(DEVICE)
```

```
    def forward(self, x):
        return self.model(x)

    def fit(self, X_train, y_train, epochs=100, batch_size=128):
        Xt = torch.tensor(X_train, dtype=torch.float32)
        yt = torch.tensor(y_train, dtype=torch.int64)
        loader = DataLoader(TensorDataset(Xt, yt),
                            batch_size=batch_size, shuffle=True)
        crit  = nn.NLLLoss()
        optim = torch.optim.Adam(self.parameters(), lr=1e-3,
weight_decay=1e-5)
        self.model.train()
        for _ in tqdm(range(epochs), desc='MLP training'):
            for xb, yb in loader:
                xb, yb = xb.to(DEVICE), yb.to(DEVICE)
                optim.zero_grad()
                crit(self(xb), yb).backward()
                optim.step()
        return self

    def score(self, X_test, y_test):
        self.model.eval()
        Xt = torch.tensor(X_test, dtype=torch.float32, device=DEVICE)
        yt = torch.tensor(y_test, dtype=torch.int64, device=DEVICE)
        with torch.no_grad():
            preds = self(Xt).argmax(dim=1)
        return (preds == yt).float().mean().item()

idx_tr, idx_te = train_test_split(
    np.arange(len(y_all)), test_size=0.2,
    stratify=y_all, random_state=42
)

mlp_results = {}
y_tr, y_te = y_all[idx_tr], y_all[idx_te]


def run_mlp_experiment(name, X_train, X_test, num_features):
    print(f'\nTraining MLP on {name} features…')
    model = MLP(num_features=num_features)
    model.fit(X_train, y_tr)
    acc = model.score(X_test, y_te)
    mlp_results[name] = acc
    print(f'  Test accuracy ({name}) : {acc:.4f}')


run_mlp_experiment('Original (4096)', f_scaled[idx_tr],
f_scaled[idx_te], num_features=4096)

for dr_name, Z in DR_CACHE.items():
```

```
    if dr_name == 'None':
        continue
    sc2 = StandardScaler()
    X_tr_z = sc2.fit_transform(Z[idx_tr])
    X_te_z = sc2.transform(Z[idx_te])
    run_mlp_experiment(dr_name, X_tr_z, X_te_z, num_features=50)

print('\n=== Q19 — MLP Test Accuracy Summary ===')
for k, v in mlp_results.items():
    print(f'  {k:<25} {v:.4f}')
```

```
Training MLP on Original (4096) features…

MLP training: 100%|██████████| 100/100 [00:04<00:00, 24.73it/s]

  Test accuracy (Original (4096)) : 0.9114

Training MLP on SVD(50) features…

MLP training: 100%|██████████| 100/100 [00:01<00:00, 51.61it/s]

  Test accuracy (SVD(50)) : 0.8992

Training MLP on UMAP(50) features…

MLP training: 100%|██████████| 100/100 [00:02<00:00, 43.35it/s]

  Test accuracy (UMAP(50)) : 0.8856

Training MLP on AE(50) features…

MLP training: 100%|██████████| 100/100 [00:02<00:00, 48.22it/s]

  Test accuracy (AE(50)) : 0.8924

=== Q19 — MLP Test Accuracy Summary ===
  Original (4096)           0.9114
  SVD(50)                   0.8992
  UMAP(50)                  0.8856
  AE(50)                    0.8924
```

```
print('### Q19 — Discussion')

orig = mlp_results.get('Original (4096)')
reduced = {k: v for k, v in mlp_results.items() if k != 'Original
(4096)'}
if orig is None or len(reduced) == 0:
    raise RuntimeError('Run q19-code first to populate mlp_results.')

best_reduced_name = max(reduced, key=reduced.get)
```

```
best_reduced_acc = reduced[best_reduced_name]
drop_best = orig - best_reduced_acc
drop_worst = orig - min(reduced.values())

print(f'Original (4096-dim) test accuracy: {orig:.4f}')
for name, acc in sorted(reduced.items(), key=lambda kv: kv[1],
reverse=True):
    print(f'  {name:<10} accuracy={acc:.4f}  drop_vs_original={orig -
acc:.4f}')

if drop_best < 0.03:
    sig_text = 'small'
elif drop_best < 0.07:
    sig_text = 'moderate'
else:
    sig_text = 'large'

print()
print('Interpretation:')
print(f'- The best reduced representation for MLP is
{best_reduced_name} at {best_reduced_acc:.4f}.')
print(f'- The performance loss versus original features is
{drop_best:.4f} ({sig_text}).')
print(f'- Worst reduced-case drop is {drop_worst:.4f}.')
print(f'- Q18 best ARI pipeline was {best_row["Reduction"]} +
{best_row["Clustering"]} (ARI={best_row["ARI"]:.4f}).')

