Dynamic k-center clustering with lifetimes

Simone Moretti *
KTH
simmor@kth.se

Paolo Pellizzoni *
Harvard University
paolo_pellizzoni@seas.harvard.edu

Andrea Pietracaprina *
University of Padova
andrea.pietracaprina@unipd.it

Geppino Pucci *
University of Padova
geppino.pucci@unipd.it

Abstract

The k-center problem is a fundamental clustering variant with applications in learning systems and data summarization. In several real-world scenarios, the dataset to be clustered is not static, but evolves over time, as new data points arrive and old ones become stale. To account for dynamicity, the k-center problem has been mainly studied under the sliding window setting, where only the N most recent points are considered non-stale, or the fully dynamic setting, where arbitrary sequences of point arrivals and deletions without prior notice may occur. In this paper, we introduce the *dynamic setting with lifetimes*, which bridges the two aforementioned classical settings by still allowing arbitrary arrivals and deletions, but making the deletion time of each point known upon its arrival. Under this new setting, we devise a deterministic $(2+\varepsilon)$-approximation algorithm with $\widetilde{O}(k/\varepsilon)$ amortized update time and memory usage linear in the number of currently active points. Moreover, we develop a deterministic $(6+\varepsilon)$-approximation algorithm that, under "tame" update sequences, has $\widetilde{O}(k/\varepsilon)$ worst-case update time and heavily sublinear working memory.

1 Introduction

Clustering, which is used to summarize and to group together data, is one of the most widely studied primitives in machine learning and data mining [21]. A common clustering task is k-center clustering, where, given a set of data points X from a metric space $(\mathcal{U}, d)$, one needs to select k centers from X such that the maximum distance from any data point at hand to its closest center is minimized. This problem is well-known to be NP-hard [18], so approximation algorithms are sought.

Motivated by many real-world scenarios, recent years have witnessed a growing interest in devising dynamic clustering algorithms [11, 2, 12], where the set of data points is not static, but rather changes dynamically, allowing new points to be inserted and others to be deleted. For example, the need for clustering dynamic data may arise for tracking stored products in a warehouse [25, 36], for analyzing the behaviors of active users in a web community [31], or studying evolving populations [16].

Dynamic clustering focuses on the analysis of (unbounded) sequences of updates (i.e., insertions or deletions of points), focusing, at any time, on the instance defined by the *active set* of points, namely, those that have been inserted and not yet deleted. The two main dynamic settings that allow both insertions and deletions are the *sliding window* setting and the *fully dynamic* setting. In the sliding window setting, the active set consists of the N most recent points, that is, whenever a new point is inserted, the oldest one is deleted [12, 6]. The focus is on designing algorithms which use $o(N)$

*equal contribution

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

memory, hence avoiding storing the entire active set. In the fully dynamic setting, both insertions and deletions are arbitrary: at any time, a new point may be inserted or an active point may be deleted [11, 2]. This latter setting is more general and therefore more challenging than the former, but known approaches typically allow memory usage linear in the size of the active set, shifting the focus to approximation quality and fast update times.

In this paper, we introduce a third setting, offering a middle-ground between the aforementioned two, which we refer to as the *dynamic setting with lifetimes*. In this setting, when a point is inserted, an arbitrary deletion time is also specified, which defines the point's lifetime. This is more general than the sliding window setting, where the lifetimes are equal and fixed for all points, but it is less general than the fully dynamic one, where deletion times are not known in advance.

The dynamic setting with lifetimes has several relevant real-world applications. For example, with reference to the aforementioned scenarios, items might arrive at a warehouse with a pre-assigned specific departure time, or subscriptions to a web community might have known expiration dates. Another concrete example is provided by public TLS certificates recorded in Certificate Transparency logs. Indeed, when a certificate is recorded, it already specifies its expiration time, so the deletion time is known upon arrival [13, 28].

Our contribution In this paper, we tackle the k-center problem under the dynamic setting with lifetimes. In particular, we introduce:

- a deterministic $(2+\varepsilon)$-approximation algorithm with $\widetilde{O}(k/\varepsilon)$ amortized update time[1] and memory usage linear in the number of currently active points;
- a deterministic $(6+\varepsilon)$-approximation algorithm that under tame update sequences (see Definition 4.3) has $\widetilde{O}(k/\varepsilon)$ worst-case update time and $\widetilde{O}(k/\varepsilon)$ working memory.

A thorough comparison of the two algorithms with previous work is carried out in Subsections 3.3 and 4.3, respectively, showing that the knowledge of lifetimes yields improved performance guarantees over the state-of-the-art algorithms in the fully dynamic and sliding window settings.

Additional related work Several works have tackled k-center in the fully dynamic setting [10, 1], focusing on variants with outliers [10, 4], on low-dimensional spaces [19, 34, 17], on minimizing recourse [26, 3, 15], and on graphs [14]. In the sliding window model, Cohen-Addad et al. [12] tackled k-center while subsequent works addressed variants [33, 37, 9], focused on low-dimensional spaces [32] or related clustering problems [6]. Dynamic settings akin to the one *with lifetimes* have been addressed recently, including the *deletions-look-ahead* model [35], the *general expiration streaming* model [5], and models with predicted deletion times [29]. More details are in Appendix A.

Organization of the paper Section 2 defines preliminary technical notions and formally specifies the k-center problem under the dynamic setting with lifetimes. Sections 3 and 4 present our algorithms and their analyses, providing, for each algorithm, a detailed comparison with previously known ones. Section 5 concludes with some final remarks. Due to space constraints, additional considerations on related and future work, some pseudocodes, and some proofs are deferred to the appendix.

2 Preliminaries

k-center Consider a metric space $(\mathcal{U}, d)$ and a finite set $X \subseteq \mathcal{U}$. For any point $p \in X$ and any subset $C \subseteq X$, we define $d(p, C) = \min_{q \in C} d(p, q)$, and for any $S \subseteq X$ we denote the *radius of C with respect to S* as $r_C(S) = \max_{p \in S} d(p, C)$.

The *k-center* problem, for a given positive integer k, requires to find a subset $C \subseteq X$ such that $|C| \leq k$ which minimizes $r_C(X)$, i.e., $C^* = \arg\min_{C \subseteq X, |C| \leq k} r_C(X)$. We denote the radius of the optimal solution by $r^*_k(X) = r_{C^*}(X)$. The points of the set C are called *centers*. Indeed, C induces a partition of X into $|C|$ clusters by assigning each point to its closest center (with ties broken arbitrarily). In general metric spaces, the problem is NP-hard, and it is impossible to achieve an approximation factor $2-\varepsilon$, for any fixed $\varepsilon > 0$, unless P = NP [24, 18].

[1] The notation $\widetilde{O}$ hides terms logarithmic in the relevant parameters.

