THE MISSING LINK: CERTIFYING AND REPAIRING APPROXIMATE DENSITY-BASED CLUSTERING

Anonymous authors
Paper under double-blind review

ABSTRACT

Density-based clustering identifies clusters of arbitrary shape and of different sizes. Its clusterings are derived from a minimum spanning tree (MST) over all pairwise distances: cutting the tree at a distance threshold gives a clustering, as in DBSCAN, and the whole tree defines a hierarchy of clusterings, as in HDBSCAN. For large datasets, scalable methods approximate the MST and build it only from the distances between nearest neighbors. Such an approximate MST can miss connections between points, and the returned clustering can then differ from the exact one without any indication. We show how to certify that a clustering obtained from an approximate MST equals the exact one, and how to repair it if it does not, with a search that examines no pair of points inside a cluster and skips pairs of clusters that lie further apart than the threshold. The key observation is that every edge of an approximate MST is a true connection between two points, so the approximation can omit connections but never invent them: a missing link can split an exact cluster into parts, but can never merge two exact clusters. Thus, it is sufficient to check for missing edges: pairs of close points that lie in different clusters but belong to the same exact cluster. If this check finds no such pair, the clustering is proven correct. Otherwise, we add the missing edges, merge the clusters they join, and check again; the number of such rounds grows only logarithmically with the number of parts into which a cluster was split. The check works on the output of any method that uses an approximate MST. On 336 approximate trees of image embeddings, all 32 failed clusterings are repaired in a single round. At 2.7 million iNaturalist images, the whole pipeline takes 87 seconds on six CPU threads, less than an exact library needs for its tree alone; the check and the repair account for 4 of them. Finally, we extend the check from a single clustering to the whole hierarchy of clusterings that the MST defines.

1 INTRODUCTION

Density-based clustering defines a cluster as a dense region of the data that is separated from other dense regions by sparse ones (Ester et al., 1996; Campello et al., 2013). Clusters can therefore have arbitrary shapes and very different sizes, and points in sparse regions are treated as noise. Two points belong to the same cluster if they are connected by a chain of close points that all lie in dense regions. To capture this, density-based methods rely on two concepts: the mutual-reachability distance (Campello et al., 2013) and minimum spanning trees (MSTs). Under the mutual-reachability distance, two points are close only if they are near each other and both lie in a dense region. Cutting an MST over all pairwise mutual-reachability distances at a distance threshold ε gives the clusters of DBSCAN (Ester et al., 1996) without border points (DBSCAN*; Campello et al., 2013). HDBSCAN selects the most stable clusters over all thresholds, and other criteria select further structures from the same tree (Draganov et al., 2025). In this paper, a clustering is the result of cutting the tree at a threshold ε; we return to HDBSCAN's selection in Section 4.3.

A direct computation of the exact MST evaluates all pairwise distances, which becomes infeasible as datasets grow. Exact geometric algorithms avoid this in low dimensions (March et al., 2010; Wang et al., 2021), but most scalable methods approximate the MST (Dell'Amico, 2019; Okkels et al., 2025): they connect each point only to a few nearest neighbors, which are often found by approximate search themselves, and compute the MST of this much smaller graph. However, the approximate MST may lack edges that the exact MST contains. Two groups of points that belong

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

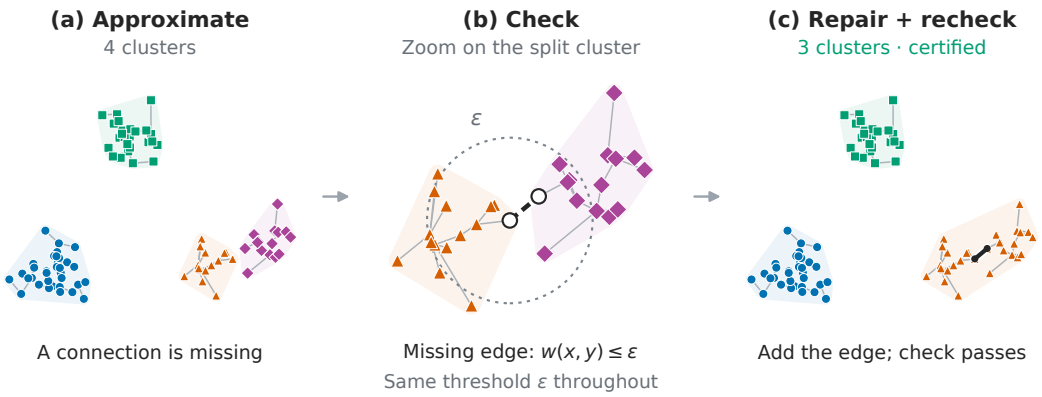

Figure 1: Check and repair at the same fixed threshold ε. (a) An approximate clustering splits one exact cluster into two. (b) A magnified view shows the missing edge found by the check: $w(x, y) \leq \varepsilon$ (dotted circle: radius ε around one endpoint). (c) Adding this edge merges the two parts; the repeated check certifies all three clusters. Gray edges are within-cluster spanning trees; colors, marker shapes and faint shading indicate clusters.

to one exact cluster can then end up in different clusters (Figure 1a). The output gives no sign of the error, because it is still a valid clustering. This paper addresses the resulting problem: given a clustering from an approximate MST with exact edge weights, decide whether it is the exact one, and repair it if it is not.

Density-based clustering is used in many domains and modalities (Campello et al., 2020), and increasingly on learned embeddings of large image and text collections (Beer et al., 2024; Grootendorst, 2022). At this scale, approximate MSTs are the norm. Even the widely used `hdbscan` library accepts an approximate MST in its default setting (McInnes et al., 2017; McInnes & Healy, 2017). The resulting errors are silent. On an embedding of $500{,}000$ iNaturalist images, the `hdbscan` default splits 12 exact clusters and changes 2.7% of the final labels, yet the adjusted Rand index (ARI) with respect to the true classes stays the same to three decimals (Table 27): the index counts pairs of points, and the split moves only $2{,}825$ points, 0.6% of the data, out of their exact clusters. Without a check, a reported clustering depends on the implementation and its approximation, and not only on the data and the definition of the algorithm.

A check seems to require the very computation that approximate methods want to avoid. The direct approach computes the exact MST and compares the two clusterings, which brings back all pairwise distances: for $500{,}000$ points, more than 10^{11} pairs. Sampling cannot replace this comparison, because a single missing edge among these pairs changes the clustering; a proof must rule out every one of them without enumerating them. In addition, many methods choose the distance threshold from the tree itself, so a check at one threshold does not cover that choice, and real data contains pairs with exactly equal distances, where floating-point rounding can flip a comparison. Finally, a check is only useful if it costs far less than the method it verifies.

The question we answer in this paper is whether one given clustering equals the exact one, and how to repair it if not. Existing work answers different questions. Approximate methods give no guarantee for an individual run (McInnes & Healy, 2017; Dell'Amico, 2019; Okkels et al., 2025), and high-probability guarantees, on cluster recovery under assumptions on the data (Jiang et al., 2020) or on the approximation of the tree (Moseley et al., 2021), do not tell whether one particular output is exact. Exact algorithms (March et al., 2010; Wang et al., 2021; McInnes, 2026) cannot check a clustering that another method produced, and classical MST verification (Komlós, 1985; King, 1997) assumes an explicit graph, whereas our graph has $N(N-1)/2$ edges and is never built. Our check works after the fact, on the output of any method. In the terminology of certifying algorithms (McConnell et al., 2011), it returns either a proof or a counterexample, and the counterexample repairs the result (see Section 5).

Our approach builds on one key observation. Approximate search is approximate in which pairs it finds, not in their distances: every pair that it returns is measured exactly, and the core distances are computed exactly. The edges of the approximate MST therefore carry their true mutual-reachability distances, and the approximate clustering can only split exact clusters, never merge them: the tree edges inside each cluster prove that its points belong together. The only possible error is therefore a missing edge, the missing link of our title: a pair of core points from different clusters within the distance threshold (Figure 1). We search for such pairs with a spatial index per cluster and skip pairs of clusters whose bounding boxes are too far apart. If the search finds no missing edge, the check passes and the clustering is proven exact. If it finds some, we merge the clusters they join and check again; we call one such pass a round. One missing edge per cluster and round suffices: every part of a split cluster is then merged with another part, so the number of parts at least halves, and the number of rounds grows only logarithmically with the number of parts into which a cluster was split (Section 3.2). A hierarchy check extends the same search to all thresholds above a chosen one, which covers methods that choose the threshold from the approximate tree (Section 3.3).

We build 336 approximate trees from NN-Descent (Dong et al., 2011) and hierarchical navigable small-world (HNSW) graphs (Malkov & Yashunin, 2018), the neighbor graphs on which graph-based approximate methods build, on embeddings of three image datasets, the largest with 500,000 images from iNaturalist, and we check the default output of the `hdbscan` library on six datasets (Table 19). The check agrees with an independent exact computation for every tree, and the repair corrects all 32 failed clusterings in one round. On all 2.7 million training images of iNaturalist, the pipeline takes 87 s, of which the check and the repair take 4 s; an exact library needs 135 s for its tree alone, and a direct computation of the clustering at a known threshold 109 s (Section 4). The hierarchy check, which also certifies the choice of the threshold, passes for 130 of the 336 trees, but for none with at least 500,000 points: there it fails or reaches our work limit. We find that HDBSCAN's cluster selection depends on the order in which implementations process merges at equal distances; a fixed tie rule reduces the largest discrepancy between approximate and exact trees in our experiments from .37 to .01 NMI (Section 4.3). Two limitations remain: the check certifies exactness with respect to the definition of the algorithm, not whether the clusters are meaningful, and it needs exact core distances, which cost far more than the check itself (5.4 against 0.09 s at 500,000 points).

In summary, our main contributions are:

- A check that decides, without building the complete graph, whether a clustering from an approximate MST equals the exact clustering at a distance threshold, and a repair that reaches the exact clustering in a logarithmic number of rounds (Sections 3.1 and 3.2).
- The theory behind both: approximation can only split exact clusters (Theorem 1), and a finite-sample condition under which the check passes (Theorem 2).
- A hierarchy check for methods that do not fix the threshold in advance but choose it from the tree, for example to obtain a given number of clusters (Beer et al., 2023; Draganov et al., 2025); it certifies all clusterings above the chosen threshold at once and returns pass, fail or, if a work limit is reached, inconclusive (Section 3.3).
- An evaluation on 336 approximate trees of three image datasets and on up to 2.7 million images (Section 4). It shows how often approximate trees give wrong clusterings, that the check finds every such case, what the check and the repair cost compared with building the tree, and that quality scores do not reveal wrong clusterings.

2 PRELIMINARIES

Let $X = (x_1, \ldots, x_N)$ be N points in $\mathbb{R}^d$, duplicates allowed, and let $\|x - y\|$ denote the Euclidean distance of two points. The statements of Section 3 hold for any metric; only the search structures, kd-trees and bounding boxes, assume vectors, and a metric index could replace them. Density is measured with one parameter μ, $1 \leq \mu < N$, which the `hdbscan` library calls `min_samples`. The core distance $\kappa_\mu(x)$ of a point x is the distance from x to its μ-th nearest other point. It is small in dense regions and large in sparse ones. The mutual-reachability (MR) distance (Campello et al., 2013) of two points is

$$w(x, y) = \max\{\kappa_\mu(x), \kappa_\mu(y), \|x - y\|\}. \tag{1}$$

Two points are close in MR distance only if they are close to each other and both lie in dense regions.

Exact clustering. Let G be the complete graph on X in which the edge between x and y has weight $w(x, y)$. G has $N(N-1)/2$ edges, and our method never builds it. For a distance threshold $\varepsilon \geq 0$, cutting G at ε means removing every edge of weight greater than ε. The connected components of the remaining graph are the clusters, and together they form the exact clustering $P_G(\varepsilon)$ at threshold ε. A point with $\kappa_\mu(x) \leq \varepsilon$ is a core point. A point with $\kappa_\mu(x) > \varepsilon$ is connected to no other point, because every edge at x has weight at least $\kappa_\mu(x)$; $P_G(\varepsilon)$ lists it as a cluster of size one, and DBSCAN* calls it noise. $P_G(\varepsilon)$ is the clustering of DBSCAN* (Campello et al., 2013), that is, of DBSCAN (Ester et al., 1996) without border points, in which a core point has at least μ other points within distance ε. Let T be a minimum spanning tree (MST) of G. Cutting T at ε gives the same clusters as cutting G at ε, for every ε (Gower & Ross, 1969; Campello et al., 2013), so the $N-1$ edges of T carry the complete clustering hierarchy. The largest edge weight on the path in T between two points is their density-connectivity distance $\mathrm{dc}_T(x, y)$ (Beer et al., 2023): the smallest threshold at which the two points belong to the same cluster.

Cluster extraction. There are several ways to read a flat clustering from T. The simplest is to cut T at a threshold ε, which gives $P_G(\varepsilon)$; this is the clustering that we certify. The threshold can be given, as in DBSCAN*, or chosen from the tree, for example so that a given number k of clusters results (Beer et al., 2023; Draganov et al., 2025). We call edges of equal weight tied. A cut at a threshold removes *all* tied edges, so the number of clusters can jump past k (our selection criterion: Section 3.3). HDBSCAN instead selects clusters from the whole hierarchy with the excess-of-mass (EOM) criterion: it condenses the hierarchy and keeps the clusters of largest persistence (Campello et al., 2013; 2015). A pass of the check at one threshold makes no statement about the clustering that EOM selects, and EOM implementations can depend on the order of tied merges (Section 4.3).

Approximate trees. A direct computation of T evaluates all pairwise distances. Scalable methods instead build an approximate nearest-neighbor (ANN) graph, in which an index such as NN-Descent (Dong et al., 2011) or HNSW (Malkov & Yashunin, 2018) joins each point to k_{nn} close points, and compute the MST of this graph. We call the result an approximate tree $\widehat{T}$. The index approximates the Euclidean neighbors, not the MR distance, and it only proposes pairs; the weight Equation (1) of each proposed pair is computed from the exact distances, which we compute separately from the graph. Hence every edge of $\widehat{T}$ is an edge of G with its exact weight, and $\widehat{T}$ is a spanning tree of G, but not necessarily a minimum one: the ANN graph may lack edges of an exact MST. The parameters μ and k_{nn} are therefore independent: μ enters only through the core distances, and k_{nn} only sets how many candidate pairs the graph contains; our experiments use $\mu \in \{5, 10\}$ and $k_{\mathrm{nn}} \in \{10, 30, 90\}$.

3 Certifying and repairing a clustering at a fixed threshold

Our own pipeline builds the approximate tree as follows. An ANN graph with k_{nn} neighbors per point supplies candidate pairs. We symmetrize the graph by keeping a pair if either point lists the other, and we weight every pair with the MR distance of Equation (1), computed from exact core distances. If the graph is disconnected, we add one bridge edge for every pair of its components: an edge of G with its exact MR weight, chosen among points near the boundaries of the two components (Section G). The approximate tree $\widehat{T}$ is the MST of this graph. We make no further assumptions about the quality of the ANN graph or of the bridge edges. All weights of $\widehat{T}$ are exact, so its only possible defect is a missing edge, and Theorem 1 shows that missing edges can only split clusters of the exact clustering, never merge them. The check itself does not depend on this construction: it accepts a tree generated by any method, and its result is correct provided that the edges inside each cluster carry their exact MR weights or upper bounds of them.

3.1 Searching for missing edges

Proposition 1 (Certificate at one threshold)**.** *Let P be a clustering of X, and suppose that every cluster of P has a spanning tree whose edges have MR weight at most ε. Then $P = P_G(\varepsilon)$ if and only if no two points x, y from different clusters of P satisfy*

$$\kappa_\mu(x) \leq \varepsilon, \qquad \kappa_\mu(y) \leq \varepsilon, \qquad \|x - y\| \leq \varepsilon, \tag{2}$$

that is, if and only if there is no missing edge.

The proposition follows from two observations (proof: Section A). The spanning trees keep every cluster of P inside one cluster of $P_G(\varepsilon)$ (Figure 1a); and if two clusters of P lay inside the same cluster of $P_G(\varepsilon)$, a path of edges with weight at most ε would lead from one to the other, and one of its edges would join two clusters of P, that is, it would be a missing edge. The spanning trees are essential: they prove that each cluster is connected at threshold ε. Without them, a clustering that wrongly merges two exact clusters has no missing edge and would pass.

To search for missing edges (Figure 1b), we first select the core points, group them by cluster, and build a kd-tree (Bentley, 1975) and a bounding box for the core points of each cluster. Only pairs of clusters whose boxes lie within distance ε of each other are searched. For such a pair, range queries with radius ε from the points of the smaller cluster into the kd-tree of the larger one enumerate candidate pairs, and Equation (2) decides for each candidate whether it is a missing edge. The geometric search can take quadratic time in the worst case. An alternative is to enumerate all pairs of core points within distance ε and to compute the connected components of this graph directly; we use it as an independent reference (Section 4.2). Our theorems are stated for real arithmetic; the implementation fixes a float64 convention and pads the search radius by a relative 10^{-9}, and a candidate pair in this padding band makes the result inconclusive instead of a pass (Section B).

3.2 Repair at a fixed threshold

A missing edge joins two clusters of the given clustering. To repair the clustering, we add the missing edges that the check found, merge the clusters they join, and check again at the same threshold ε (Figure 1c). We call one such pass a round. In every round, the search looks for one missing edge per cluster. A cluster can be without a missing edge, but only a complete search of its candidate pairs establishes this. Algorithm 1 (Section C.2) lists the steps and the three possible results.

Lemma 1 (number of rounds)**.** *Suppose that every cluster of the current clustering lies inside one cluster of $P_G(\varepsilon)$, and let $p_{\max}$ be the largest number of current clusters inside one cluster of $P_G(\varepsilon)$. If every round finds a missing edge for every cluster that has at least one, the repair reaches $P_G(\varepsilon)$ within $\lceil \log_2 p_{\max} \rceil$ rounds.*

Each part of a split cluster of $P_G(\varepsilon)$ has a missing edge to another part, so every part is merged with at least one other part in the round, and the number of parts at least halves; this is the contraction argument of Borůvka's algorithm. The bound needs two conditions: every cluster with a missing edge must receive one in every round (a global limit on the edges per round can leave parts waiting for many rounds, although every merge remains valid), and the threshold must stay fixed (if a selection criterion chooses a new ε after every round, the repair still terminates, provided that no work limit stops it, but the bound does not apply; Lemma 3).

3.3 The hierarchy check

So far the threshold ε was given. In practice, it is often chosen from the tree by a selection criterion, and a pass of the check at the selected threshold does not certify this choice: on the exact tree, the criterion might have selected a different threshold. The hierarchy check closes this gap with one search that covers every threshold the criterion inspects, so that the selected clustering and its labels are certified together. Our criterion sweeps the thresholds of $\widehat{T}$ and selects the highest threshold at which the number of clusters with at least m points equals a target k, or comes closest to it. It keeps up to k of these clusters, the largest ones, and every other point receives the label of its nearest assigned point, so that the final clustering labels every point (details and tie rule: Section A).

We denote the selected threshold by t. The check of Section 3.1, run with $\varepsilon = t$, certifies the complete clustering at this threshold, before clusters are kept and points are reassigned. These two steps depend only on the certified clustering and on the coordinates, so they need no further check. To certify the selection, the approximate and the exact tree must give the same clustering at every threshold that the criterion inspects. Repeating the check, with strict inequality, at every distinct edge weight above t is sufficient (Corollary 1 and Lemma 4) but expensive. We instead use one search for all thresholds $s \geq t$.

Proposition 2 (restricted cycle condition). *Let $\widehat{T}$ be a spanning tree of G with exact MR weights, and let $\mathrm{dc}_{\widehat{T}}(i,j)$ be the largest edge weight on the path between i and j in $\widehat{T}$. Cutting $\widehat{T}$ and cutting G give the same clustering at every threshold $s \geq t$ if and only if, for every pair of points i, j,*

$$\mathrm{dc}_{\widehat{T}}(i,j) \leq \max\{w(i,j), t\}. \tag{3}$$

The condition compares two ways of connecting i and j: the direct edge of G and the path in $\widehat{T}$. Above t, the path must never contain a heavier edge than the direct edge. This is the cycle property of minimum spanning trees, restricted to thresholds above t and evaluated on a graph that is never built. We prove this in Section C.

The search does not enumerate all pairs. It examines pairs of nodes of a hierarchical index that follows the merges of $\widehat{T}$: every node holds the points that one merge joins, their bounding box, and their smallest core distance, which give a lower bound on $w(i,j)$ and an upper bound on $\mathrm{dc}_{\widehat{T}}(i,j)$ for all pairs between two nodes. If the bounds imply Equation (3), all these pairs are settled at once; otherwise a node is split into its children (see Section C for more details). The index needs $O(N \log N + Nd)$ memory, the work remains quadratic in the worst case, and pairs of nodes that are undecided at the work limit make the result inconclusive.

If the target k is reached at t, a pass for all thresholds $s \geq t$ certifies the selected clustering and the labels of the unassigned points. If k is not reached, the criterion depends on the whole sweep, and the hierarchy check has to start at zero. A guarantee for HDBSCAN's EOM selection needs more: the complete hierarchy and a selection that is determined by this hierarchy under a stated tie rule (see Section 4.3).

4 Experiments

In extensive experiments, we investigate four aspects of the check and the repair. (1) Soundness (Section 4.1): how often does an approximate tree lead to a wrong clustering, and does the check detect every such case ? (2) Running time (Section 4.2): how many rounds does the repair need, and what do the check and the repair cost compared with building the tree? (3) Ties (Section 4.3): how much of the difference between HDBSCAN's EOM selection on approximate and on exact trees is caused by the approximation, and how much by ties? (4) Quality scores (Section 4.4): does a wrong clustering show in quality scores?

Our real-world experiments comprise three image datasets: STL-10 (Coates et al., 2011) with $13{,}000$ images, CIFAR-100 (Krizhevsky, 2009) with $60{,}000$, and iNaturalist-2021 (Van Horn et al., 2021) with $500{,}000$. The target k and clustering quality are based on their class labels. We use two frozen backbones, DINOv2 and DINOv3 (Oquab et al., 2024; Siméoni et al., 2025), two encoders that map the backbone features to 10 dimensions, and five random seeds, leading to 20 embeddings per dataset. At this dimension, the exact references and core distances remain computable at $500{,}000$ points; higher dimensions: Sections E.4 and 6. We use an autoencoder (AE) and an encoder trained with a density-based adaptation of the supervised contrastive loss (SupCon) (Khosla et al., 2020) as encoders; details, settings, software versions and file hashes: Sections D, F and I. The experiments on the complete training sets of iNaturalist-2021 and ImageNet-1k (Deng et al., 2009; Russakovsky et al., 2015) (Section 4.2) use separate embeddings of all 2.7 and 1.28 million images (Sections E.2 and E.3).

4.1 Check results on approximate trees

This experiment measures how often an approximate tree leads to a clustering that differs from the exact one, and whether the check detects every such case (Table 1). For each of the 20 embeddings per dataset and each $\mu \in \{5, 10\}$, we compute an exact reference with the exact Borůvka mode of the `hdbscan` library, which builds the MST of the complete graph without a neighbor graph, and recompute its edge weights with the float64 expression of Section B: 40 exact references per dataset. All thresholds are selected by the criterion of Section 3.3 with k equal to the number of classes (10, 100 and 273) and $m = 10$; every reference clustering passes the check. We then build approximate trees from NN-Descent and HNSW graphs with $k_{\mathrm{nn}} = 30$, two per reference (240 in total), and, on the four embeddings of seed zero per dataset, with $k_{\mathrm{nn}} = 10$ and 90 (96 in total); every ANN graph is built independently (Section D).

Table 1: Check and hierarchy check on 336 approximate trees at the threshold selected on each tree ($k = \#$ classes, $m = 10$; 16, 80 and 16 trees per dataset for $k_{\mathrm{nn}} = 10$, 30 and 90); every result of the check agrees with the exact reference. Inconclusive: a work limit was reached (Section C).

		check		hierarchy check		
Dataset	k_{nn}	pass	fail	pass	fail	inconclusive
STL-10	10	10	6	0	16	0
	30	77	3	56	24	0
	90	16	0	12	4	0
CIFAR-100	10	8	8	2	14	0
	30	78	2	50	8	22
	90	16	0	10	2	4
iNaturalist	10	4	12	0	16	0
	30	79	1	0	39	41
	90	16	0	0	1	15
All		304	32	130	124	82

The check passes for 304 trees and fails for the other 32, and every result agrees with the exact reference. With $k_{\mathrm{nn}} = 30$, 234 of 240 trees pass; with $k_{\mathrm{nn}} = 10$, only 22 of 48; with $k_{\mathrm{nn}} = 90$, all 48. With $k_{\mathrm{nn}} = 10$, the share of failed checks grows with the size of the dataset: 6, 8 and 12 of 16 trees fail at 13,000, 60,000 and 500,000 points. Wrong clusterings are therefore rare at a common setting and frequent when few neighbors are used. However, users cannot tell the two situations apart without a check. The hierarchy check passes for 130 trees, and in each of these cases the final clustering equals the one from the exact reference. It passes for no tree with at least 500,000 points within the work limits.

Exact trees. The hierarchy check also applies to trees from exact methods: on the 120 exact references, 64 checks pass, 10 fail and 46 are inconclusive. In the ten failed cases, the library's tree contains an edge that is heavier than a competing edge by one floating-point step once all weights use the same float64 expression, which changes the clustering only between the two weights and does not affect this section. An exact algorithm thus does not by itself guarantee the exact hierarchy under a fixed floating-point convention; a check does. On 1,000 small random problems with ties and duplicates, all 23,368 evaluations of Equation (3) and all 6,630 passing hierarchy checks agree with a brute-force computation (Section B).

4.2 Running time of the repair and of the check

Repair. The repair corrects all 32 clusterings that failed the check in Section 4.1: the threshold stays fixed, an exact cluster was split into two to four parts, and after one round every repaired clustering passes the check and equals the exact reference (Section E.1). To test the repair under heavier damage, we deliberately damage 300 neighbor graphs of synthetic DENSIRED data (Jahn et al., 2024) with 20,000 and 50,000 points (Section D.3), each starting from ten clusters at a fixed threshold, and compare our repair with two variants that add at most 1,000 or 100,000 missing edges per round regardless of the cluster. In all 900 runs, the final clustering passes the check and equals the exact result; our repair needs one or two rounds, the two variants a median of nine and seven (Figure 5). A cluster is split into at most ten parts here, so the experiment does not show how the number of rounds grows for more parts. When the threshold is selected again after every round, the check still never passes wrongly, but 113 of 300 repairs reach our limit of 20 rounds (Table 28).