# Compare clustering-best reduction with classification-best reduction
using actual outputs.
ari_best_reduction = str(best_row['Reduction'])
if ari_best_reduction in reduced:
    ari_best_acc = reduced[ari_best_reduction]
    gap = best_reduced_acc - ari_best_acc
    if abs(gap) < 0.02:
        print('- Clustering and classification are largely consistent
across reduced features (accuracy gap < 0.02).')
    elif gap > 0:
        print(f'- There is a mismatch: best-clustering reduction
({ari_best_reduction}) is {gap:.4f} below the best reduced MLP
accuracy.')
        print('- This can happen because ARI rewards unsupervised
cluster separation, while MLP exploits supervised decision
boundaries.')
    else:
        print('- Best-clustering reduction also has the strongest
reduced-feature MLP accuracy.')
else:
    print(f'- Note: Q18 winner reduction ({ari_best_reduction}) was
not evaluated in reduced-feature MLP results.')
```

```
### Q19 — Discussion
Original (4096-dim) test accuracy: 0.9114
  SVD(50)    accuracy=0.8992  drop_vs_original=0.0123
  AE(50)     accuracy=0.8924  drop_vs_original=0.0191
  UMAP(50)   accuracy=0.8856  drop_vs_original=0.0259

Interpretation:
- The best reduced representation for MLP is SVD(50) at 0.8992.
- The performance loss versus original features is 0.0123 (small).
- Worst reduced-case drop is 0.0259.
- Q18 best ARI pipeline was UMAP(50) + HDBSCAN (ARI=0.6307).
- Clustering and classification are largely consistent across reduced
features (accuracy gap < 0.02).
```

PART 1 — Task 3: Held-Out Steam Game Profiling (Q9–Q12)

```python
from pathlib import Path
import os

def resolve_steam_csv(file_names, explicit_path=None):
    cwd = Path.cwd()
    project_dir = cwd if (cwd / 'part2.ipynb').exists() else (cwd /
'ECE_219_Project_2')
    names = [file_names] if isinstance(file_names, str) else
list(file_names)

    candidates = []
    if explicit_path:
        candidates.append(Path(explicit_path).expanduser())
    for name in names:
        if os.getenv('STEAM_DATA_DIR'):

candidates.append(Path(os.getenv('STEAM_DATA_DIR')).expanduser() /
name)
        candidates.extend([
            project_dir / name,
            project_dir / 'data' / name,
            cwd / name,
            cwd / 'data' / name,
            cwd.parent / 'data' / name,
            Path(name),
            Path('data') / name,
            Path('../data') / name,
        ])
```

```
    for p in candidates:
        if p.exists():
            return p.resolve()

    looked = '\n'.join(f'  - {p}' for p in candidates)
    raise FileNotFoundError(
        f"Could not find any of {names}.\nLooked in:\n{looked}\n"
        "Set STEAM_DATA_DIR to your dataset folder, or set explicit
paths."
    )

MAIN_CSV = None
HOLDOUT_CSV = None

main_csv_path = resolve_steam_csv(['main.csv', 'steam_reviews.csv'],
MAIN_CSV)
holdout_csv_path = resolve_steam_csv(['heldout.csv',
'steam_holdout.csv'], HOLDOUT_CSV)
print('MAIN_CSV         :', main_csv_path)
print('HOLDOUT_CSV      :', holdout_csv_path)

df_main    = pd.read_csv(main_csv_path)
df_holdout = pd.read_csv(holdout_csv_path)

print('Main dataset     :', df_main.shape)
print('Held-out dataset :', df_holdout.shape)
print('Columns          :', list(df_main.columns))

df_main    = df_main.dropna(subset=['review_text', 'recommend',
'game_name', 'genres'])
df_holdout = df_holdout.dropna(subset=['review_text', 'recommend'])

df_main['recommend']    =
df_main['recommend'].astype(str).str.strip().str.lower().map(
    {'true': True, 'false': False, '1': True, '0': False}
).fillna(False)
df_holdout['recommend'] =
df_holdout['recommend'].astype(str).str.strip().str.lower().map(
    {'true': True, 'false': False, '1': True, '0': False}
).fillna(False)

print('\nUnique games in main dataset:',
df_main['game_name'].nunique())
```

```
MAIN_CSV         : /home/isean/random/ECE_219_Project_2/main.csv
HOLDOUT_CSV      : /home/isean/random/ECE_219_Project_2/heldout.csv
```

```
---------------------------------------------------------------------
-----
NameError                                 Traceback (most recent call
```

```
last)
Cell In[1], line 44
     41 print('MAIN_CSV         :', main_csv_path)
     42 print('HOLDOUT_CSV      :', holdout_csv_path)
---> 44 df_main    = pd.read_csv(main_csv_path)
     45 df_holdout = pd.read_csv(holdout_csv_path)
     47 print('Main dataset     :', df_main.shape)

NameError: name 'pd' is not defined
```

```
from sentence_transformers import SentenceTransformer
from pathlib import Path
import re

