**1. Data Ingestion & Preprocessing**

* **Load images with torchvision.datasets.ImageFolder or a custom Dataset that reads train\_labels.csv for scene IDs.**
* **Resize & normalize to a fixed size (e.g. 512×512) using torchvision.transforms or [Albumentations].**
* **Augment (color jitter, random crop/flip, blur) during training of retrieval models to improve invariance.**

**2. Global Descriptor Extraction & Training**

1. **Backbone + Pooling**
   * **Start from a pre‑trained ResNet‑50/101 or EfficientNet-B4 in PyTorch.**
   * **Replace the last global‐average pool with either:**
     + **Generalized Mean (GeM) pooling (learnable p‑norm)**
     + **NetVLAD layer (e.g. K = 64 clusters).**
2. **Metric Learning**
   * **Supervised Contrastive Loss: treat each scene as a “class,” optimize positives (same scene) vs. negatives (other scenes/outliers).**
   * **Triplet Margin Loss: for each anchor image, sample one positive and one hard negative.**
   * **Use [PyTorch Lightning] or plain PyTorch training loops with an online miner (e.g. batch hard mining).**
3. **Output**
   * **A fixed‑length descriptor vector (e.g. 2048‑D) per image, L2‑normalized.**

**3. Candidate Pair Selection**

* **Compute pairwise cosine similarities (or faiss approximate nearest‐neighbors) between all test‑set descriptors.**
* **For each image, keep the top‑K neighbors (e.g. K = 50). This drastically reduces the number of image pairs passed to the next stage.**

**4. Local Feature Matching & Geometric Verification**

1. **Keypoint Detection & Description**
   * **Use SuperPoint (or [D2‑Net]/[R2D2]) implemented in PyTorch/Kornia to extract dense keypoints + descriptors.**
2. **Matching**
   * **Feed overlapping descriptor sets into SuperGlue (graph‑neural matcher) or LoFTR for robust, outlier‑resistant matches.**
3. **RANSAC Filtering**
   * **On each image pair, run RANSAC to fit a fundamental matrix or homography with OpenCV’s cv2.findFundamentalMat(…).**
   * **Discard pairs with fewer than a threshold (e.g. 15) inliers.**

**5. Scene Clustering**

* **Graph construction: nodes = images, edges = verified matches.**
* **Connected components: each component is an initial scene cluster.**
* **Refinement:**
  + **Merge small components into “outliers” if size < min\_size (e.g. 3 images).**
  + **Optionally run DBSCAN or HDBSCAN on global descriptors to further split/merge clusters.**

**6. 3D Reconstruction & Pose Estimation**

* **pycolmap integration: Python bindings for COLMAP**
  1. **Feed each cluster’s images + matches into COLMAP’s incremental mapper.**
  2. **COLMAP outputs camera poses (R, t) and sparse 3D point clouds.**
* **Bundle Adjustment: fully optimize R and t using COLMAP’s built‑in BA or external tools (e.g. gtsam) for maximum accuracy.**

**7. Pose‐Only Regression (Optional)**

* **For images with insufficient matches, you can train a small regression head:**
  + **Input: global descriptor**
  + **Output: 6‑DOF pose (parameterized as quaternion + translation)**
  + **Loss: MSE on R and t against ground truth from train\_labels.csv.**
* **This acts as a “backup” for outlier images that you still want to assign to a cluster.**

**8. Submission Assembly**

* **For each test image:**
  + **dataset: folder name**
  + **scene: cluster ID (e.g. cluster1, cluster2, or outliers)**
  + **rotation\_matrix: flatten COLMAP’s R in row‑major, ;‑separated (or nan if no pose)**
  + **translation\_vector: same for t (or nan)**
* **Write out exactly as dataset,scene,image,rotation\_matrix,translation\_vector.**

**Key Model Recommendations & Resources**

| **Component** | **Model / Library** | **Notes** |
| --- | --- | --- |
| **Global retrieval** | **ResNet‑50 + GeM + NetVLAD (K=64)** | **[Arandjelović & Zisserman, 2016]** |
| **Local features** | **SuperPoint + SuperGlue (or LoFTR)** | **Official PyTorch implementations exist** |
| **Matching & RANSAC** | **OpenCV’s solvePnP / findFundamentalMat** | **Lightweight, CPU‑based** |
| **SfM pipeline** | **pycolmap** | **Python‑friendly COLMAP bindings** |
| **Clustering** | **scikit‑learn DBSCAN / HDBSCAN** | **Tune eps based on descriptor dims** |
| **Regression head** | **MLP with quaternion output** | **Only if needed for sparse cases** |

**Why this works**

* **Global+local hybrid gives both speed (global pruning) and precision (local verification).**
* **Metric learning on scenes leverages the provided poses to create high‑quality descriptors.**
* **COLMAP remains the gold‑standard for SfM; pycolmap lets you script it in Python.**
* **The clustering→SfM division cleanly separates grouping from pose estimation, matching the challenge’s two sub‑tasks.**