The dynamic setting with lifetimes Let X be a dataset whose subsets represent instances of a given optimization problem[2]. In the *dynamic with lifetimes setting*, each element $p \in X$ is endowed with an *arrival time* $t_{\text{arr}}(p) \in \mathbb{N}$ and a *deletion time* $t_{\text{del}}(p) \in \mathbb{N}$, with $t_{\text{arr}}(p) < t_{\text{del}}(p)$, and we say that p *expires* (i.e., its lifetime ends) at time $t_{\text{del}}(p)$. We assume that for each time $t \geq 0$, there exists at most one element $p \in X$ with $t_{\text{arr}}(p) = t$. At any time $t \geq 0$, we say that an element p is *active* if $t_{\text{arr}}(p) \leq t < t_{\text{del}}(p)$, and define, accordingly, the *active set* X_t as

$$X_t = \{p \in X \ : \ t_{\text{arr}}(p) \leq t < t_{\text{del}}(p)\}.$$

Also, we assume that an oracle communicates the arrivals of elements from X, ordered by their arrival time. The oracle reveals a point p and its deletion time $t_{\text{del}}(p)$ only at time $t_{\text{arr}}(p)$ and provides no information about future arrivals. An algorithm in this dynamic setting maintains a suitable representation of X_t, by running a primitive UPDATE(p) upon the arrival of a new element p, and, at any time t, is able to compute a solution with respect to instance X_t by running a primitive QUERY(t).

The goals that the algorithm should pursue are:

1. Good approximation of the optimal solution for X_t;
2. Query time sublinear in $|X_t|$;
3. Update time sublinear in $|X_t|$ (possibly amortized);
4. Memory requirements at most linear in $|X_t|$ (*Optionally*: memory sublinear in $|X_t|$).

Note that the main requirements are shared with both the fully dynamic and sliding window settings. The optional one, typical of the sliding window but not of the fully dynamic setting, makes the algorithm design more challenging. Under this requirement, our setting is analogous to the *general expiration streaming model*, which was recently introduced in Blank et al. [5] as an extension of the streaming model, thus focusing mainly on memory requirements.

Dynamic k-center with lifetimes Let X be a set of points with arrival and deletion times, as described above. The dynamic k-center with lifetimes problem on X requires that, at any given time t, a set of k centers $C_t \subseteq X_t$ be determined, whose radius $r_{C_t}(X_t)$ is close to the optimal one $r^*_k(X_t)$ for X_t. An algorithm for this problem consists of an UPDATE procedure, which deals with insertions and deletions, so to maintain a suitable representation of the active set X_t, and a QUERY procedure, which, at any given time t, returns a solution C_t for X_t. The approximation ratio of the algorithm is the maximum of $r_{C_t}(X_t)/r^*_k(X_t)$ over any set X_t.

Let $d_{\min}$ and $d_{\max}$ be the minimum and maximum distances between any two distinct points of X, and define the *aspect ratio* of X as $\Delta = d_{\max}/d_{\min}$. As standard in most previous works, our algorithms will assume that $d_{\min}$ and $d_{\max}$ are known.

3 A $(2+\varepsilon)$-approximation algorithm

This section presents a deterministic algorithm for dynamic k-center with lifetimes, which attains an almost-tight $(2+\varepsilon)$ approximation ratio, requiring memory linear in $|X_t|$ and $O((\log \Delta/\varepsilon) \cdot k + \max_t \log |X_t|)$ amortized update time. We assume that $|X_t| > k$ whenever a query is made. Queries when $|X_t| \leq k$ can be handled separately by returning the entire active set.

The algorithm's UPDATE and QUERY procedures, described later in this section, crucially exploit a data structure based on several guesses for the optimal radius. Specifically, since the optimal radius clearly falls in $[d_{\min}, d_{\max}]$, we define the following set of guesses:

$$\Gamma = \left\{(1+\beta)^i : \lfloor \log_{1+\beta} d_{\min} \rfloor \leq i \leq \lceil \log_{1+\beta} d_{\max} \rceil \right\}, \tag{1}$$

where $\beta = \varepsilon/2$, and $\varepsilon > 0$ is the user-defined approximation tolerance. At a high level, for each $\gamma \in \Gamma$ we dynamically maintain a data structure, akin to the one used in [11], consisting of the sets:

- $C^\gamma = (c^\gamma_1, \ldots, c^\gamma_{\ell^\gamma})$, a list of $\ell^\gamma \leq k$ centers, such that for each $1 \leq i \neq j \leq \ell^\gamma$, we have $d(c^\gamma_i, c^\gamma_j) > 2\gamma$;

[2] Although we define the dynamic setting with lifetimes with respect to optimization problems, several computational problems can be cast in this setting.

Algorithm 1: UPDATE$_{2+\varepsilon}(p, t)$

1 **while** $Q \neq \emptyset$ *and* $\min_{q \in Q} t_{\mathrm{del}}(q) \leq t$ **do**
2 Remove $q = \mathrm{argmin}_{q \in Q} t_{\mathrm{del}}(q)$ from Q
3 **for** $\gamma \in \Gamma$ **do**
4 DELETE(q, γ)
5 **if** $p \neq$ NULL **then**
6 **for** $\gamma \in \Gamma$ **do**
7 INSERT(p, γ)
8 Insert p in Q

Algorithm 2: INSERT(p, γ)

1 **for** $i = 1, \ldots \ell^\gamma$ **do**
2 **if** $d(p, c_i^\gamma) \leq 2\gamma$ **then**
3 Insert p in D_i^γ, and **goto** line 8
4 **if** $\ell^\gamma < k$ **then**
5 Insert p in C^γ and in $D_{\ell^\gamma+1}^\gamma$
6 $\ell^\gamma \leftarrow \ell^\gamma + 1$
7 **else** Insert p in U^γ;
8 RECLUSTERING(γ)

- $D^\gamma = (D_1^\gamma, \ldots, D_{\ell^\gamma}^\gamma)$, a list of $\ell^\gamma \leq k$ disjoint sets of points, such that for each $1 \leq i \leq \ell^\gamma$ and $x \in D_i^\gamma$, we have $d(x, c_i^\gamma) \leq 2\gamma$;
- U^γ, a set of unclustered points such that, for each $x \in U^\gamma$ and $1 \leq i \leq \ell^\gamma$, we have $d(x, c_i^\gamma) > 2\gamma$. Moreover $U^\gamma \neq \emptyset$ implies $\ell^\gamma = k$.

Intuitively, for a given guess γ, the sets D_i^γ's are clusters of radius 2γ centered at the points c_i^γ's. If γ is too small, we will not be able to cover the entire active set with at most k clusters, and the remaining uncovered points are thus stored in the set U^γ.

To keep track of which points need to be deleted from the active set, we maintain a copy of all points in a priority queue Q indexed on their deletion time, with ties broken by the arrival time to guarantee a unique ordering. We implicitly assume the same tie-breaking in proofs, so every comparison involving deletion times is understood with respect to the resulting total order. Given this total order, with a slight abuse of notation, in this section we implicitly reindex all operations by consecutive integers in execution order, so that $t_{\mathrm{del}}(p) - 1$ denotes the step immediately before the deletion of p.

The D_i^γ's and U^γ's are implemented as doubly-linked lists, and each entry in Q has a pointer to the entries of the D_i^γ's and U^γ's corresponding to the same point, so to allow these latter entries to be identified in $O(1)$ time. By construction, if $U^\gamma = \emptyset$, then C^γ provides a solution for X_t of radius $\leq 2\gamma$, while, if $U^\gamma \neq \emptyset$, then C^γ plus any point of U^γ yields a set of $k+1$ points of X_t at pairwise distance $> 2\gamma$, which is a witness that no solution of radius γ for X_t exists.

Update procedure Procedure UPDATE$_{2+\varepsilon}(p, t)$ (see Algorithm 1 for the pseudocode) is called when a new point p arrives (in this case $t = t_{\mathrm{arr}}(p)$), or at any time t when expired points, if any, need to be deleted from the data structure (in this case, $p =$ NULL). First, for each $q \in Q$ with $t_{\mathrm{del}}(q) \leq t$, q is removed from Q and procedure DELETE(q, γ) is invoked for each $\gamma \in \Gamma$. Then, if $p \neq$ NULL, procedure INSERT(p, γ) is invoked for each $\gamma \in \Gamma$, and p is added to the priority queue Q.