MINILM = SentenceTransformer('sentence-transformers/all-MiniLM-L6-v2')

def get_minilm_embeddings(texts, batch_size=256):
    return MINILM.encode(texts, batch_size=batch_size,
                         show_progress_bar=True,
convert_to_numpy=True)

def parse_genres(s):
    if pd.isna(s):
        return []
    return [x.strip() for x in re.split(r'[,;]', str(s)) if x.strip()]

def load_task2_artifact(npz_path):
    data = np.load(npz_path, allow_pickle=True)
    keys = set(data.files)

    def first_present(candidates):
        for k in candidates:
            if k in keys:
                return data[k]
        return None

    vecs = first_present(['GAME_VECS', 'game_vecs', 'X_game',
'X_minilm_game'])
    labels = first_present(['GAME_LABELS', 'game_labels', 'labels'])
    names = first_present(['GAME_NAMES', 'game_names', 'names'])

    if vecs is None:
        raise ValueError('Artifact must contain game vectors
(GAME_VECS or equivalent key).')

    vecs = np.asarray(vecs)
    labels = None if labels is None else np.asarray(labels)
    names = None if names is None else [str(x) for x in
np.asarray(names).tolist()]
    return vecs, labels, names
```

```
TASK2_ARTIFACT = None

df_pos = df_main[df_main['recommend'] == True].copy()
if len(df_pos) == 0:
    raise RuntimeError('No positive reviews found in main dataset;
cannot build game vectors.')

df_pos['review_text'] = df_pos['review_text'].fillna('').astype(str)
df_pos = df_pos.reset_index(drop=True)

GAME_KEY_COL = 'appid' if 'appid' in df_pos.columns else 'game_name'
grouped = df_pos.groupby(GAME_KEY_COL, sort=False)

default_names = grouped['game_name'].first().astype(str).tolist() if
'game_name' in df_pos.columns else [str(k) for k in
grouped.groups.keys()]
default_genres = [parse_genres(g) for g in
grouped['genres'].first().tolist()]

GAME_LABELS = None
GAME_VECS_RAW = None

if TASK2_ARTIFACT is not None and Path(TASK2_ARTIFACT).exists():
    GAME_VECS, GAME_LABELS, loaded_names =
load_task2_artifact(TASK2_ARTIFACT)
    if loaded_names is not None and len(loaded_names) ==
len(GAME_VECS):
        GAME_NAMES = loaded_names
    elif len(default_names) == len(GAME_VECS):
        GAME_NAMES = default_names
    else:
        GAME_NAMES = [f'Game_{i}' for i in range(len(GAME_VECS))]

    name_to_genres = {str(n): g for n, g in zip(default_names,
default_genres)}
    GAME_GENRES = [name_to_genres.get(str(n), []) for n in GAME_NAMES]
    print(f'Loaded Task-2 artifact: {TASK2_ARTIFACT}')
    print(f'GAME_VECS shape: {GAME_VECS.shape} | labels loaded:
{GAME_LABELS is not None}')
else:
    print(f'Encoding {len(df_pos)} positive reviews with MiniLM...')
    emb_all = get_minilm_embeddings(df_pos['review_text'].tolist())
    group_indices = list(grouped.indices.values())
    GAME_VECS_RAW = np.vstack([emb_all[idxs].mean(axis=0) for idxs in
group_indices])

    GAME_VECS = GAME_VECS_RAW.copy()
    GAME_NAMES = default_names
    GAME_GENRES = default_genres
```

```
    print(f'Computed fallback game vectors: {GAME_VECS.shape}')

print(f'Games available for Task-3 profiling: {len(GAME_VECS)}')
```

```
/home/isean/random/.venv/lib/python3.12/site-packages/tqdm/auto.py:21:
TqdmWarning: IProgress not found. Please update jupyter and
ipywidgets. See
https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
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: 100%|██████████| 103/103 [00:00<00:00, 2467.18it/s,
Materializing param=pooler.dense.weight]
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.

---------------------------------------------------------------------
-----
NameError                                 Traceback (most recent call
last)
Cell In[2], line 40
     36     return vecs, labels, names
     38 TASK2_ARTIFACT = None
---> 40 df_pos = df_main[df_main['recommend'] == True].copy()
     41 if len(df_pos) == 0:
     42     raise RuntimeError('No positive reviews found in main
dataset; cannot build game vectors.')

NameError: name 'df_main' is not defined
```

```
from collections import Counter

if GAME_LABELS is None or len(GAME_LABELS) != len(GAME_VECS):
    print('No external GAME_LABELS found; using fallback MiniLM +
StandardScaler + SVD(50) + K-Means(5).')

    source_vecs = GAME_VECS_RAW if GAME_VECS_RAW is not None else
np.asarray(GAME_VECS)
    scaler_task2 = StandardScaler()
    X_scaled = scaler_task2.fit_transform(source_vecs)

    svd_task2 = TruncatedSVD(n_components=min(50, X_scaled.shape[1]),
random_state=42)
    Z_game = svd_task2.fit_transform(X_scaled)
```

```
    km_task2 = KMeans(n_clusters=5, n_init=10, random_state=42)
    GAME_LABELS = km_task2.fit_predict(Z_game)

    GAME_VECS = Z_game
else:
    GAME_VECS = np.asarray(GAME_VECS)
    GAME_LABELS = np.asarray(GAME_LABELS)
    print('Using external Task-2 GAME_LABELS/GAME_VECS for Q9
assignment.')