Running time. Figure 2a shows the running time of the check given exact core distances: 0.01, 0.01 and 0.09 s at 13,000, 60,000 and 500,000 points on six CPU threads, whereas computing the clustering directly from all pairs within distance ε takes 0.15, 0.09 and 2.62 s (all variants and sizes: Table 4). The direct computation depends on the core points at the threshold, not on N: STL-10 has 9,739, CIFAR-100 only 7,997. The exact core distances are the larger cost: computing them and running the check takes 5.51 s at 500,000 points, or 2.32 s if only core points are determined exactly.

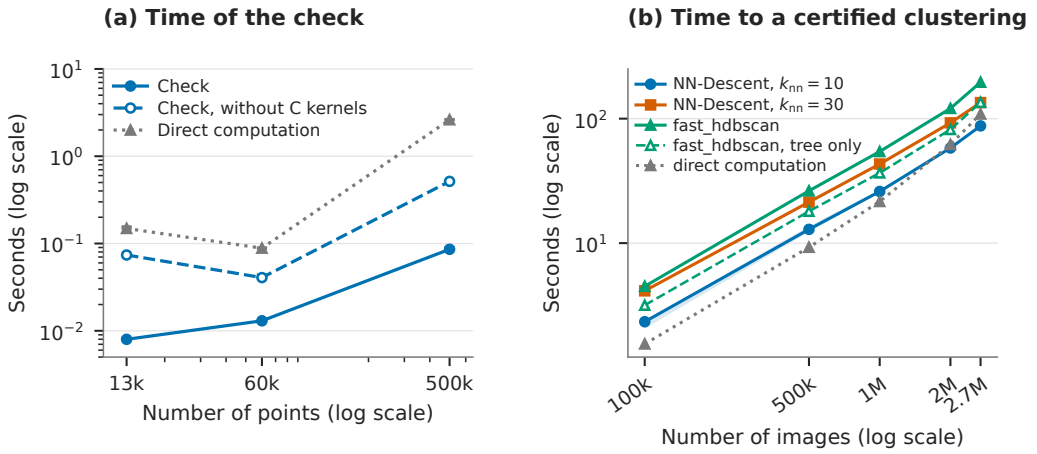

Figure 2: Running time. (a) The check, given exact core distances, and a direct computation of the clustering from all pairs within distance ε (DINOv2 AE embedding of seed zero, $\mu = 10$). (b) Time to a certified clustering at a fixed threshold, including core distances, tree construction, check and repair, on nested subsets of an embedding of all 2.7 million training images of iNaturalist-2021; the NN-Descent lines are our pipeline, the dashed line is the tree of `fast_hdbscan` alone, and the dotted line is a direct computation of the clustering at the known threshold, without any tree. Medians over three runs on six CPU threads, full ranges as bars and bands, logarithmic axes. Protocol, stages and memory: Section E.2.

Figure 2b compares the total time to a certified clustering, including core distances, tree construction, check and repair, in separate processes on six CPU threads, on nested random subsets of an embedding of all 2,686,843 training images of iNaturalist-2021 (Section E.2); 72 further runs on the three datasets are in Section E.1. Every final clustering passes the check and equals the exact reference. All runs with $k_{\mathrm{nn}} = 10$ fail the first check, with up to 27 parts per exact cluster, and are repaired in one round; with more neighbors and for the exact libraries, the first check passes. At 2.7 million points, NN-Descent with $k_{\mathrm{nn}} = 10$ takes 87 s in total: 60 s for the core distances, 24 s for the tree, and 4 s for the check and the repair. The exact library `fast_hdbscan` takes 196 s under the same protocol, of which 135 s are spent on building its tree. At every size, the certified clustering from NN-Descent with $k_{\mathrm{nn}} = 10$ is available earlier than the exact tree of the library without any check (87 against 135 s at 2.7 million points, and 27 against 53 s on ImageNet-1k, Section E.3), but the library needs less memory (1.93 against 4.17 GiB). When the threshold is known in advance, no tree is needed: a direct computation of the DBSCAN* clustering from all pairs of core points within distance ε is the fastest method up to one million points (22 against 26 s at one million), but at 2 and 2.7 million points, where more pairs lie within the threshold, it falls behind the certified clustering from the approximate tree (109 against 87 s at 2.7 million points; Figure 2b). The tree and the check also serve the case in which the threshold is selected from the tree or several thresholds are needed: given the core distances, one check costs 0.1 to 0.3 s on iNaturalist-500k, whereas the direct computation costs 3 to 119 s depending on the threshold (Table 30).

4.3 Ties and HDBSCAN's cluster selection

HDBSCAN's EOM selection is the most common way to extract clusters from the tree, and our check does not cover it (Section 3.3). How much of the difference between EOM on an approximate and on the exact tree is caused by the approximation, and how much by ties, that is, by edges of exactly equal weight? Ties do not affect the clustering at any threshold. HDBSCAN* removes tied edges simultaneously (Campello et al., 2015), but the implementations that we inspected merge tied clusters two at a time (Table 2), so the condensed tree, from which EOM selects, depends on the order of these merges. We pass each of the 240 approximate trees with $k_{\mathrm{nn}} = 30$ and its exact reference to the EOM code of the `hdbscan` library (Section D.2). With respect to the classes, the

median absolute NMI difference is 7.75×10^{-5}, the maximum is .374, and 12 differences exceed .01. We therefore fix the tie rule: all merges at the same distance are treated as one merge, which is split into pairwise merges in the order of the smallest point index. With it, the median absolute NMI difference becomes zero, the maximum drops to .011, and only two of 240 differences exceed .01 (Table 3 and Figure 4).

4.4 Exactness and clustering quality

A passed check proves a clustering exact; it does not make it better. For the 32 failed clusterings of Section 4.1, we compare the clustering of the approximate tree with the repaired one, which equals the exact clustering in all 32 cases. The errors are not always small: up to 49% of the points lie in an exact cluster that the approximate tree had split (median 1.7%), and the ARI between the two clusterings is as low as .77 (median .999). Quality scores with respect to the classes do not reveal these errors. In 24 of the 32 cases, the ARI changes by less than .001, and the direction is not consistent: the repair raises the ARI in 18 cases and lowers it in 14, on STL-10 by up to .14. Exactness is a property that quality scores cannot replace: it attributes differences between results to the data and parameters, not to the approximation. Whether density-based clustering suits a dataset is a separate question (Sections E.5 and F.1).

5 Related work

Building the tree. Approximate HDBSCAN (McInnes & Healy, 2017), FISHDBC (Dell'Amico, 2019), cuML HDBSCAN (Raschka et al., 2020), sngDBSCAN (Jiang et al., 2020), sDBSCAN (Xu & Pham, 2024), and methods that build the MST from a nearest-neighbor graph (Okkels et al., 2025) or from incremental similarity searches (Schubert, 2025) make density-based clustering faster. Geometric algorithms (de Berg et al., 2019; Wang et al., 2021; McInnes, 2026) are exact by design (`fast_hdbscan` for $d \leq 30$); our check applies to the output of both groups.

Checking a result. The condition of Proposition 2 and the repair rounds use classical ideas from MST verification and from Borůvka's algorithm (Komlós, 1985; King, 1997). The search is the bichromatic primitive of dual-tree Borůvka (March et al., 2010) on the robust single linkage graph at level ε (Chaudhuri & Dasgupta, 2010). New is what the approximate tree lets us ask of this primitive: since approximation can only split exact clusters, the tree edges inside a cluster already prove that cluster, and only the emptiness of the MR graph between clusters remains to be shown. The check therefore examines no pair inside a cluster, and its cost does not grow with the number of pairs within distance ε (Table 30). The emptiness query between two point sets is also the hard core of exact DBSCAN, for which Gan & Tao (2015) give a conditional lower bound in three and more dimensions; our search is quadratic in the worst case. The recovery theorem specializes analyses of neighborhood graphs (Maier et al., 2009) and robust single linkage (Chaudhuri & Dasgupta, 2010).

6 Limitations and conclusion

Our check proves that a clustering equals the exact clustering at a threshold ε under the given parameters. It does not choose the threshold or the number of clusters, judge whether the clusters are meaningful, or certify the clusters that HDBSCAN's EOM criterion selects. The bound on the number of rounds needs a fixed threshold and complete searches, and the implementation can return inconclusive. Our main experiments use 10-dimensional embeddings, and the selection criterion is given the number of classes. In higher dimensionalities, the exact core distances, which every exact method needs, dominate the cost (up to 3,895 s at 128 dimensions), whereas the check and the repair take at most 410 s and still find and correct every wrong clustering (Section E.4); the projection filter of Section C.4 prunes most candidate pairs of their search.

Our check either proves that a clustering from an approximate tree is exact or returns missing edges, which the repair adds; for all 32 failed clusterings of Table 1, one round sufficed, and check and repair took 4 of the 87 s at 2.7 million points. Future work includes datasets with billions of points, such as the 1.8 billion sources of Gaia DR3 (Gaia Collaboration et al., 2023), a hierarchy check that stays within its work limit at 500,000 points, and a certificate for EOM under a fixed tie rule.

Reproducibility statement

The code for all experiments is provided in an anonymous repository at `https://anonymous.4open.science/r/SLICE-2B1B`. Section I lists, for every table and figure, the script that produces it, its input files, and the hashes of the stored results. The experiments on real data use 60 embeddings and one embedding each of the complete iNaturalist-2021 and ImageNet-1k training sets, which the provided pipeline computes from the public datasets with fixed random seeds. Sections D and I give the settings, random seeds, software versions and hardware of every experiment. Runs that reached a work limit are reported as inconclusive and are included in all counts. The implementation of the check is covered by 48 unit tests, its C kernels by 85 further tests against the NumPy and SciPy code, and by a comparison with a brute-force computation on the complete graph for 1,000 small problems.

Ethics statement

This work proposes a method for verifying and repairing the output of clustering algorithms. It involves no human subjects, no private data and no new data collection: all experiments use public benchmark datasets under their published terms (STL-10, CIFAR-100, iNaturalist-2021, ImageNet-1k, MNIST, Fashion-MNIST, KMNIST, USPS, COIL-20, COIL-100, Optdigits, Pendigits and the synthetic DENSIRED generator), and the image features come from publicly released frozen backbones. We do not foresee harms specific to the method: it does not enable new applications of clustering but makes the result of an existing one independent of the approximation that produced it, which supports reproducible reporting. The hardware and the running times of all experiments are listed in Section I.

AI use statement

Language-model assistants were used as tools under the authors' direction: for writing and refactoring experiment code, including the C kernels and the scripts that generate the tables and figures from the stored records. Additionally, we used generative AI tools to edit the text for readability. The problem formulation, the theory and its proofs, the design of the experiments and the interpretation of the results are the authors' own. The authors reviewed every change to code and text, every reported number is linked to a stored record (Section I), and the assistants were not used to generate or select scientific claims.

References

Paul Balister, Béla Bollobás, Amites Sarkar, and Mark Walters. Connectivity of random k-nearest-neighbour graphs. *Advances in Applied Probability*, 37(1):1–24, 2005. doi: 10.1239/aap/1113402397.

Anna Beer, Andrew Draganov, Ellen Hohma, Philipp Jahn, Christian M. M. Frey, and Ira Assent. Connecting the dots – density-connectivity distance unifies DBSCAN, k-center and spectral clustering. In *KDD*, pp. 80–92, 2023. doi: 10.1145/3580305.3599283.

Anna Beer, Pascal Weber, Lukas Miklautz, Collin Leiber, Walid Durani, Christian Böhm, and Claudia Plant. SHADE: Deep density-based clustering. In *ICDM*, pp. 675–680, 2024. doi: 10.1109/ICDM59182.2024.00075.

Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975. doi: 10.1145/361002.361007.

Ricardo J. G. B. Campello, Davoud Moulavi, and Jörg Sander. Density-based clustering based on hierarchical density estimates. In *PAKDD*, pp. 160–172, 2013. doi: 10.1007/978-3-642-37456-2_14.

Ricardo J. G. B. Campello, Davoud Moulavi, Arthur Zimek, and Jörg Sander. Hierarchical density estimates for data clustering, visualization, and outlier detection. *ACM Transactions on Knowledge Discovery from Data*, 10(1):5:1–5:51, 2015. doi: 10.1145/2733381.

Ricardo J. G. B. Campello, Peer Kröger, Jörg Sander, and Arthur Zimek. Density-based clustering. *WIREs Data Mining and Knowledge Discovery*, 10(2):e1343, 2020. doi: 10.1002/widm.1343.

Kamalika Chaudhuri and Sanjoy Dasgupta. Rates of convergence for the cluster tree. In *NIPS*, pp. 343–351, 2010. URL https://proceedings.neurips.cc/paper/2010/hash/b534ba68236ba543ae44b22bd110a1d6-Abstract.html.

Tarin Clanuwat, Mikel Bober-Irizar, Asanobu Kitamoto, Alex Lamb, Kazuaki Yamamoto, and David Ha. Deep learning for classical Japanese literature. *arXiv:1812.01718*, 2018.

Adam Coates, Andrew Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature learning. In *AISTATS*, pp. 215–223, 2011. URL https://proceedings.mlr.press/v15/coates11a.html.

Mark de Berg, Ade Gunawan, and Marcel Roeloffzen. Faster DBSCAN and HDBSCAN in low-dimensional Euclidean spaces. *International Journal of Computational Geometry & Applications*, 29(1):21–47, 2019. doi: 10.1142/S0218195919400028.

Matteo Dell'Amico. FISHDBC: Flexible, incremental, scalable, hierarchical density-based clustering for arbitrary data and distance. *arXiv:1910.07283*, 2019. URL https://arxiv.org/abs/1910.07283.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *CVPR*, pp. 248–255, 2009. doi: 10.1109/CVPR.2009.5206848.

Wei Dong, Moses Charikar, and Kai Li. Efficient K-nearest neighbor graph construction for generic similarity measures. In *WWW*, pp. 577–586, 2011. doi: 10.1145/1963405.1963487.

Andrew Draganov, Pascal Weber, Rasmus Jørgensen, Anna Beer, Claudia Plant, and Ira Assent. Ultrametric cluster hierarchies: I want 'em all! In *NeurIPS*, pp. 128190–128241, 2025. doi: 10.52202/085713-4268. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/ba1293b663ddd4a29c6854e4d3bf766a-Abstract-Conference.html.

Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*, pp. 226–231, 1996. URL https://cdn.aaai.org/KDD/1996/KDD96-037.pdf.

Gaia Collaboration, A. Vallenari, A. G. A. Brown, T. Prusti, et al. Gaia Data Release 3: Summary of the content and survey properties. *Astronomy & Astrophysics*, 674:A1, 2023. doi: 10.1051/0004-6361/202243940.

Junhao Gan and Yufei Tao. DBSCAN revisited: Mis-claim, un-fixability, and approximation. In *SIGMOD*, pp. 519–530, 2015. doi: 10.1145/2723372.2737792.

John C. Gower and Gavin J. S. Ross. Minimum spanning trees and single linkage cluster analysis. *Journal of the Royal Statistical Society, Series C: Applied Statistics*, 18(1):54–64, 1969. doi: 10.2307/2346439.

Maarten Grootendorst. BERTopic: Neural topic modeling with a class-based TF-IDF procedure. *arXiv:2203.05794*, 2022. URL https://arxiv.org/abs/2203.05794.

Xifeng Guo, Long Gao, Xinwang Liu, and Jianping Yin. Improved deep embedded clustering with local structure preservation. In *IJCAI*, pp. 1753–1759, 2017. doi: 10.24963/ijcai.2017/243.

Jonathan J. Hull. A database for handwritten text recognition research. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(5):550–554, 1994. doi: 10.1109/34.291440.

Philipp Jahn, Christian M. M. Frey, Anna Beer, Collin Leiber, and Thomas Seidl. Data with density-based clusters: A generator for systematic evaluation of clustering algorithms. In *ECML PKDD*, pp. 3–21, 2024. doi: 10.1007/978-3-031-70368-3_1.

Heinrich Jiang, Jennifer Jang, and Jakub Łącki. Faster DBSCAN via subsampled similarity queries. In *NeurIPS*, pp. 22407–22419, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/fdf1bc5669e8ff5ba45d02fded729feb-Abstract.html.

Markelle Kelly, Rachel Longjohn, and Kolby Nottingham. The UCI machine learning repository. https://archive.ics.uci.edu, 2023.

Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. Supervised contrastive learning. In *NeurIPS*, pp. 18661–18673, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/d89a66c7c80a29b1bdbab0f2a1a94af8-Abstract.html.

Valerie King. A simpler minimum spanning tree verification algorithm. *Algorithmica*, 18(2):263–270, 1997. doi: 10.1007/BF02526037.

János Komlós. Linear verification for spanning trees. *Combinatorica*, 5(1):57–65, 1985. doi: 10.1007/BF02579443.

Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009. URL https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791.

Collin Leiber, Lena G. M. Bauer, Benjamin Schelling, Christian Böhm, and Claudia Plant. Dip-based deep embedded clustering with k-estimation. In *KDD*, pp. 903–913, 2021. doi: 10.1145/3447548.3467316.

Markus Maier, Matthias Hein, and Ulrike von Luxburg. Optimal construction of k-nearest-neighbor graphs for identifying noisy clusters. *Theoretical Computer Science*, 410(19):1749–1764, 2009. doi: 10.1016/j.tcs.2009.01.009.

Yu. A. Malkov and D. A. Yashunin. Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs, 2018. URL https://arxiv.org/abs/1603.09320v4. Version 4.

William B. March, Parikshit Ram, and Alexander G. Gray. Fast Euclidean minimum spanning tree: Algorithm, analysis, and applications. In *KDD*, pp. 603–612, 2010. doi: 10.1145/1835804.1835882.

Ross M. McConnell, Kurt Mehlhorn, Stefan Näher, and Pascal Schweitzer. Certifying algorithms. *Computer Science Review*, 5(2):119–161, 2011. doi: 10.1016/j.cosrev.2010.09.009.

Leland McInnes. fast_hdbscan: A fast multicore implementation of HDBSCAN for low-dimensional Euclidean data. https://github.com/TutteInstitute/fast_hdbscan, 2026. URL https://pypi.org/project/fast-hdbscan/0.3.2/. Version 0.3.2.

Leland McInnes and John Healy. Accelerated hierarchical density based clustering. In *ICDM Workshops*, pp. 33–42, 2017. doi: 10.1109/ICDMW.2017.12.

Leland McInnes, John Healy, and Steve Astels. hdbscan: Hierarchical density based clustering. *Journal of Open Source Software*, 2(11):205, 2017. doi: 10.21105/joss.00205.

Leland McInnes, John Healy, Nathaniel Saul, and Lukas Großberger. UMAP: Uniform manifold approximation and projection. *Journal of Open Source Software*, 3(29):861, 2018. doi: 10.21105/joss.00861.

Benjamin Moseley, Sergei Vassilvitskii, and Yuyan Wang. Hierarchical clustering in general metric spaces using approximate nearest neighbors. In *AISTATS*, pp. 2440–2448, 2021. URL https://proceedings.mlr.press/v130/moseley21a.html.

Sameer A. Nene, Shree K. Nayar, and Hiroshi Murase. Columbia object image library (COIL-100). Technical Report CUCS-006-96, Columbia University, 1996a.

Sameer A. Nene, Shree K. Nayar, and Hiroshi Murase. Columbia object image library (COIL-20). Technical Report CUCS-005-96, Columbia University, 1996b.

Camilla Birch Okkels, Erik Thordsen, Martin Aumüller, Arthur Zimek, and Erich Schubert. Approximate single-linkage clustering using graph-based indexes: MST-based approaches and incremental searchers. In *SISAP*, pp. 233–247, 2025. doi: 10.1007/978-3-032-06069-3_19.

Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy V. Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, Mahmoud Assran, Nicolas Ballas, Wojciech Galuba, Russell Howes, Po-Yao Huang, Shang-Wen Li, Ishan Misra, Michael Rabbat, Vasu Sharma, Gabriel Synnaeve, Hu Xu, Hervé Jégou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. DINOv2: Learning robust visual features without supervision. *TMLR*, 2024. URL https://openreview.net/forum?id=a68SUt6zFt.

Sebastian Raschka, Joshua Patterson, and Corey Nolet. Machine learning in Python: Main developments and technology trends in data science, machine learning, and artificial intelligence. *Information*, 11(4):193, 2020. doi: 10.3390/info11040193.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet large scale visual recognition challenge. *International Journal of Computer Vision*, 115 (3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

Erich Schubert. Automatic indexing for similarity search in ELKI. In *SISAP*, pp. 205–213, 2022. doi: 10.1007/978-3-031-17849-8_16.

Erich Schubert. Hierarchical density-based clustering using incremental similarity search. In *SISAP*, pp. 263–270, 2025. doi: 10.1007/978-3-032-06069-3_21.

Oriane Siméoni, Huy V. Vo, Maximilian Seitzer, Federico Baldassarre, Maxime Oquab, Cijo Jose, Vasil Khalidov, Marc Szafraniec, Seungeun Yi, Michaël Ramamonjisoa, Francisco Massa, Daniel Haziza, Luca Wehrstedt, Jianyuan Wang, Timothée Darcet, Théo Moutakanni, Leonel Sentana, Claire Roberts, Andrea Vedaldi, Jamie Tolan, John Brandt, Camille Couprie, Julien Mairal, Hervé Jégou, Patrick Labatut, and Piotr Bojanowski. DINOv3. *arXiv:2508.10104*, 2025. URL https://arxiv.org/abs/2508.10104.

Grant Van Horn, Elijah Cole, Sara Beery, Kimberly Wilber, Serge Belongie, and Oisin Mac Aodha. Benchmarking representation learning for natural world image collections. In *CVPR*, pp. 12884–12893, 2021. URL https://openaccess.thecvf.com/content/CVPR2021/html/Van_Horn_Benchmarking_Representation_Learning_for_Natural_World_Image_Collections_CVPR_2021_paper.html.

Yiqiu Wang, Shangdi Yu, Yan Gu, and Julian Shun. Fast parallel algorithms for Euclidean minimum spanning tree and hierarchical spatial clustering. In *SIGMOD*, pp. 1982–1995, 2021. doi: 10.1145/3448016.3457296.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. *arXiv:1708.07747*, 2017.

Junyuan Xie, Ross Girshick, and Ali Farhadi. Unsupervised deep embedding for clustering analysis. In *ICML*, pp. 478–487, 2016. URL https://proceedings.mlr.press/v48/xieb16.html.

HaoChuan Xu and Ninh Pham. Scalable DBSCAN with random projections. In *NeurIPS*, pp. 27978–28008, 2024. doi: 10.52202/079017-0877.

A PROOFS

This appendix proves the statements of the main text about the check at one threshold, the repair, and the repeated check at several thresholds. It also describes a rule for choosing the number of neighbors, and it states and proves the finite-sample recovery theorem.

A.1 SETUP AND PROOFS OF THE MAIN STATEMENTS

This subsection recalls the setting of Section 3, fixes the notation, and proves the statements that support the check, the repair and the selection criterion. The graph G is the complete graph on X with exact MR weights w. For a weighted graph H and a threshold t, the graph $G_{\le t}(H)$ keeps the edges of H whose own weights are at most t. The approximate tree is built from a neighbor graph $\widehat{G}$, which is a connected subgraph of G with weights $\hat{w} \ge w$. We call this inequality the monotone property. In our pipeline, the neighbor graph is the symmetrized ANN graph together with its bridge edges. The weights are exact if $\hat{w} \equiv w$ on $E(\widehat{G})$. In the implementation, the weights follow the float64 reference, and the floating-point conventions and restrictions listed in Section B apply. The approximate tree $\widehat{T}$ is an MST of $\widehat{G}$. For $2 \le k \le N-1$, the k-Center cut removes the $k-1$ heaviest edges of the approximate tree, which gives the clustering $\widehat{P}_k$, and ε is the k-th largest weight of $\widehat{T}$.

Lemma 2 (cutting a graph and cutting its MST)**.** *Let H be a connected weighted graph and T_H any MST of H. For every threshold t, the connected components of $G_{\le t}(T_H)$ equal the connected components of $G_{\le t}(H)$.*

Proof. Because $G_{\le t}(T_H) \subseteq G_{\le t}(H)$, the connected components of the subgraph refine those of $G_{\le t}(H)$. Conversely, suppose that u, v lie in one connected component of $G_{\le t}(H)$ but the path in T_H from u to v uses an edge e with weight $> t$. Removing e splits T_H into two sides that separate u from v. Some edge e' of the path from u to v in $G_{\le t}(H)$ crosses this split and has weight $\le t < w(e)$. Then $T_H - e + e'$ is a spanning tree of strictly smaller weight, which contradicts the minimality of the MST. The exchange is valid under arbitrary ties because the inequality $w(e') \le t < w(e)$ is strict. □

Theorem 1 (the cut under missing edges and overestimated weights)**.** (a) *For every threshold s, each connected component of $G_{\le s}(\widehat{T})$ lies inside one cluster of $P_G(s)$: missing edges can split clusters of the exact clustering, but never merge them.* (b) *Let $2 \le k \le N-1$, let the cut on $\widehat{T}$ be unambiguous (its $(k-1)$-st largest weight strictly exceeds ε), recall that ε is fixed as the k-th largest weight of $\widehat{T}$, and let $P_k^\star := P_G(\varepsilon)$ be the exact clustering at this threshold, whose clusters are the connected components of $G_{\le \varepsilon}(G)$. If each cluster of $P_k^\star$ is connected within $G_{\le \varepsilon}(\widehat{G})$, then $\widehat{P}_k = P_k^\star$.*

Lemma 3 (termination of the repair)**.** *Assume exact weights and that every round cuts the current approximate tree at a threshold ε, that is, keeps exactly its edges of weight at most ε; the threshold may change between rounds, as with an unambiguous k-Center cut or the selection criterion. Then no missing edge that the check reports is an existing edge of the neighbor graph. Adding all reported missing edges, with their exact weights, strictly enlarges the edge set. Therefore the loop of check and repair terminates, and when the check passes, Proposition 1 applies to the cut of the last round.*

Corollary 1 (repeated check at several thresholds, with ties)**.** *Assume exact weights and let $v_1 > \cdots > v_m$ be the distinct values among the $L+1$ largest weights of $\widehat{T}$. We call the check strict when it uses strict inequalities at a threshold v_j and is applied to the clustering of the approximate tree just below v_j. If the strict check passes for $j = 1, \ldots, m-1$, then the approximate tree and the exact tree give the same clustering at every threshold $t \ge v_m$. The distances at which clusters merge, which we call merge distances, then also coincide in (v_m, ∞), counted with multiplicity. Any cluster extraction that is determined by the clusterings at these thresholds gives the same result on the approximate tree and on the exact tree. A selection with the excess-of-mass (EOM) criterion that is truncated to these thresholds is covered only if its condensation and its tie rule are determined by the clusterings at these thresholds. An arbitrary binary merge tree that a library builds with its default edge order need not satisfy this premise (Section 4.3). The weights need not be distinct. A group of tied merge distances is certified as one merge, and the clustering at a threshold does not depend on how ties are broken.*

Details of the selection criterion. The criterion of Section 3.3 sweeps the thresholds of the tree and counts at each threshold the clusters with at least m points. It selects the highest range of thresholds at which this number equals the target k. If no threshold yields exactly k such clusters, it selects the closest number that occurs, preferring the larger number and then the higher threshold. All thresholds of the selected range give the same clustering, and we use the midpoint of the range. If no cluster has at least m points, the clustering is reported as degenerate. At the selected threshold, the criterion keeps the $\min(k, q)$ largest clusters, where q is the number of clusters with at least m points. All other points are unassigned, and each of them receives the label of its nearest assigned point in Euclidean distance. All ties are broken by the smallest point index, so the final clustering is a deterministic function of the tree.