INSERT(p, γ) (see Algorithm 2 for the pseudocode) inserts p in the cluster of D^γ of lowest index whose center is at distance at most 2γ from p. If no such cluster exists, then if D^γ comprises $\ell^\gamma < k$ clusters, then the procedure creates a new cluster by making p its center, otherwise, it inserts p in U^γ. At the end of INSERT(p, γ) a reclustering procedure, described later, is called.

DELETE(p, γ), where p is an expired point, works as follows (see Algorithm 3). If p is not a center, it is simply removed from its cluster in D^γ. If p is a center c_i^γ, the procedure deletes c_i^γ and tries to recluster all points of D_i^γ in clusters with index $> i$. Intuitively, each of these points is inserted in the cluster where it would have been inserted if c_i^γ had not existed. Then, all cluster indexes greater than i are decremented by 1, making them contiguous again, and, if $U^\gamma \neq \emptyset$, the point of U^γ with the latest deletion time becomes the center of a new cluster. Finally, the reclustering procedure is called.

Note that the expensive part of the deletion procedure, namely, re-assigning points and creating a new cluster, only happens if a cluster center is deleted. In order to limit these costly operations, we aim at selecting centers with largest deletion times. In particular, this is enforced when a new center from U^γ must be selected (Lines 17-23 of DELETE(p, γ)), and when the reclustering is performed at the end of INSERT and DELETE, which is described next.

For a given $\gamma \in \Gamma$, we define as *persistent* those clustered points which expire after their respective clusters' centers, and *vanishing* those which either expire earlier than their clusters' centers or are unclustered. Note that only persistent points can change cluster because of the deletion of their center, as vanishing points expire before the deletion of their center. We partition each cluster D_i^γ

Algorithm 3: DELETE(p, γ)

if $p \notin C^\gamma$ **then**
  Remove p from U^γ or D^γ
  goto line 24
Let i be such that $p = c_i^\gamma$
for $x \in D_i^\gamma$ with $x \neq p$ **do**
  for $j = i+1, \ldots, \ell^\gamma$ **do**
    if $d(x, c_j^\gamma) \leq 2\gamma$ **then**
      Insert x in D_j^γ, and **break**
  if x *was not inserted* **then**
    if $\ell^\gamma < k$ **then**
      Insert x in C^γ and in $D_{\ell^\gamma+1}^\gamma$;
      $\ell^\gamma \leftarrow \ell^\gamma + 1$
    else Insert x in U^γ;
Remove c_i^γ and D_i^γ;
$\ell^\gamma \leftarrow \ell^\gamma - 1$;
Rename indexes of C^γ, D^γ to be contiguous
if $U^\gamma \neq \emptyset$ **then**
  $\ell^\gamma \leftarrow \ell^\gamma + 1$;
  Pick $u \in U^\gamma$ s.t. $t_{\text{del}}(u)$ is maximal;
  Remove u from U^γ, insert in C^γ, D^γ as $c_{\ell^\gamma}^\gamma$
  for $x \in U^\gamma$ **do**
    if $d(x, u) \leq 2\gamma$ **then**
      Remove x from U^γ, insert in $D_{\ell^\gamma}^\gamma$
RECLUSTERING(γ)

Algorithm 4: RECLUSTERING(γ)

$i \leftarrow$ min index h s.t. $\sum_{j=h}^{\ell^\gamma} |P_j^\gamma| > |U^\gamma| + \sum_{j=h}^{\ell^\gamma} |V_j^\gamma|$
if *no such* i *exists* **then**
  return
$U' \leftarrow U^\gamma \cup (\cup_{j=i}^{\ell^\gamma} D_j^\gamma)$
Remove centers c_j^γ and clusters D_j^γ $\forall j \geq i$;
$\ell^\gamma \leftarrow i - 1$;
for $j = i, \ldots, k$ **do**
  if $U' = \emptyset$ **then**
    break
  $\ell^\gamma \leftarrow j$;
  Pick $u \in U'$ s.t. $t_{\text{del}}(u)$ is maximal;
  Remove u from U', insert in C^γ, D^γ as c_j^γ;
  for $x \in U'$ **do**
    if $d(x, u) \leq 2\gamma$ **then**
      Remove x from U', insert in D_j^γ
$U^\gamma \leftarrow U'$

into persistent and vanishing points as $P_i^\gamma := \{x \in D_i^\gamma : \ t_{\text{del}}(c_i^\gamma) < t_{\text{del}}(x)\}$ and $V_i^\gamma := \{x \in D_i^\gamma : \ t_{\text{del}}(c_i^\gamma) \geq t_{\text{del}}(x)\}$. In practice, we maintain two counters per cluster for $|P_i^\gamma|$ and $|V_i^\gamma|$, which can be updated in $O(1)$ time for each insertion and deletion.

Procedure RECLUSTERING(γ) (see Algorithm 4), first searches for the smallest index i such that

$$\sum_{j=i}^{\ell^\gamma} |P_j^\gamma| > |U^\gamma| + \sum_{j=i}^{\ell^\gamma} |V_j^\gamma|. \tag{2}$$

If such i exists, then it reclusters the points of $U^\gamma \cup (\cup_{j=i}^{\ell^\gamma} D_j^\gamma)$ into $\ell^\gamma - i + 1$ clusters, selecting, for each such new cluster, a center with latest deletion time. Note that, after reclustering, some points might remain unclustered, but the newly created clusters comprise only vanishing points.

The following remark shows that the reclustering performed after each invocation of INSERT or DELETE is crucial to obtain sublinear amortized update time, as without it a linear number of updates may require a quadratic number of operations.

Remark 3.1. *There exists a sequence of $O(n)$ calls to* UPDATE$_{2+\varepsilon}$ *which, without the use of* RECLUSTERING*, would perform a total of $\Theta(n^2)$ operations.*

Query procedure At any time t, an approximate solution to k-center for the active set X_t can be computed calling QUERY$_{2+\varepsilon}(t)$ (see Algorithm 7 for the pseudocode). We assume that when QUERY$_{2+\varepsilon}(t)$ is invoked, the data structure contains no expired points. This can be ensured by checking that the minimum deletion time of a point in Q is $> t$, and, if this is not the case, invoking UPDATE$_{2+\varepsilon}$(NULL, t) before QUERY$_{2+\varepsilon}(t)$. The procedure simply returns the set C^{γ^*}, with $\gamma^* := \min_{\gamma \in \Gamma}\{\gamma : \ U^\gamma = \emptyset\}$, as a set of centers.

The following subsections analyze the approximation ratio, and the time and memory requirements of the above algorithm. For convenience, we define a *time step* as one call of INSERT or DELETE. Since arrival times are distinct and Q breaks ties for deletion times based on arrival times, we can assume, without loss of generality, that each call corresponds to a unique time step. For any time step $t \geq 0$, we use X_t to denote the active set, and $C^{\gamma,t}, D^{\gamma,t}, U^{\gamma,t}, Q^t$ to denote the various components of the data structure, *after* the t_{th} call of INSERT or DELETE. Time step $t = 0$ refers to the initial phase of the algorithm, when nothing is yet inserted. We may drop t when clear from the context.