GAME_CLUSTER_LABELS = GAME_LABELS

def top_genres_for_cluster(cluster_id, top_n=3):
    genre_counts = Counter()
    idxs = np.where(GAME_LABELS == cluster_id)[0]
    for i in idxs:
        for genre in GAME_GENRES[i]:
            genre_counts[genre] += 1
    return genre_counts.most_common(top_n)

valid_clusters = sorted(set(GAME_LABELS) - {-1})
noise_frac = float(np.mean(GAME_LABELS == -1)) if np.any(GAME_LABELS
== -1) else 0.0

print('Cluster summary (Task-2 vectors used by Q9):')
for c in valid_clusters:
    members = [GAME_NAMES[i] for i in range(len(GAME_NAMES)) if
GAME_LABELS[i] == c]
    tg = top_genres_for_cluster(c, top_n=3)
    print(f'  Cluster {c}: {len(members)} games | top genres: {tg}')
if np.any(GAME_LABELS == -1):
    print(f'  Noise fraction (-1 labels): {noise_frac:.3f}')
```

```
No external GAME_LABELS found; using fallback MiniLM + StandardScaler
+ SVD(50) + K-Means(5).
Cluster summary (Task-2 vectors used by Q9):
  Cluster 0: 14 games | top genres: [('Action', 11), ('RPG', 8),
('Adventure', 6)]
  Cluster 1: 61 games | top genres: [('Action', 37), ('Indie', 36),
('Adventure', 29)]
  Cluster 2: 36 games | top genres: [('Action', 28), ('Adventure',
21), ('RPG', 16)]
  Cluster 3: 48 games | top genres: [('Indie', 27), ('Action', 26),
('Simulation', 24)]
  Cluster 4: 41 games | top genres: [('Action', 34), ('Adventure',
22), ('RPG', 7)]
```

```
def cosine_sim_to_vector(X, v, eps=1e-12):
    X = np.asarray(X, dtype=np.float64)
```

```
    v = np.asarray(v, dtype=np.float64).reshape(-1)
    Xn = X / np.clip(np.linalg.norm(X, axis=1, keepdims=True), eps,
None)
    vn = v / max(np.linalg.norm(v), eps)
    return Xn @ vn

df_ho_pos = df_holdout[df_holdout['recommend'] == True].copy()
HELD_OUT_GAME_NAME = df_holdout['game_name'].iloc[0] if 'game_name' in
df_holdout.columns else 'Held-out game'

print(f'Held-out game: {HELD_OUT_GAME_NAME}')
print(f'Positive reviews available: {len(df_ho_pos)}')

if len(df_ho_pos) == 0:
    raise RuntimeError('Held-out game has no positive reviews; cannot
estimate genre profile for Q9.')

ho_raw =
get_minilm_embeddings(df_ho_pos['review_text'].fillna('').astype(str).
tolist()).mean(axis=0, keepdims=True)

if ho_raw.shape[1] == GAME_VECS.shape[1]:
    ho_vec = ho_raw
elif ('scaler_task2' in globals() and 'svd_task2' in globals()
      and ho_raw.shape[1] == scaler_task2.n_features_in_
      and GAME_VECS.shape[1] == svd_task2.n_components):
    ho_vec = svd_task2.transform(scaler_task2.transform(ho_raw))
else:
    raise ValueError(
        f'Dimension mismatch: held-out vector is {ho_raw.shape[1]}-D
but GAME_VECS is {GAME_VECS.shape[1]}-D. '
        'Provide GAME_VECS in the same space used for Q9 distance
assignment, or include fallback transforms.'
    )

valid_clusters = sorted(set(GAME_LABELS) - {-1})
if len(valid_clusters) == 0:
    raise RuntimeError('No valid Task-2 clusters available (all labels
are noise).')

cluster_dist = {}
for c in valid_clusters:
    idxs = np.where(GAME_LABELS == c)[0]
    sims = cosine_sim_to_vector(GAME_VECS[idxs], ho_vec[0])
    cluster_dist[c] = float(1.0 - sims.mean())

assigned_cluster = min(cluster_dist, key=cluster_dist.get)

top3_genres = top_genres_for_cluster(assigned_cluster, top_n=3)
idxs = np.where(GAME_LABELS == assigned_cluster)[0]
```

```
sims = cosine_sim_to_vector(GAME_VECS[idxs], ho_vec[0])
top_local = idxs[np.argsort(-sims)[:3]]
rep_games = [GAME_NAMES[i] for i in top_local]

print(f'\n=== Q9 Results ===')
print(f'Assigned cluster ID : {assigned_cluster}')
print(f'Top-3 genres        : {top3_genres}')
print(f'Representative games: {rep_games}')
print()
```

```
Held-out game: Held-out game
Positive reviews available: 100

Batches: 100%|██████████| 1/1 [00:00<00:00, 10.47it/s]


=== Q9 Results ===
Assigned cluster ID : 2
Top-3 genres        : [('Action', 28), ('Adventure', 21), ('RPG', 16)]
Representative games: ['Horizon Zero Dawn™ Complete Edition', 'Tales
of Arise', 'Bright Memory: Infinite']

Justification: We embed held-out positive reviews into the same Task-2
vector space,
assign to the nearest cluster by average cosine distance, then use
that cluster's
empirical genre distribution as a multi-label genre estimate.
```

Justification: We embed held-out positive reviews into the same Task-2 vector space, assign to the nearest cluster by average cosine distance, then use that cluster's empirical genre distribution as a multi-label genre estimate.