Lemma 4 (agreement of the selection criterion)**.** *Let the selection criterion choose the threshold t on $\widehat{T}$, and suppose that the criterion reaches the given number of clusters at this threshold. If the strict check of Corollary 1 passes at every distinct weight of $\widehat{T}$ above t, then on the exact tree the selection criterion chooses the clustering that the approximate tree has at t. The range of thresholds that the criterion selects on the exact tree may extend below t. The two final clusterings coincide after the same deterministic assignment of the unassigned points to their nearest assigned points.*

Proof of Theorem 1. (a) By the monotone property, an edge with $\hat{w} \le s$ has $w \le s$. Therefore $G_{\le s}(\widehat{T}) \subseteq G_{\le s}(\widehat{G}) \subseteq G_{\le s}(G)$, and the connected components of the subgraphs refine those of $G_{\le s}(G)$, which are the clusters of $P_G(s)$. (b) As in (a), with $s = \varepsilon$, the connected components of $G_{\le \varepsilon}(\widehat{G})$ refine $P_k^\star$. By the condition of (b), these connected components equal $P_k^\star$, with exactly k clusters. By Lemma 2 applied to $\widehat{G}$ with weights $\hat{w}$, the graph $G_{\le \varepsilon}(\widehat{T})$ has the same k connected components, so $\widehat{T}$ has exactly $k-1$ edges of weight $> \varepsilon$. These edges are its $k-1$ heaviest edges, because all remaining tree edges have $\hat{w} \le \varepsilon$, and the strict gap that the theorem assumes makes the choice of the $(k-1)$ heaviest edges unambiguous. Removing these edges gives $P_k^\star$. □

Proof of Proposition 1. The spanning trees of the clusters put every cluster of P inside a cluster of $P_G(\varepsilon)$. If no pair of points from different clusters satisfies Equation (2), then no edge of the complete graph with weight at most ε joins different clusters, so the connected components of the complete graph cut at this threshold are exactly the clusters of P. Conversely, a missing edge has weight at most ε and joins two different clusters, which contradicts $P = P_G(\varepsilon)$. For an unambiguous cut into k clusters, Lemma 2 transfers the same clustering at this threshold to every exact MST, which then has exactly $k-1$ edges above ε. □

Proof of Lemma 3. Assume $\hat{w} \equiv w$ on $E(\widehat{G})$. An existing edge (u, v) with $w(u, v) \le \varepsilon$ puts u, v in one connected component of $G_{\le \varepsilon}(\widehat{G})$ and hence, by Lemma 2 applied to $\widehat{G}$, in one cluster of the cut at ε. A missing edge joins two different clusters, so no missing edge is an existing edge. Adding all missing edges with their exact weights strictly enlarges $E(\widehat{G})$, so the loop terminates within $\binom{N}{2}$ rounds. When the check passes, Proposition 1 applies to the cut of the last round. If the weights are only monotone, a missing edge can be an existing edge with an overestimated weight. An implementation must then overwrite the weight of this edge with the smaller exact weight. An element-wise maximum never lowers a weight, so with this operation the loop need not terminate. Our pipeline uses exact weights. In this case the element-wise maximum only inserts entries that were absent and is harmless. □

Proof of Corollary 1. Assume exact weights, and let $v_1 > \cdots > v_m$ be the distinct values among the $L+1$ largest weights of $\widehat{T}$. The map from a threshold to the clustering at this threshold, $t \mapsto \mathrm{components}(G_{\le t}(\widehat{T}))$, does not depend on the choice of the MST (Lemma 2) or on how ties are broken. Removing all edges of weight above t is unambiguous even when weights are tied, so the object that is certified is well defined. Write Q_j for the clustering of the approximate tree on the interval $[v_{j+1}, v_j)$. The strict check with index j excludes pairs of points from different clusters of Q_j that satisfy $\kappa_\mu(x) < v_j$, $\kappa_\mu(y) < v_j$ and $\|x - y\| < v_j$, all with strict inequality. The intervals $[v_{j+1}, v_j)$ for $j = 1, \ldots, m-1$, together with $[v_1, \infty)$, cover $[v_m, \infty)$. Fix j and $t \in [v_{j+1}, v_j)$. Each cluster of Q_j is spanned by edges of the approximate tree with weight $\le v_{j+1} \le t$, so each cluster is connected in $G_{\le t}(G)$, and the exact clustering at t coarsens Q_j. A strict coarsening requires an

edge of the complete graph between two different clusters with weight $\leq t < v_j$. The three terms of this weight, which are the two core distances and the Euclidean distance, are then each strictly below v_j, and the strict check excludes such a pair. Therefore the approximate tree and the exact tree give the same clustering on each interval, including its left endpoint. The point $t = v_j$ itself lies in the interval of index $j - 1$, or in $[v_1, \infty)$ for $j = 1$, so every endpoint is covered. In particular, a pair of points from different clusters with weight exactly v_j, whether it is a tree edge that merges two clusters or another pair of equal weight, cannot change the clustering on $[v_j, v_{j-1})$, which the strict check with index $j - 1$ covers. For $t \geq v_1$ both clusterings consist of a single cluster. The agreement of the clusterings at every threshold in $[v_m, \infty)$ determines every merge distance in the open interval (v_m, ∞), together with the number of clusters that merge there (a merge at v_m itself would need the clustering just below v_m). Therefore the merge distances in (v_m, ∞), counted with multiplicity, coincide. The multiset of MST edge weights is the same for all MSTs of a graph by the standard exchange argument, so the merge distances of $\widehat{T}$ are well defined without reference to a particular tree. □

Proof of Lemma 4. Let $v_1 > \cdots > v_{\ell-1}$ be the distinct weights of $\widehat{T}$ above t and v_ℓ the largest weight not exceeding t, so that $t \in [v_\ell, v_{\ell-1})$. The hypothesis is exactly that of Corollary 1 for $j = 1, \ldots, \ell-1$. Therefore the approximate tree and the exact tree give the same clustering at every threshold in $[v_\ell, \infty) \supseteq [t, \infty)$, and hence the same number of clusters with at least m points at every threshold in this interval. We call this number the count. On the approximate tree, the selection criterion chose the range of thresholds that contains t as the highest range on which the count equals k. On $[t, \infty)$ the count on the exact tree equals the count on the approximate tree, so any range of the exact tree with count k that lies above t would also be such a range of the approximate tree, which contradicts maximality. At t itself, the exact tree also has count k. On the exact tree, the highest range with count k therefore contains t. This range may extend below t, where nothing is certified. However, the clustering that the criterion selects is the exact clustering at t, which coincides with the clustering of the approximate tree. The assignment of the unassigned points is a deterministic function of the clustering, the cluster sizes and the coordinates, so the final clusterings coincide. If the given number of clusters is not reached, the selection criterion compares counts over the whole sweep, including thresholds below t, and the lemma does not apply. □

A.2 A RULE FOR CHOOSING THE NUMBER OF NEIGHBORS

This subsection describes a rule that chooses the number of neighbors of the neighbor graph in our pipeline, and it states which parameters the rule does not set. A kNN graph on n uniformly random points needs $k = \Theta(\log n)$ neighbors to be connected with high probability (Balister et al., 2005). With a fixed number of neighbors, the graph can therefore become disconnected as the sample grows. Our rule doubles k_{nn} until the number of connected components of the neighbor graph, counted before the bridge edges are added, reaches a plateau. It then uses the number of neighbors at the elbow of this curve. Under the assumptions of Section A.3, the exact neighbor graph has the true number of connected components over the range given in Corollary 2. An approximate neighbor graph can contain additional edges, and its number of connected components can reach a plateau too early. The check detects differences from the exact clustering at the tested threshold (Section G). The rule sets only the number of neighbors. Our pipeline still requires μ, a minimum cluster size, a given number of clusters for the final clustering, and the hyperparameters of the encoder. EOM chooses the number of clusters itself.

A.3 FINITE-SAMPLE RECOVERY: INFORMAL STATEMENT

This subsection states informally the finite-sample condition under which the check passes, which the main text lists among its contributions. The next subsection gives the assumptions, the formal statement and the proof.

Theorem 2 (finite-sample recovery of a certified clustering, informal)**.** *Let f be a density whose super-level set $\{f \geq \lambda\}$ has k compact, path-connected, (c_0, r_0)-thick connected components with pairwise separation at least $3R$, and let the neighbor graph contain the exact symmetrized k_{nn}-NN graph with exact weights. There are constants C_0, C_1 such that for $\mu \geq C_0 \log(n/\delta)$, $k_{\mathrm{nn}} \geq \max(\mu, C_1 \log(n/\delta))$ and $\rho_{\mathrm{up}}(k_{\mathrm{nn}}) \leq R/2$, the following holds with probability at least $1 - 2\delta$. The k-Center cuts of the approximate tree and of the exact tree both equal the clustering of the sample*

by the k connected components. The cut gap, which is the difference between the lightest removed and the heaviest kept tree weight, is at least $\frac{5}{2}R$, and the check passes.

The proof follows in the next subsection of Section A. It uses two-sided Chernoff bounds on kNN distances, that is, on the distances from the sample points to their nearest neighbors of a given rank, and a connectivity argument over a net whose balls all contain sample points. The proof specializes the analysis of cluster identification by Maier et al. (2009) and the rates for robust single linkage by Chaudhuri & Dasgupta (2010) to our pipeline. The constants grow as 5^d and 10^d with the data dimension. Assumption (A3) sets $f = 0$ outside the super-level set. This assumption excludes noise points around the clusters and overlapping tails of the clusters. The theorem therefore covers only a restricted setting. For approximate neighbor graphs and for data with noise, the check decides at run time whether the clustering agrees with the exact one.

A.4 Finite-sample recovery: formal statement and proof

This subsection lists the assumptions, states the formal version of the recovery theorem, and proves it. Let f be a density on $\mathbb{R}^d$, let $\lambda > 0$ be a density level whose super-level set has connected components $A_1, \dots, A_k$, and fix a working threshold R. We call these connected components the clusters of the density, and we call the sample points that lie in one of them a cluster sample.

Assumptions.

- (A1) (thickness) Each A_i is compact, path-connected, and (c_0, r_0)-thick, that is, $\mathrm{vol}(B(x,s) \cap A_i) \geq c_0 \,\mathrm{vol}(B(x,s))$ for all $x \in A_i$ and $s \leq r_0$. In addition, $R \leq r_0$. Connectedness alone would suffice through η-chain connectedness, but we assume path-connectedness to keep the exposition simple.
- (A2) (separation) $\mathrm{dist}(A_i, A_j) \geq 3R$ for $i \neq j$.
- (A3) (density bounds, no mass outside the clusters) $\lambda \leq f \leq \Lambda$ on $\bigcup_i A_i$ and $f = 0$ elsewhere. The n sample points are drawn i.i.d. from the distribution with density f, with $n \geq 4$, so exact distance ties among sample points occur with probability zero.
- (G) (assumptions on the graph) $\widehat{G}$ contains the exact symmetrized k_{nn}-NN graph (an edge is present when either endpoint lists the other), carries exact MR weights on all its edges, and is connected. The bridge edges provide, with exact weights, the edges between connected components that are needed to make the graph connected.

Lemma 5 (two-sided bounds on the kNN distance)**.** *Let v_d be the volume of the unit ball, let $0 < \delta < 1$, and define for $k \leq n-1$*

$$\rho_{\mathrm{lo}}(k) = \left(\tfrac{k}{2\Lambda v_d n}\right)^{1/d}, \qquad \rho_{\mathrm{up}}(k) = \left(\tfrac{2k}{c_0 \lambda v_d n}\right)^{1/d}.$$

There are universal constants c, C_0 such that the following holds. If $\mu \geq C_0 \log(n/\delta)$ and $\rho_{\mathrm{up}}(k) \leq r_0$, then with probability at least $1-\delta$, uniformly over the sample points $x \in \bigcup_i A_i$ and all ranks $k' \in \{\mu, \dots, k\}$, the distance from the sample point to its nearest neighbor of this rank satisfies $r_{k'}(x) \in [\rho_{\mathrm{lo}}(k'), \rho_{\mathrm{up}}(k')]$.

Proof. Condition on $x = X_i$. The remaining $n-1$ points are i.i.d. from f. For $\rho \leq r_0$, the count of these points in $B(x,\rho)$ is $\mathrm{Bin}(n-1, q)$ with $c_0 \lambda v_d \rho^d \leq q \leq \Lambda v_d \rho^d$. The lower bound uses (A1) and (A3), and the upper bound uses (A3). At $\rho = \rho_{\mathrm{lo}}(k')$ the mean is at most $k'/2$. At $\rho = \rho_{\mathrm{up}}(k')$ the mean is at least $2k'(1-1/n) \geq \frac{3}{2}k'$ for $n \geq 4$. Multiplicative Chernoff bounds give $\Pr[\mathrm{count} \geq k'] \leq e^{-ck'}$ and $\Pr[\mathrm{count} < k'] \leq e^{-ck'}$ for a universal $c > 0$. A union bound over all n sample points and all $k' \in \{\mu, \dots, k\}$ bounds the failure probability by

$$n \sum_{k' \geq \mu} 2e^{-ck'} \;\leq\; \frac{2n\, e^{-c\mu}}{1 - e^{-c}} \;\leq\; \delta$$

once $\mu \geq C_0 \log(n/\delta)$. No interpolation between dyadic values is needed. □

Lemma 6 (close pairs are edges of the neighbor graph)**.** *On the event of Lemma 5, and on the almost-sure event of (A3) that no distances are tied, every pair of sample points $x, y \in \bigcup_i A_i$ with $\|x - y\| \leq \rho_{\mathrm{lo}}(k_{\mathrm{nn}})$ is an edge of the exact symmetrized k_{nn}-NN graph, and every such edge between points of one A_i has MR weight at most $\rho_{\mathrm{up}}(k_{\mathrm{nn}})$, provided that $\mu \leq k_{\mathrm{nn}}$.*

Proof. The inequality $r_{k_{\mathrm{nn}}}(x) \geq \rho_{\mathrm{lo}}$ means that fewer than k_{nn} points lie strictly within distance ρ_{lo} of x. Because no distances are tied, every point at distance at most ρ_{lo} is therefore among the k_{nn} nearest neighbors of x, and the symmetrized graph keeps the edge. The core distances satisfy $\kappa_\mu(\cdot) = r_\mu(\cdot) \leq \rho_{\mathrm{up}}(\mu) \leq \rho_{\mathrm{up}}(k_{\mathrm{nn}})$, and the distance of the pair is at most $\rho_{\mathrm{lo}} \leq \rho_{\mathrm{up}}$. The MR weight is the maximum of these three quantities. □

Lemma 7 (connectivity through an occupied net)**.** *Let $\rho = \rho_{\mathrm{lo}}(k_{\mathrm{nn}})$ and let $\mathcal{N}_i$ be a maximal $(\rho/5)$-separated subset of A_i, hence a $(\rho/5)$-net. There is a constant $C_1 = C_1(c_0, \lambda, \Lambda, d)$ such that if $k_{\mathrm{nn}} \geq C_1 \log(n/\delta)$ and $\rho_{\mathrm{up}}(k_{\mathrm{nn}}) \leq r_0$, then with probability at least $1 - \delta$ every ball $B(z, \rho/5)$ with $z \in \mathcal{N}_i$ contains a sample point of A_i. Consequently, on the event of Lemma 5, every two sample points of A_i are joined in the neighbor graph by a path whose edges have length at most ρ.*

Proof. Cardinality. The balls $B(z, \rho/10)$ with $z \in \mathcal{N}_i$ are disjoint, and each has probability mass at least $\lambda c_0 v_d (\rho/10)^d$ by (A1) and (A3). The masses sum to at most one, so $|\mathcal{N}_i| \leq \left(\lambda c_0 v_d (\rho/10)^d\right)^{-1} = O(n/k_{\mathrm{nn}})$. This bound uses only the masses of disjoint balls and needs no control of Minkowski enlargements. *Occupancy.* Each set $B(z, \rho/5) \cap A_i$ has mass at least $c_0 \lambda v_d (\rho/5)^d = c' k_{\mathrm{nn}}/n$ by (A1) and (A3), with $c' = c_0 \lambda / (2 \cdot 5^d \Lambda)$. The number of sample points in this set is therefore binomial with mean at least $c' k_{\mathrm{nn}}$. This number is zero with probability at most $e^{-c' k_{\mathrm{nn}}} \leq \delta / (k\,|\mathcal{N}_i|)$ once $k_{\mathrm{nn}} \geq C_1 \log(n/\delta)$, where we use the cardinality bound and $k \leq n$ (absorbed into C_1). A union bound over $\mathcal{N}_i$ and over the k clusters of the density gives a total failure probability of at most δ. *Chain.* Let $u, v \in A_i$ be sample points. Path-connectedness (A1) gives a continuous path $\gamma \subset A_i$ from u to v. Choose points $u = q_0, \ldots, q_m = v$ on this path with $\|q_j - q_{j+1}\| \leq \rho/5$. Let $z_j \in \mathcal{N}_i$ with $\|z_j - q_j\| \leq \rho/5$, and let s_j be a sample point of A_i in $B(z_j, \rho/5)$, where we take $s_0 = u$ and $s_m = v$. Then

$$\|s_j - s_{j+1}\| \leq \tfrac{\rho}{5} + \tfrac{\rho}{5} + \tfrac{\rho}{5} + \tfrac{\rho}{5} + \tfrac{\rho}{5} = \rho,$$

where the five terms are the distances from the sample point to the center of its ball, from this center to the path, along one step of the path, from the path to the next center, and from that center to its sample point. Each hop from s_j to s_{j+1}, including the end hops of length at most $3\rho/5$, is an edge of the neighbor graph by Lemma 6. □

Theorem 3 (finite-sample recovery of a certified clustering, formal)**.** *Assume (A1) to (A3) and (G), $\mu \geq C_0 \log(n/\delta)$, $k_{\mathrm{nn}} \geq \max(\mu, C_1 \log(n/\delta))$, and $\rho_{\mathrm{up}}(k_{\mathrm{nn}}) \leq R/2$. Then the following four statements hold with probability at least $1 - 2\delta$.*

1. *The sample points of each A_i form one connected component of $G_{\leq R}(\widehat{G})$, and no edge of G of weight at most R joins distinct A_i, A_j. Hence the connected components of both $G_{\leq R}(\widehat{G})$ and $G_{\leq R}(G)$ are exactly the k cluster samples.*
2. *Consequently, the k-Center cuts of the approximate tree and of the exact tree both equal the clustering into the cluster samples. The heaviest tree weight that the cut keeps satisfies $\varepsilon_{\mathrm{cut}} \leq \rho_{\mathrm{up}} \leq R/2$, every removed edge has weight at least $3R$, and the cut gap is $\Delta_k \geq 3R - \rho_{\mathrm{up}} \geq \frac{5}{2} R$.*
3. *The check of Proposition 1, run at threshold $\varepsilon_{\mathrm{cut}}$, passes.*
4. *The exact symmetrized k_{nn}-NN graph, cut at threshold R, has exactly k connected components.*

Proof. Claim 1: connectivity within each cluster of the density is Lemma 7, whose path edges have MR weight at most $\rho_{\mathrm{up}} \leq R/2 < R$ by Lemma 6. These edges lie in $\widehat{G}$ with exact weights by (G). Between different clusters of the density, (A2) gives $\|x - y\| \geq 3R$, so $w(x, y) \geq 3R > R$. Therefore no edge between different clusters of the density exists at threshold R in G, and a fortiori none exists in $\widehat{G}$. Claim 2: with exactly k connected components at threshold R in both graphs, Lemma 2 gives $k - 1$ tree edges above R in each tree. The tree edges that the cut keeps have weight at most ρ_{up}, the threshold at which the clusters of the density are already connected, and the removed tree edges join different clusters of the density and have weight at least $3R$. Claim 3 is immediate from (A2): a missing edge would be an edge of the complete graph with weight at most $\varepsilon_{\mathrm{cut}} \leq R/2 < 3R$ between different clusters of the density, and such an edge does not exist. Claim

4 follows from the connectivity within each cluster of the density, whose edges have weight at most $\rho_{\mathrm{up}} \leq R/2$, and from the absence of edges between different clusters of the density at threshold R. The failure probability is one δ for Lemma 5 and one for Lemma 7. □

Note that the theorem involves two thresholds. The quantity R is the working threshold of the assumptions, which sets the separation of the clusters of the density, while $\varepsilon_{\mathrm{cut}}$, the heaviest tree weight that the cut keeps, is the threshold at which the check runs. The theorem shows $\varepsilon_{\mathrm{cut}} \leq R/2$, so the two thresholds never coincide, and the premise of the check refers to the observable quantity.

Corollary 2 (one-sided guarantee for the elbow)**.** *Fix the sample, so that the exact k'-NN graphs are nested in k' on the almost-sure event that no distances are tied. Under the assumptions of Theorem 3, with the event of Lemma 5 taken at upper index $k_{\max}$ where $\rho_{\mathrm{up}}(k_{\max}) \leq R/2$, for every $k'_{\mathrm{nn}} \in [\max(\mu, C_1 \log(n/\delta)),\ k_{\max}]$ the exact symmetrized k'_{nn}-NN graph has exactly k connected components, the k cluster samples, without cutting the graph at a threshold. The plateau of the elbow over this whole range therefore lies at the true number of connected components.*

Proof. On the event of Lemma 5 at upper index $k_{\max}$ (admissible because $\rho_{\mathrm{up}}(k_{\max}) \leq R/2 \leq r_0$), every k'-NN distance with $k' \leq k_{\max}$ is at most $\rho_{\mathrm{up}}(k_{\max}) \leq R/2 < 3R$. By (A2), no k'-NN edge, in either direction of the symmetrization, therefore joins different clusters of the density at any k'_{nn} in the range. Connectivity within each cluster of the density at the base $k'_{\mathrm{nn}} = \max(\mu, C_1 \log(n/\delta))$ is Lemma 7, and nesting preserves this connectivity for larger k'_{nn}. □

Plateaus that occur too early, below the lower end of this range, are not excluded, and the check does not detect a plateau at a number of connected components that differs from the number of clusters of the density. If the cut of the approximate tree at the selected k equals the cut of the exact MST at that same k, the check passes, whether or not k matches the number of clusters of the density. However, the check does detect every difference between the clustering from the approximate tree and the exact clustering. These differences include exact clusters that are split because the number of neighbors is too small, which is the error that the rule for choosing the number of neighbors addresses.

B Implementation of the check

This section describes the implementation of the check at one threshold. It lists the floating-point conventions and the input validation to which the main text refers. Proposition 1 states the check in real arithmetic. Of its two requirements, the implementation in `src/cut_certificate.py` tests the absence of missing edges, and the caller provides the spanning trees of the clusters. The implementation first validates its input. The given clustering must assign a nonnegative integer label to every point, the core distances and the threshold must be finite and nonnegative, and the coordinates must be finite. The coordinates are cast to float64. The supported dimension is $1 \leq d \leq 10^6$, and nonzero coordinates must have a magnitude in $[10^{-100}, 10^{100}]$. The implementation rejects input outside these ranges and reports no result for it. The search has four steps.

1. Select the core points with a relative padding, that is, the points with $\kappa_\mu(i) \leq \varepsilon(1+\eta)$, $\eta = 10^{-9}$. Sort their indices by cluster with a stable sort and split them into one group per cluster. Build a kd-tree and a coordinate bounding box for each nonempty group. With a core points, grouping costs $O(N + a \log a)$, because the labels of all N points are not scanned again for every cluster.
2. For each pair of groups, compute the Euclidean gap between their bounding boxes. Skip the pair only if this gap exceeds $\varepsilon(1+\eta)$. Otherwise, query the smaller group against the tree of the larger group within this padded distance, in chunks of 256 points to bound the memory for the candidate pairs.
3. Recompute the distance of every candidate pair with the expression `sqrt(einsum(...j,...j->..., diff, diff))` on float64 coordinate differences, and take the maximum with the two core distances. The weights of tree edges and of missing edges are computed with this same expression. The exact core distances come from the exact mode of SciPy's `cKDTree.query`.

4. The check uses the predicate $w \leq \varepsilon$. The strict check at the distinct edge weights of a hierarchy uses $w < \varepsilon$. The search stops at the limit on the number of edges per round, and the strict predicate is applied before a pair counts toward this limit. Pairs of a tie, whose weight equals the threshold, can therefore not exhaust the limit and hide a later pair that satisfies the strict predicate. In a strict check, a weight equal to the threshold is not an error, because all merges at the same weight count as one merge. A pair whose weight differs from the threshold but lies within the relative padding is flagged separately as a boundary pair.

A valid missing edge makes the check fail even if the search is incomplete. The check passes only if the search is complete, returns no missing edge and flags no boundary pair. If an incomplete search returns no missing edge, the result is inconclusive. A result with boundary pairs only is inconclusive. Code that uses the result must therefore read the status and the completeness of the search; an empty list of missing edges or a single Boolean value is not a certificate. Likewise, the hierarchy check in Section C never reports a pass when it reaches a work limit. The tests include duplicate points and the threshold zero, and points with distinct indices at the same coordinates count as distinct neighbors.

Floating-point arithmetic. In real arithmetic, the gap between two bounding boxes is a lower bound on the distance of every pair of points from the two boxes, so skipping pairs of groups loses no missing edge. In the float64 implementation, the search uses a conservatively padded distance, and the unpadded predicate makes the final decision. This separation is designed for the supported range of moderate coordinate magnitudes. We do not assume that the distances that SciPy accumulates equal the NumPy results bit for bit, and we do not claim a formally verified rounding bound for every traversal inside `cKDTree`. In particular, if an evaluation has a relative error of at most θ, a computed distance of at most ε implies a real distance of at most $\varepsilon/(1-\theta)$, not merely $\varepsilon(1+\theta)$. The padding, the input validation and the rechecks of boundary pairs are engineering measures that we validated against exhaustive comparisons. They do not certify a result in arbitrary-precision real arithmetic.

The public interface casts the coordinates before it queries and checks them. Without this cast, a float32 input could lead to a false pass for a pair at the boundary. The scripts behind the tables of this paper cast their input themselves, so no reported result is affected. The tests also cover the positive lower bound on the query distance that the search for core points uses at very small thresholds, a comparison of clusterings that distinguishes noise from clusters, the fixed tie rule, the validity of the spanning trees of the clusters, and inconclusive results. The test suite has 48 passing tests. A separate experiment with fixed random seeds checks $1{,}000$ small problems on the complete graph, including ties, duplicates and cases in which the given number of clusters cannot be reached. It compares the results with independently computed clusterings and with exact references for the repair computed on the dense graph.