3.1 Approximation

First, we prove that the algorithm maintains the following invariant. This can be then used to prove the approximation guarantee for $\text{QUERY}_{2+\varepsilon}$.

Lemma 3.2. *For any time step t, the following properties hold for each $\gamma \in \Gamma$, and $x \in X_t$:*

1. *if $d(x, C^{\gamma,t}) > 2\gamma$, then $x \in U^{\gamma,t}$;*
2. *if $d(x, C^{\gamma,t}) \leq 2\gamma$, then $x \in D_i^{\gamma,t}$ with $i = \text{argmin}_{j=1,\dots \ell^{\gamma,t}}\{c_j \in C^{\gamma,t} \colon\ d(x, c_j) \leq 2\gamma\}$.*

Moreover, if $U^{\gamma,t} \neq \emptyset$ then $|C^{\gamma,t}| = k$.

At a high level, the invariant states that all points that are not within 2γ of a center in C^γ are maintained in U^γ (which can happen only if there are already k centers), and that if a point is within 2γ of a center in C^γ, then it is assigned to the cluster with the lowest index.

Theorem 3.3. *For any time step t, the solution returned by $\text{QUERY}_{2+\varepsilon}(t)$ is a $(2+\varepsilon)$-approximation for the k-center problem on the active set X_t.*

Proof. Recall that the algorithm returns $C^{\gamma^*,t}$, with $\gamma^* = \min_{\gamma\in\Gamma}\{\gamma \colon\ U^{\gamma,t} = \emptyset\}$. If $\gamma^* \leq d_{\min}$, the theorem trivially holds. Suppose $\gamma^* > d_{\min}$, and let $\gamma' = \gamma^*/(1+\beta) \in \Gamma$. Clearly, $U^{\gamma',t} \neq \emptyset$, hence $|C^{\gamma',t}| = k$. Let $W = C^{\gamma',t} \cup \{x\}$, for an arbitrary $x \in U^{\gamma',t}$. By construction and by Lemma 3.2 we have that $\forall p \neq q \in W$, $d(p,q) > 2\gamma'$. Moreover, since $|W| = k+1$, we have that $r_k^*(X_t) \geq \min_{p\neq q\in W} d(p,q)/2 > \gamma' = \gamma^*/(1+\beta)$, as at least two distinct points of W are in the same optimal cluster. By Lemma 3.2, $\max_{x\in X_t} d(x, C^{\gamma^*,t}) \leq 2\gamma^* < 2(1+\beta)r_k^*(X_t) \leq (2+\varepsilon)r_k^*(X_t)$. □

3.2 Running time and memory complexity

Consider a sequence of n consecutive calls to the $\text{UPDATE}_{2+\varepsilon}$ procedure, which must involve $\Theta(n)$ insertions and $O(n)$ deletions. In what follows, we analyze the performance of our algorithm relatively to these calls. Let us focus on an arbitrary guess $\gamma \in \Gamma$. We first analyze the overall cost of the reclustering operations.

Lemma 3.4. *If a point $x \in X$ is vanishing for γ at time t_0, then x remains vanishing for all t such that $t_0 \leq t < t_{\text{del}}(x)$.*

Proposition 3.5. *The total number of operations made by all calls of $\text{RECLUSTERING}(\gamma)$ is $O(n \cdot k)$.*

We now estimate the overall costs of insertions and deletions. For insertions, the following proposition holds because of Proposition 3.5 and because, besides the call to $\text{RECLUSTERING}(\gamma)$, $\text{INSERT}(p, \gamma)$ takes $O(k)$ time.

Proposition 3.6. *The total number of operations made by all calls of $\text{INSERT}(p, \gamma)$ is $O(n \cdot k)$.*

Consider now the deletions. In what follows, for time t and $x \in X_t$, and for each $\gamma \in \Gamma$, let $c_{ID}^{\gamma,t}(x)$ be the index of the cluster in D^γ that x is in at time t, and let $c_{ID}^{\gamma,t}(x) = k+1$ if $x \in U^\gamma$.

To bound the total number of operations made by deletions we first show that once a point becomes vanishing the number of deleted centers with index smaller than the point's index is not greater than $k+1$. This information, together with Property (2), will later help upper bound the number of re-homing operations made at Line 5.

Lemma 3.7. *Let $x \in X$ and $\gamma \in \Gamma$. Call $\mathcal{Q}_x^\gamma$ the set of centers c such that at time $t = t_{\text{del}}(c) - 1$ it holds that x is vanishing for γ and $c_{ID}^{\gamma,t}(c) \leq c_{ID}^{\gamma,t}(x)$. Then, $|\mathcal{Q}_x^\gamma| \leq k+1$.*

Proposition 3.8. *The total number of operations made by all calls of $\text{DELETE}(p, \gamma)$ is $O(n \cdot k)$.*

Proof. First, we bound the number of operations done by the re-assignment of cluster points in the loop at Line 5. We begin by noticing that when c_i is deleted, only persistent points contribute to the loop at Line 5. For a single such point $x \in D_i, x \neq c_i$, Lines 6-13 perform $O\big(c_{ID}^{\gamma,t}(x) - c_{ID}^{\gamma,t-1}(x)\big)$ operations, i.e., proportional to the difference between the current index i and the index of the cluster that it will end up in. If $c_{ID}^{\gamma,t}(x)$ was non-decreasing over time, then the maximum number

of operations for x across all timesteps would be $O(k)$. However this is not the case because of re-indexing due to center deletions.

Given that x must be a persistent point, we are only interested in events in which $c_{ID}^{\gamma,t}(x)$ decreases and x remains a persistent point. The only such case occurs during the relabeling process in a deletion that involves a center (Line 16) with index $< c_{ID}^{\gamma,t-1}(x)$. In this case $c_{ID}^{\gamma,t}(x) = c_{ID}^{\gamma,t-1}(x) - 1$, so each center deletion adds 1 to the maximum number of re-assignment operations for each persistent point with index greater or equal to its own.

Let $\mathcal{D} := \{c : c \in C^{\gamma,t_{\text{del}}(c)-1}\}$ be the set of all deleted centers at any time. We remark that, for $x \in X$, superscripts $t_{\text{del}}(x) - 1$ denote states right before the deletion of x. We get that the total number of re-assignment operations is:

$$O\Big(n{\cdot}k{+}\sum_{c\in\mathcal{D}}\sum_{i\geq c_{ID}^{\gamma,t_{\text{del}}(c)-1}(c)}|P_i^{\gamma,t_{\text{del}}(c)-1}|\Big) = O\Big(nk{+}\sum_{c\in\mathcal{D}}\Big(|U^{\gamma,t_{\text{del}}(c)-1}|{+}\sum_{i\geq c_{ID}^{\gamma,t_{\text{del}}(c)-1}(c)}|V_i^{\gamma,t_{\text{del}}(c)-1}|\Big)\Big),$$

where we used the negation of Property (2), which holds after reclustering. If we fix $c \in \mathcal{D}$ in the last summation and call $j := c_{ID}^{\gamma,t_{\text{del}}(c)-1}(c)$ we get that, for each $x \in U^{\gamma,t_{\text{del}}(c)-1} \cup \bigcup V_{i\geq j}^{\gamma,t_{\text{del}}(c)-1}$, x is by definition vanishing and $c_{ID}^{\gamma,t_{\text{del}}(c)-1}(c) \leq c_{ID}^{\gamma,t_{\text{del}}(c)-1}(x)$. Therefore we have:

$$\begin{aligned}\sum_{c\in\mathcal{D}}\Big(|U^{\gamma,t_{\text{del}}(c)-1}| + \sum_{i\geq c_{ID}^{\gamma,t_{\text{del}}(c)-1}(c)}|V_i^{\gamma,t_{\text{del}}(c)-1}|\Big) &\leq \sum_{x\in X}\Big|\Big\{c\in\mathcal{D}:\ {\scriptstyle x \text{ is vanishing at time } t_{\text{del}}(c)-1 \atop \text{and } c_{ID}^{\gamma,t_{\text{del}}(c)-1}(c)\leq c_{ID}^{\gamma,t_{\text{del}}(c)-1}(x)}\Big\}\Big| \\ &= \sum_{x\in X}|\mathcal{Q}_x^\gamma| \leq n\cdot(k+1),\end{aligned}$$

where we used a double counting argument by rearranging the summation order, and then applied Lemma 3.7. Hence the total number of operations made by the re-assignment of cluster points in the deleting function over all calls of $\textsc{Delete}(p,\gamma)$ is $O(n \cdot k)$. Second, we bound the number of operations made by the creation of a new cluster in the deleting function (Line 17). When creating a new cluster we scan each element in U for finding the one with greater t_{del} and then we do the same for finding which points are within distance 2γ. Therefore, the total number of operations is $O(\sum_{c\in\mathcal{D}}|U^{\gamma,t_{\text{del}}(c)-1}|)$, which, as above, is $O(n \cdot k)$.

Finally, all other operations in the deletion procedure take $O(k)$ time, which concludes the proof. □

Based on Propositions 3.6 and 3.8, we can prove the bound for update times. We also state the bound for query times, as well as the memory requirements of our algorithm.

Theorem 3.9. *The total number of operations made by a sequence of n consecutive calls to* $\textsc{Update}_{2+\varepsilon}$ *is* $O(n(\max_t \log|X_t| + (\log\Delta/\varepsilon)k))$.

Theorem 3.10. *The number of operations made by each* $\textsc{Query}_{2+\varepsilon}$ *call is* $O((\log\Delta/\varepsilon)+k)$.

Theorem 3.11. *The maximum number of points maintained in memory when processing the pointset X is* $O((\log\Delta/\varepsilon)\max_t|X_t|)$.

Interestingly, our update times are tight, up to a factor $O(\log\Delta/\varepsilon)$, due to the lower bound of $\Omega(|X|k)$ distance queries given by Bateni et al. [1, Theorem 54], which holds even in the offline setting.

3.3 Comparison with related work

As we mentioned in the introduction, the dynamic setting with lifetimes bridges the sliding window and fully dynamic frameworks. We compare our algorithm to the existing fully dynamic ones, as they can be used even in the setting with lifetimes (while sliding windows ones would lose all guarantees).

Our algorithm mimics the fully-dynamic algorithm from Chan et al. [11], yet by exploiting the known deletion times it lowers the amortized update times from $O\big(\frac{\log\Delta}{\varepsilon}k^2\big)$ to $O\big(\frac{\log\Delta}{\varepsilon}k\big)$, while having the same approximation ratio. In fact, this allows us to have faster updates even compared to the more complex $(2+\varepsilon)$-approximation algorithm by Bateni et al. [1], which has amortized $O\big((\log\Delta/\varepsilon)\max_t\log|X_t|(k+\max_t\log|X_t|)\big)$ update time. Importantly, since our algorithm is deterministic, it works even against metric-adaptive adversaries, while the aforementioned two can

only deal with oblivious ones. Therefore, in settings where lifetimes are available, it would be wasteful to use a fully-dynamic algorithm, as they are more computationally expensive and with less guarantees, rather than our algorithm with lifetimes. In fact, making use of lifetimes, our algorithm can *sidestep* the metric-adaptive lower bound of Bateni et al. [1, Thm. 3], which rules out fully dynamic $O(1)$-approximation algorithms with polylogarithmic update time for a range of values of k against metric-adaptive adversaries. Recent work by Grilnberger and Skarlatos [20] obtains a randomized $O(1)$-approximation with $\widetilde{O}(k)$ update time against point-adaptive adversaries, which does not contradict the aforementioned lower bound, as the metric is fixed in advance.

The recent paper by Blank et al. [5] addresses a dynamic setting akin to the one with lifetimes, while focusing primarily on the memory requirements. Indeed, their algorithm requires $O\big((\log \Delta/\varepsilon) \cdot k^3\big)$ memory, as opposed to our requirements, which are linear in the active set size. However, their algorithm features a high $O\big((\log \Delta/\varepsilon) \cdot k^4\big)$ update time, and, more importantly, features only a $(6k + 2 + \varepsilon)$ approximation ratio, which is much worse than our almost-tight $(2 + \varepsilon)$-approximation.

4 A space-efficient algorithm with constrained lifetimes

The algorithm developed in the previous section achieves an almost-tight approximation ratio, but requires memory which is linear in the size of the active set. In this section, we devise an algorithm which requires much smaller memory, constant with respect to the size of the active set, at the cost of a slightly worse approximation guarantee. As before, we assume that $|X_t| > k$ whenever a query is made. Queries when $|X_t| \leq k$ can be handled by maintaining the k points with largest deletion times.

Let $\varepsilon > 0$ be a user-defined approximation tolerance, and let Γ be the set of guesses defined by Equation 1, instantiated now with $\beta = \varepsilon/6$. At any time t, the algorithm maintains, for each $\gamma \in \Gamma$, a data structure, which is able to provide a solution of radius 6γ or a certificate that no solution of radius γ exists. The data structure, which closely matches the one from Cohen-Addad et al. [12] , consists of the following sets:

- $A^\gamma = (a_1^\gamma, \ldots, a_{\ell^\gamma}^\gamma) \subseteq X_t$ a set of $\ell^\gamma \leq k + 1$ *attractors* such that $d(a_i^\gamma, a_j^\gamma) > 2\gamma, \forall i \neq j$. For $x \in X_t$, $a \in A^\gamma$, we say that x is attracted by a if $t_{\mathrm{arr}}(x) \geq t_{\mathrm{arr}}(a)$ and $d(x, a) \leq 2\gamma$.
- $R^\gamma_{\mathrm{act}} = (r_1^\gamma, \ldots, r_{\ell^\gamma}^\gamma) \subseteq X_t$ a set of *representatives*, where, for $1 \leq i \leq \ell^\gamma$, r_i is the point with latest deletion time among the ones attracted by $a_i \in A^\gamma$, and it is denoted with $rep^\gamma(a_i) = r_i$. We use $R^\gamma_{\mathrm{orph}} \subseteq X_t$ to denote the set of representatives whose attractors are no longer in A^γ, and define $R^\gamma = R^\gamma_{\mathrm{act}} \cup R^\gamma_{\mathrm{orph}}$.

Intuitively, for a given guess γ, the data structure maintains up to $k + 1$ clusters of radius 2γ, centered at the points a_i^γ's. While in the data structure from Section 3 we maintained the entire clusters in memory, here we only maintain the point with longest lifetime for each cluster, called a representative.

Observe that if $|A^\gamma| \geq k + 1$, there are $k + 1$ points at distance $> 2\gamma$, which implies that $\gamma < r^*_k(X_t)$. Conversely, if $|A^\gamma| \leq k$, we will show that any point of X_t is at distance at most 6γ from R^γ.