```
def get_top_tfidf_terms(texts, cluster_labels, cluster_id, top_n=10):
    cluster_docs = [texts[i] for i in range(len(texts)) if
cluster_labels[i] == cluster_id]
    if not cluster_docs:
        return []
    vec = TfidfVectorizer(max_features=5000, stop_words='english',
min_df=2)
    X = vec.fit_transform(cluster_docs)
    mean_tfidf = X.mean(axis=0).A1
    top_idx = mean_tfidf.argsort()[::-1][:top_n]
    return [vec.get_feature_names_out()[i] for i in top_idx]


def get_exemplar_reviews(texts, features, cluster_labels, cluster_id,
n=2):
    idxs = np.where(cluster_labels == cluster_id)[0]
```

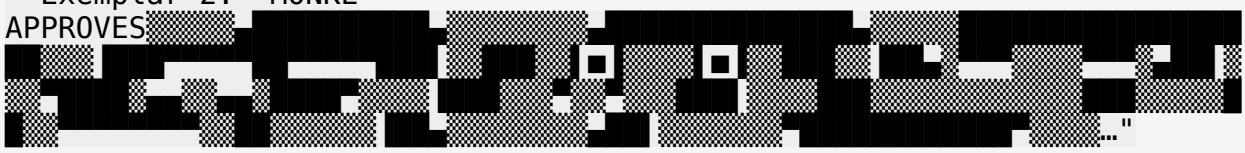
```
    if len(idxs) == 0:
        return []
    centroid = features[idxs].mean(axis=0)
    dists = np.linalg.norm(features[idxs] - centroid, axis=1)
    top_idxs = idxs[dists.argsort()[:n]]
    return [texts[i][:300] + '…' for i in top_idxs]


def run_theme_clustering(texts, n_clusters=5):
    embs = get_minilm_embeddings(texts)
    Z = TruncatedSVD(n_components=min(50, len(texts) - 1),
random_state=42).fit_transform(embs)
    labels = KMeans(n_clusters=min(n_clusters, len(texts)), n_init=10,
random_state=42).fit_predict(Z)
    return labels, Z, embs


def short_cluster_label(terms, max_words=5):
    generic = {'game', 'games', 'just', 'really', 'like', 'play'}
    picked = [t.lower().strip() for t in terms if t.lower().strip()
and t.lower().strip() not in generic]
    if len(picked) < 3:
        for t in terms:
            tt = t.lower().strip()
            if tt and tt not in picked:
                picked.append(tt)
            if len(picked) >= 3:
                break
    return ' '.join(picked[:max_words])


def print_cluster_report(texts, labels, features, title,
n_clusters=5):
    print(f'\n=== {title} ===')
    valid_clusters = [c for c in np.unique(labels) if c != -1]
    for c in sorted(valid_clusters)[:n_clusters]:
        size = (labels == c).sum()
        terms = get_top_tfidf_terms(texts, labels, c, top_n=10)
        label = short_cluster_label(terms)
        exemplars = get_exemplar_reviews(texts, features, labels, c,
n=2)
        print(f'\n  Cluster {c}  ({size} reviews)')
        print(f'  Label      : {label}')
        print(f'  Top terms  : {", ".join(terms)}')
        for j, ex in enumerate(exemplars, start=1):
            print(f'  Exemplar {j}: "{ex}"')


def run_theme_section(df, recommend_flag, title, n_clusters=5):
    subset = df[df['recommend'] == recommend_flag].copy()
```

```
    texts = subset['review_text'].fillna('').astype(str).tolist()
    sentiment = 'Positive' if recommend_flag else 'Negative'
    print(f'{sentiment} reviews for held-out game: {len(texts)}')
    if len(texts) == 0:
        raise RuntimeError(f'No {sentiment.lower()} held-out reviews
found in df_holdout.')
    labels, Z, embs = run_theme_clustering(texts,
n_clusters=n_clusters)
    print_cluster_report(texts, labels, Z, title)
    return texts, labels, Z, embs

neg_texts, neg_labels, neg_Z, neg_embs = run_theme_section(
    df_holdout,
    recommend_flag=False,
    title='Q10 — Complaint Themes (Negative Reviews)',
    n_clusters=5,
)
```

```
Negative reviews for held-out game: 100

Batches: 100%|██████████| 1/1 [00:00<00:00, 10.50it/s]