The timing tables and the tables on several thresholds of the hierarchy that use a relative tolerance on merge heights state this tolerance and their version of the check. The strict comparisons of the cycle condition do not treat two weights as equal merely because their relative difference is below 10^{-9}; a difference of one unit in the last place counts. This convention explains why the tree of an exact library with recomputed weights can fail the hierarchy check while its clustering at the selected threshold still passes the check.

C Extensions: guarantees and implementation

This section contains the proofs and the implementation details for the hierarchy check, the repair and two extensions of the check, and it defines the fixed tie rule. All statements concern indexed points, the fixed parameter μ of the core distance, and the complete MR graph of Equation (1). The geometric inequalities are stated in real arithmetic, and the implementation follows the floating-point conventions of Section B.

C.1 THE HIERARCHY CHECK

This subsection proves the restricted cycle condition of the main text and describes the search over pairs of index nodes that the hierarchy check uses.

Proof of Proposition 2. For necessity, fix i, j and take $s = \max\{w(i,j), t\}$. The complete graph connects i and j at threshold s, so agreement requires a path in the tree whose largest edge weight is at most s. For sufficiency, fix $s \geq t$. Every edge of the complete graph with weight at most s can be replaced by a path in the tree whose largest edge weight is at most $\max\{w(i,j), t\} \leq s$. Concatenating these paths preserves every connection in the complete graph. Conversely, each tree edge is an edge of the complete graph with the same weight, so a connection in the tree is a connection in the complete graph. □

To check the condition without enumerating every pair, let a node of the index contain the point set A and a representative a, and let $\operatorname{span}(A) = \max_{i \in A} \operatorname{dc}_{\widehat{T}}(a, i)$. For another node B with representative b, the ultrametric inequality of the dc-distance gives

$$\operatorname{dc}_{\widehat{T}}(i,j) \leq U(A,B) := \max\{\operatorname{span}(A), \operatorname{span}(B), \operatorname{dc}_{\widehat{T}}(a,b)\}, \tag{4}$$

$$w(i,j) \geq L(A,B) := \max\{\min_{i \in A} \kappa_\mu(i), \min_{j \in B} \kappa_\mu(j), \operatorname{dist}(\operatorname{box} A, \operatorname{box} B)\}. \tag{5}$$

A pair of nodes is therefore settled if $U \leq t$ or $U \leq L$. Otherwise we split a node, and pairs of single points are decided by the condition itself. The index can be a bounding-volume hierarchy that follows the merges of the tree or a spatial one; both are valid. Our experiments use the former. The dc-distances come from a binary-lifting index on the tree, which requires $O(N \log N)$ memory, and the geometry adds $O(Nd)$. The hierarchy check therefore does not inherit the linear memory use that we measured for the check at one threshold. In the worst case, the work is quadratic.

The implementation verifies the weights of the tree edges before it uses them. The geometric bounds are padded conservatively, and comparisons of close pairs in the leaves use the same NumPy expression as the check. An open pair of nodes is never counted as settled because a work limit is reached. The default work limits are 50 million pairs of nodes, 100 million pairs of points and one million boundary pairs. Reaching one of them makes the result inconclusive. A fail comes with a pair of points that does not satisfy Equation (3).

Consequence for the selection criterion. If the criterion reaches the given number of clusters at t on the approximate tree, agreement for all $s \geq t$ fixes the highest range of thresholds with this number of clusters and the clustering in this range. The exact reference cannot hide a higher range that qualifies. The deterministic choice of the kept clusters and the assignment of each unassigned point to its nearest assigned point (1-NN assignment) then agree as well. If the criterion cannot reach the given number of clusters, agreement only above the threshold that it selects instead is not sufficient. In this case our hierarchy check starts at zero, so the two trees offer the same numbers of clusters throughout the sweep and lead to the same selection. These arguments concern our selection criterion, not HDBSCAN's excess-of-mass (EOM) criterion, which depends on lower thresholds that are not checked.

C.2 REPAIR AT A FIXED THRESHOLD

This subsection lists the steps of the check and the repair (Algorithm 1), proves the lemma of the main text on the number of rounds of the repair, and states what the bound does not cover.

Proof of Lemma 1. Fix a cluster of $P_G(\varepsilon)$ that contains $p > 1$ current clusters, and call it the exact cluster. Its points are connected by edges of weight at most the threshold, so the quotient of this connected graph by the current clusters is connected. Every current cluster therefore has a missing edge, and a complete search with one missing edge per cluster finds at least one for each. The undirected graph of the found missing edges on the current clusters has no isolated vertices, so each of its connected components contains at least two current clusters. After contraction, the exact cluster contains at most $\lfloor p/2 \rfloor$ current clusters. Current clusters never merge across exact clusters, since the added edges have weight at most ε. Iterating proves the bound. An exact cluster that already consists of one current cluster requires no added edge. □

Algorithm 1 Check and repair a clustering at a fixed threshold ε

Require: points X, exact core distances κ_μ, threshold ε, clustering P, for every cluster of P a spanning tree whose edges have exact MR weight at most ε, and a mode, CHECK or REPAIR
Ensure: PASS, FAIL or INCONCLUSIVE, and the current clustering
repeat
  $E \leftarrow$ for every cluster with a missing edge, one such edge ▷ search of Section 3.1
  if $E = \emptyset$ **then**
    return PASS if every search was complete and no candidate lies in the padding band, otherwise INCONCLUSIVE
  end if
  if the mode is CHECK **then**
    return FAIL and E
  end if
  merge the clusters that E joins, and add to the spanning trees the edges of E that do not close a cycle ▷ ε stays fixed
until the limit on the number of rounds is reached
return INCONCLUSIVE and E

The bound counts rounds, not comparisons of individual points or the final search that confirms the pass. Searching for one missing edge of a cluster can still be expensive. The experiment keeps ε fixed; rebuilding the tree and selecting a new ε is a different algorithm. A test with 64 points that starts from single-point clusters and $1{,}000$ random repairs that are compared with the complete graph complement the 300 deliberately damaged neighbor graphs, in which every initial clustering has exactly ten clusters.

C.3 Determining only the core points

This subsection describes the variant of the check that determines only the core points exactly, whose cost the main text reports. At a fixed threshold, the check needs only $\kappa_\mu(x) \leq \varepsilon$, which is equivalent to finding at least μ distinct other indexed points within distance ε. An ANN search can prove this fact by returning enough such points. A repeated index and the index of the query point itself do not count, whereas distinct points at the same coordinates do. If the ANN search returns too few such points, the query point remains undecided. For the undecided points, exact k-NN queries with a distance bound follow, and the returned distance is compared with the threshold. Too few points from the ANN search never prove that a point is not a core point. The distance bound is at least 10^{-150}, which avoids an underflow of the squared bound for duplicate points and very small thresholds. The final comparison still uses the actual ε.

This variant still requires the spanning trees of the clusters. It answers a question about one threshold and provides neither exact core distances nor MR weights for the missing edges that a repair would add. Its timings therefore measure the search for core points and the search for missing edges with given spanning trees, not a pipeline that computes every quantity only on demand. On the embeddings of our experiments, this variant finds the same core points as the computation from all exact core distances. Inputs in float32 are cast before every geometric computation.

C.4 Two extensions of the check: summary

Two further results extend the check; the next two subsections state them formally. The first is a margin. If every point moves by less than a computable bound, a certified clustering stays exact (Proposition 3). All 120 clusterings from the exact references have a positive bound. The bounds are small, with a median of 4.25×10^{-7}, and every tested perturbation within the bound leaves the clustering certified (Section D.4). The second result makes the search cheaper in the original feature space. A projection that does not increase any distance, such as the orthogonal projection onto the leading principal components, can be used as a filter: the search runs in fewer dimensions and cannot miss a pair (Proposition 4). We tested the filter in 576 checks on the original features, each with 64 held-out query points; it misses no true neighbor. At 64 dimensions and $\mu = 10$, a PCA projection keeps 14.1% of the candidate pairs and a learned projection 39.3%, so the learned projection removes fewer candidate pairs than PCA (Section D.5 and Figure 3).

C.5 A SUFFICIENT PERTURBATION MARGIN

This subsection states and proves the margin of Section C.4, a bound on the displacement of the points below which a certified clustering stays exact.

Proposition 3 (margin for a fixed clustering)**.** *Suppose that every cluster of P has a spanning tree whose edges have MR weight at most a, and that all MR weights between points of different clusters are strictly greater than $b_L > a$. Then $P = P_G(\varepsilon)$ for every $\varepsilon \in [a, b_L]$. If each indexed point is moved by at most δ, the same clustering is the exact clustering of the moved points at the fixed midpoint $(a + b_L)/2$ whenever $4\delta < b_L - a$.*

Proof. The statement about the interval follows from Proposition 1. By the triangle inequality, a displacement of at most δ per point changes each pair distance by at most 2δ. If all entries of a finite list change by at most 2δ, so does any order statistic of fixed rank: at least the same number of entries remain below the old statistic plus that bound, and the reverse argument gives the lower bound. Thus each core distance changes by at most 2δ, as does the maximum of two core distances with a pair distance. At the fixed midpoint, the weights in the spanning trees of the clusters are at most $a+2\delta < (a+b_L)/2$, while the weights between different clusters exceed $b_L-2\delta > (a+b_L)/2$. □

Here b_L is a lower bound on the MR weights between different clusters that complete searches for missing edges have proven, not an estimate of the minimum from sampled pairs. Because the inequality is strict, the guarantee covers an open range of displacements, and the reported value $(b_L - a)/4$ is the supremum of this range. Small clusters or clusters of unassigned points can determine this value. The guarantee does not show that the selection criterion selects the same threshold after the points have moved, or that unassigned points keep their nearest assigned points. If a stored spanning tree no longer satisfies the threshold after a larger perturbation, another path inside the cluster may exist. A check that does not pass is, thus, no proof that the clustering changed.

C.6 A SAFE PROJECTION FILTER

This subsection states and proves the property behind the projection filter of the main text: a projection that does not increase any distance can filter the search without missing a pair.

Proposition 4 (contractive projection with residual norm)**.** *Let $U \in \mathbb{R}^{d\times q}$ be a matrix with one row per dimension of the data and one column per dimension of the projection. Suppose that $\|U\|_2 \le 1$, and define*

$$\phi_U(x) = \big(U^\top x,\ \|(I - UU^\top)x\|\big).$$

Then $\|\phi_U(x) - \phi_U(y)\| \le \|x - y\|$ for all x, y. An exhaustive range search in the projected space therefore cannot discard a neighbor that lies within the same distance in the original space.

Proof. Put $z = x - y$ and $A = UU^\top$. The reverse triangle inequality bounds the difference of the two residual norms by $\|(I - A)z\|$. Thus the squared distance in the projected space is at most $z^\top[A + (I - A)^2]z$. The eigenvalues of A lie in $[0, 1]$, and $\lambda + (1 - \lambda)^2 \le 1$ on that interval. The expression is therefore at most $\|z\|^2$. □

The metric of the original space and the parameter of the core distance remain fixed. A projection changes only the set of candidate pairs, and an accepted edge still requires a distance computation in the original space. The implementation scales U conservatively to enforce the contraction, and it pads the range comparisons. The proposition gives a lower bound for the search. It is neither a new clustering objective nor a claim that a learned projection is better than established projections.

C.7 THE FIXED TIE RULE

This subsection defines the fixed tie rule that the main text applies to EOM, and it gives a small example in which the EOM selection of the library depends on the order of tied edges. At each distinct weight, we find the connected groups that form when the clusters from just below that weight merge. We visit the clusters of each group in the order of their smallest original point index and build a binary chain for the group, in which every pairwise merge receives that same weight as its height. The binary linkage represents one merge of several clusters, and its pairwise merges

need not be edges of the MR graph. Induction over the distinct weights shows that the construction depends only on the clusterings at all thresholds and on the point indices. Hence trees with identical dc-distances produce identical linkages and deterministic EOM outputs. Reindexing the points may lead to a different order of the pairwise merges. The rule is a convention and not a definition of a unique EOM solution that is invariant under permutations of the points.

The example uses $X = I_{40}$, $\mu = 5$, and a minimum cluster size of ten. All Euclidean and MR distances between distinct points are $\sqrt{2}$. A path is an exact MST, whatever the order of its tied edges. Across 100 random orders of the tied edges, EOM with the default edge order of the library returns two clusters in 93 cases and no cluster in 7, whereas all linkages under the fixed tie rule are identical. The example shows that the result depends on how the library encodes ties for the cluster extraction, and no approximate neighbor search is involved.

Tie handling in existing implementations. Table 2 lists how the definition of HDBSCAN* and the implementations that we inspected treat edges of equal weight. The definition removes tied edges simultaneously, so that a cluster can split into more than two clusters at one level and no order is involved. The implementations sort the edges of the tree by weight, merge them one at a time with a union-find structure, and condense the resulting binary tree node by node. Tied edges are processed in the order in which the sort returns them: an order that is unspecified in `hdbscan`, scikit-learn and ELKI (Schubert, 2022), and the order of the endpoint indices in `fast_hdbscan`. Two trees that describe the same hierarchy but list their tied edges differently can therefore lead to different condensed trees. The fixed tie rule first groups all merges at one distance and then orders the merges within a group by the smallest point index, so that the result depends only on the dc-distances and on the order of the points.

Table 2: Treatment of edges of equal weight (ties). Versions are the ones used in this paper; the entries follow from the cited definition and from the source code of the libraries.

	Order of tied edges	Tied merges
HDBSCAN* (Campello et al., 2015)	removed simultaneously	one level per weight
`hdbscan` 0.8.44	unspecified (`argsort` by weight)	one binary merge at a time
scikit-learn 1.9.0	unspecified (`argsort` by weight)	one binary merge at a time
`fast_hdbscan` 0.3.2	by the indices of the endpoints	one binary merge at a time
ELKI 0.8.0	unspecified (heap by weight)	one binary merge at a time*
Fixed tie rule (ours)	grouped by weight, then smallest point index	same result for equal dc-distances

* The development version of ELKI additionally flattens a cluster that arises at the same distance as its parent.

D Experimental protocols and complete results

This section gives the protocols and the complete results for the experiments of the main text: the approximate trees and their exact references, excess of mass (EOM) under two kinds of tie handling, the repair of deliberately damaged neighbor graphs, the two extensions, and the timing of the check.

D.1 Embeddings, approximate trees and exact references

This subsection describes the embeddings, the approximate trees and the exact references behind the check results on approximate trees in the main text.

The experiments combine three datasets, two backbones (DINOv2/v3), two encoders with ten-dimensional embeddings (AE/SupCon), and the seeds $0, 1, 2, 3, 4$. We use every point: $13{,}000$ STL-10, $60{,}000$ CIFAR-100 and $500{,}000$ iNaturalist images. With $\mu = 5, 10$, these combinations give 120 exact references. For each exact reference, we build approximate trees from an NN-Descent graph and from an HNSW graph with 30 neighbors per point, which gives the 240 main runs. With $10, 90$ neighbors per point, we build trees only for seed zero, which adds 48 additional runs each. Because the additional runs reuse the embeddings of seed zero, they should not be pooled with the main runs as independent repetitions of the five seeds.

NN-Descent runs ten iterations and requests one additional neighbor, because each point appears in its own neighbor list and is removed afterward. HNSW uses $M = 16$, a construction/query

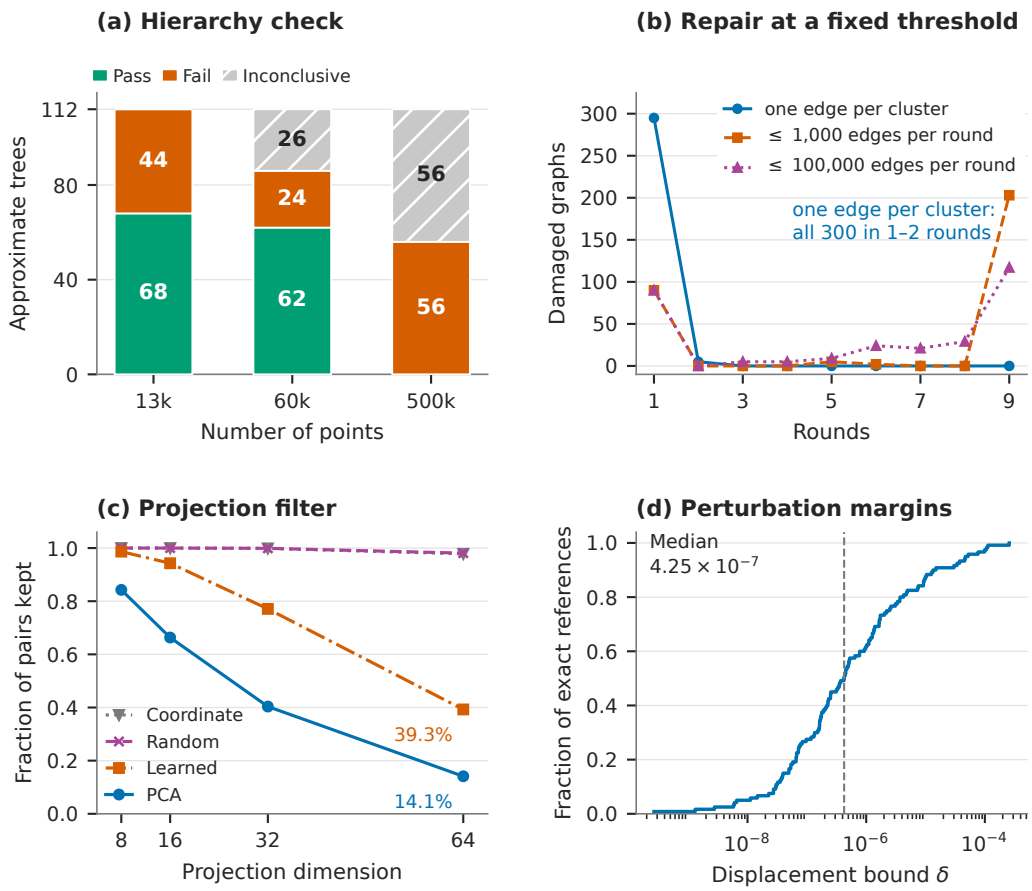

Figure 3: Results of the hierarchy check, the repair, the projection filter and the perturbation margins. **(a)** Results of the hierarchy check on the 336 approximate trees (112 per dataset). Hatched sections mark inconclusive results, in which the search reached a work limit. **(b)** Number of repair rounds on the 300 deliberately damaged neighbor graphs. For each graph, all repair variants start from the same clustering with ten clusters at a fixed threshold. **(c)** Fraction of all pairs of a query and a point that the projection filter keeps, for 64 held-out queries against each complete feature set at $\mu = 10$, averaged over six feature sets and three seeds. Every projection method appends a residual norm, and none misses a true neighbor. PCA and coordinate selection are deterministic, so their repeated runs are controls and not independent samples. The fractions measure how many pairs the filter removes, not the total running time. **(d)** Empirical distribution of the displacement bound, that is, the supremum of the displacements that the margin guarantee covers, for the clusterings of the 120 exact references. The guarantee holds at a fixed midpoint threshold and for displacements strictly below the bound.

$\mathrm{ef} = \max(100, 2\,k_{\mathrm{nn}})$, which is at least twice the number of neighbors, one insertion thread for reproducibility, and six query threads. The weights of all pairs in an ANN graph are recomputed from float64 coordinates and exact core distances before the graph is symmetrized and the tree is built. If an ANN graph is disconnected, we add bridge edges between points near the boundaries of its connected components, as in our pipeline. This step applies to 52 graphs, and all reported results use these boundary bridge edges. A control that instead joins the connected components of these graphs by a deterministic chain of bridge edges is stored separately and enters no reported result. Every ANN graph is built independently for its number of neighbors. Therefore, the edges of a graph with fewer neighbors need not be a subset of the edges of a graph with more neighbors.

The exact reference is computed with the exact mode of the hdbscan library, version 0.8.44 (`approx_min_span_tree=False`), and the weights of its tree edges are recomputed with the same expression as for the approximate trees. Recomputing the weights does not automatically certify the complete hierarchy of this tree. However, for every exact reference, the clustering at the selected threshold passes the check. To compare an approximate tree with its exact reference, we cut both trees at the threshold that the selection criterion selects on the approximate tree, and we record whether the two clusterings are identical. This comparison relies on a tree from an independent exact method. It is not a separate check of the exact reference at every such threshold. For the hierarchy check, in contrast, the selection criterion sweeps each tree separately, and the hierarchy check covers the thresholds that the selection on the approximate tree requires.

The given number of clusters is the number of evaluation classes: ten for STL-10, 100 for CIFAR-100, and 273 orders for iNaturalist. The cluster extraction therefore uses side information, and the SupCon encoders also used the number of classes during training. The minimum cluster size is ten. Section 3.3 specifies the remaining steps of the selection criterion: it selects the highest range of thresholds with the given number of clusters, falls back to the closest number of clusters that occurs, keeps the largest clusters, and gives each unassigned point the label of its nearest assigned point in Euclidean distance, a 1-NN assignment with a deterministic tie rule. A quality score with respect to the classes is computed after this assignment and describes the final clustering. The check, in contrast, covers the complete clustering at the selected threshold, including the unassigned points.

D.2 EOM, TIE HANDLING AND UNCERTAINTY

This subsection gives the protocol of the EOM comparison between approximate trees and their exact references, which the main text reports in its experiment on ties, and it describes the bootstrap intervals.

All EOM runs use the EOM selection of the hdbscan library with a minimum cluster size of ten, and they pass the tested μ as the library's minimum number of samples. EOM with the default edge order receives the tree as a list of edges that is sorted by recomputed weight and then by the endpoints, with the smaller point index first. The library builds its binary merge tree from this list. This procedure differs from the library's own fit routine, which would build its own tree. EOM with the fixed tie rule uses the same weighted trees, but it builds the binary merge tree as described in Section C. For both kinds of tie handling, the results of all 120 exact references and all 336 approximate trees are available. Two EOM results count as identical only if the same points carry the noise label -1 and the remaining clusters coincide. The reported ARI/NMI values follow the standard definitions, in which noise is one further label, and the fraction of noise points is recorded separately. These values must therefore not be mixed with the EOM tables in which every noise point receives the label of its nearest assigned point (1-NN assignment).

Table 3: Absolute NMI difference between EOM on an approximate tree and EOM on its exact reference, for the 240 main runs with 30 neighbors per point. In each row, both trees of a pair are processed with the same tie handling. Both rows use identical weighted trees.

Tie handling	median $\lvert\Delta\mathrm{NMI}\rvert$	maximum	pairs above .01
Default edge order	7.75×10^{-5}	.37431	12
Fixed tie rule	0	.01051	2

We do not call the fit routine of the library, which would build its own tree; the EOM code receives our trees directly. Compared with each other, the EOM results on an approximate tree and on its exact reference are identical for 30 of the 240 pairs under the default edge order; their ARI has a median of .999, is below .9 for eight pairs, and is negative in the worst case ($-.15$). EOM with the fixed tie rule runs on all 456 trees.

Of the 240 pairs of an approximate tree and its exact reference in the main runs, 115 have exactly equal dc-distances. For these pairs, EOM with the fixed tie rule returns identical results, including the noise points. For another 53 pairs, the dc-distances differ only by less than 10^{-9} in relative terms, and for 72 pairs the hierarchies differ more. The two NMI differences above .01 under the fixed tie rule belong to the same exact reference, STL-10 DINOv2 SupCon with seed three and

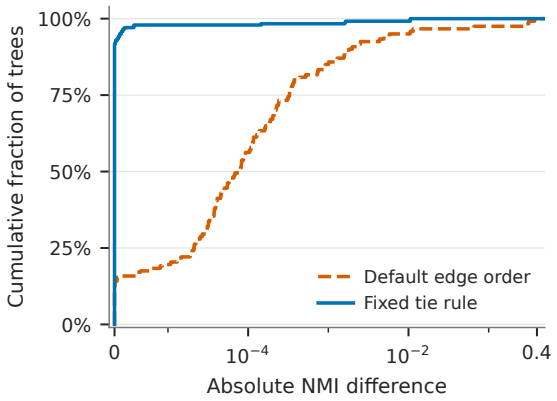

Figure 4: Distribution of the absolute NMI difference between EOM on an approximate tree and EOM on the exact reference, over the 240 trees with $k_{\mathrm{nn}} = 30$, with the default edge order of the library and with the fixed tie rule (Section 4.3). The horizontal axis is linear up to 10^{-5} and logarithmic above, so that exact zeros remain visible.

$\mu = 10$, once with NN-Descent and once with HNSW. Two such runs are correlated and do not count as independent cases.

An additional diagnostic varies the order of tied edges at random and rounds the weights to fewer decimals. It covers every exact reference for which a main run has $|\Delta\mathrm{NMI}| > .01$ under the default edge order. Each variant has a time limit of 30 s, and each of these cases has a time limit of 120 s in total. Of 180 planned variants, 120 complete, eight reach the time limit, and 52 are not run because the time limit of their case is used up. We set these limits because, on two large iNaturalist trees, the library's EOM selection takes very long for some orders of tied edges. This diagnostic is therefore incomplete, and we count its missing results neither as negative results nor as passed checks. The EOM results with the default edge order and with the fixed tie rule are complete.

Paired intervals for differences in quality use 10,000 bootstrap resamples of the five pairs of seeds. They are computed separately for every combination of dataset, backbone, embedding, density parameter and approximate method. These intervals are exploratory and are not corrected for multiple comparisons, and five seeds give limited precision. We developed the protocol with pilot runs and did not preregister it.

D.3 Protocol of the repair at a fixed threshold

This subsection gives the protocol of the repair experiment on deliberately damaged neighbor graphs, which the main text reports in its paragraph on the repair (Section 3.2).

The experiment uses 300 configurations of deliberately damaged neighbor graphs. The configurations are a fixed selection and not a full Cartesian grid of the following parameters. The data are DENSIRED samples with 20k or 50k points, $\mu = 5$, and the given number of clusters $k = 10$. The ANN graphs have between two and thirty neighbors per point, NN-Descent stops after $1, 2, 10$ iterations, and a fraction of up to .9 of the edges is deleted at random. As the exact reference, an independent implementation of Prim's algorithm on the complete graph with $O(N)$ memory runs once per input size. For each configuration, the initial threshold and the initial clustering with ten clusters are the same for the three repair variants and for the comparison with the exact reference. Every initial cluster has a spanning tree whose edges have weight at most this threshold.

