SceneSqueezer: Learning to Compress Scene for Camera Relocalization Supplementary Material

Luwei Yang[1*] Rakesh Shrestha[1*] Wenbo Li[2] Shuaicheng Liu[3]
Guofeng Zhang[2] Zhaopeng Cui[2†] Ping Tan[1,4†]

[1] Simon Fraser University [2] State Key Lab of CAD & CG, Zhejiang University
[3] University of Electronic Science and Technology of China [4]Alibaba XR Lab

{luweiy, rakeshs, pingtan}@sfu.ca, {wenboli, zhangguofeng, zhpcui}@zju.edu.cn,
liushuaicheng@uestc.edu.cn

This supplementary document aims to provide additional details to make our submission self-contained. It includes: 1) System details; 2) Training details; 3) Registration of a query image at runtime; 4) Additional ablation studies and time costs;

A. System Details

Co-visible Frames Clustering: We use "average" linkage for hierarchical clustering since it is less sensitive to outliers. After obtaining the hierarchically clustered dendrogram, flat clusters are formed such that the distance between samples in a cluster is less than a pre-defined distance threshold. We implement this with `scipy`'s [19] `linkage` and `fcluster` functions respectively.

2D Keypoint Features: Our 2D keypoint feature is based on SuperPoint [3]. It encodes the Superpoint feature and its confidence along with the 2D keypoint position using SuperGlue's [14] keypoint encoder. Concretely, let i be a keypoint at position $\mathbf{x}_i$ in frame m with SuperPoint feature $\tilde{\mathbf{F}}_i$ and confidence p_i. Its encoded feature $\mathbf{F}_i$ is then generated by:

$$\mathbf{F}_i = \tilde{\mathbf{F}}_i + \texttt{MLP}(\hat{\mathbf{x}}_i, p_i). \tag{1}$$

Here, $\hat{\mathbf{x}}_i$ denotes the normalized keypoint position in the range $(-1, 1)$. The MLP layer `MLP` transforms the normalized position and the keypoint's confidence into a 256-channel feature.

3D Point Features: The 3D *anchor* point feature is generated by fusing its 2D observations. Let $\mathbf{X}_k$ be a 3D point in anchor set $\mathbf{A}$ with its 2D observations in verification frames $\{\mathbf{V}_n\}$ denoted by $\{\mathbf{x}_n^k\}$. The feature of point $\mathbf{X}_k$ is the average of $\{\mathbf{x}_n^k\}$ from `MeanPooling` operation:

$$\mathbf{F}_k = \texttt{MeanPooling}(\{\mathbf{x}_n^k\}) \tag{2}$$

*Equal contribution, †Corresponding authors

Geometric Similarity: The reprojection distance between a 3D anchor point $\mathbf{X}_i \in \mathbb{R}^3$ and a 2D keypoint $\mathbf{x}_j \in \mathbb{R}^2$ in verification frame $\mathbf{V}_n$ is given by:

$$r_{ij} = \|\pi(\xi_n, \mathbf{X}_i) - \mathbf{x}_j\|.$$

Here, ξ_n is the camera pose of frame $\mathbf{V}_n$, π is the 3D projection function that projects a 3D point to 2D image plane given camera intrinsic and extrinsic parameters.

The geometric similarity M_{ij} between point i and keypoint j is then defined as:

$$M_{ij} = \exp(-\log(r_{ij})) \tag{3}$$

Anchor-Verification Fusion using Attention: We use the multi-head attention mechanism [18] to fuse the aggregated verification features $\{\mathbf{O}_{V_n \to A}\}$ from all the verification frames with the anchor features $\mathbf{F}_A$ to obtain aggregated anchor features $\mathbf{F}_A^{agg} \in \mathbb{R}^{m_A \times C}$ (Equation 4 of the main text). Here, m_A is the number of 3D anchor points and C is the dimension of the features. Multi-head attention aggregates information from different representation subspaces of the features by training multiple attentions in parallel. Each attention (known as "head") has 3 inputs: key ($\mathbf{K}$), value ($\mathbf{V}$) and query ($\mathbf{Q}$). For an anchor point $i \in [1, m_A]$, these are obtained as follows:

$$\begin{aligned} \mathbf{K} &= \text{MLP}_K(\mathbf{F}_A^i), \\ \mathbf{V} &= \text{MLP}_V(\{\mathbf{V}_n^i\}), \\ \mathbf{Q} &= \text{MLP}_Q(\{\mathbf{V}_n^i\}), \end{aligned} \tag{4}$$

where $\{\mathbf{V}_n^i\} \in \mathbb{R}^{N \times C}$ are the aggregated verification features corresponding to the 3D point i with N being the number of verification frames. We have $\mathbf{K} \in \mathbb{R}^{1 \times C'}$, $\mathbf{V} \in \mathbb{R}^{N \times C'}$ and $\mathbf{Q} \in \mathbb{R}^{N \times C'}$ where C' is the dimension