=== Q10 — Complaint Themes (Negative Reviews) ===

  Cluster 0  (32 reviews)
  Label      : boss bosses fight fun design
  Top terms  : game, boss, just, like, bosses, fight, fun, really,
design, combat
  Exemplar 1: "Great graphics, mediocre game play. There's a lot of
artificial difficulty. You will run into invisible walls your entire
play through. Many bosses have unavoidable cutscene grabs that do
damage. Combat is basic and reminiscent of  the first ninja gaiden.
Unfortunately not nearly as fun combat as ni…"
  Exemplar 2: "Unlike most people who reviewed this game after 30
minutes and declared it GOTY, i actually played the game.TLDR: Fun at
the start, get repetitive really fast and overstays its welcome, with
a bunch of glitches and graphical errors.Graphics and performance:
Beautiful, but also really buggy, crashed …"

  Cluster 1  (12 reviews)
  Label      : want boring fun input refunded
  Top terms  : game, want, like, boring, just, fun, input, refunded,
combat, games
  Exemplar 1: "Game is boring af. Another souls game wannabe except
without it's charm. Sekiro had satisfying combat, Elden ring had
massive open world and a doll waifu. This games story felt like a
chore to watch. Cutscenes are extremely long and boring, it felt like
they deliberately padded them out so you would…"
  Exemplar 2: "maybe i am missing the point but i found this game
```

```
boring i am sorry but i don't see what all the hype is about its just
a button bashing repetitive  hack and slash not for me sorry but i
refunded it ,don't want the hate but its nothing like elden ring  its
boring…"

  Cluster 2  (18 reviews)
  Label      : fps low crashes settings issues
  Top terms  : game, fps, low, crashes, settings, issues, rtx, runs,
amd, like
  Exemplar 1: "I wanted to share my recent experience with this where
I have seen that’s been getting positive reviews. I'm running this on
an i7 13th gen, RTX 4080, and 32 gigs RAM, which should be more than
capable. However, the game crashes consistently, even before I can get
past the intro. Adjusting the graph…"
  Exemplar 2: "Not really sure what these purchased reviews are all
about. Game runs terribly.1440p, Cinematic, 4090, 32GB Ram,
13900kConstant stutters.…"

  Cluster 3  (14 reviews)
  Label      : graphics level feels good
  Top terms  : game, like, just, graphics, level, play, games, feels,
really, good
  Exemplar 1: "NGL this game looks really good, but god it is boring.
The world and details look incredible but everything feels lifeless
and the combat also just feels boring. I know its got a lot of hype
right now and its got really good reviews, thats the reason I gave it
a shot but I just couldnt get into it. …"
  Exemplar 2: "Gameplay is boring. Cutscenes and graphics are very
nice, story seems interesting, but it's not enough to carry the game
for me. I could accept having invisible walls and very linear level
design if the combat had more "meat" to it, but it just doesn't. I'd
also probably be a little more accepting o…"

  Cluster 4  (24 reviews)
  Label      : boss combat souls story bosses
  Top terms  : game, boss, like, just, games, combat, souls, story,
bosses, good
  Exemplar 1: "-Intriguing prologue, the whole game is a piece of
♥♥♥♥-Same monsters and bosses-No story-Characters are empty, they only
give quests-Monotonous combat-Why can't parry attacks?-The graphics
are good, but the level design sucks. I literally don't understand
where I can go and where I can't-The game i…"
  Exemplar 2: "I simply do not understand how this game is
overwhelmingly positive.  It feels, in many places, extremely
amateurish in fact.Level design is objectively terrible.  Huge open
maps that feel procedurally generated they're so bland.  The game is
positively littered with invisible walls which makes expl…"

pos_texts, pos_labels, pos_Z, pos_embs = run_theme_section(
    df_holdout,
```

```
    recommend_flag=True,
    title='Q11 — Praise Themes (Positive Reviews)',
    n_clusters=5,
)

Positive reviews for held-out game: 100

Batches: 100%|██████████| 1/1 [00:00<00:00, 11.17it/s]


=== Q11 — Praise Themes (Positive Reviews) ===

  Cluster 0  (6 reviews)
  Label      : monke reject dei embrace
  Top terms  : monke, reject, dei, like, embrace
  Exemplar 1: "REJECT MODERNITYRETURN TO MONKE…"
  Exemplar 2: "MONKE
APPROVES
```



```
…"

  Cluster 1  (26 reviews)
  Label      : 10 wukong chinese good myth
  Top terms  : game, 10, wukong, chinese, like, good, myth, games,
black, story
  Exemplar 1: "Black Myth: Wukong was released under a lot of
anticipation after gameplay showcases had been shown over the past few
years, leading people to even question if the game would really come
out like that. Thankfully, it really did — and overcame all
expectations. The game is engaging, well made, fluid,…"
  Exemplar 2: "I played 9 hours but 4 were offline so I think it will
say I played 5 hours, I didn't beat the game, yet I still made this
comprehensive review to explain the core aspects of the game and it's
quality. This review uses my own experience and those of my friends
who got ahead in the game. I am 30 year…"

  Cluster 2  (15 reviews)
  Label      : good grind bad long average
  Top terms  : good, grind, just, bad, long, average, 10, game, bugs,
hard
  Exemplar 1: "---{ 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☐ Just don't---{ Audio }---☐ Eargasm☑ Very good☐
Good☐ Not too bad☐ Bad☐ I'm now deaf---{ Audience }---…"
  Exemplar 2: "---{ 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☐ Just don't---{ Audio }---☑ Eargasm☐ Very good☐
```