In every round, our repair searches for one missing edge for each cluster that still has one. The two other variants add up to 1,000 or 100,000 missing edges per round, regardless of the cluster. All 900 repaired clusterings pass the check and equal the clustering of the exact reference at the same threshold. Our repair needs one or two rounds, with a median of one. The two variants with a global limit on the number of edges per round need a median of nine and seven rounds and at most nine. The experiment in Table 28 uses the same configurations but asks a different question, because its

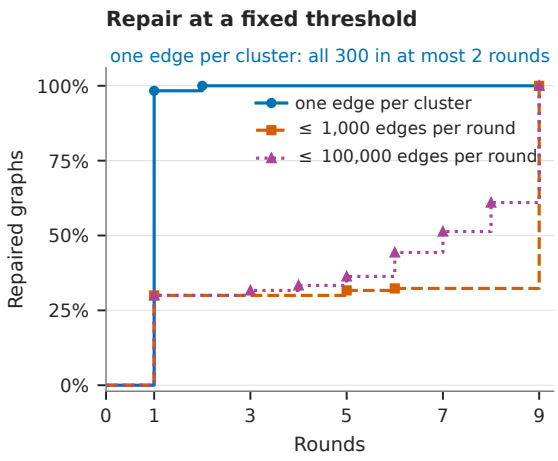

Figure 5: Fraction of the 300 damaged neighbor graphs that are repaired after a given number of rounds, for our repair with one missing edge per cluster and round and for two variants with a global limit on the number of edges per round. Every variant finally passes the check and equals the exact result. In total, our repair adds at most 9 edges to a tree with 19,999 or 49,999 edges; the two variants add up to 9,000 and 900,000.

threshold changes between rounds. Its results, in which some repairs stop at a limit on the number of rounds, therefore stand next to the results of this subsection and are not replaced by them.

D.4 Margin and perturbation protocol

This subsection gives the protocol of the perturbation experiment for the margin, which is the first of the two extensions (Section C.4).

For all 120 exact references, the clustering at the selected threshold stays certified on an interval of thresholds with positive length, which separates the weights inside the clusters from the weights between the clusters. The displacement bound, that is, the supremum of the displacements that the guarantee covers, has a minimum of 2.65×10^{-10}, a median of 4.25×10^{-7} and a maximum of 2.58×10^{-4}. Each clustering is tested under five perturbations: random displacements of 0.25, 0.75 and 1.5 times its displacement bound, a cast of the coordinates to float32, and a quantization to float16. These are 600 tests. Every point is moved by a random displacement that is bounded by the stated value. After the perturbation, we test both conditions of the check again at the fixed midpoint threshold: the spanning trees of the clusters from before the perturbation must still have edge weights of at most this threshold, and the search must find no missing edge. In every tested case within the bound, the clustering is certified again. The stored coordinates are already float32 values before they are cast for the check, so the cast to float32 changes nothing and serves as a control. If the clustering is not certified after a displacement beyond the bound or after the quantization, the sufficient condition has not established stability. Such a result does not show that the clustering has changed.

D.5 Protocol of the projection filter on the original features

This subsection gives the protocol of the experiment with the projection filter on the original features, which is the second of the two extensions (Section C.4).

We use all points of six sets of frozen 384-dimensional features (three datasets, two backbones). The features are cast to float64, and every row is normalized with an offset of 10^{-9} in the denominator. A fixed permutation with seed 73019 selects up to 8,192 points for fitting the projections (at most half of the feature set) and the next 64 points as queries. The queries are evaluated against the complete feature set, so only the fitting points and the queries are disjoint. The class labels of the evaluation are not used for fitting. We test coordinate selection, random orthogonal projection, PCA, and the

learned projection at the projection dimensions $8, 16, 32, 64$ and with the seeds $0, 1, 2$. These are 288 configurations, which include 72 fits of the learned projection. The residual norm adds one coordinate to each stated projection dimension.

The objective of the learned projection uses local negative pairs, that is, pairs of nearby fitting points whose distance exceeds the core distance for $\mu = 10$ on the fitting points. The fit takes 150 Adam steps at learning rate $.01$, each followed by a QR retraction with consistent column signs. All methods enforce the contraction bound before they append the residual norm. The true neighbor sets are computed exhaustively from the float64 distances between the queries and all points on the GPU, and distances at the boundary of a neighbor set are evaluated again with NumPy. With both $\mu = 5, 10$, this gives 576 comparisons of a filter with the true neighbor sets. No comparison contains a false negative. At projection dimension $d = 64$ and $\mu = 10$, the mean fraction of kept pairs is $.1412$ for PCA, $.3927$ for the learned projection, $.9809$ for the random projection and $.9783$ for coordinate selection. PCA and coordinate selection are deterministic. Their repeated runs are therefore controls and not independent observations from which an uncertainty could be estimated. The exact distances are computed once and reused to confirm correctness and to count the kept pairs. No statement about speed treats these distances as free. The experiment measures how many pairs a safe filter removes for queries against a complete feature set. It does not certify a clustering in high dimensional space, and it does not measure the total time of exact filtering.

D.6 Timing protocol of the check

This subsection gives the protocol and all measurements for the cost of the check in the main text.

Table 4: Time of the check and of a direct computation of the clustering: median seconds over three repetitions on six cores of a cluster node (protocol: Section E.1), on the DINOv2 AE embeddings of seed zero with $\mu = 10$. The first four rows receive the exact core distances, and their times do not include computing them. Every variant of the check receives the spanning trees of the clusters, whereas the direct computation finds the clusters itself from all pairs within the threshold. The last three rows include the queries that determine the core distances or the core points, but not the construction of the spanning trees of the clusters.

Method	13k	60k	500k
Check with a scan of all points per cluster	.081	.447	38.719
Check with core points collected first	.008	.013	.086
the same, without the C kernels	.074	.041	.515
Direct computation from all pairs within the threshold	.149	.088	2.616
Exact core distances + check	.066	.539	5.514
Core points from exact bounded queries + check	.051	.228	2.317
Core points from ANN and exact bounded queries + check	.040	.284	2.503

The methods run in three repetitions, in a random order per repetition, within one process per implementation that is pinned to six physical cores of one socket, with the thread pools limited to six threads (Section E.1). We report medians; the minima and maxima are stored with the records. The first row times an implementation of the check that scans the labels of all points once for every cluster, and it serves only as a baseline for this comparison. The implementation in the second row first collects the core points and then groups them by cluster, which avoids these scans. The scans are particularly costly on iNaturalist, where the clustering contains many small clusters of unassigned points. Three C kernels, compiled with OpenMP, replace NumPy and SciPy code: the deduplication of the edges of the ANN graph, the bounding-box filter of the check, and a kd-tree that computes the exact core distances and answers the range queries of the check and of the repair. The kd-tree evaluates distances with the same float64 expression as SciPy's `cKDTree`, so its core distances are identical bit for bit, and its search returns the same missing edges in the same order as the Python implementation. The third row runs the second implementation without these kernels: the results are identical, and at 500k points the kernels reduce the time of the check from 0.52 to 0.09 s. No timing in this table uses a GPU. Table 30 uses the same protocol. The synthetic scaling tables (Tables 22 and 29) keep their own conventions for the hardware threads and for the implementation of the check; their times should not be compared directly with this table.

E PROTOCOLS OF THE TOTAL-COST AND SENSITIVITY EXPERIMENTS

This section gives the protocols and the complete results of five experiments: the total cost of obtaining a certified clustering, the total cost on the complete training sets of iNaturalist-2021 and ImageNet-1k, the check and the repair in higher dimensionalities, and the sensitivity of our final clustering to the given number of clusters and the minimum cluster size.

E.1 TOTAL COST OF TREE CONSTRUCTION, CHECK AND REPAIR

This subsection supports the paragraph on the total cost in the main text: it describes the methods and the timing protocol, and it reports all runs.

The comparison uses the same three DINOv2 AE embeddings of seed zero as the timing of the check: 13k, 60k and 500k points in ten dimensions, with $\mu = 10$. For each embedding, the threshold is the one that the selection criterion selects on the approximate tree from the NN-Descent graph with ten neighbors per point. Every method and every repair round uses this threshold. The task is the clustering at this threshold. Selecting the threshold, assigning the unassigned points, and excess of mass (EOM) are not part of this comparison.

We build the NN-Descent and HNSW graphs with 10, 30 and 90 neighbors per point independently of each other, and we add boundary bridge edges to disconnected graphs as in the previous section. The two exact libraries are `fast_hdbscan` and hdbscan in its exact Borůvka mode. We call their MST routines directly, so that no time is spent on EOM, which is not part of the task. Every method recomputes the exact core distances in float64. The edges of the trees from the two exact libraries are reweighted with these core distances and the common distance expression. The core distances that these libraries compute internally still count toward their construction time. Every method receives the check, the repair if the check fails, and a final check after the repair. In this way, the certified clustering is defined identically for every method, and we do not need to assume that the libraries use the same floating-point arithmetic as the check.

The 72 runs consist of three repetitions of eight methods on three embeddings, in a fixed random order. The eight methods are NN-Descent and HNSW with three numbers of neighbors each, and the two exact libraries. All timings of this subsection, of Tables 4 and 30 and of Section E.2 were measured on one node of a compute cluster with two Intel Xeon Platinum 8558P processors (192 hardware threads, 2 TiB RAM), in Slurm jobs that each held 48 hardware threads and 160 GB of the node. A few small jobs of other users, which used the GPUs of the node and two to eight threads each, ran at the same time; the one-minute load average of the node before every measurement is stored with the records (median 1.5, maximum 4.7). The runs are executed one after the other, each in a fresh process pinned to six physical cores of one socket, with every thread pool limited to six threads. A second array with the isolated timing of the check and the regime sweep (Tables 4 and 30) ran on six other cores of the node during the runs at 100k and 2.69M points; the 36 runs at 500k to 2M points that overlapped with it were repeated alone, and their first versions are kept with the records. A further 18 runs repeat the six ANN methods without the C kernels of Table 4. HNSW inserts points with one thread for reproducibility and answers queries with six threads. The ANN graphs are built with seed zero in every repetition, so the ranges describe only the variability of the execution. Each process imports the same libraries and warms up on 128 synthetic points. The total time starts when the float64 coordinates are loaded. It includes the queries for the core distances, the construction of the ANN graph or of the library's tree, the MST of the ANN graph or the reweighting of the library's tree edges, the cut at the threshold, the check and the repair. Loading, warm-up, the comparison with independently computed results and the writing of result files are excluded. The peak resident set size (RSS) of the process is read at the end of the timed interval. It includes the runtime with the imported libraries, the loaded data, the warm-up and the preparation of the exact reference, which is the same in every process. The peak RSS is therefore not the additional memory that a method needs for its graph. Tables 5 and 6 report all runs and the stages at 500k points. The sum of the median stage times need not equal the median total time.

All 72 final clusterings pass the check and equal the clustering that is computed from the directly enumerated pairs within the threshold. The 18 runs of the ANN methods with ten neighbors per point fail the first check and need one repair round each. All other 54 runs pass the first check. The 18 runs without the C kernels give the same trees and clusterings; they differ only in the time of the

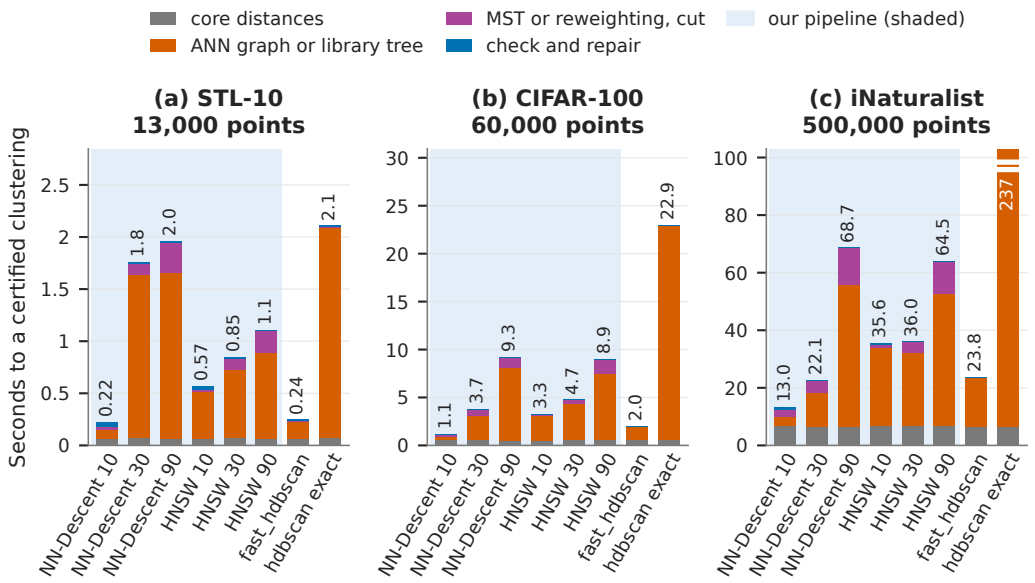

Figure 6: Total time to obtain a certified clustering at a fixed threshold on the DINOv2 AE embedding of seed zero, split into stages: the common exact core distances, the ANN graph or the library's tree, the MST of the ANN graph or the reweighting of the tree edges together with the cut, and the check with the repair. Every method uses the same threshold per dataset. Segments are the medians of the stage times over three runs in separate processes, and the number on a column is the median total time; the two can differ slightly. Warm-up and loading are not included. The panels use separate linear time axes; the column of hdbscan exact on iNaturalist is cut. The six shaded columns are our pipeline with the given ANN method and k_{nn}, the two columns on the right are the exact libraries. The exact libraries compute their own core distances inside their tree construction, in addition to the common exact core distances. Memory: Table 5.

core distances, the graph stage and the check (Table 6). NN-Descent with ten neighbors per point is the fastest method on all three embeddings, on STL-10 by a small margin (0.22 against 0.24 s for `fast_hdbscan`). The library has the smallest peak RSS on all three embeddings. At 500k points, NN-Descent with ten neighbors per point is 1.8× faster in total than `fast_hdbscan`, with the common core distances included for both methods. The library builds its own tree in 17.18 s (Table 6 and Figure 6), so even the library alone, without the common core distances and the check, is slower than the certified total of NN-Descent (12.99 s). Of this total, the check and the repair take 0.76 s. This comparison neither removes the library's own computation of core distances from its time, nor does it treat the peak RSS of a whole process as the additional memory of a method.

In a separate experiment, we repair every one of the 32 clusterings that fail the check among the 336 approximate trees. For each of them, we confirm the edge weights of the spanning trees of the clusters with the core distances of the exact reference, and we keep the threshold of the failed check fixed. We compare the repaired clustering with the clustering of the exact reference at this threshold, run the check on it again, and test the bound on the number of rounds with the actual number of parts into which an exact cluster is split before the repair. These repairs are executed concurrently with other experiments. They therefore support statements about correctness and about the number of rounds, but not about speed.

E.2 Total cost on the complete iNaturalist-2021 training set

This subsection supports the paragraph on the total cost in the main text: it describes the embedding and the timing protocol, and it reports all runs, including the peak memory (Figure 7).

Table 5: Total cost of obtaining a certified clustering at a fixed threshold: median seconds over three runs in separate processes (the minima and maxima are stored with the records), and the largest peak RSS of a process in GiB. Every row includes the recomputed common core distances, the check and, if the check fails, the repair at the fixed threshold. For the two exact libraries, the cost of their own tree construction is included in addition to the common core distances. Rows marked Python (one run each) run without the three C kernels of Table 4; all other rows, including the exact libraries, use them.

	Total seconds			Peak GiB		
Method, neighbors	13k	60k	500k	13k	60k	500k
NN-Descent, 10	0.217	1.15	12.99	0.91	1.01	1.88
NN-Descent, 30	1.76	3.73	22.07	0.97	1.17	2.70
NN-Descent, 90	1.97	9.31	68.69	1.28	1.63	4.05
HNSW, 10	0.566	3.25	35.61	0.91	0.99	1.73
HNSW, 30	0.848	4.74	36.00	0.95	1.16	2.59
HNSW, 90	1.10	8.95	64.50	1.05	1.62	3.99
`fast_hdbscan`	0.244	2.01	23.78	0.89	0.91	1.10
hdbscan exact	2.11	22.92	236.8	0.90	0.95	1.29
NN-Descent, 10 (Python)	0.660	1.52	16.64	0.91	1.02	1.86
NN-Descent, 30 (Python)	1.52	5.12	31.98	0.97	1.17	2.69
NN-Descent, 90 (Python)	2.65	12.99	102.5	1.28	1.59	4.09
HNSW, 10 (Python)	0.873	3.62	37.03	0.91	1.01	1.70
HNSW, 30 (Python)	1.27	5.93	46.65	0.95	1.18	2.57
HNSW, 90 (Python)	1.91	10.04	98.30	1.05	1.58	4.03

Table 6: Median seconds per stage on iNaturalist-500k, rows as in Table 5. The column graph contains the MST of the ANN graph for the ANN methods, or the reweighting of the tree edges for the exact libraries. The column cut contains cutting the tree at the threshold. The last column contains the first check, the repair if the first check fails, and the check after the repair.

Method, neighbors	core distances	build	graph	cut	check/repair
NN-Descent, 10	6.93	3.15	2.24	0.014	0.759
NN-Descent, 30	6.55	11.75	4.14	0.010	0.093
NN-Descent, 90	6.62	49.43	12.50	0.009	0.091
HNSW, 10	6.80	27.24	1.07	0.009	0.480
HNSW, 30	6.81	25.57	3.52	0.009	0.090
HNSW, 90	6.85	45.83	11.17	0.009	0.090
`fast_hdbscan`	6.42	17.18	0.058	0.013	0.096
hdbscan exact	6.64	229.8	0.049	0.013	0.092
NN-Descent, 10 (Python)	8.07	2.95	4.44	0.009	1.17
NN-Descent, 30 (Python)	7.48	9.46	14.50	0.010	0.527
NN-Descent, 90 (Python)	7.58	47.73	46.67	0.010	0.545
HNSW, 10 (Python)	7.47	24.06	4.35	0.009	1.15
HNSW, 30 (Python)	7.49	24.21	14.41	0.009	0.527
HNSW, 90 (Python)	7.80	43.35	46.62	0.009	0.527

Embedding. We extract the features of all $2{,}686{,}843$ training images of iNaturalist-2021 with the frozen DINOv3 ViT-S/16 backbone (384 dimensions), with the same preprocessing as in the other iNaturalist experiments. One autoencoder with the layers 384–512–256–128–10 and a mirrored decoder is trained on all features for 100 epochs with Adam (learning rate 10^{-3}, batch size 256, seed zero). Its outputs are normalized to unit length and stay fixed for all subset sizes. The subsets with 100k, 500k, 1M and 2M points are prefixes of one uniform random permutation of the images, so they are nested. The subset with 500k points is therefore not the set of 500k images of the other experiments. Extracting the features takes 80 minutes and training the autoencoder 106 minutes on one A100. These times are not part of the totals.

Protocol. The protocol is that of Section E.1, with $\mu = 10$, six CPU threads, separate processes, and three repetitions per method and size, which gives 45 runs, and a further 10 runs of NN-Descent without the C kernels. The methods are NN-Descent with 10 and 30 neighbors per point

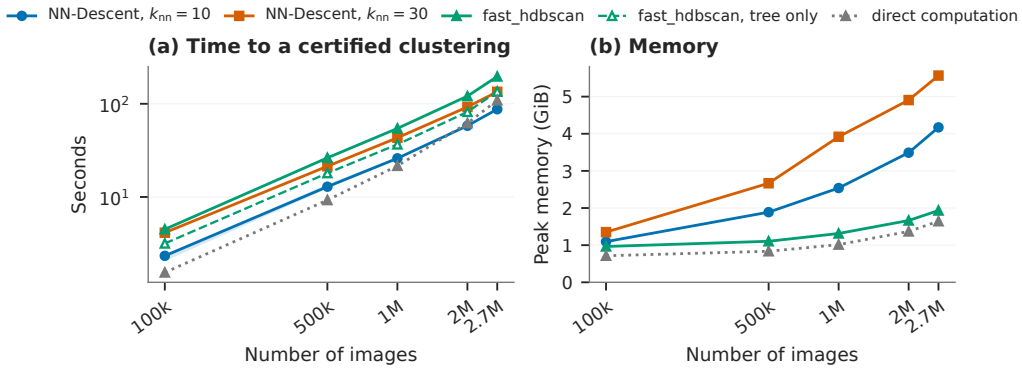

Figure 7: Total cost on the complete iNaturalist-2021 training set. (a) Time to obtain a certified clustering at a fixed threshold, including core distances, tree construction, check and repair, on nested subsets of one embedding of all 2,686,843 images. The dashed line is the time in which `fast_hdbscan` builds its tree alone, without the common core distances and without the check; the dotted line is the direct computation of the clustering at the known threshold. (b) Largest peak memory of a process. Points are medians over three runs in separate processes on six CPU threads, and shaded bands are full ranges. Stages: Section E.2.

and `fast_hdbscan`. In addition, we time the direct computation of the clustering at the known threshold, which enumerates all pairs of core points within distance ε in batches and joins them with a union–find structure, without any tree (`reproduce/direct_dbscan_star.py`; 15 runs). For every size, the threshold is fixed before the timing. It is the threshold that the selection criterion selects, with $k = 273$ and $m = 10$, on the approximate tree from the NN-Descent graph with ten neighbors per point. At 2.69M points, 755,747 points are core points at this threshold. The task is the clustering at this threshold. The experiment therefore measures cost and correctness at a fixed threshold in the 10-dimensional embedding. It does not compare clustering quality, and it does not certify the choice of the threshold.

Results. Table 7 reports all runs. All 55 final clusterings pass the check, and the 15 direct computations return the same clustering. All 20 runs of NN-Descent with ten neighbors per point fail the first check and need one repair round. Before the repair, an exact cluster is split into at most 27 parts. At every size, the total of NN-Descent with ten neighbors per point is below the time in which `fast_hdbscan` builds its tree alone (column build), by 26 to 35 percent. For NN-Descent with ten neighbors per point, the core distances are the largest stage at every size. The direct computation is the fastest method up to 1M points and second to NN-Descent with ten neighbors per point at 2M and 2.69M points (61.9 against 58.1 s and 108.8 against 87.5 s), where the enumeration of the pairs within the threshold takes 23 and 48 s; its peak RSS is the smallest at every size, and among the tree-based methods, `fast_hdbscan` has the smallest peak RSS at every size.

Independent comparison. A separate program enumerates all pairs of core points within the threshold distance in batches and computes their connected components with a union–find structure. At 2.69M points it enumerates 20,845,860 pairs. The resulting clustering equals the final clustering of every run. Timed under the protocol of the other methods, the same enumeration is the direct computation in Table 7: at 2.69M points, 61 s for the core distances and 48 s for the pairs and their components. The program also recomputes the weight of every stored tree edge, verifies that the subsets are nested, and recomputes the core distance of 32 points per size by an exhaustive search.

E.3 Total cost on ImageNet-1k

We apply the protocol of Section E.2 to the 1,281,167 training images of ImageNet-1k (Deng et al., 2009; Russakovsky et al., 2015): frozen DINOv3 ViT-S/16 features, the same autoencoder (384–512–256–128–10, 100 epochs, seed zero, outputs normalized to unit length; extraction 23 and training 45 minutes on one H200), the threshold that the selection criterion picks, with

Table 7: Total cost on nested subsets of the complete iNaturalist-2021 training set: median seconds [minimum, maximum] over three runs in separate processes on six CPU threads, median seconds per stage, and the largest peak RSS of a process in GiB. The column build contains the construction of the ANN graph or, for `fast_hdbscan`, of the library's tree, which includes the library's own core distances. The column graph contains the MST of the ANN graph or the reweighting of the tree edges. The column check/repair contains the first check, the repair if the first check fails, and the check after the repair. Every failed first check is repaired in one round. Parts is the largest number of parts into which an exact cluster is split before the repair. Rows marked Python (one run each) run NN-Descent without the C kernels. The row direct computation needs no tree: its column graph contains the enumeration of the pairs of core points within distance ε and their connected components, and it needs no check.

N	Method, neighbors	Total seconds [range]	core distances	build	graph	check/repair	GiB	first check	parts
100k	NN-Descent, 10	2.34 [2.05, 2.37]	1.32	0.570	0.261	0.138	1.10	fail	4
	NN-Descent, 30	4.14 [4.06, 4.15]	1.31	2.07	0.722	0.022	1.35	pass	1
	`fast_hdbscan`	4.50 [4.48, 4.62]	1.33	3.17	0.009	0.029	0.96	pass	1
	NN-Descent, 10 (Python)	3.01 [3.01, 3.01]	1.39	0.578	0.803	0.234	1.11	fail	4
	NN-Descent, 30 (Python)	6.04 [6.04, 6.04]	1.37	2.07	2.53	0.074	1.34	pass	1
	direct computation	1.55 [1.51, 1.57]	1.29	—	0.257	—	0.72	—	—
500k	NN-Descent, 10	12.90 [12.30, 13.69]	7.89	2.92	1.85	0.431	1.89	fail	4
	NN-Descent, 30	21.35 [20.96, 22.26]	7.79	9.22	4.15	0.073	2.67	pass	1
	`fast_hdbscan`	26.30 [25.54, 27.05]	8.09	18.02	0.071	0.076	1.10	pass	1
	NN-Descent, 10 (Python)	17.46 [17.46, 17.46]	9.66	2.40	4.38	1.03	1.84	fail	4
	NN-Descent, 30 (Python)	34.10 [34.10, 34.10]	9.76	9.35	14.45	0.538	2.67	pass	1
	direct computation	9.26 [8.85, 9.52]	8.11	—	1.15	—	0.84	—	—
1M	NN-Descent, 10	25.99 [25.44, 26.90]	17.19	5.19	2.68	0.923	2.54	fail	8
	NN-Descent, 30	43.20 [42.55, 43.29]	16.73	17.85	8.50	0.158	3.92	pass	1
	`fast_hdbscan`	54.41 [53.25, 55.45]	17.45	36.65	0.152	0.164	1.31	pass	1
	NN-Descent, 10 (Python)	36.40 [36.40, 36.40]	18.89	5.19	9.47	2.84	2.48	fail	8
	NN-Descent, 30 (Python)	67.69 [67.69, 67.69]	18.53	17.84	29.95	1.36	3.93	pass	1
	direct computation	21.59 [21.20, 22.10]	17.66	—	3.93	—	1.01	—	—
2M	NN-Descent, 10	58.14 [57.66, 58.92]	38.23	11.34	5.93	2.65	3.49	fail	27
	NN-Descent, 30	92.49 [92.49, 93.76]	37.93	37.10	17.18	0.568	4.91	pass	1
	`fast_hdbscan`	120.8 [119.7, 124.4]	37.98	81.82	0.327	0.591	1.66	pass	1
	NN-Descent, 10 (Python)	118.7 [118.7, 118.7]	71.72	11.40	19.47	16.06	3.58	fail	27
	NN-Descent, 30 (Python)	177.8 [177.8, 177.8]	71.41	37.20	63.49	5.64	4.89	pass	1
	direct computation	61.92 [61.88, 61.92]	38.98	—	22.93	—	1.37	—	—
2.69M	NN-Descent, 10	87.46 [86.82, 87.56]	59.61	15.88	7.96	3.80	4.17	fail	23
	NN-Descent, 30	134.6 [134.3, 134.8]	59.74	50.68	23.43	0.870	5.57	pass	1
	`fast_hdbscan`	196.2 [192.2, 196.9]	58.99	135.3	0.446	0.892	1.93	pass	1
	NN-Descent, 10 (Python)	206.1 [206.1, 206.1]	122.8	15.90	27.30	40.03	4.12	fail	23
	NN-Descent, 30 (Python)	271.6 [271.6, 271.6]	122.5	50.70	86.29	12.00	6.33	pass	1
	direct computation	108.8 [108.1, 109.3]	61.11	—	47.78	—	1.64	—	—

$k = 1{,}000$ and $m = 10$, on the NN-Descent tree with ten neighbors, and three repetitions per method in fresh processes on six cores. These runs were measured on an exclusively allocated node with two Intel Xeon Platinum 8480C processors (`factorial/extract_imagenet.py`, `factorial/embed_inat_full.py`, `extended/inat_scaling.py` and `slurm_timing.py`; records `results/imagenet_full_20260925/timing_idle/`; a first run on a shared node under load, kept in `timing/`, also gives the certified clustering earlier than the library's tree alone, 28 against 43 s). Table 8 reports all runs. NN-Descent with ten neighbors fails the first check, with an exact cluster split into three parts, is repaired in one round and takes 27 s in total, of which the check and the repair take 1.3 s; the library needs 53 s for its tree alone and 68 s with the common core distances and the check, and the direct computation at the known threshold takes 17 s and, as at one million points of iNaturalist, is the fastest method. Every final clustering equals the reference clustering.