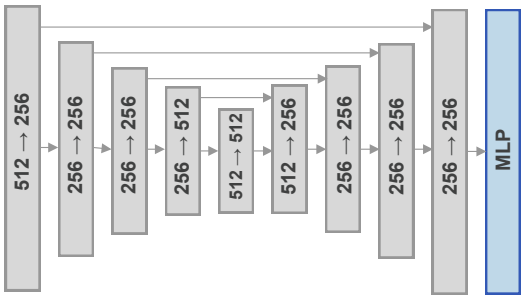

Figure A. **Point Transformer Structure** Each rectangle represents a `PointTransformer` Block [20], the inner text indicates input and output dimension of the point feature (*in* $\rightarrow$ *out*). The blocks are stacked in a UNet [12] style.

of the features after their respective multilayer perceptrons $\text{MLP}_{\{\}}$. We use $C' = 64$ in our network.

The attention weights of j^{th} head is computed using the scaled dot-product of $\mathbf{K}$, $\mathbf{V}$ and $\mathbf{Q}$

$$\text{head}_j(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{N}})\mathbf{V}. \tag{5}$$

The outputs of the independent heads are then concatenated and linearly transformed to obtain the fused anchor feature of i.

$$\begin{aligned} \mathbf{F}_A^{agg^i} &= \text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) \\ &= [\text{head}_1; ...; \text{head}_h]\mathbf{W}^O, \end{aligned} \tag{6}$$

where $\mathbf{W}^O$ is a parameter matrix to be learned and h is the number of attention heads. We set $h = 4$ in our network.

Point Transformer Structure: Our Point Transformer structure is shown in Figure A. Two mostly identical networks are used to generate distinctiveness scores $\mathbf{d}$ and pairwise proximity matrix $\mathbf{K}$ of 3D points. We stack multiple `PointTransformerBlocks` [20] in a UNet [12] scheme as it can better fuse local and global information of the point cloud. Similar structures have also been used for point cloud classification and segmentation tasks with impressive results [20]. We set sampling ratio to 0.25 in all blocks. The input to our Point Transformer is the normalized 3D point positions (in the range $[-1, 1]$) concatenated with their features from Section 3.2 of the main paper. A one-layered MLP is added at the end of the point transformer to produce output with desired dimensions. Concretely, we set the output dimensions to 1 to obtain the distinctiveness scores $\mathbf{d}$ of the points and 64 to yield the point features that are used for computing the proximity matrix $\mathbf{K}$.

Solving QP Problem: During training, we use differentiable convex optimization solver `CVXPYLayers` [1] for solving the QP problem which allows us to backpropagate the error gradients to $\mathbf{d}$, $\mathbf{K}$ and τ to train our network. However, training with more than 1000 points is not feasible using `CVXPYLayers` since it requires considerable GPU memory. Therefore, we randomly select 1000 points from the anchor point set at each iteration. The weight τ that balances the two terms in the QP problem (Equation 3 in the main paper) is also learned. We set its lower bound to 0.1 in order to prevent extreme solutions that ignore the distinctiveness scores altogether.

For testing, we implement the efficient Sequential Minimal Optimization based QP solver from [9] to compress points from each co-visible cluster. It solves the problem iteratively and optimizes only 2 points in one iteration. It therefore supports problems with a large number of variables (*i.e.* points). The algorithm is summarized in Algorithm 1.