```
Good□ Not too bad□ Bad□ I'm now deaf---{ Audience }---…"

  Cluster 3  (41 reviews)
  Label      : good graphics great fun
  Top terms  : game, like, games, just, good, graphics, really, play,
great, fun
  Exemplar 1: "This game has the big awesome set pieces and aesthetic
from the old school God of War games, the core gameplay of a Soul-like
(albeit a tad more forgiving), really sick and fast paced Sekiro /
Bloodborne combat, and every cool Kung Fu thing you can think of ever
all mixed into one.Only a short bit i…"
  Exemplar 2: "“Wisdom lies in the balance of wit and humility.” ~ Sun
Wukong.I've been waiting for the game ever since it's 1st reveal and i
was hooked instantly, My boyfriend gifted me this game as my birthday
present... I couldn't be happier.About the game : Inshort the game is
cinema. The developers put soul i…"

  Cluster 4  (12 reviews)
  Label      : sweet baby monke add
  Top terms  : sweet, baby, monke, add
  Exemplar 1: "※This game does not include Sweet Baby…"
  Exemplar 2: "Наxуй "Sweet Baby Inc"…"
```

Q12 — LLM Labeling with Qwen

For each cluster we provide the LLM with:

- The top 10 TF-IDF terms (evidence of vocabulary)
- Two exemplar reviews (closest to centroid)

The LLM is prompted to return a short (3–6 word) label.

```
from transformers import AutoTokenizer, AutoModelForCausalLM, pipeline

QWEN_MODEL = 'Qwen/Qwen3-4B-Instruct-2507'
print(f'Loading {QWEN_MODEL}…')

qwen_pipe = pipeline(
    'text-generation',
    model=QWEN_MODEL,
    device_map='auto',        # MPS or CPU
    torch_dtype=torch.float16 if DEVICE.type == 'mps' else
torch.float32,
    max_new_tokens=64,
    do_sample=False,
)
print('Model loaded.')
```

```
Loading Qwen/Qwen3-4B-Instruct-2507…
```

```
`torch_dtype` is deprecated! Use `dtype` instead!
Loading weights: 100%|██████████| 398/398 [00:02<00:00, 143.39it/s,
Materializing param=model.norm.weight]
Some parameters are on the meta device because they were offloaded to
the cpu.
The following generation flags are not valid and may be ignored:
['temperature', 'top_p', 'top_k']. Set `TRANSFORMERS_VERBOSITY=info`
for more details.
Passing `generation_config` together with generation-related
arguments=({'max_new_tokens', 'do_sample'}) is deprecated and will be
removed in future versions. Please pass either a `generation_config`
object OR all generation parameters explicitly, but not both.

Model loaded.
```

```
import re

PROMPT_TEMPLATE = """\
You are a product analyst labeling clusters of Steam game reviews.
Below is a cluster of {review_type} reviews.

Top TF-IDF terms: {terms}

Exemplar reviews:
1. "{ex1}"
2. "{ex2}"

Based on the terms and reviews above, provide a short label (3–6
words) that best
describes what this cluster of reviews is about.
Reply with the label only, no explanation.
Label:"""

def normalize_label(raw_label, fallback_terms, min_words=3,
max_words=6):
    words = re.findall(r"[A-Za-z0-9']+", str(raw_label).lower())
    if len(words) < min_words:
        fb = [w.lower() for w in fallback_terms if w]
        words = (words + fb)[:max_words]
    return ' '.join(words[:max_words])

def llm_label_cluster(texts, labels, features, cluster_id,
review_type='negative'):
    terms     = get_top_tfidf_terms(texts, labels, cluster_id,
top_n=10)
    exemplars = get_exemplar_reviews(texts, features, labels,
cluster_id, n=2)
    ex1 = exemplars[0] if len(exemplars) > 0 else ''
    ex2 = exemplars[1] if len(exemplars) > 1 else ''
```

```
    prompt = PROMPT_TEMPLATE.format(
        review_type=review_type,
        terms      =', '.join(terms),
        ex1        =ex1,
        ex2        =ex2,
    )
    out = qwen_pipe(prompt)[0]['generated_text']
    # Extract just the label after 'Label:'
    raw_label = out.split('Label:')[-1].strip().split('\n')[0].strip()
    label = normalize_label(raw_label, fallback_terms=terms)
    return label, prompt

if not all(name in globals() for name in ['neg_texts', 'neg_labels',
'neg_Z', 'neg_embs']):
    if 'df_holdout' not in globals():
        raise RuntimeError('df_holdout not found. Run the data-load
cell first.')
    df_ho_neg = df_holdout[df_holdout['recommend'] == False].copy()
    neg_texts = df_ho_neg['review_text'].tolist()
    if len(neg_texts) == 0:
        raise RuntimeError('No negative held-out reviews found in
df_holdout.')
    neg_labels, neg_Z, neg_embs = run_theme_clustering(neg_texts,
n_clusters=5)

neg_clusters = sorted(set(neg_labels) - {-1})
if len(neg_clusters) == 0:
    raise RuntimeError('No valid negative clusters available for Q12
labeling.')