E.4 Check and repair in higher dimensionalities

The main experiments use 10-dimensional embeddings. To see how the check and the repair behave in higher dimensionalities on real data, we run the pipeline on PCA projections of the frozen 384-dimensional DINOv2 ViT-S/14 features of the three datasets, centered and without whitening, with

Table 8: Total cost on the complete ImageNet-1k training set; columns as in Table 7.

N	Method, neighbors	Total seconds [range]	core distances	build	graph	check/repair	GiB	first check	parts
1.28M	NN-Descent, 10	26.93 [26.93, 26.94]	14.63	7.21	3.74	1.32	2.85	fail	3
	NN-Descent, 30	51.96 [51.79, 51.97]	14.62	25.26	11.73	0.215	4.24	pass	1
	`fast_hdbscan`	68.05 [67.09, 68.36]	14.62	53.01	0.157	0.222	1.43	pass	1
	direct computation	17.41 [17.40, 17.43]	14.61	—	2.81	—	1.13	—	—

$d \in \{16, 32, 64, 128\}$ for all three datasets and all 384 dimensions for STL-10 and CIFAR-100 (`reproduce/highdim_check.py`, records `results/highdim_20260925_kd/`). For every dataset and dimension, a reference process computes the exact core distances ($\mu = 10$) with SciPy's kd-tree, the threshold that the selection criterion of Section 3.3 picks on an NN-Descent tree with 10 neighbors (k equal to the number of classes, $m = 10$), and the exact clustering at this threshold by the direct computation from all pairs of core points within the threshold; this clustering passes the check in every case. Then one fresh process per method measures, on six cores of a shared machine with two Intel Xeon Gold 6326 processors and with the C kernels of Table 4, the stages of the pipeline: exact core distances, ANN graph, tree, cut, check, repair and final check. Every final clustering equals the reference clustering (Table 9).

Three observations. First, the check finds every wrong clustering at every dimension, and the repair corrects it: with 10 neighbors, NN-Descent splits an exact cluster into up to 66 parts, and the repair needs one round in all cases but one, where it needs two (45 parts at 64 dimensions on iNaturalist; the bound of Lemma 1 is six, and seven for 66 parts); with 30 neighbors, the first check passes in all cases but one: at 128 dimensions on iNaturalist, HNSW misses four edges and splits an exact cluster into four parts; the check fails, and one round of the repair adds the four edges. A larger neighbor budget is no guarantee in higher dimensionalities; the check is. Second, the cost that grows with the dimension is the exact core distances: on iNaturalist-500k they take 67 s at 16 dimensions and 3,895 s at 128, whereas one check takes at most 113 s, and the check, the repair and the final check together at most 410 s; on CIFAR-100 the core distances grow from 2 s at 16 dimensions to 151 s at 384, the check stays below 2 s. Third, as on the 10-dimensional embeddings, the threshold that the criterion selects leaves most points of CIFAR-100 and iNaturalist unassigned: 3,430 to 9,066 of the 60,000 and 31,983 to 190,212 of the 500,000 points are core points, whereas on STL-10 7,793 to 11,210 of the 13,000 points are; on iNaturalist, the number of core points and with it the work of the check grow with the dimension.

E.5 Sensitivity to the number of clusters and the minimum cluster size

This subsection tests how sensitive our final clustering is to the given number of clusters k and to the minimum cluster size m; it describes the settings and reports all results. In short, all 90 resulting clusterings pass the check. Their ARI stays above that of k-means in all 18 settings on average ($+.146$ to $+.336$) and in 89 of the 90 runs. The selection criterion reaches the given k in only 9 of the 90 runs. In the other 81 it selects the closest number of clusters that occurs, which can give the same threshold for different k. The kept clusters contain 7.2% to 34.4% of the points, and all other points receive their label from the nearest assigned point.

We use all ten iNaturalist AE embeddings (two backbones, five seeds) and do not retrain any encoder. The trees are the exact references with $\mu = 10$. We combine the given numbers of clusters 218, 273 and 328 (approximately -20%, the nominal number of classes, and $+20\%$) with the minimum cluster sizes 5, 10 and 20, which yields 90 clusterings. The check runs on the complete clustering at each selected threshold. The final labels give every unassigned point the label of its nearest assigned point (1-NN assignment), as in the main text. We fit k-means with ten initializations and the seed of the embedding for each of the three numbers of clusters, which gives 30 fits. Each fit serves for all three minimum cluster sizes, because k-means does not depend on the minimum cluster size. Both methods are given the same number of clusters. The experiment tests how sensitive the cluster extraction is to its two settings on fixed embeddings that were trained for reconstruction only. It does not test model selection for an unknown number of classes, and it does not test SupCon training without the number of classes.

Table 10 reports every setting as an average over the five seeds of each backbone. The column assigned gives the percentage of points that lie in the kept clusters. All other points are unassigned

Table 9: Check and repair on PCA projections of the 384-dimensional DINOv2 features, one run per cell, six threads, seconds. "Var." is the fraction of the variance that the projection keeps, "core points" their number at the selected threshold, and "cores" the time of the exact core distances in the run with NN-Descent and 10 neighbors. For NN-Descent with 10 neighbors: result of the first check, largest number of parts of a split exact cluster, time of the first check, the repair and the final check together, and total time to the certified clustering; for NN-Descent and HNSW with 30 neighbors, the time of the first check and the total time (†: this check fails, 4 parts, and its time includes the repair and the final check).

					NN-Descent, $k_{\mathrm{nn}} = 10$				NN-Descent, 30		HNSW, 30	
Dataset	d	var.	core points	cores	check	parts	check + repair	total	check	total	check	total
STL-10	16	0.35	7,793	0.12	pass	1	0.01	0.26	0.01	1.82	0.01	0.80
STL-10	32	0.49	10,087	0.34	fail	2	0.13	0.59	0.03	2.08	0.03	1.26
STL-10	64	0.64	10,420	0.89	fail	2	0.48	1.51	0.14	2.73	0.14	2.11
STL-10	128	0.81	11,146	1.98	fail	5	1.09	3.39	0.24	4.49	0.22	3.80
STL-10	384	1.00	11,210	6.04	fail	3	5.34	12.35	0.93	10.87	0.93	10.83
CIFAR-100	16	0.28	3,430	2.07	fail	2	0.08	2.66	0.02	5.13	0.01	6.61
CIFAR-100	32	0.40	9,066	7.30	fail	2	0.21	8.13	0.06	10.76	0.06	12.74
CIFAR-100	64	0.56	8,022	23.75	fail	3	0.46	25.23	0.14	28.14	0.13	30.65
CIFAR-100	128	0.74	7,774	57.10	fail	2	1.11	60.19	0.33	64.10	0.34	67.62
CIFAR-100	384	1.00	8,756	151	fail	3	7.61	192	1.73	168	1.85	178
iNaturalist	16	0.28	31,983	67.31	fail	8	0.46	74.09	0.08	89.81	0.07	127
iNaturalist	32	0.42	69,141	471	fail	15	2.92	483	0.88	502	0.88	543
iNaturalist	64	0.59	190,212	2263	fail	45	257	2622	74.25	2500	26.37	2337
iNaturalist	128	0.78	186,978	3895	fail	66	340	4396	113	4449	410†	4747

and receive the label of their nearest assigned point. The number of kept clusters can differ from the given number of clusters, because the selection criterion falls back to the closest number of clusters that occurs. Settings that share an embedding, and settings that reuse one k-means fit, are correlated comparisons. The stored results also contain the scores per seed, a test that the nominal setting reproduces the scores obtained for the same exact references in the previous section, and separate scores for the assigned and for the unassigned points. The separate scores describe different sets of points and cannot be compared with each other.

All 90 clusterings pass the check. All 18 mean ARI differences over five seeds favor our final clustering, and they range from .146 to .336. The smallest mean difference occurs for DINOv2 at $(k, m) = (218, 5)$, where our final clustering reaches an ARI of .263 and k-means .117, compared with .404 and .097 at the nominal setting $(273, 10)$. The setting with the smallest mean difference also contains the only paired difference that is not positive (seed one, .079 versus .115). With DINOv3, the mean ARI of our final clustering ranges from .391 to .400. The selection criterion reaches the given number of clusters in only nine clusterings, all of them with DINOv2. In the remaining 81 clusterings, it selects the closest number of clusters that occurs. Equal scores for different given numbers of clusters therefore do not show that the number of classes was recovered. Over the individual clusterings, the percentage of assigned points ranges from 7.2% to 34.4%.

F Factorial study: full tables

This section contains the comparison of our final clustering with k-means and with HDBSCAN's selection (Section F.1), and it supports the discussion of HDBSCAN's cluster selection in the main text. It also contains the full tables of the factorial study and the protocol with which the embeddings were trained; Figure 9 summarizes three of its results. The study combines three image datasets and two frozen backbones with three embeddings of the images: the features of the frozen backbone (frozen), the output of an autoencoder (AE), and the output of a supervised contrastive encoder (SupCon). We call one combination of dataset, backbone and embedding a configuration. In the tables, iNat abbreviates iNaturalist, and v2 and v3 abbreviate the backbones DINOv2 and DINOv3.

All tables of this section use the training protocol described below and the fit routine of the hdbscan library with the tie handling of the library. In some runs, HDBSCAN's excess-of-mass (EOM) criterion returns at most four clusters, and we then say that EOM collapses. How often EOM collapses

Table 10: Sensitivity of our final clustering on the iNaturalist AE embeddings: all 18 settings, each with five seeds. The second and third columns contain the given number of clusters and the minimum cluster size. The ARI of our final clustering is given as mean±population standard deviation. The other scores, the percentage of assigned points and the number of kept clusters are means. The k-means result for the same number of clusters is repeated for every minimum cluster size. The last column counts the passed checks, which cover the complete clustering at the selected threshold.

			ARI		NMI		assigned	kept	check
Backbone	k	m	Ours	k-means	Ours	k-means	%	clusters	passes
DINOv2	218	5	0.263 ± 0.115	0.117	0.444	0.507	30.6	218.0	5/5
DINOv2	218	10	0.406 ± 0.064	0.117	0.513	0.507	15.1	212.2	5/5
DINOv2	218	20	0.405 ± 0.066	0.117	0.509	0.507	14.3	163.6	5/5
DINOv2	273	5	0.412 ± 0.062	0.097	0.513	0.506	14.8	270.0	5/5
DINOv2	273	10	0.404 ± 0.063	0.097	0.512	0.506	15.0	214.2	5/5
DINOv2	273	20	0.405 ± 0.066	0.097	0.509	0.506	14.3	163.6	5/5
DINOv2	328	5	0.400 ± 0.055	0.085	0.512	0.505	14.3	276.0	5/5
DINOv2	328	10	0.404 ± 0.063	0.085	0.512	0.505	15.0	214.2	5/5
DINOv2	328	20	0.405 ± 0.066	0.085	0.509	0.505	14.3	163.6	5/5
DINOv3	218	5	0.391 ± 0.036	0.086	0.440	0.447	10.1	185.4	5/5
DINOv3	218	10	0.397 ± 0.012	0.086	0.441	0.447	9.3	133.0	5/5
DINOv3	218	20	0.400 ± 0.014	0.086	0.439	0.447	9.1	94.0	5/5
DINOv3	273	5	0.391 ± 0.036	0.075	0.440	0.446	10.1	185.4	5/5
DINOv3	273	10	0.397 ± 0.012	0.075	0.441	0.446	9.3	133.0	5/5
DINOv3	273	20	0.400 ± 0.014	0.075	0.439	0.446	9.1	94.0	5/5
DINOv3	328	5	0.391 ± 0.036	0.064	0.440	0.445	10.1	185.4	5/5
DINOv3	328	10	0.397 ± 0.012	0.064	0.441	0.445	9.3	133.0	5/5
DINOv3	328	20	0.400 ± 0.014	0.064	0.439	0.445	9.1	94.0	5/5

in these tables is an observation about these settings. The frequencies do not isolate an effect of the ANN search, and they do not show a general advantage of contrastive training. EOM with the fixed tie rule, which runs on trees whose edge weights are recomputed with one common expression, is reported separately in Section 4.3 and table 3. That experiment does not repeat every run of the fit routine that is reported here. Unless stated otherwise, NMI and ARI are computed after each noise point has received the label of its nearest assigned point in Euclidean distance. We call this step 1-NN assignment, and we apply it in the same way to all compared methods.

F.1 Comparison with k-means and with HDBSCAN's selection

This experiment asks when density-based clustering pays off, and how the clustering that our pipeline certifies compares with k-means and with HDBSCAN's own selection. For each of the 60 embeddings we compare three methods: k-means, HDBSCAN's EOM selection on the exact tree with `min_samples` $= \mu = 10$, and the final clustering of our pipeline (Section 3.3) with $\mu = 10$. k-means and our selection criterion are given the number of classes k; EOM chooses the number of clusters itself. Points that EOM labels as noise receive the label of their nearest assigned point, as in our pipeline. All 60 final clusterings of our pipeline pass the check.

On iNaturalist, the largest dataset, whose 273 classes have very different sizes, density-based clustering pays off, as Table 11 and Figure 8 show. Our certified clustering reaches an ARI between .34 and .42 in all four configurations, about four times that of k-means. An analysis of the class sizes links this advantage to the large classes, which the density-based clustering keeps together, whereas k-means splits them. NMI, which is less sensitive to such splits, is nearly equal for the two methods (.515 against .507 on the analyzed embedding). At the selected threshold, 84% of the points of this embedding are unassigned and receive their label from the nearest assigned point (Section F.2). Our selection criterion is also the more robust of the two density-based extractions. It never collapses, whereas EOM returns at most four clusters in 9 of the 30 runs with DINOv3 features; Section 4.3 shows for one such case that the collapse depends only on the order of tied merges. Averaged over the four configurations, our pipeline is therefore ahead of EOM on iNaturalist (.39 against .27) and on CIFAR-100 (.28 against .24). Where EOM does not collapse, the two are within one standard

Table 11: Adjusted Rand index (ARI, mean±std over five seeds) on the 10-dimensional embeddings. k-means and our pipeline are given the number of classes; EOM runs on the exact tree and chooses the number of clusters itself. "Collapsed" counts the seeds in which EOM returns at most four clusters. NMI and further settings: Tables 14 and 18.

Dataset	Backbone	Encoder	k-means	EOM	collapsed	Ours
STL-10	DINOv2	AE	**.959** ±.008	.566 ±.220	0/5	.475 ±.075
		SupCon	**.837** ±.023	.608 ±.148	0/5	.359 ±.171
	DINOv3	AE	**.909** ±.015	.457 ±.172	0/5	.341 ±.038
		SupCon	.730 ±.104	**.745** ±.038	0/5	.427 ±.092
CIFAR-100	DINOv2	AE	**.391** ±.012	.325 ±.012	0/5	.271 ±.015
		SupCon	**.420** ±.014	.285 ±.038	0/5	.334 ±.007
	DINOv3	AE	**.401** ±.005	.082 ±.117	3/5	.247 ±.020
		SupCon	**.343** ±.013	.260 ±.037	0/5	.257 ±.023
iNaturalist	DINOv2	AE	.097 ±.003	**.435** ±.088	0/5	.404 ±.063
		SupCon	.100 ±.006	**.440** ±.094	0/5	.417 ±.067
	DINOv3	AE	.075 ±.001	.000 ±.000	4/5	**.397** ±.012
		SupCon	.091 ±.005	.205 ±.170	2/5	**.335** ±.045

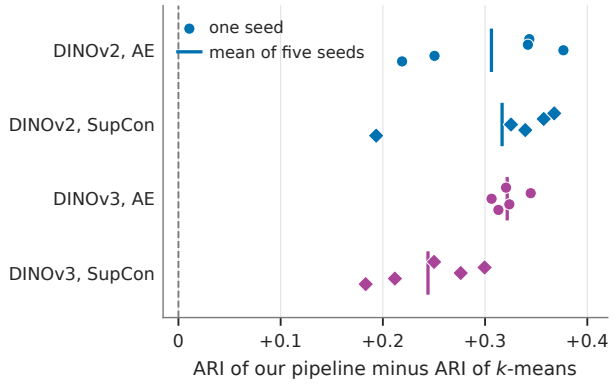

Figure 8: Difference in ARI between the final clustering of our pipeline and k-means on iNaturalist (500,000 images). Rows give the backbone and the encoder; every point is one seed, and the vertical line is the mean of the five seeds. Both methods are given $k = 273$ (Table 11). The difference is positive in all 20 runs.

deviation on iNaturalist, and EOM is ahead on STL-10. On the two balanced datasets, k-means remains the best choice when the number of classes is known. Most important for this paper, our final clustering can be certified at its threshold. The check passes in all 60 cases, so the final clustering equals the one that the exact tree gives at the same threshold. That the exact tree selects the same threshold is certified only by the hierarchy check, which does not pass at 500,000 points within our work limits (Section 4.1). An EOM result needs the hierarchy check over the complete hierarchy (Section 3.3).

The EOM results in Tables 11 and 18 use the tie handling of the library, because they describe what a user of the library obtains; we did not recompute them with the fixed tie rule of Section 4.3. EOM collapses, that is, it returns at most four clusters, in 2 of the 30 runs on SupCon embeddings and in 7 of the 30 runs on AE embeddings. This holds under these settings and this tie handling, and it does not show a general advantage of the training method. For our final clustering, the difference in quality between SupCon and AE is not consistent across datasets and backbones (Tables 13 and 14).

Quality before and after the repair. Table 12 supports Section 4.4. It lists, for the 32 clusterings that failed the check in Section 4.1, how much the repair changes the clustering and its ARI with

respect to the classes. The threshold stays fixed, and after the repair every clustering equals the exact one.

Table 12: Clustering quality before and after the repair for the 32 clusterings that failed the check (Section 4.4). Embedding: backbone, encoder, seed and μ. Parts: largest number of parts into which an exact cluster was split. Split: percentage of the points that lie in an exact cluster that the approximate tree had split. Between: ARI between the clustering before and the clustering after the repair. The last four columns give the ARI with respect to the classes, first with noise points as clusters of size one (singletons), then for the final labels of our pipeline, in which every unassigned point receives the label of its nearest assigned point. Script: `reproduce/repair_quality.py`.

						singletons		final labels	
Dataset	Embedding	Index, k_{nn}	parts	split	between	before	after	before	after
CIFAR-100	DINOv2, AE, 0, 10	HNSW, 10	3	0.2	1.000	.055	.055	.247	.247
CIFAR-100	DINOv2, AE, 0, 10	NN-Descent, 10	3	0.2	1.000	.055	.055	.247	.247
CIFAR-100	DINOv2, SupCon, 0, 10	HNSW, 10	2	0.1	1.000	.048	.048	.330	.330
CIFAR-100	DINOv2, SupCon, 0, 10	NN-Descent, 10	2	0.2	.999	.048	.048	.330	.330
CIFAR-100	DINOv2, SupCon, 3, 10	HNSW, 30	2	1.0	.959	.057	.057	.338	.336
CIFAR-100	DINOv2, SupCon, 3, 10	NN-Descent, 30	2	1.0	.959	.057	.057	.338	.336
CIFAR-100	DINOv3, AE, 0, 10	HNSW, 10	2	1.1	.974	.051	.051	.264	.261
CIFAR-100	DINOv3, AE, 0, 10	NN-Descent, 10	2	1.1	.974	.051	.051	.264	.261
CIFAR-100	DINOv3, SupCon, 0, 10	HNSW, 10	2	1.7	.999	.153	.153	.254	.254
CIFAR-100	DINOv3, SupCon, 0, 10	NN-Descent, 10	2	1.7	.999	.153	.153	.254	.254
iNaturalist	DINOv2, AE, 0, 10	HNSW, 10	4	6.1	.999	.091	.091	.444	.445
iNaturalist	DINOv2, AE, 0, 10	NN-Descent, 10	4	6.1	.999	.091	.091	.444	.445
iNaturalist	DINOv2, AE, 0, 5	HNSW, 10	2	0.1	1.000	.161	.161	.467	.467
iNaturalist	DINOv2, AE, 0, 5	NN-Descent, 10	2	1.0	1.000	.161	.161	.467	.467
iNaturalist	DINOv2, SupCon, 0, 10	HNSW, 10	2	7.8	.986	.125	.124	.453	.444
iNaturalist	DINOv2, SupCon, 0, 10	NN-Descent, 10	2	7.8	.986	.125	.124	.453	.444
iNaturalist	DINOv2, SupCon, 0, 5	HNSW, 10	2	0.0	1.000	.112	.112	.466	.466
iNaturalist	DINOv2, SupCon, 0, 5	HNSW, 30	2	0.0	1.000	.112	.112	.466	.466
iNaturalist	DINOv3, AE, 0, 10	HNSW, 10	4	0.8	1.000	.044	.044	.402	.401
iNaturalist	DINOv3, AE, 0, 10	NN-Descent, 10	4	0.8	1.000	.044	.044	.402	.401
iNaturalist	DINOv3, SupCon, 0, 10	HNSW, 10	3	5.1	1.000	.108	.108	.333	.332
iNaturalist	DINOv3, SupCon, 0, 10	NN-Descent, 10	3	5.1	1.000	.108	.108	.333	.332
iNaturalist	DINOv3, SupCon, 0, 5	NN-Descent, 10	2	3.3	1.000	.069	.069	.381	.381
STL-10	DINOv2, AE, 0, 10	HNSW, 10	2	45.4	.771	.473	.333	.584	.392
STL-10	DINOv2, AE, 0, 10	NN-Descent, 10	2	45.4	.771	.473	.333	.584	.392
STL-10	DINOv2, SupCon, 1, 10	HNSW, 30	2	6.1	.999	.407	.410	.412	.416
STL-10	DINOv2, SupCon, 1, 10	NN-Descent, 30	2	5.3	.999	.407	.410	.412	.416
STL-10	DINOv2, SupCon, 1, 5	HNSW, 30	2	0.8	1.000	.409	.409	.412	.412
STL-10	DINOv3, AE, 0, 10	HNSW, 10	2	35.5	.980	.295	.288	.376	.366
STL-10	DINOv3, AE, 0, 10	NN-Descent, 10	2	35.5	.980	.295	.288	.376	.366
STL-10	DINOv3, SupCon, 0, 10	HNSW, 10	3	49.2	.946	.372	.361	.370	.360
STL-10	DINOv3, SupCon, 0, 10	NN-Descent, 10	3	49.2	.946	.372	.361	.370	.360

F.2 Full tables and training protocol

Protocol. We extract 384-dimensional features from the frozen backbones DINOv2 ViT-S/14 and DINOv3 ViT-S/16. The AE and SupCon both use a $[512, 256, 128] \to 10$ encoder, which is first trained for 100 epochs with a reconstruction loss. SupCon adds two contrastive training phases of 200 epochs each. The density-based supervision of SupCon uses the target numbers of clusters $4k$, $2k$, and k, where k is the true number of classes. The SupCon embedding therefore uses the number of classes, even when the extraction method that runs on this embedding chooses the number of clusters itself.

The check runs on the 10-dimensional embedding with μ=10 and k_{nn}=30, and the repair may use at most five rounds. Our selection criterion uses a minimum cluster size of 10. EOM runs in the hdbscan library with a minimum cluster size of 10 and minimum samples 5, unless a row states another value. We call these two values the fixed setting of EOM. k-means uses ten restarts.

Every AE and SupCon configuration has five seeds, which all come from one training run on an A100 GPU. All 60 learned embeddings are stored, and the other experiments on real data reuse them. We report means and standard deviations for configurations that involve randomness. Configurations on frozen features without a standard deviation use one fixed seed, also for NN-Descent and k-means. The experiments that cluster the frozen features directly are also single runs. A preliminary run of the study is not part of the tables. The preliminary run used three seeds for the DINOv3 and iNaturalist configurations, and only some of its embeddings were stored.

Table 13: NMI of three extraction methods on the features of the frozen backbones and on the two learned embeddings (mean±std over five seeds). A method marked with † is given the true number of classes k, which serves as the target of our selection criterion, whereas EOM chooses k itself. CERT means that, in every seed, the clustering at the selected threshold passes the check and the final clustering equals the one from the exact reference. Both statements hold for 60 of 60 embeddings (Section F). In the frozen rows, our selection criterion runs on the 384-dimensional features without the check. Embeddings: the frozen rows use the 384-dimensional backbone features (L2 normalized). The AE, which is trained with a reconstruction loss only, and SupCon, which is trained with a density-based contrastive loss, both map these features to 10 dimensions. The EOM column uses the fit routine of the hdbscan library, and its values depend on the tie handling of the library (Section 4.3). The other tables of Section F include STL-10 and ARI.

		k-means†	Ours†	EOM
CIFAR-100 (DINOv2)	frozen	.661 ±.002	.493	.001 (collapse, 38 min)
	AE	.659 ±.006	.624 ±.011 CERT	.400 ±.326
	SupCon	.677 ±.004	.657 ±.003 CERT	.648 ±.028
CIFAR-100 (DINOv3)	frozen	.695	.538	.640
	AE	.670 ±.003	.611 ±.008 CERT	.518 ±.258
	SupCon	.644 ±.007	.640 ±.006 CERT	.628 ±.020
iNat-500k (DINOv2)	frozen	.509	.418	.000 (collapse, 44 h)
	AE	.506 ±.001	.512 ±.003 CERT	.315 ±.257
	SupCon	.512 ±.001	.529 ±.005 CERT	.422 ±.211
iNat-500k (DINOv3)	frozen	.450	.356 (19.6 h)	—
	AE	.446 ±.001	.441 ±.002 CERT	.092 ±.183
	SupCon	.439 ±.001	.460 ±.003 CERT	.454 ±.009

Comparison with the exact reference and unassigned points. For all 60 learned embeddings, we compare our results with the exact Borůvka tree of the hdbscan library, using `certify_external_trees.py`. Every clustering at the selected threshold passes the check, two of them after one round of repair and the others without repair. The complete final clusterings, including the selected threshold and the labels of the unassigned points, equal the ones from the exact references. After the minimum cluster size is applied, the fraction of points that are unassigned and receive their label by 1-NN assignment is 62 to 89 percent on CIFAR-100, 79 to 93 percent on iNaturalist, and 1 to 49 percent on STL-10. On the frozen 384-dimensional features, our final clusterings are not checked.