Algorithm 1: Sequential Minimal Optimization

Input: $\mathbf{d}, \mathbf{K}, \tau, v, max\_iter$
Output: $\boldsymbol{\alpha}$

$\boldsymbol{\alpha} \leftarrow \text{initialize}(\mathbf{d}, \nu)$
$m \leftarrow \text{size}(\boldsymbol{\alpha})$
$itr = 0$
while $itr \leq max\_iter$ **do**
  $i, j \leftarrow \text{select\_pair}(\boldsymbol{\alpha})$
  $\Delta \leftarrow \boldsymbol{\alpha}_i + \boldsymbol{\alpha}_j$
  $\theta_1 \leftarrow 2 \sum_{l \neq i, l \neq j} \boldsymbol{\alpha}_l \mathbf{K}_{il}$
  $\theta_2 \leftarrow 2 \sum_{l \neq i, l \neq j} \boldsymbol{\alpha}_l \mathbf{K}_{jl}$
  $T \leftarrow \tau(\mathbf{d}_i - \mathbf{d}_j) - \theta_1 + \theta_2$
  $\boldsymbol{\alpha}_i^{'} \leftarrow \frac{1}{2}(\frac{T}{2(1 - \mathbf{K}_{ij} + \Delta)})$
  $\boldsymbol{\alpha}_i^{*} \leftarrow \max(0, \min(\min(\frac{1}{vm}), \Delta), \boldsymbol{\alpha}_i^{'})$
  $\boldsymbol{\alpha}_j \leftarrow \Delta - \boldsymbol{\alpha}_i^{*}; \boldsymbol{\alpha}_i \leftarrow \boldsymbol{\alpha}_i^{*}$
  $itr \leftarrow itr + 1$

Here, "initialize$(\mathbf{d}, \nu)$" initializes $\boldsymbol{\alpha}$ by assigning the corresponding entries of the top $\lceil \nu m \rceil$ distinct points to $\frac{1}{\nu m}$ and the rest to 0. This initialization procedure was proposed by [9] and has shown to greatly reduce the convergence time. The "select_pair$(\boldsymbol{\alpha})$" operation randomly selects a pair of points (i, j) such that at least one of them has non-zero entry in $\boldsymbol{\alpha}$. The maximum number of iterations max_iter is set to 20000.

Feature Quantization: Our feature quantization module is an auto-encoder network with Differentiable Soft Quantization (DSQ) [4] function as the network's bottleneck. The input to the module is 128 dimensional point features. The encoder is a 2 layered MLP where each layer consecutively outputs latent features of dimensions 256 and 128. The decoder is a mirrored version of the encoder.

Let $\dot{\mathbf{F}}_i$ be the 128-dim feature of point i produced by the encoder. The DSQ layer first transforms it with an MLP and then normalizes it to $(-0.5, 0.5)$ using the sigmoid func-

tion:

$$\ddot{\mathbf{F}}_i = \text{sigmoid}(\texttt{MLP}(\dot{\mathbf{F}}_i)) - 0.5. \quad (7)$$

To obtain the quantized feature $\hat{\mathbf{F}}_i$, we then use the differentiable soft quantization function (Equation 5 in [4]) to map $\ddot{\mathbf{F}}_i$ from the interval $(-0.5, 0.5)$ to $[0, 2^b - 1]$. Here, we set the bit width $b = 8$ to quantize our features to 8-bit unsigned integers. The differentiable soft quantization [4] is summarized in Algorithm 2. We refer our readers to the original paper for detailed derivations.

Algorithm 2: Differentiable Soft Quantization

Input: $\ddot{\mathbf{F}}_i, l_Q, u_Q, b, \gamma$
Output: $\hat{\mathbf{F}}_i$

$\Delta \leftarrow \frac{u_Q - l_Q}{2^b - 1}$ ▷ interval width
$\ddot{\mathbf{F}}_i \leftarrow \ddot{\mathbf{F}}_i + \text{ReLU}(l_Q - \ddot{\mathbf{F}}_i)$ ▷ clip lower bound
$\ddot{\mathbf{F}}_i \leftarrow \ddot{\mathbf{F}}_i - \text{ReLU}(\ddot{\mathbf{F}}_i - u_q)$ ▷ clip upper bound
$i \leftarrow \text{trunc}(\frac{\ddot{\mathbf{F}}_i - l_Q}{\Delta})$ ▷ index of interval
$m \leftarrow l_Q + (i + 0.5)\Delta$
$s \leftarrow \frac{1}{1-\gamma}$
$k \leftarrow \frac{1}{\Delta}\log(\frac{2}{\gamma}) - 1$
$\phi \leftarrow s\text{tanh}(k(\ddot{\mathbf{F}}_i - m))$ ▷ asymptotic function
$\hat{\mathbf{F}}_i \leftarrow l_Q + \Delta(i + \frac{1}{2}(\phi + 1))$

The decoder consists of 2 fully-connected layers. The first layer takes the 128-dimensional quantized 8-bit unsigned integer features from the DSQ and outputs 128-dimensional 32-bit floating point features. The second layer transforms the features further to 256-dimensional floats which are used for feature matching during runtime localization after compression.