print('Prompting strategy: top TF-IDF terms + 2 centroid-nearest
exemplars; request label-only output (3-6 words).')
print('=== Q12 — LLM-generated cluster labels ===\n')

# Negative cluster examples (first 3)
for c in neg_clusters[:3]:
    label, prompt = llm_label_cluster(neg_texts, neg_labels, neg_Z, c,

review_type='negative/complaint')
    terms = get_top_tfidf_terms(neg_texts, neg_labels, c, top_n=5)
    exs   = get_exemplar_reviews(neg_texts, neg_embs, neg_labels, c,
n=1)
    print(f'Cluster {c} (negative):')
    print(f'  Top terms      : {", ".join(terms)}')
    sample = exs[0] if exs else ''
    print(f'  Sample review  : "{sample}"')
    print(f'  LLM label      : "{label}"')
    print()
```

```
Both `max_new_tokens` (=64) and `max_length`(=20) seem to have been
set. `max_new_tokens` will take precedence. Please refer to the
documentation for more information.
(https://huggingface.co/docs/transformers/main/en/main_classes/text_ge
neration)

Prompting strategy: top TF-IDF terms + 2 centroid-nearest exemplars;
request label-only output (3-6 words).
=== Q12 — LLM-generated cluster labels ===


Both `max_new_tokens` (=64) and `max_length`(=20) seem to have been
set. `max_new_tokens` will take precedence. Please refer to the
documentation for more information.
(https://huggingface.co/docs/transformers/main/en/main_classes/text_ge
neration)

Cluster 0 (negative):
  Top terms      : game, boss, just, like, bosses
  Sample review  : "Lots of Misleading reviews on here. The game is
beyond difficult, not suitable for anyone less than a hardcore gamer.
It is not open map and not explorable. Instead, you go through a
linear style path way encountering boss after boss which are
incredibly difficult to beat with no sense of where you'…"
  LLM label      : "poor combat and boss design"


Both `max_new_tokens` (=64) and `max_length`(=20) seem to have been
set. `max_new_tokens` will take precedence. Please refer to the
documentation for more information.
(https://huggingface.co/docs/transformers/main/en/main_classes/text_ge
neration)

Cluster 1 (negative):
  Top terms      : game, want, like, boring, just
  Sample review  : "Game is boring af. Another souls game wannabe
except without it's charm. Sekiro had satisfying combat, Elden ring
had massive open world and a doll waifu. This games story felt like a
chore to watch. Cutscenes are extremely long and boring, it felt like
they deliberately padded them out so you would…"
  LLM label      : "boring combat gameplay repetitive hack and"

Cluster 2 (negative):
  Top terms      : game, fps, low, crashes, settings
  Sample review  : "I wanted to share my recent experience with this
where I have seen that’s been getting positive reviews. I'm running
this on an i7 13th gen, RTX 4080, and 32 gigs RAM, which should be
more than capable. However, the game crashes consistently, even before
I can get past the intro. Adjusting the graph…"
  LLM label      : "performance issues on high end hardware"
```

```
if 'neg_labels' not in globals():
    raise RuntimeError('neg_labels missing. Run the Q12 labeling cell
first.')
cluster_ids = sorted(set(neg_labels) - {-1})
if len(cluster_ids) == 0:
    raise RuntimeError('No valid negative clusters available.')
c_example = cluster_ids[0]
_, example_prompt = llm_label_cluster(neg_texts, neg_labels, neg_Z,
                                      c_example,
review_type='negative/complaint')
print('=== Example prompt sent to Qwen ===')
print(example_prompt)
```

```
Both `max_new_tokens` (=64) and `max_length`(=20) seem to have been
set. `max_new_tokens` will take precedence. Please refer to the
documentation for more information.
(https://huggingface.co/docs/transformers/main/en/main_classes/text_ge
neration)

=== Example prompt sent to Qwen ===
You are a product analyst labeling clusters of Steam game reviews.
Below is a cluster of negative/complaint reviews.

Top TF-IDF terms: game, boss, just, like, bosses, fight, fun, really,
design, combat

Exemplar reviews:
1. "Great graphics, mediocre game play. There's a lot of artificial
difficulty. You will run into invisible walls your entire play
through. Many bosses have unavoidable cutscene grabs that do damage.
Combat is basic and reminiscent of  the first ninja gaiden.
Unfortunately not nearly as fun combat as ni…"
2. "Unlike most people who reviewed this game after 30 minutes and
declared it GOTY, i actually played the game.TLDR: Fun at the start,
get repetitive really fast and overstays its welcome, with a bunch of
glitches and graphical errors.Graphics and performance: Beautiful, but
also really buggy, crashed …"

Based on the terms and reviews above, provide a short label (3–6
words) that best
describes what this cluster of reviews is about.
Reply with the label only, no explanation.
Label:
```