Update procedure When a new point p arrives, procedure $\textsc{Update}_{6+\varepsilon}(p)$ is called (see Algorithm 5 for the pseudocode) to handle the insertion of p, as well as possible deletions of expired points. For every $\gamma \in \Gamma$, the data structures are updated as follows. First, all points expired before time $t_{\mathrm{arr}}(p)$ are removed from A^γ and R^γ. Then, if p is at distance $> 2\gamma$ from every attractor $a \in A^\gamma$, then p is added to A^γ, and to R^γ as representative of itself (i.e., $p = rep^\gamma(p)$), and procedure $\textsc{Cleanup}(p, \gamma)$ is invoked to possibly evict unnecessary points from A^γ and R^γ. Otherwise, an attractor $a \in A^\gamma$ with $d(a, p) \leq 2\gamma$ and $t_{\mathrm{del}}(rep^\gamma(a)) < t_{\mathrm{del}}(p)$, if any, is arbitrarily chosen and p becomes the new representative $rep^\gamma(a)$. If no such attractor is found, p is discarded.

Procedure $\textsc{Cleanup}(p, \gamma)$ (see Algorithm 6 for the pseudocode), works as follows. If $|A^\gamma| = k + 2$, then the attractor with minimum deletion time is removed from the set. After this removal, or if the size of A^γ is $k + 1$ at the start of the procedure, all representatives q such that $t_{\mathrm{del}}(q) < t_{\min}$, where $t_{\min} = \min_{a \in A^\gamma} t_{\mathrm{del}}(a)$ is the minimum deletion time of an attractor, are removed from R^γ. These removals are justified by the following argument. If $|A^\gamma| = k + 1$, then these $k + 1$ points, which are at distance greater than 2γ from one another, provide a certificate that no solution of radius $\leq \gamma$ exists for any $X_{t'}$ with $t \leq t' < t_{\min}$, hence γ will not be a useful guess until $t_{\min}$, and there is no need to keep points that expire in this time interval.

Algorithm 5: UPDATE$_{6+\varepsilon}(p)$

foreach $\gamma \in \Gamma$ **do**
  Remove points x of A^γ and R^γ with $t_{\mathrm{del}}(x) \le t_{\mathrm{arr}}(p)$
  $E \leftarrow \{a \in A^\gamma : d(a,p) \le 2\gamma\}$
  if $E = \emptyset$ **then**
    $A^\gamma \leftarrow A^\gamma \cup \{p\}$
    $rep^\gamma(p) \leftarrow p$; $R^\gamma \leftarrow R^\gamma \cup \{p\}$
    CLEANUP(p, γ)
  else
    $E' \leftarrow \{a \in E \ : \ t_{\mathrm{del}}(rep^\gamma(a)) < t_{\mathrm{del}}(p)\}$
    if $E' \neq \emptyset$ **then**
      $a \leftarrow$ arbitrary attractor in E'
      $R^\gamma \leftarrow R^\gamma \setminus \{rep^\gamma(a)\}$
      $rep^\gamma(a) \leftarrow p$; $R^\gamma \leftarrow R^\gamma \cup \{p\}$

Algorithm 6: CLEANUP(p, γ)

if $|A^\gamma| = k + 2$ **then**
  $a_{\min} \leftarrow \mathrm{argmin}_{a \in A^\gamma} t_{\mathrm{del}}(a)$;
  Remove $a_{\min}$ from A^γ;
if $|A^\gamma| = k + 1$ **then**
  $t_{\min} \leftarrow \min_{a \in A^\gamma} t_{\mathrm{del}}(a)$;
  Remove all points q with $t_{\mathrm{del}}(q) < t_{\min}$ from R^γ

Query procedure At any time t, to obtain a solution to k-center for the active set X_t, procedure QUERY$_{6+\varepsilon}(t)$ is invoked (see Algorithm 8 in the appendix for the pseudocode). First, for each $\gamma \in \Gamma$, all points p with $t_{\mathrm{del}}(p) \le t$ are removed from A^γ and R^γ. Then, the procedure searches the minimum guess $\gamma \in \Gamma$ such that $|A^\gamma| \le k$ and a 2γ-covering C^γ of R^γ of size at most k can be found using the simple greedy strategy by Hochbaum and Shmoys [23], and this set C^γ is returned as the solution.

4.1 Approximation

In what follows, we denote with superscript t the states of data structures either after the call of UPDATE$_{6+\varepsilon}(p)$, with $t_{\mathrm{arr}}(p) = t$, or after the execution of the first for loop in QUERY$_{6+\varepsilon}(t)$. To prove the approximation ratio, we first prove that the algorithm maintains the following invariant.

Lemma 4.1. *After any execution of* UPDATE$_{6+\varepsilon}(p)$*, the following properties hold for each* $\gamma \in \Gamma$*, and* $t = t_{\mathrm{arr}}(p)$:

1. *If* $|A^{\gamma,t}| \le k$*, then* $\forall x \in X_t$ *we have* $d(x, R^{\gamma,t}) \le 4\gamma$.
2. *If* $|A^{\gamma,t}| = k + 1$*, then* $\forall x \in X_t$ *such that* $t_{\mathrm{del}}(x) \ge \min_{a \in A^{\gamma,t}} t_{\mathrm{del}}(a)$*, we have* $d(x, R^{\gamma,t}) \le 4\gamma$.

At a high level, the invariant states that if there are at most k attractors, then any active point is at distance at most 4γ from a representative. If however there are $k + 1$ attractors, which implies that γ is too small to be the correct guess, then only the points with deletion time later than the earliest deletion time of an attractor are guaranteed to be at distance at most 4γ from a representative.

Based on the invariants, we can prove the approximation ratio of the solutions returned by QUERY$_{6+\varepsilon}$.

Theorem 4.2. *For any time* t*, the solution returned by* QUERY$_{6+\varepsilon}(t)$ *is a* $(6 + \varepsilon)$*-approximation for the* k*-center problem on the active set* X_t.

4.2 Running time and memory complexity

We now analyze the number of points maintained by the data structure employed by our algorithm, which directly influences the time and memory requirements. The analysis is made parametric with respect to a specific notion of ordering of the input dataset X, which is defined below and it is a crucial novel ingredient introduced by our paper.

Definition 4.3. *For an integer parameter* $H \ge 0$*, we say that the set* X *is* H-ordered *if for every pair of points* $p, q \in X$ *such that* $t_{\mathrm{arr}}(p) < t_{\mathrm{arr}}(q)$ *and at least* H *points arrive between* p *and* q*, i.e.,* $|\{x \in X : \ t_{\mathrm{arr}}(p) < t_{\mathrm{arr}}(x) < t_{\mathrm{arr}}(q)\}| \ge H$*, we have that* $t_{\mathrm{del}}(p) < t_{\mathrm{del}}(q)$.

Intuitively, the above notion provides a quantitative characterization of how arbitrary the points' arrival and departure times are, and a low value of H means that points are deleted roughly in the order in which they arrive. For example, in the sliding-window setting, or in general when points

are deleted in the order they arrive, the pointsets are 0-ordered. More generally, any first-in-first-out (FIFO) policy is 0-ordered, and hence this class strictly contains the sliding-window setting.