Quantized AP-GeM Global Descriptor: For the coarse image retrieval based localization, we compress and store the global descriptor of one frame from each cluster. We quantize the AP-GeM [11] descriptor of an image using the same technique described in sub-section **Feature Quantization**. Specifically, we apply the encoder MLP to the 2048-dim AP-GeM descriptor which converts it to a 1024-dim float feature. Next, a DSQ [4] layer quantizes the feature to 1024-dim 8-bit integers. A decoder symmetric to the encoder is used to recover the original information from the quantized feature. To train the quantization model, we fix the parameters of the AP-GeM backbone and use APLoss [11] to update weights of the quantization MLPs. At each iteration, we take 10 samples from the training set of *RobotCar Seasons* dataset for computing APLoss while making sure that there are at least two positive samples. Additionally, hard-negative mining [17] is applied every 5 epochs. We train the model for 30 epochs in total.

Note that only the quantized global descriptors are stored for runtime localization and the decoder network is applied to the quantized global descriptors only when registering a query image.

B. Training Details

Training Data: Our training data is sampled from the *RobotCar Seasons* [8] dataset. Since the dataset captures a large city-scale scene, exhaustively computing pairwise co-visibility distances between the image frames is expensive. We therefore sample a subset of reference images where any two frames are at least 6m and 45°apart before applying clustering based on the co-visibility distance. The subset sampling is also done using the same hierarchical clustering technique [10]. We first group the frames using their pairwise camera distance and then further split the groups based on their angular distance. From each group, we only keep one frame that is closest to the group's centroid. Next, we use co-visible frames clustering (Section 3.1 in main paper) with maximum inter-cluster distance 10 on the selected subset. If any group has less than 20 frames we add images that are co-visible to the existing frames in the group from the images outside of the selected subset to ensure that each cluster has at least 20 frames.

We use hierarchical clustering because it does not require us to set a prespecified number of clusters. Instead, a tree structure of cluster hierarchies is generated based on pairwise distances between the frames. The cluster hierarchies are "flattened" [19] into independent clusters by enforcing a maximum inter-cluster distance.

Query frames: While the co-visible clusters are sampled only from the reference set `overcast-reference`, the training set contains different scenarios such as `dawn`, `dusk`, `night` etc. Therefore, for robustness against different weather conditions we use images from the training set as query frames. Note that the number of images in the training set is quite limited and there might not be enough query frames for every co-visible cluster. For any cluster that has no query image from the training set, we use images from the co-visible set (at most 5) that were not used as anchor or verification frames as query frames.

Training Process: Since compressing a scene point cloud using the differentiable QP solver [1] is time-intensive, we train our network in two stages. In the first stage, we only train the point transformer that computes the distinctiveness scores $\mathbf{d}$ using the loss L_d (Equation 6 of the main document) for 30 epochs. This stage does not require us to solve the QP problem. In the second stage, we train our entire network with both L_d and L_ξ losses for 15 more epochs or until convergence. The loss L_ξ utilizes the $\boldsymbol{\alpha}$ estimated by the QP solver (Equation 7 of the main paper). We use $H = 64$ hypotheses, with each consists of 4 sampled 2D-3D correspondences, and an inlier threshold of 12 pixels while solving the PnP problem [6] for each hypothesis.

	K.College	S.Facade	O.Hospital	St.Church	Aachen
3D Points	230K	61K	214K	308K	1.3M
DB Frames	1220	231	895	1487	4328
Co-visible clusters	14	6	24	46	1935
Query Frames	343	103	182	530	922

Table A. Test dataset statistics

C. Registration of Query Frames

Test Dataset: After training, we test our model on the *Cambridge Landmarks* [5] and *Aachen Day-Night* [15] datasets. We use the training set of these datasets as reference frames and follow the technique described in Section B to sample co-visible frames. The *Cambridge Landmarks* contains 1080P video sequences and covers small to medium scale outdoor scenes. The larger *Aachen Day-Night* dataset on the other hand is captured using sparsely distributed cameras at image resolution 1600x1063 pixels. We therefore do not apply the reference frames pre-sampling procedure of Section B for this dataset. The dataset statistics is listed in Table A.