In the preliminary run, three of five STL-10 AE seeds reached the limit of five rounds. None of them returned a false certificate, and the configuration had NMI $.768 \pm .043$. Training these three seeds again, on CPU and on a GPU of the H100 class that the preliminary run used, gives six runs that all pass the check without repair. The five seeds of this configuration in the tables also pass without repair. The three embeddings that reached the limit were not stored, so the cause is unknown. This observation is a further reason to check every run instead of relying on a fixed number of neighbors.

Scope of the quality comparison on iNaturalist. This paragraph and the analysis that follows support the iNaturalist results of the quality comparison in Section F.1. They state what the comparison shows, and they explain why ARI and NMI differ. Our selection criterion is given the true number of classes. On iNaturalist, whose classes are the taxonomic orders, our final clustering reaches an ARI of about .40 to .42 in selected AE and SupCon settings, the best EOM configurations reach .43 to .44, and k-means at $k = 273$ reaches about .10. These numbers describe the stated settings, and they do not show that two methods are statistically equivalent. On the one embedding that we analyzed, 84 percent of the points lie in clusters below the minimum cluster size and receive the label of their nearest assigned point. The certificate covers the clusters of these points at the selected threshold. The certificate does not show that the assigned labels are semantically correct. k-means scores higher on CIFAR-100 and STL-10. Compressing the features to a low-dimensional embedding makes the check affordable. However, the choice of the embedding and of the extraction method still has a large influence on the scores.

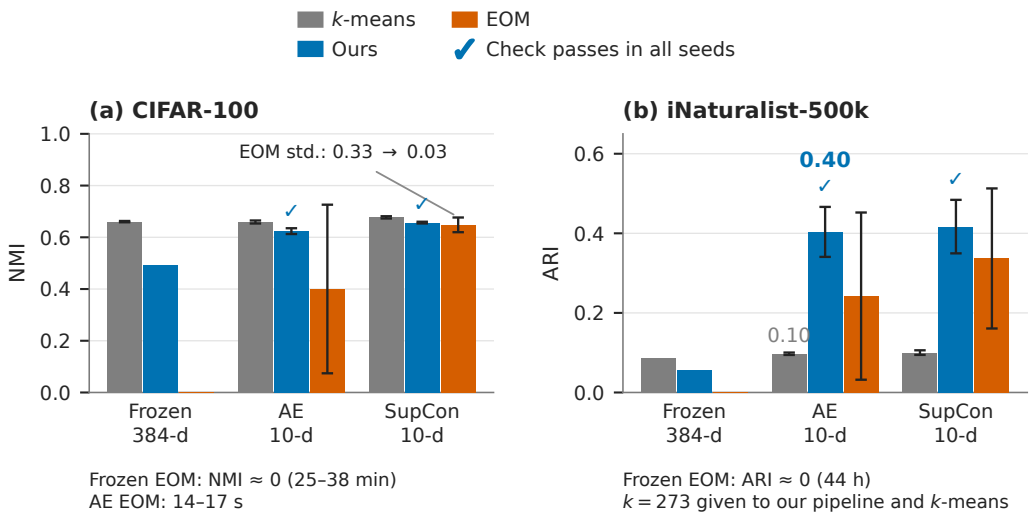

Figure 9: Three results of the factorial study on DINOv2 features. Bars are means over five seeds, error bars are standard deviations over the seeds, and a check mark means that the check passes in every seed, after at most one round of repair. Table 13 and the other tables of Section F give all results, including DINOv3 and STL-10. (a) CIFAR-100: on the frozen 384-dimensional features, EOM collapses at its fixed setting (NMI .001 after 25–38 min), whereas on the 10-dimensional embedding of the AE, which is trained with a reconstruction loss only, the extraction works again and takes seconds. With SupCon, fewer EOM runs collapse under this setting, which the drop of the standard deviation $\pm.33 \to \pm.03$ shows. How often EOM collapses also depends on minimum samples and on the order in which the library processes tied merges (Section 4.3). (b) iNaturalist-500k with the taxonomic orders as classes: given the true k, our final clustering reaches ARI .40 to .42 in its best configurations. The best EOM runs in which minimum samples matches the density parameter of our pipeline reach .43 to .44, and k-means reaches .10. For every learned embedding, the clustering at the selected threshold passes the check at $N{=}500$k.

Why ARI and NMI differ on iNaturalist. The 273 orders have very different sizes. The five largest orders contain 33 percent of the 500k images (71,500, 33,900, 26,900, 17,550, and 14,550 images), and the twenty largest contain 63 percent. The median order has 200 images, and 103 orders have at most 100.

On the iNaturalist DINOv2 AE embedding of seed 0 (`inat_metric_analysis.py`), k-means at $k{=}273$ needs 20 to 81 clusters to cover 90 percent of each of the ten largest orders. Its largest cluster for an order contains 7 to 41 percent of that order. Our final clustering needs 1 to 46 clusters and keeps 10 to 93 percent of an order in its largest cluster. ARI counts pairs of points, so splitting these large orders lowers ARI strongly: over all points, k-means scores .10 and our final clustering scores .45. Restricted to the twenty largest orders, the scores are .09 and .46. The scores that are based on entropy differ much less: NMI is .507 against .515, and the adjusted mutual information (AMI) is .500 against .509. Homogeneity is higher for k-means (.60 against .55), and completeness is higher for our final clustering (.48 against .44). Running k-means with 228 clusters, the number of clusters in our final clustering, gives ARI .11. Using the same number of clusters therefore does not remove the difference.

At the selected threshold, 84 percent of the points of this embedding lie in clusters below the minimum cluster size. These points are unassigned, and their final labels come from 1-NN assignment. The 16 percent of the points that lie in the kept clusters have ARI .89. The certificate covers the complete clustering at the selected threshold, including the small clusters. For comparison, the fraction of unassigned points is 8 percent on MNIST (LeCun et al., 1998), 3 percent on FMNIST (Xiao et al., 2017), and zero on the trained DENSIRED embedding.

Table 14: Full results of the factorial study: NMI and ARI of every extraction method for every dataset, backbone and embedding (mean±std over five seeds for the AE and SupCon configurations). $\tilde{k}$ is the median number of clusters that a method returns. The column "passes" gives the number of runs in which the check passes, after at most one round of repair (two runs needed one), out of the number of runs that were checked. Extraction methods: k-means (†) and our final clustering (rows "Ours (k)") are both given the true k; a dash marks a run that does not exist (EOM on the frozen DINOv3 features of iNaturalist) or a clustering that was not checked. EOM uses 1-NN assignment and runs at its fixed setting (`min_samples` 5; other values on the frozen features: Table 16) and, on the same embeddings, with minimum samples matched to μ (`min_samples` 10, rows "EOM (μ)"). On the frozen iNaturalist features, the EOM run took 44.4 hours in the 384-dimensional feature space and collapsed to two clusters. The same collapse occurs on the frozen CIFAR-100 features, and the iNaturalist run cost 70× more. On the AE and SupCon embeddings, EOM runs in seconds. Restricted to the seeds in which EOM does not collapse, the NMI of EOM on the AE embedding against the SupCon embedding is .666 against .648 (CIFAR-100, DINOv2), .647 against .628 (DINOv3), .525 against .528 (iNaturalist, DINOv2) and .457 against .454 (iNaturalist, DINOv3, where one AE seed does not collapse).

			DINOv2				DINOv3			
Dataset	Embedding	Extraction	NMI	ARI	$\tilde{k}$	passes	NMI	ARI	$\tilde{k}$	passes
STL-10	frozen	k-means†	.916 ±.020	.847 ±.064	10	—	.916	.918	10	—
		Ours (k)	.528	.153	10	—	.636	.317	10	—
		EOM	.853	.771	29	—	.789	.644	29	—
	AE	k-means†	.956 ±.006	.959 ±.008	10	—	.915 ±.009	.909 ±.015	10	—
		Ours (k)	.757 ±.041	.475 ±.075	10	5/5	.663 ±.041	.341 ±.038	10	5/5
		EOM	.810 ±.026	.692 ±.057	39	—	.699 ±.096	.485 ±.170	34	—
		EOM (μ)	.806 ±.031	.629 ±.110	27	—	.731 ±.114	.531 ±.207	20	—
	SupCon	k-means†	.913 ±.020	.837 ±.023	10	—	.851 ±.042	.730 ±.104	10	—
		Ours (k)	.681 ±.111	.359 ±.171	10	5/5	.727 ±.042	.427 ±.092	10	5/5
		EOM	.831 ±.040	.630 ±.126	16	—	.845 ±.006	.773 ±.015	25	—
		EOM (μ)	.813 ±.067	.599 ±.150	15	—	.840 ±.006	.754 ±.030	23	—
CIFAR-100	frozen	k-means†	.661 ±.002	.384 ±.001	100	—	.695	.432	100	—
		Ours (k)	.493	.044	49	—	.538	.123	49	—
		EOM	.001	.000	2	—	.640	.231	126	—
	AE	k-means†	.659 ±.006	.391 ±.012	100	—	.670 ±.003	.401 ±.005	100	—
		Ours (k)	.624 ±.011	.271 ±.015	75	5/5	.611 ±.008	.247 ±.020	71	5/5
		EOM	.400 ±.326	.201 ±.164	153	—	.518 ±.258	.196 ±.105	124	—
		EOM (μ)	.656 ±.008	.297 ±.036	106	—	.260 ±.317	.105 ±.132	2	—
	SupCon	k-means†	.677 ±.004	.420 ±.014	100	—	.644 ±.007	.343 ±.013	100	—
		Ours (k)	.657 ±.003	.334 ±.007	80	5/5	.640 ±.006	.257 ±.023	52	5/5
		EOM	.648 ±.028	.262 ±.095	82	—	.628 ±.020	.216 ±.071	64	—
		EOM (μ)	.651 ±.007	.261 ±.035	71	—	.638 ±.012	.236 ±.057	52	—
iNat-500k	frozen	k-means†	.509	.086	273	—	.450	.068	273	—
		Ours (k)	.418	.057	243	—	.356	.066	150	—
		EOM	.000	.000	2	—	—	—	—	—
	AE	k-means†	.506 ±.001	.097 ±.003	273	—	.446 ±.001	.075 ±.001	273	—
		Ours (k)	.512 ±.003	.404 ±.063	216	5/5	.441 ±.002	.397 ±.012	133	5/5
		EOM	.315 ±.257	.242 ±.210	881	—	.092 ±.183	.070 ±.140	3	—
		EOM (μ)	.524 ±.012	.427 ±.065	593	—	.000 ±.001	.000 ±.000	2	—
	SupCon	k-means†	.512 ±.001	.100 ±.006	273	—	.439 ±.001	.091 ±.005	273	—
		Ours (k)	.529 ±.005	.417 ±.067	222	5/5	.460 ±.003	.335 ±.045	75	5/5
		EOM	.422 ±.211	.337 ±.176	723	—	.454 ±.009	.336 ±.086	231	—
		EOM (μ)	.432 ±.211	.355 ±.194	437	—	.448 ±.012	.265 ±.070	100	—

G FRAGMENTED ANN GRAPHS: BRIDGE EDGES OR MORE NEIGHBORS

This section supports the construction of the approximate tree in Section 3, in which a disconnected ANN graph receives bridge edges. The data are 10^6 points of the synthetic DENSIRED generator

Table 15: NMI on CIFAR-100 when the AE is replaced by a compression method that is not a trained encoder (PCA, random projection, UMAP (McInnes et al., 2018)). Values are means over three seeds where the extraction involves randomness. PCA and the projections are deterministic. The UMAP-10 values are means over five seeds with the default settings of `umap-learn` and have a standard deviation of at most .003 NMI. The AE rows are repeated from Table 13 for comparison. No linear compression to 10 dimensions makes the density-based extraction work: EOM stays collapsed (median $k \leq 4$), and every extraction method scores below its value on the AE embedding. On DINOv3 with PCA-10, the repair reaches its limit of five rounds in all three seeds, so these clusterings of our pipeline are not certified. In contrast, UMAP-10, which is nonlinear and not a neural network, scores higher than the AE embedding with every extraction method. Its final clustering passes the check without repair in every seed, and EOM never collapses on it at `min_samples` 5, 10 or 25. On iNat-500k (three seeds), our final clustering on UMAP-10 reaches NMI .418 and ARI .106, against .512 and .404 on the AE embedding (`run_hygiene.py umap`).

Backbone	Compression	k-means†	Ours (k)	EOM
DINOv2	PCA-10	.483	.400	.066
DINOv2	PCA-10 whitened	.489	.388	.059
DINOv2	random projection 10	.244	.073	.004
DINOv2	UMAP-10	.709	.681	.696
DINOv2	AE-10 (learned)	.659	.624	.400
DINOv3	PCA-10	.489	.294	.002
DINOv3	PCA-10 whitened	.481	.276	.004
DINOv3	random projection 10	.217	.027	.014
DINOv3	UMAP-10	.741	.721	.728
DINOv3	AE-10 (learned)	.670	.611	.518

Table 16: EOM and leaf selection of the hdbscan library on the frozen CIFAR-100 features for four values of minimum samples (`run_hygiene.py frozen-sweep`). The features are L2-normalized and 384-dimensional. All runs use minimum cluster size 10 and 1-NN assignment of the noise points. Each entry gives the NMI and, in parentheses, the number of clusters k that the library returns. Bold entries are collapsed runs with at most four clusters. Each run takes 39 to 40 minutes and labels 16 to 92 percent of the points as noise before the assignment. The EOM results on frozen features in the other tables use minimum samples 5. On DINOv3, the result differs by one cluster between two machines (.641 and 127 here, .640 and 126 in Table 14). We did not trace the cause. At 384 dimensions the library builds its tree with the exact Prim algorithm, so these EOM and leaf results are computed on exact trees.

`min_samples`	DINOv2		DINOv3	
	EOM	leaf	EOM	leaf
5	**.001** (2)	.628 (150)	.641 (127)	.646 (181)
11	**.003** (2)	.622 (100)	.624 (87)	.632 (111)
25	.577 (57)	.604 (67)	**.002** (2)	.624 (71)
50	**.039** (4)	.552 (52)	.579 (46)	.596 (55)
AE embedding, fixed setting	.400 (mean), .664 (median)		.518 (mean), .642 (median)	

in 10 dimensions, mapped to 10 dimensions by an autoencoder with hidden layers of 500, 500 and 2,000 units that is trained on these points; we call its output the trained embedding of this section. We say that an ANN graph is *fragmented* if it is disconnected, that is, if it consists of at least two connected components. For this embedding, we report how the number of components depends on the number of neighbors k_{nn}, and we compare the two ways of handling a fragmented graph: bridge edges followed by the check, and a larger number of neighbors.

With this encoder, the ANN graph of the trained embedding with 10^6 points is disconnected for a fixed number of neighbors (Figure 10): before bridge edges are added, the number of connected components falls $15{,}548 \rightarrow 253 \rightarrow 10$ as k_{nn} grows $10 \rightarrow 30 \rightarrow 50$. The resulting error is real, but it is an error of the cluster extraction: HDBSCAN's excess-of-mass (EOM) criterion on the disconnected graph returns 260 clusters (NMI .62), whereas k-means on the same embedding

Table 17: NMI of EOM for three values of minimum samples on the same embeddings (`run_cell.py -eom-min-samples`). The embeddings were trained again for this test: CIFAR-100 with seeds 0 to 4 and iNat-500k with seeds 0, 3, 4. EOM runs at our fixed setting `min_samples` $= 5$, at 10 (which matches μ) and at 25. Bold entries are collapsed runs with at most four clusters. On the AE embeddings, the seeds that collapse change with minimum samples, and a larger value can both remove and cause a collapse. On the SupCon embeddings, EOM does not collapse at any tested setting.

Backbone	Encoder	`min_samples`	seed 0	seed 1	seed 2	seed 3	seed 4
DINOv2	AE	5	.667	.664	.668	**.001**	**.001**
DINOv2	AE	10	.656	.641	.664	.663	.654
DINOv2	AE	25	.638	.650	.654	**.002**	**.001**
DINOv3	AE	5	.647	.662	**.003**	.638	.642
DINOv3	AE	10	.661	**.001**	**.002**	**.001**	.637
DINOv3	AE	25	.570	.633	**.002**	.592	**.017**
DINOv3	SupCon	5	—	—	—	.645	.635
DINOv3	SupCon	10	—	—	—	.646	.637
DINOv3	SupCon	25	—	—	—	.649	.634
iNat-500k, DINOv2	AE	5	**.000**	—	—	.522	.526
iNat-500k, DINOv2	AE	10	.530	—	—	.500	.534
iNat-500k, DINOv2	AE	25	**.000**	—	—	.489	.530
iNat-500k, DINOv2	SupCon	5	—	—	—	.511	.533
iNat-500k, DINOv2	SupCon	10	—	—	—	.531	.533
iNat-500k, DINOv2	SupCon	25	—	—	—	.523	.544
iNat-500k, DINOv3	AE	5	—	—	—	.457	**.000**
iNat-500k, DINOv3	AE	10	—	—	—	**.000**	**.000**

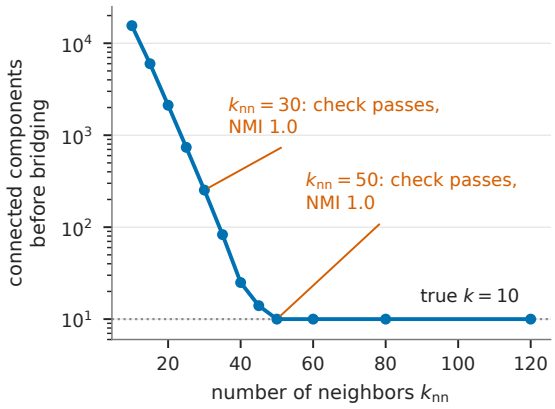

Figure 10: Fragmentation on the trained embedding with 10^6 points. The curve shows the number of connected components of the ANN graph before bridge edges are added, for a fixed number of neighbors. The annotations give the result of the check for our final clustering at two numbers of neighbors. Our pipeline returns the exact clustering even at k_{nn}=30, where the ANN graph consists of 253 connected components.

scores 1.0. Our pipeline returns the exact result in this tested case with the same fixed number of neighbors. Exact core distances, bridge edges and the check give the exact clustering (NMI 1.0, PASS, zero missing edges) at k_{nn}=30, where the ANN graph consists of 253 connected components. Self-calibration of the number of neighbors (Section A.2) is therefore a way to tune the cost, and correctness does not depend on it. Whether fragmentation occurs also depends on the encoder: with a smaller encoder (two hidden layers of 64 units, output dimension 8), the ANN graph is never disconnected at 10^6 points. The number of neighbors alone therefore does not determine whether the final clustering is exact.

Table 18: EOM results of the fit routine of the hdbscan library in its approximate and in its exact mode on the same 60 learned embeddings (`run_cell.py -load-latent -eom-exact`). Minimum samples is 5 or 10, and the second value matches μ. The columns under "exact tree" use `approx_min_span_tree=False`. This exact mode does not by itself certify the hierarchy under our floating-point convention. All runs use the tie handling of the library, so the comparison does not separate the effect of the approximation from the effect of the order of tied merges (Section 4.3). We report mean $\pm sd$ over five seeds, the number of collapsed seeds (at most four clusters), and the number of seeds in which both trees give the same number of clusters. The number of clusters differs in 83 of 120 pairs of runs. The exact mode can both add and remove collapsed seeds. For CIFAR-100 DINOv3 AE at 5, one seed collapses on the approximate tree and three seeds collapse on the exact tree. For iNaturalist DINOv2 SupCon at 10, one seed collapses on the approximate tree and none on the exact tree. Over CIFAR-100 and iNaturalist together, EOM on the exact tree collapses in $12/20$ AE runs at 5 and in $7/20$ at 10. For SupCon, the counts are $1/20$ and $2/20$.

			approximate tree		exact tree			
Dataset (backbone)	Encoder	`min_samples`	NMI	collapsed	NMI	ARI	collapsed	same k
STL-10 (v2)	AE	5	.810 ±.026	0/5	.807 ±.014	.677 ±.047	0/5	3/5
STL-10 (v2)	AE	10	.806 ±.031	0/5	.761 ±.116	.566 ±.220	0/5	0/5
STL-10 (v2)	SupCon	5	.831 ±.040	0/5	.830 ±.040	.629 ±.126	0/5	3/5
STL-10 (v2)	SupCon	10	.813 ±.067	0/5	.819 ±.066	.608 ±.148	0/5	3/5
STL-10 (v3)	AE	5	.699 ±.096	0/5	.699 ±.096	.485 ±.170	0/5	3/5
STL-10 (v3)	AE	10	.731 ±.114	0/5	.705 ±.092	.457 ±.172	0/5	3/5
STL-10 (v3)	SupCon	5	.845 ±.006	0/5	.843 ±.011	.762 ±.033	0/5	4/5
STL-10 (v3)	SupCon	10	.840 ±.006	0/5	.838 ±.008	.745 ±.038	0/5	4/5
CIFAR-100 (v2)	AE	5	.400 ±.326	2/5	.401 ±.326	.202 ±.166	2/5	1/5
CIFAR-100 (v2)	AE	10	.656 ±.008	0/5	.662 ±.002	.325 ±.012	0/5	1/5
CIFAR-100 (v2)	SupCon	5	.648 ±.028	0/5	.663 ±.009	.308 ±.042	0/5	0/5
CIFAR-100 (v2)	SupCon	10	.651 ±.007	0/5	.655 ±.010	.285 ±.038	0/5	0/5
CIFAR-100 (v3)	AE	5	.518 ±.258	1/5	.262 ±.319	.104 ±.133	3/5	0/5
CIFAR-100 (v3)	AE	10	.260 ±.317	3/5	.249 ±.304	.082 ±.117	3/5	3/5
CIFAR-100 (v3)	SupCon	5	.628 ±.020	0/5	.639 ±.012	.256 ±.037	0/5	0/5
CIFAR-100 (v3)	SupCon	10	.638 ±.012	0/5	.642 ±.012	.260 ±.037	0/5	0/5
iNat-500k (v2)	AE	5	.315 ±.257	2/5	.312 ±.255	.190 ±.176	2/5	2/5
iNat-500k (v2)	AE	10	.524 ±.012	0/5	.528 ±.005	.435 ±.088	0/5	0/5
iNat-500k (v2)	SupCon	5	.422 ±.211	1/5	.425 ±.213	.354 ±.193	1/5	1/5
iNat-500k (v2)	SupCon	10	.432 ±.211	1/5	.530 ±.011	.440 ±.094	0/5	0/5
iNat-500k (v3)	AE	5	.092 ±.183	4/5	.000 ±.000	.000 ±.000	5/5	2/5
iNat-500k (v3)	AE	10	.000 ±.001	4/5	.001 ±.001	.000 ±.000	4/5	4/5
iNat-500k (v3)	SupCon	5	.454 ±.009	0/5	.447 ±.006	.356 ±.047	0/5	0/5
iNat-500k (v3)	SupCon	10	.448 ±.012	0/5	.274 ±.223	.205 ±.170	2/5	0/5

H BENCHMARKS AND ADDITIONAL TABLES

This section contains the tables and figures that support the experiments of the main text on clustering quality, on the cost of the check and of the repair, and on ties. It compares the guarantees of scalable methods (Table 20) and reports the check on trees built by other libraries, the effect of the failed checks on the final clustering, an experiment with deliberately damaged neighbor graphs, cases in which an approximate tree deviates from the exact one deep in the hierarchy (Table 26), the scaling of our pipeline with the number of points and with the dimension, and the comparison of our trained pipeline with SHADE and with further clustering methods.

Ties occur in real data. The tree of the USPS embedding (Hull, 1994), which we use in the comparison with the methods of Table 21, contains three ties among its 25 largest edge weights and 182 ties in total. An assumption that all edge weights are distinct would therefore fail on this embedding. For this reason, the hierarchy check treats edges of equal weight together and does not count equal weights as an error (Section 3.1).

The scores for one dataset differ between the tables of this section, because the tables use different protocols. Every caption names its protocol. Table 31 uses the default protocol of our implementation with seed 0. Tables 21 and 25 use the embeddings of seed 0 that we exported for the comparison with ELKI. The text of this section reports means over five seeds with the full training schedule.