The assumption of H-ordered sets is realistic. A concrete example is provided by the aforementioned TLS certificates recorded in Certificate Transparency logs [13, 28]. These certificates are valid for 160 hours since issuance and have therefore a set expiration time, and they are usually recorded in a Certificate Transparency log shortly after issuance. Taking the logging time as arrival, we then have that their deletion order closely follows their arrival order, with a few inversions due to uneven logging delay, which yields a stream with low but nonzero H. As another example, consider a stream with batch processing (e.g., buffers in TCP packet routers, or intermediate storage for industrial manufacturing). Items arrive in sequence and are grouped into batches of size H, with each batch being processed as a unit. While the deletion order of items within a batch may change, batches themselves are served in arrival order, resulting in an H-ordered set.

Importantly, the algorithm does not require the knowledge of H, which is only used by the analysis to bound the size of the data structure. Indeed, intuitively, the attractors and their active representatives can be at most $O(k)$, so the only points that could grow up to $|X_t|$ are orphaned representatives. Under H-ordered sets, however, their number is bounded by $O(k + H)$. We then have the following.

Lemma 4.4. *Suppose that X is H-ordered, for some $H \geq 0$. Then, for any guess $\gamma \in \Gamma$ and time t,*

$$|A^{\gamma,t}| + |R^{\gamma,t}| = O(\min\{(k + H), |X_t|\}) .$$

The next theorems, based on Lemma 4.4, state upper bounds for the update and query running times, and for the memory complexity. Let $\mathrm{pr}(t)$ be the time of the last update/query operation before t.

Theorem 4.5. $\textsc{Update}_{6+\varepsilon}(p)$ *requires* $O\big((\log \Delta/\varepsilon) \min\{(k+H), |X_{\mathrm{pr}(t_{\mathrm{arr}}(p))}|\}\big)$ *operations.*

Theorem 4.6. $\textsc{Query}_{6+\varepsilon}(t)$ *requires* $O\big((\log \Delta/\varepsilon)k \cdot \min\{(k + H), |X_{\mathrm{pr}(t)}|\}\big)$ *operations.*

Theorem 4.7. *The maximum number of points maintained in memory when processing the pointset X is $O((\log \Delta/\varepsilon) \min\{(k + H), \max_t |X_t|\})$.*

Note that, for $H \in O(k)$, update times and the memory complexity are bounded by $O((\log \Delta/\varepsilon)k)$.

4.3 Comparison with related work

The memory and time complexities of the algorithm presented in this section depend on the tameness of the update sequences. Importantly, the memory usage never exceeds the one of the algorithm presented in Section 3, and, for small H (e.g., $H \in O(k)$), it is significantly smaller memory, while attaining comparable update time. Moreover, unlike any existing fully-dynamic algorithm in general metrics, the bound on the operations is *worst-case* rather than amortized. These gains, however, come at the cost of a (slightly) worse approximation ratio.

Our setting can be viewed as a generalization of the sliding window one. Indeed, our algorithm extends the sliding window one of Cohen-Addad et al. [12] by allowing arbitrary deletions. Despite the increased generality, we match the same memory and time complexities when $H = 0$. As discussed in Section 3.3, Blank et al. [5] present an algorithm for k-center in a setting akin the one with lifetimes, which stores $O\big((\log \Delta/\varepsilon)\, k^3\big)$ points regardless of H-ordering and is more memory-efficient than ours for large H (i.e., $H \in \Omega(k^3)$). However, their algorithm has expensive update times, and, more importantly, a worse non-constant approximation ratio, growing linearly with k.

5 Conclusions

In this paper, we explored the dynamic setting *with lifetimes*, devising constant-approximation k-center clustering algorithms. The new setting represents a trade-off between the fully dynamic and the sliding window settings, and offers several opportunities for future work. It is interesting to study variants of k-center problem (e.g., dealing with outliers [4] or minimizing recourse [26]), and to apply the new setting to other problems, including clustering with different objectives [2], graph problems [14], and machine unlearning problems [7]. Moreover, a compelling goal is to consider a more general setting where only approximate estimates of deletion times are available, linking the framework to the broader field of *algorithms with predictions* [30, 29]. Finally, whether the approximation-memory trade-off achieved by our space-efficient algorithm is optimal remains open, since no lower bound specific to the setting with lifetimes is currently known.

References

[1] MohammadHossein Bateni, Hossein Esfandiari, Hendrik Fichtenberger, Monika Henzinger, Rajesh Jayaram, Vahab Mirrokni, and Andreas Wiese. Optimal fully dynamic k-center clustering for adaptive and oblivious adversaries. In *Symposium on Discrete Algorithms (SODA)*, 2023.

[2] Sayan Bhattacharya, Martín Costa, Silvio Lattanzi, and Nikos Parotsidis. Fully dynamic k-clustering in O(k) update time. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

[3] Sayan Bhattacharya, Martin Costa, Ermiya Farokhnejad, Silvio Lattanzi, and Nikos Parotsidis. Almost optimal fully dynamic k-center clustering with recourse. In *International Conference on Machine Learning (ICML)*, 2025.

[4] Leyla Biabani, Annika Hennes, Denise La Gordt Dillie, Morteza Monemizadeh, and Melanie Schmidt. Improved guarantees for fully dynamic k-center clustering with outliers in general metric spaces. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

[5] Lotte Blank, Sergio Cabello, Mohammad Taghi Hajiaghayi, Robert Krauthgamer, Sepideh Mahabadi, André Nusser, Jeff M Phillips, and Jonas Sauer. The expiration streaming model: Diameter, k-center, counting, sampling, and friends. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 37–1, 2026.

[6] Michele Borassi, Alessandro Epasto, Silvio Lattanzi, Sergei Vassilvitskii, and Morteza Zadimoghaddam. Sliding window algorithms for k-clustering problems. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[7] Lucas Bourtoule, Varun Chandrasekaran, Christopher A Choquette-Choo, Hengrui Jia, Adelin Travers, Baiwu Zhang, David Lie, and Nicolas Papernot. Machine unlearning. In *2021 IEEE Symposium on Security and Privacy (SP)*, 2021.

[8] Jan van den Brand, Sebastian Forster, Yasamin Nazari, and Adam Polak. On dynamic graph algorithms with predictions. In *Symposium on Discrete Algorithms (SODA)*, 2024.

[9] Matteo Ceccarello, Andrea Pietracaprina, Geppino Pucci, and Francesco Visonà. Fair center clustering in sliding windows. In *International Conference on Extending Database Technology, (EDBT) 2026*, pages 196–207, 2026.

[10] T-H Hubert Chan, Silvio Lattanzi, Mauro Sozio, and Bo Wang. Fully dynamic k-center clustering with outliers. *Algorithmica*, 86(1):171–193, 2024.

[11] TH Hubert Chan, Arnaud Guerqin, and Mauro Sozio. Fully dynamic k-center clustering. In *World Wide Web Conference (WWW)*, pages 579–587, 2018.

[12] Vincent Cohen-Addad, Chris Schwiegelshohn, and Christian Sohler. Diameter and k-center in sliding windows. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 19–1, 2016.

[13] David Cooper, Stefan Santesson, Stephen Farrell, Sharon Boeyen, Russell Housley, and William Polk. Internet x. 509 public key infrastructure certificate and certificate revocation list (crl) profile, 2008.

[14] Emilio Cruciani, Sebastian Forster, Gramoz Goranci, Yasamin Nazari, and Antonis Skarlatos. Dynamic algorithms for k-center on graphs. In *Symposium on Discrete Algorithms (SODA)*, 2024.

[15] Sebastian Forster and Antonis Skarlatos. Dynamic consistent k-center clustering with optimal recourse. In *Symposium on Discrete Algorithms (SODA)*, 2025.