For each co-visible cluster, we store: 1) 1024 bytes global descriptor of the frame that has the most observed 3D points; 2) selected 3D points' positions and quantized descriptors.

Query Frame Registration: We match and register a query frame $\mathbf{I}_Q$ to the compressed scene. Our registration pipeline is similar to hloc [13]. First, we retrieve the co-visible clusters that have similar appearance to the query frame $\mathbf{I}_Q$. We follow [13] and select 15 closest candidates for the *Cambridge Landmarks* scenes and 50 candidates for the larger *Aachen Day-Night* scenes. Next, we use SuperGlue [14] matcher to establish 2D-3D correspondences between the query image $\mathbf{I}_Q$ and 3D points from from each candidate cluster. RANSAC+PnP [6] is then applied to register the query frame against each cluster individually. We use the implementation of COLMAP [16] for RANSAC+PnP with an inlier re-projection error threshold of 12 pixels. Finally, we use all the 2D-3D correspondences from the clusters that were successfully registered to obtain the query frame camera pose ξ_Q with another RANSAC+PnP.

D. Additional Ablation Studies

Size/Performance comparison with uncompressed representation: Table B shows the size/performance of uncompressed and compressed representations on `St.M Church` scene. '*Uncompressed*' uses the full image database along with all the raw (1024 bytes) SuperPoint features. For '*Compressed*', we first perform co-visible clustering to generate clusters, and select points with different compression ratios from each cluster. The selected points' features are further quantized to 128-bytes. When the scene is compressed to 11.2 MB, the accuracy is comparable to the uncompressed 2009 MB representation.

	Compressed					*Uncomp.*
Size (MB)	1.18	1.93	3.51	6.63	11.85	2009
Recall (%)	86.9	91.3	93.0	94.2	94.8	95.1

Table B. Percentage of query frames localized within 0.25m translation error and 1° rotation error when using compressed/uncompressed representations on `St.M Church`.

Our	QP+R.Sift [9]	KC [7]	KCP [2]
484	18	328	798

Table C. Compression time (seconds) of different methods.

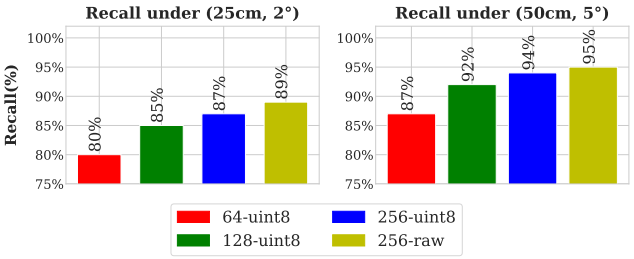

Figure B. **Localization accuracy with different quantized feature dimensions.** We evaluate the performance of 64, 128 and 256 dimensional quantized features along with the 256-dim raw point features for reference.

Feature Quantization: To study the impact of feature quantization, we trained three auto-encoders that produce 64, 128 and 256 dimensional quantized 8-bit integer features, denoted by `64-uint8`, `128-uint8` and `256-uint8` respectively. These quantized feature are used in our localization pipeline to estimate query image poses. The compression ratio is set to 1.5% for all cases. We evaluate the final localization accuracy using the metric recall under $(25cm, 2°)$ and $(50cm, 5°)$ pose error on the `Shop Facade` scene. We also include the performance of raw point features without quantization (denoted as `256-raw`) for reference. The results are shown in Figure B.

As can be seen from Figure B, the performance of `256-uint8` is slightly lower than `256-raw`. Reducing the dimensions by half (`128-uint8`) further decreases the accuracy, and we see considerable deterioration with `64-uint8`, especially for recall under $(25cm, 2°)$ error. Hence, in our framework we choose `128-uint8` quantized features because it provides a good tradeoff between memory usage and localization performance.

Compression Time: In Table C we report the time required for scene compression by our system (compression ratio 1.5%), QP+R.Sift [9], KC [7] and KCP [2] on the `St.Marys Church` scene from the *Cambridge Landmarks* dataset. For QP+R.Sift, KC and KCP we present the compression time reported by [9]. Our compression time is higher than QP+R.Sift because the computation of **d** and **K**

	Time
Co-visible Frames Clustering	14s
Computing **d** and **K**	165s
Solving QP	319s
Total Compression	484s

Table D. Time taken by each step to process `St.Mary's Church` scene, note that the time for computing **d**/**K** and solving QP considers all 46 clusters, both of which occupy large portions of the total time.