Table 19: Results of the check on trees built by other libraries (`certify_external_trees.py`). The runs use the unoptimized implementation of the check, not the optimized one of Table 4. For every tree, we recompute the $N-1$ edge weights in float64 from exact core distances, apply our selection criterion with the given k and the minimum cluster size m, and then check and repair the resulting clustering. Each cell gives the result of the first check and the time to build the tree. A cell "fail v, r rounds" means that the first check failed with v missing edges and that r rounds of repair gave the exact clustering. The column "identical" states whether the final clusterings from all trees of the row, including the exact references, are equal. The exact references are the exact Borůvka mode of the hdbscan library and, up to 10^5 points, an independent implementation of Prim's algorithm on the complete graph that needs $O(N)$ memory ("dense Prim"). The `fast_hdbscan` library computes its weights in float32. Recomputing them changes them by at most 4.2×10^{-6} in relative terms and changes none of the tested clusterings.

dataset (N, d, k, μ, m)	Ours	`fast_hdbscan`	hdbscan default	hdbscan exact	dense Prim	identical
DENSIRED (10^5, 10, 10, 5, 1)	pass, 26 s	pass, 1 s	pass, 13 s	pass, 13 s	pass, 641 s	yes
DENSIRED (10^6, 10, 10, 5, 1)	pass, 58 s	pass, 35 s	pass, 753 s	pass, 921 s	—	yes
trained embedding (10^6, 8, 10, 5, 10)	pass, 67 s	pass, 4 s	pass, 116 s	pass, 130 s	—	yes
iNat-500k AE embedding (5×10^5, 10, 273, 10, 10)	pass, 36 s	pass, 6 s	fail 48, 2 rounds, 190 s	pass, 317 s	—	yes
MNIST embedding (7×10^4, 10, 10, 5, 10)	pass, 23 s	pass, 0.5 s	fail 4, 1 round, 7 s	pass, 5 s	pass, 172 s	yes
FMNIST embedding (7×10^4, 10, 10, 5, 10)	pass, 23 s	pass, 0.4 s	fail 6, 1 round, 8 s	pass, 8 s	pass, 164 s	yes

Table 20: Guarantees of scalable methods for density-based clustering. "Sparse MR MST": the method computes the MST of a sparse graph with MR weights. "Check" and "hierarchy check": the method proves for the individual run that the clustering at one threshold, or the clusterings and merge distances of the upper part of the hierarchy, equal those of the exact computation on the complete graph. "Repair and detection of ANN errors": the method detects at run time that the approximate neighbor search changed the result and then adds the missing edges until it reaches the exact clustering. "A-priori guarantee": a statistical or approximation guarantee that holds before the method runs. For our pipeline, this guarantee assumes a neighbor graph that contains all exact neighbors (Section A.3). Exact algorithms (Wang et al., 2021; McInnes, 2026) are correct by design, and the table separates this guarantee from an additional check of the individual run. A passed hierarchy check covers only the thresholds that were checked, and a search that reaches its work limit remains inconclusive. The rows group the methods that the main text cites.

	sparse MR MST	check	hierarchy check	repair & detection of ANN errors	a-priori guarantee
hdbscan library default	—	—	—	—	—
FISHDBC, cuML HDBSCAN	✓	—	—	—	—
sngDBSCAN	—	—	—	—	✓
ANN hierarchical-clustering bound	✓	—	—	—	✓
graph-index HDBSCAN* methods	✓	—	—	—	—
Ours	✓	✓	✓	✓	✓

We compare our trained pipeline with SHADE, which stores the dense distance matrix, on MNIST (LeCun et al., 1998), Fashion-MNIST (Xiao et al., 2017), KMNIST (Clanuwat et al., 2018), USPS (Hull, 1994), Optdigits and Pendigits (Kelly et al., 2023), and COIL-20 and COIL-100 (Nene et al., 1996b;a). Both methods use the same encoder architecture on each dataset (Table 24). Our pipeline is better on MNIST (NMI .840 against .530), FMNIST, COIL-20 and COIL-100. On the two COIL datasets it is also the best of all compared methods, which include HDBSCAN on a UMAP embedding (McInnes et al., 2018), DipDECK (Leiber et al., 2021), DEC (Xie et al., 2016) and IDEC (Guo et al., 2017) (Table 31). The two methods are equal on Pendigits. Our pipeline is worse where the sparse graph estimates the density less reliably, namely on data with few dimensions or small N, and on KMNIST, where the embedding limits the quality. Table 23 compares the peak memory. SHADE needs $58\times$ the memory of our pipeline at 10^5 points, and beyond this size its distance matrix cannot be allocated. While we developed the check, it exposed a bug in the threshold search of our pipeline: the check passed for a clustering with one cluster per point, which the search returned when no threshold reached the requested number of clusters. The fix, which selects the closest

number that occurs, raises the ARI on USPS from .000 to .627, now for a certified clustering, and restores the contrastive supervision that had degenerated without any warning.

Table 21: Exact HDBSCAN* compared with our trained pipeline. The exact method is ELKI's `HDBSCANLinearMemory` with cluster extraction by excess of mass (EOM) and `minPts` $= 5$. It is the method that the graph-index methods (Okkels et al., 2025; Schubert, 2025) accelerate. We run it on the original features ("raw") and on the embedding of our pipeline ("emb"), and we compare it with the cluster extraction of the pipeline itself. Scores are NMI/ARI on the embeddings of seed 0 that we exported for this comparison. The number of clusters k is given in parentheses where the extraction returns many more clusters than classes. The column "Ours" reports the default cluster extraction of the pipeline, whereas the .815 in the text of Section H is the score of our final clustering, which we computed on the exported USPS points and certified.

Dataset	[HDB*]+raw	[HDB*]+emb	Ours
USPS	.680/.587	.772/.613	.811/.777
Synth-low	.541/.069 (k=177)	.865/.676	.904/.855
Synth-high	.527/.055 (k=199)	.829/.668	.822/.691
MNIST	—	.708/.459	**.840/.711**
FMNIST	—	.503/.207	.546/.253
COIL-100	—	.929/.725	.945/.784
COIL-20	.888/.702	.889/.743	.907/.814
Optdigits	.847/.769	.783/.635	.767/.577

Table 22: Time and memory of our pipeline, including the check, on DENSIRED (d=10, k=10, CPU). In every run, the check returns PASS, the NMI $= 1.0$, and the number of clusters k is recovered. The column "check" gives the time of the search for missing edges alone. The total time is the sum of the four timed stages (NN-Descent including its JIT compilation, exact core distances, graph with MST and cut, check); the generation of the data is not included. Rows give the median over three seeds at 10^6 points, over two seeds at 10^5 points and one seed above. Peak RSS is the peak resident memory of the process. The column "dense matrix" gives the size of the distance matrix ($N^2 \cdot 8$ bytes) that SHADE and related dense methods would have to store. This size is computed, not measured. Figure 11 shows the times of the libraries that we actually ran.

N	peak RSS	total time	check	dense matrix
10^5	1.5 GB	16 s	0.05 s	80 GB
10^6	4.1 GB	58 s	0.47 s	8 TB
2×10^6	6.5 GB	2.0 min	1.0 s	32 TB
4×10^6	11.7 GB	4.1 min	2.3 s	128 TB
8×10^6	21.1 GB	8.6 min	5.1 s	512 TB

Table 23: Peak memory on DENSIRED for complete runs of our trained pipeline, including the training of the encoder. The memory of our pipeline is measured at every N. Table 22 measures only the clustering and the check, which explains its lower values at the same N. The memory of SHADE is measured up to 100k points. For larger inputs, the entry is the size of its distance matrix, $N^2 \cdot 8$ bytes, which we computed exactly and marked with †.

N	Ours (GiB)	SHADE (GiB)	ratio
50k	1.9	32.8	17×
100k	2.2	128	58×
200k	2.7	298†	110×
500k	3.9	1863†	478×
1M	6.0	7451†	1242×

We repeat this ablation on the USPS embedding, that is, on the same points that ELKI clusters in Table 21. All three stages of Table 25 already give the clustering of the exact reference for both numbers of neighbors, and the check finds no missing edge. On these points, the EOM extraction of

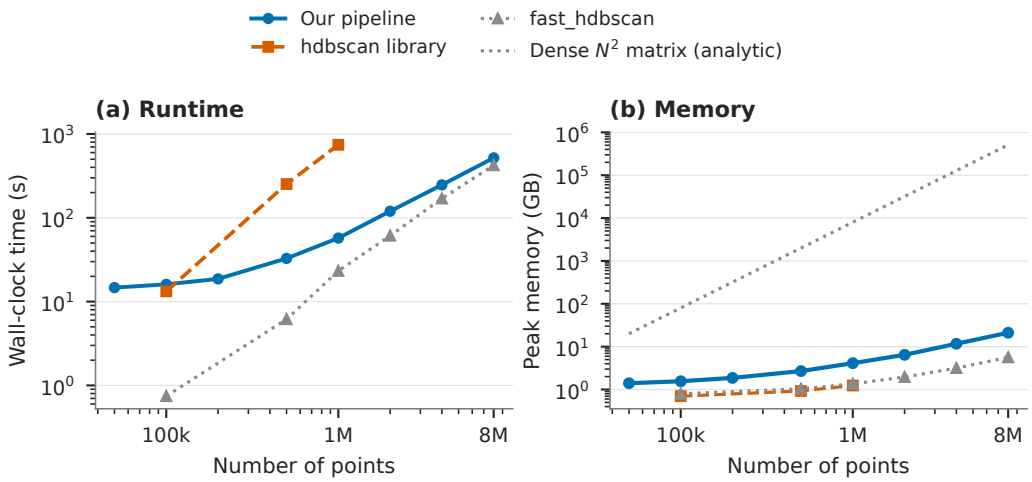

Figure 11: Scaling of our pipeline on DENSIRED (d=10). (a) Time of our pipeline (sum of its timed stages, including the check; last seed per size, Table 22), compared with the hdbscan library in its default setting and with `fast_hdbscan`, an exact parallel Borůvka method that computes in float32 and does not check its result. All three reach the same clustering quality at every size. The time of the hdbscan library grows faster than linearly. `fast_hdbscan` is faster than our pipeline by 2.5× at 10^6 points and by 1.2× at 8×10^6 points. (b) Measured peak memory of our pipeline compared with the size of the dense N^2 distance matrix that SHADE requires.

Table 24: Comparison of our trained pipeline with SHADE on the larger real datasets (NMI and ARI, all N points, means over five seeds). Both methods use the same encoder architecture. Our pipeline uses its own density-based cluster extraction and assigns the remaining points to their nearest assigned point (1-NN). These scores therefore belong to a different experiment than the comparison of final clusterings in Table 11.

	NMI		ARI	
Dataset (N)	Ours	SHADE	Ours	SHADE
MNIST (70k)	**.840**	.530	**.711**	.401
FMNIST (70k)	**.546**	.454	**.253**	.234
Pendigits (11k)	**.825**	.823	.723	**.751**
KMNIST (70k)	.058	**.396**	.013	**.190**

ELKI reaches an NMI of .772. Our final clustering reaches .815 on the same points and is proven to be the exact clustering. When both methods use the same cluster extraction, the scores are .772 for ELKI against .774 for our pipeline, in the comparison with the methods of Table 21.

Every repair that ended with a pass returned the clustering of the exact reference. The repairs that reached the limit on the number of rounds are those for which the neighbor graph had thousands of connected components before bridging. These are all graphs with k_{nn}=2, and the graphs with $k_{\text{nn}} \leq 5$ and at least 60 percent of the edges deleted, which have 9,000 to 45,000 connected components at N=50k. Eight of these repairs had already reached the clustering of the exact reference, and the check still failed for the reason given in the caption when the repair reached the limit. With $k_{\text{nn}} \geq 10$ and up to 60 percent of the edges deleted, the repair ends with a pass within nine rounds.

The repository contains further benchmarks of our implementation. The first is the quality comparison of Table 31. Our pipeline is best on COIL-20 and COIL-100, SHADE is best on the two synthetic datasets, and HDBSCAN on a generic UMAP embedding is best on the two simple digit datasets with k=10. The second benchmark covers six datasets with few dimensions or small N, on which SHADE is better. Our pipeline is not designed for such data. The third benchmark measures which fraction of the edges within distance ε NN-Descent finds. This recall is $\geq$ .985 at k_{nn}=6

Table 25: Ablation of the stages of our pipeline, before and after the repair, on one fixed Pendigits embedding (seed 0). Each stage adds one mechanism to the MST of the sparse MR graph, which is the construction that our pipeline shares with the graph-index methods. "=exact" states whether the clustering of the stage equals the clustering of the exact reference on the complete graph, when both use the same selection criterion and the same assignment of unassigned points. "Components" counts the connected components of the graph of the stage. That the check fails only for the larger number of neighbors is specific to this embedding. When we retrain the encoder with the same configuration, the check finds more than 10^3 missing edges for both numbers of neighbors, and the repair reaches the clustering of the exact reference within two rounds.

k_{nn}	stage	components	check	NMI	ARI	=exact
10	kNN graph only	7	—	.663	.385	no
10	+bridging	1	PASS	.698	.405	yes
10	+repair	1	PASS (0 rounds)	.698	.405	yes
30	kNN graph only	5	—	.664	.386	no
30	+bridging	1	FAIL (358 missing edges)	.664	.386	no
30	+repair	1	PASS (1 round, 358 edges)	.698	.405	yes

Table 26: Three cases from the comparison of the check with exact references on the complete graph for DENSIRED data with up to 2,500 points (108 configurations, `verify_certificate.py`), evaluated in float64, in which the approximate tree deviates from the exact tree deep in the hierarchy. In each case, the clustering at the selected threshold equals the exact clustering and passes the check. At lower thresholds, however, the distances at which clusters merge (merge heights) deviate from the exact ones for NN-Descent graphs with many neighbors. The hierarchy check, carried out here by repeating the check with strict inequality threshold by threshold from the top of the hierarchy, fails before it reaches these merges. "Last pass" gives the largest number of clusters up to which every check passes. "Probes" gives the number of thresholds, out of ten, at which the approximate and the exact tree give the same clustering. A clustering at a threshold does not depend on how ties are broken. "False pass" states whether a check passed for thresholds at which the two clusterings differ.

Dataset	seed	k_{nn}	max. rel. height deviation	probes	last pass	false pass
DENSIRED-1200	0	50	.140	9/10	4	no
DENSIRED-1200	1	100	.060	10/10	5	no
DENSIRED-2500	0	100	.102	10/10	4	no

and 1.0 at the number of neighbors that our pipeline uses by default. The fourth benchmark is a comparison with the exact tree for $N \leq 2500$, whose question the experiments with the check in Section 4.1 also answer. Finally, we report our final clusterings on real embeddings with the full training schedule and five seeds. Every one of them passes the check. The scores are USPS .808 $\pm$.031 NMI / .725 $\pm$.061 ARI; Optdigits .830 $\pm$.020 / .743 $\pm$.033; COIL-20 .915 $\pm$.005 / .794 $\pm$.007 (three seeds); Pendigits .668 $\pm$.048 / .360 $\pm$.098. On Pendigits, the EOM extraction remains better, and we report its score together with the score of our final clustering.

I Reproducibility index

All scripts are in the directory `reproduce/` of the code repository. The lists below give the scripts and the stored records of every experiment, and `reproduce/figs/` regenerates the figures and the table rows of the paper from these records.

Hardware and software. The CPU times of the synthetic scaling experiments (Tables 22 and 29 and Figure 11) are measured on one node with two Intel Xeon Gold 6326 processors (64 hardware threads, 503 GiB RAM) without other jobs; NN-Descent and the kd-tree queries use all threads, `fast_hdbscan` is timed after a JIT warm-up on 2,000 points, and the manifest marks the measurement at dimension 100 as taken while other jobs were running. All other CPU times (Tables 4 and 30, Sections E.1 and E.2) come from `reproduce/slurm_timing.py`, run as Slurm jobs on one

Table 27: What the failed checks of Table 19 change (`alarm_severity.py`). The table considers the approximate tree that the hdbscan library builds in its default setting on the three real embeddings, before and after the repair. "Edges" gives the number of missing edges that the first check reports. Common threshold: at the threshold t_0 that is selected on the unrepaired tree, we report the number of exact clusters that the unrepaired tree splits, the points inside these clusters, and the points outside the largest part of each split cluster. In every case, each cluster of the unrepaired clustering lies inside one exact cluster. Final clustering: we compare the final clustering from the unrepaired tree, which consists of the kept clusters and the unassigned points assigned to their nearest assigned point (1-NN), with the final clustering after the repair. The latter equals the final clustering from the exact tree in every case. We report the numbers of kept clusters, the fraction of labels that change after optimal matching, the ARI between the two labelings, and the ARI with respect to the true classes before and after the repair. Cost: the time to build the tree, to compute the exact core distances, and to check and repair the clustering, taken from the runs of Table 19, which ran without other jobs on the node.

		common threshold t_0			final clustering, before vs after				end-to-end cost (s)		
Embedding	edges	split	in split	displaced	kept	changed	ARI	truth ARI	default+rep.	exact+check	`fast_hdbscan`+check
MNIST (7×10^4, k=10)	4	1	621	75	10 / 10	.02%	.9997	.5514 / .5514	7.2	5.1	0.8
FMNIST (7×10^4, k=10)	6	1	20,465	166	10 / 10	.03%	.9992	.2207 / .2207	8.8	8.6	0.8
iNat-500k AE (5×10^5, k=273)	48	12	10,907	2,825	239 / 228	2.7%	.985	.4452 / .4450	366	381	74

Table 28: Check and repair on deliberately damaged neighbor graphs when the threshold is selected again after every round (`adversarial_audit.py`). On DENSIRED (d=10, k=10, μ=5), we damage the neighbor graph on purpose. We use $k_{\mathrm{nn}} \in \{2, 3, 4, 5, 6, 10, 30\}$, stop NN-Descent after 1, 2 or 10 iterations, delete 0 to 90 percent of the edges of the graph at random, and join the connected components of the graph by random pairs of points instead of bridge edges chosen among sampled points near the component boundaries. The random pairs are still edges of G with exact weights, so the assumption of Theorem 1 holds, but nothing directs a bridge edge toward the light edge of the exact tree that the graph lacks. We compare every clustering with the clustering of an exact reference, an implementation of Prim's algorithm on the complete graph that needs $O(N)$ memory and shares no code with our pipeline. "False pass": the check passes although the clustering differs from the reference. "Fail, different": the check fails and the clustering differs from the reference. "Fail, equal": the check fails although the clustering already equals the reference, because a random heavy bridge edge raises the selected threshold ε, and the check finds missing edges at this higher threshold. "Repaired": the repair ends with a pass. "Stopped": the repair reaches the limit on the number of rounds.

N	graphs	false pass	fail, different	fail, equal	repaired	stopped (20 rounds)
20,000	252	0	178	74	164	88
50,000	48	0	32	16	23	25

cluster node under the protocol of Section E.1; its 172 records, with the load average, the node and the CPU affinity of every measurement, are stored under `results/slurm_timing_699943/`. The experiment in higher dimensionalities (Section E.4) ran on six cores of the Xeon Gold 6326 node under the same protocol, and the ImageNet-1k timings (Section E.3) on an exclusively allocated node with two Intel Xeon Platinum 8480C processors. Software: Python 3.12.8 (3.12.12 on the cluster nodes), NumPy 2.5.2, SciPy 1.18.0, scikit-learn 1.9.0, pynndescent 0.6.0, hnswlib 0.8.0, hdbscan 0.8.44, `fast_hdbscan` 0.3.2, ELKI 0.8.0 and PyTorch 2.5.1. The encoders are trained on one A100, H100 or H200; all experiments start from the stored embeddings, because training on a GPU need not be bitwise deterministic.

Scripts and records by experiment.

- Check and hierarchy check on the 336 approximate trees, ties, margin and projection filter (Sections C.4, 4.1 and 4.3): `extended/latent_suite.py`, `repair_suite.py`, `projection_suite.py`, `timing_suite.py`, `validation.py`, `report.py`; records under `results/extended_20260914/` with `REPORT.md`, the comparison

Table 29: Cost of the stages of our pipeline for different dimensions of the data at $N{=}10^6$ (DENSIRED, $k_{\mathrm{nn}}{=}30$, $\mu{=}5$, bridge edges chosen among sampled points near the component boundaries, CPU, seconds; `certificate_scaling.py`). "Total" is the sum of the four stages, each the median over the available runs; at $d{=}100$, the row uses $k_{\mathrm{nn}}{=}150$ and was measured while other jobs were running. The check itself stays cheap up to $d{=}64$. The stage that grows is the query for the exact core distances with a scipy kd-tree. `fast_hdbscan`, timed on the same data, builds the exact tree only for $d \leq 30$ (23 and 227 s at $d{=}10$ and 20); for $d > 30$ it switches to an approximate tree from an NN-Descent graph, so its times at $d \geq 32$ are not those of an exact method. The check saves work in two ways. Only clusters that contain core points enter the check (564 of the $421{,}899$ clusters on the iNaturalist embedding), and the check uses the bounding boxes to exclude all but $1{,}361$ of their $158{,}766$ pairs. On DENSIRED, the check examines 1 of 45 pairs of clusters.

d	NN-Descent	exact core dist.	graph, MST, cut	check	total	`fast_hdbscan`
10	34	13	12	0.5	60	23
20	42	253	13	0.6	308	227
32	46	671	14	1.0	732	31
64	51	1638	17	2.3	1708	35
100	185	3825	62	178	4250	34

Table 30: Cost of the check for different given numbers of clusters (`regime_sweep.py`; protocol of Section E.1, medians over three runs). Rows with the same selected threshold are merged. We report the number $q(t)$ of all clusters at the selected threshold, the number of core points, the number of pairs of core points within the threshold distance ("radius edges"), the pairs of clusters that the check searches out of all pairs of clusters with core points, and the time of the check. The direct method enumerates the radius edges with `cKDTree.query_pairs` and computes their connected components. Its clusterings equal the clusterings from the approximate tree in every row. The last column gives the time in which `fast_hdbscan` builds the exact tree with Borůvka's algorithm, which does not depend on k. The time of the direct method follows the number of radius edges, the time of the check does not: the check is faster in every row, by a factor of at least 23 on iNaturalist and of more than $1{,}300$ on DENSIRED. Building the exact tree once costs more than any single check, but less than the direct method in all rows except iNaturalist with $k{\geq}100$.

Input	k	$q(t)$	core pts	radius edges	cluster pairs	check (s)	direct (s)	exact tree (s)
MNIST ($7{\times}10^4$, $m{=}10$)	2	834	69,169	$3.6{\times}10^8$	3 / 3	0.05	43.3	1.04
MNIST ($7{\times}10^4$, $m{=}10$)	5	2,025	67,982	$2.7{\times}10^8$	10 / 21	0.04	31.5	1.04
MNIST ($7{\times}10^4$, $m{=}10$)	10	5,803	64,208	$1.1{\times}10^8$	4 / 55	0.03	12.1	1.04
MNIST ($7{\times}10^4$, $m{=}10$)	30	11,454	58,563	$3.1{\times}10^7$	5 / 136	0.02	4.97	1.04
DENSIRED (10^5, $m{=}1$)	2	2	100,000	$1.7{\times}10^9$	1 / 1	0.05	320	2.62
DENSIRED (10^5, $m{=}1$)	5	5	100,000	$1.4{\times}10^9$	1 / 10	0.02	178	2.62
DENSIRED (10^5, $m{=}1$)	10	10	100,000	$8.7{\times}10^8$	1 / 45	0.02	107	2.62
DENSIRED (10^5, $m{=}1$)	30	30	99,980	$2.8{\times}10^8$	0 / 45	0.01	35.6	2.62
DENSIRED (10^5, $m{=}1$)	100	100	99,910	$2.7{\times}10^8$	0 / 45	0.01	34.7	2.62
DENSIRED (10^5, $m{=}1$)	300	300	99,710	$2.6{\times}10^8$	0 / 45	0.01	30.0	2.62
DENSIRED (10^5, $m{=}1$)	1000	1,000	99,010	$2.4{\times}10^8$	0 / 45	0.01	28.5	2.62
DENSIRED (10^5, $m{=}1$)	3000	3,000	97,013	$2.1{\times}10^8$	3 / 78	0.01	26.0	2.62
DENSIRED (10^5, $m{=}1$)	10000	10,000	90,067	$1.8{\times}10^8$	118 / 2,211	0.02	22.5	2.62
iNat-500k AE ($5{\times}10^5$, $m{=}10$)	2	23,122	476,881	$1.6{\times}10^8$	2 / 3	0.30	119	15.56
iNat-500k AE ($5{\times}10^5$, $m{=}10$)	5	107,798	392,257	$5.6{\times}10^7$	59 / 1,485	0.29	57.4	15.56
iNat-500k AE ($5{\times}10^5$, $m{=}10$)	10	125,666	374,408	$4.7{\times}10^7$	77 / 2,701	0.29	51.1	15.56
iNat-500k AE ($5{\times}10^5$, $m{=}10$)	30	176,920	323,241	$2.9{\times}10^7$	218 / 12,880	0.25	35.8	15.56
iNat-500k AE ($5{\times}10^5$, $m{=}10$)	100	291,491	208,887	$9.8{\times}10^6$	1,283 / 71,253	0.24	12.6	15.56
iNat-500k AE ($5{\times}10^5$, $m{=}10$)	273	421,899	78,665	$1.6{\times}10^6$	1,361 / 158,766	0.10	2.77	15.56

of every run, and `provenance/` (hashes of inputs, results and source code, resolved package versions).

- Repair of the 32 failed clusterings, sensitivity to k and m, total cost (Sections E.1 and E.2): `extended/submission_study.py`, `extended/inat_scaling.py`, `direct_dbscan_star.py`; records under `results/submission_20260916/` and `results/slurm_timing_699943/`; `repair_quality.py` scores the repaired clusterings (Table 12). ImageNet-1k (Section E.3):

Table 31: Clustering quality of our trained pipeline compared with SHADE and with the further methods named in the text of this section (NMI/ARI, default protocol of our implementation, seed 0). The best NMI of each row is in bold.

Dataset	Ours	SHADE	UMAP+HDB	DipDECK	DEC	IDEC
Synth-low	.935/.897	**.993**/.994	.529/.054	.442/.223	.709/.442	.575/.320
Synth-high	.806/.658	**.978**/.983	.525/.055	.582/.336	.575/.316	.531/.296
USPS	.830/.803	.696/.668	**.841**/.712	.805/.726	.770/.666	.765/.663
Optdigits	.722/.469	.847/.838	**.920**/.905	.860/.812	.867/.815	.890/.869
COIL-20	**.920**/.839	.882/.674	.859/.681	.855/.689	.793/.578	.772/.560
COIL-100	**.948**/.800	.867/.569	.865/.512	.715/.113	.864/.584	.846/.590

`factorial/extract_imagenet.py`, `factorial/embed_inat_full.py`, `extended/inat_scaling.py`, `slurm_timing.py`; records under `results/imagenet_full_20260925/`.

- Trees built by other libraries (Tables 19 and 27): `certify_external_trees.py`, `alarm_severity.py`; damaged neighbor graphs (Table 28): `adversarial_audit.py`; ablation of the repair (Table 25): `repair_ablation.py`.
- Check against the direct computation for several thresholds (Table 30): `regime_sweep.py`; scaling with N and d (Tables 22 and 29): `certificate_scaling.py`, `baselines/run_fast_hdbscan_scaling.py`; higher dimensionalities on real features (Table 9): `highdim_check.py`.
- Factorial study of the embeddings (Section F): `factorial/extract_features.py`, `factorial/run_cell.py` (`-eom-exact` for Table 18, `-eom-min-samples` for Table 17), `factorial/run_linear_compression.py` (Table 15); merged results in `factorial/_out_cells_merged.jsonl`.
- Exact HDBSCAN* with ELKI (Table 21): `sisap/`; quality scores on iNaturalist: `inat_metric_analysis.py`; correctness of the check against exact references on the complete graph (108 configurations): `verify_certificate.py`, `independent_spotcheck.py`.

Implementation and tests. The check is `src/cut_certificate.py`, its C kernels are `src/native_accel.py` and `src/native_kdtree.py`; the repair, the margin and the variant that determines only the core points are in `src/certificate_extensions.py`, the hierarchy check in `src/hierarchy_certificate.py`, the projection filter in `src/safe_projection.py`, and the fixed tie rule in `src/canonical_hierarchy.py`. The tests in `tests/` cover the correctness of the check, the termination of the repair, the strict check at several thresholds, numerical boundary cases, the C kernels against the NumPy and SciPy code, and, on four problems, that the search misses no pair; `extended/validation.py` compares the hierarchy check, the selection and the repair with a brute-force computation on $1{,}000$ small problems (Section B).