	Top-5	Top-15	Top-30	Top-50
Time (sec.)	0.7	1.2	1.9	2.2

Table E. Query frame registration time w.r.t top-k candidates.

is more involved with the Transformer while the QP+R.Sift uses more efficient handcrafted rules. In addition, we use a custom, unoptimized version of their QP solver written in python while they use a closed source C++ implementation; Those however do not hinder the application of our method in practical situations since the scene compression is done offline and an end-user directly uses the compressed scene for online localization. The detailed breakdown of the time required (in seconds) is listed in Table D.

Registration Time: Our framework requires 1.2 seconds to register a query frame. This includes retrieval of top-15 cluster candidates (0.16s), aggregating 2D-3D matches using SuperGLUE against the 15 cluster candidates (1.02s), and registering the query frame using RANSAC + PnP on all 2D-3D correspondences (0.02s). Our runtime performance is mainly determined by the number of top-k candidates selected from image retrieval. The query frame registration time with different 'k' are reported in Table E.

References

[1] Akshay Agrawal, Brandon Amos, Shane Barratt, Stephen Boyd, Steven Diamond, and J Zico Kolter. Differentiable convex optimization layers. In *NeurIPS*, 2019. 2, 3

[2] Song Cao and Noah Snavely. Minimal scene descriptions from structure from motion models. In *CVPR*, 2014. 4

[3] Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Superpoint: Self-supervised interest point detection and description. In *CVPRW*, 2018. 1

[4] Ruihao Gong, Xianglong Liu, Shenghu Jiang, Tianxiang Li, Peng Hu, Jiazhen Lin, Fengwei Yu, and Junjie Yan. Differentiable soft quantization: Bridging full-precision and low-bit neural networks. In *CVPR*, 2019. 2, 3

[5] Alex Kendall, Matthew Grimes, and Roberto Cipolla. Posenet: A convolutional network for real-time 6-dof camera relocalization. In *ICCV*, 2015. 4

[6] Vincent Lepetit, Francesc Moreno-Noguer, and Pascal Fua. Epnp: An accurate o (n) solution to the pnp problem. *IJCV*, 81(2):155–166, 2009. 3, 4

[7] Yunpeng Li, Noah Snavely, and Daniel P Huttenlocher. Location recognition using prioritized feature matching. In *ECCV*, 2010. 4

[8] Will Maddern, Geoffrey Pascoe, Chris Linegar, and Paul Newman. 1 year, 1000 km: The oxford robotcar dataset. *The International Journal of Robotics Research*, 36(1):3–15, 2017. 3

[9] Marcela Mera-Trujillo, Benjamin Smith, and Victor Fragoso. Efficient scene compression for visual-based localization. In *3DV*, 2020. 2, 4

[10] Daniel Müllner. Modern hierarchical, agglomerative clustering algorithms. *arXiv preprint arXiv:1109.2378*, 2011. 3

[11] Jerome Revaud, Jon Almazán, Rafael S Rezende, and Cesar Roberto de Souza. Learning with average precision: Training image retrieval with a listwise loss. In *ICCV*, 2019. 3

[12] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*. Springer, 2015. 2

[13] Paul-Edouard Sarlin, Cesar Cadena, Roland Siegwart, and Marcin Dymczyk. From coarse to fine: Robust hierarchical localization at large scale. In *CVPR*, 2019. 4

[14] Paul-Edouard Sarlin, Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Superglue: Learning feature matching with graph neural networks. In *CVPR*, 2020. 1, 4

[15] Torsten Sattler, Tobias Weyand, Bastian Leibe, and Leif Kobbelt. Image retrieval for image-based localization revisited. In *BMVC*, 2012. 4

[16] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. In *CVPR*, 2016. 4

[17] Kah-Kay Sung. Learning and example selection for object and pattern detection. 1996. 3

[18] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017. 1

[19] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. https://docs.scipy.org/doc/scipy/reference/generated/scipy.cluster.hierarchy.fcluster.html. 1, 3

[20] Hengshuang Zhao, Li Jiang, Jiaya Jia, Philip HS Torr, and Vladlen Koltun. Point transformer. In *ICCV*, 2021. 2