[16] Pawel Gajer, Rebecca M Brotman, Guoyun Bai, Joyce Sakamoto, Ursel ME Schütte, Xue Zhong, Sara SK Koenig, Li Fu, Zhanshan Ma, Xia Zhou, et al. Temporal dynamics of the human vaginal microbiota. *Science translational medicine*, 4(132):132ra52–132ra52, 2012.

[17] Jinxiang Gan and Mordecai J Golin. Fully dynamic k-center in low dimensions via approximate furthest neighbors. In *Symposium on Simplicity in Algorithms (SOSA)*, 2024.

[18] Teofilo F Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985.

[19] Gramoz Goranci, Monika Henzinger, Dariusz Leniowski, Christian Schulz, and Alexander Svozil. Fully dynamic k-center clustering in low dimensional metrics. In *Workshop on Algorithm Engineering and Experiments (ALENEX)*, 2021.

[20] Mara Grilnberger and Antonis Skarlatos. Adaptive fully dynamic k-center clustering with (near-) optimal worst-case guarantees. *arXiv preprint arXiv:2604.01726*, 2026.

[21] Christian Hennig, Marina Meila, Fionn Murtagh, and Roberto Rocci. *Handbook of cluster analysis*. CRC press, 2015.

[22] Monika Henzinger, Barna Saha, Martin P. Seybold, and Christopher Ye. On the complexity of algorithms with predictions for dynamic graph problems. In *Innovations in Theoretical Computer Science (ITCS)*, 2024.

[23] Dorit S Hochbaum and David B Shmoys. A unified approach to approximation algorithms for bottleneck problems. *Journal of the ACM (JACM)*, 33(3):533–550, 1986.

[24] Wen-Lian Hsu and George L Nemhauser. Easy and hard bottleneck location problems. *Discrete Applied Mathematics*, 1(3):209–215, 1979.

[25] Hicham Kalkha, Azeddine Khiat, Ayoub Bahnasse, and Hassan Ouajji. Enhancing warehouse efficiency with time series clustering: A hybrid storage location assignment strategy. *IEEE Access*, 12:52110–52126, 2024.

[26] Jakub Lacki, Bernhard Haeupler, Christoph Grunau, Rajesh Jayaram, and Vaclav Rozhoň. Fully dynamic consistent k-center clustering. In *Symposium on Discrete Algorithms (SODA)*, 2024.

[27] Silvio Lattanzi and Sergei Vassilvitskii. Consistent k-clustering. In *International Conference on Machine Learning (ICML)*, 2017.

[28] Ben Laurie, Eran Messeri, and Rob Stradling. Certificate transparency version 2.0, 2021.

[29] Quanquan C. Liu and Vaidehi Srinivas. The predicted-updates dynamic model: Offline, incremental, and decremental to fully dynamic transformations. In *Conference on Learning Theory (COLT)*, 2024.

[30] Michael Mitzenmacher and Sergei Vassilvitskii. Algorithms with predictions. *Communications of the ACM*, 65(7):33–35, 2022.

[31] Donn Morrison, Ian McLoughlin, Alice Hogan, and Conor Hayes. Evolutionary clustering and analysis of user behaviour in online forums. In *Proceedings of the International AAAI Conference on Web and Social Media*, 2012.

[32] Paolo Pellizzoni, Andrea Pietracaprina, and Geppino Pucci. Adaptive k-center and diameter estimation in sliding windows. *International Journal of Data Science and Analytics*, 14(2): 155–173, 2022.

[33] Paolo Pellizzoni, Andrea Pietracaprina, and Geppino Pucci. k-center clustering with outliers in sliding windows. *Algorithms*, 15(2):52, 2022.

[34] Paolo Pellizzoni, Andrea Pietracaprina, and Geppino Pucci. Fully dynamic clustering and diversity maximization in doubling metrics. In *Algorithms and Data Structures Symposium*, 2023.

[35] Binghui Peng and Aviad Rubinstein. Fully-dynamic-to-incremental reductions with known deletion order (e.g. sliding window). In *Symposium on Simplicity in Algorithms (SOSA)*, 2023.

[36] Mukundan Srinivasan and Young B Moon. A comprehensive clustering algorithm for strategic analysis of supply chain networks. *Computers & industrial engineering*, 36(3):615–633, 1999.

[37] Yanhao Wang, Francesco Fabbri, Michael Mathioudakis, and Jia Li. Fair max–min diversity maximization in streaming and sliding-window models. *Entropy*, 25(7):1066, 2023.

Appendix

A Comparison with settings akin to the one *with lifetimes*

Dynamic settings akin to the one *with lifetimes* introduced here have recently been proposed in the literature.

Peng and Rubinstein [35] address the *deletions-look-ahead* dynamic model, where the order of deletions of elements of the active set is known to the algorithm, like in the dynamic setting with lifetimes. They show that any incremental algorithm with worst-case update time T_u and query time T_q can be turned into an algorithm that can support both insertions and deletions in the deletions-look-ahead model with $O(T_u \log |X|)$ amortized update time, $O(T_q)$ query time, and a $\Theta(\log |X|)$ memory overhead. Crucially, the total number of elements that are inserted or deleted from the active set, i.e., $|X|$, can be exponentially bigger than the number of active elements at any point in time, i.e., $\max_t |X_t|$. Because of this, the proposed technique would fail the requirement of the dynamic setting with lifetimes requiring update times sublinear in $|X_t|$.

The recent paper of [5] introduces the *general expiration streaming model*, which generalizes the sliding window model by allowing for arbitrary deletion orders, yet elements are endowed with their deletion time, like in the dynamic setting with lifetimes. Since this model stems from the streaming one, the focus seems to be primarily on sublinear memory usage. For a comparison with their algorithm, see Section 4.3.

Finally, several settings have been introduced to inform dynamic algorithms with predicted or uncertain (rather than exact) deletion times of elements, inserting these works in the broader *algorithms with predictions* framework [30]. Among these, Liu and Srinivas [29] utilizes the *predicted-update dynamic model*, extending the setting of Peng and Rubinstein [35] to allow for predicted deletion times, and show that also under this setting some incremental or offline algorithms can be turned into fully-dynamic ones, with $O(\mathrm{polylog}|X|)$ overhead. Moreover, both Brand et al. [8] and Henzinger et al. [22] endow fully dynamic algorithms with predicted deletion times of points, in a setting akin to the predicted-update dynamic model of Liu and Srinivas [29], but focus on graph problems.

B Limitations and future work

In this paper, we developed two algorithms for k-center in the newly introduced dynamic setting *with lifetimes*. While the introduction of this setting and the new algorithms offer several advantages over algorithms for the fully dynamic or the sliding window setting (in terms of efficiency and generality, respectively), we envision that future work can address several limitations of the present work.

First, developing lower bounds on the trade-offs between approximation guarantees and time complexities for k-center in the dynamic setting with lifetimes, and understanding how they differ from the ones in the fully dynamic setting (e.g., the ones presented in Bateni et al. [1]), would be a major step forward for the field. Additionally, it would be interesting to understand how these lower bounds change under the requirement of sublinear working memory (i.e., making the dynamic setting with lifetimes analogous to the general expiration streaming model [5]), and how they relate to the ones in the sliding window model [12].

Second, it would be interesting to relax the assumptions of the algorithms we developed, for example by removing the need for the knowledge of the aspect ratio Δ of the metric space, or by allowing for inexact deletion times, which would bridge the dynamic setting with lifetimes to the algorithm with predictions literature [30, 8, 22].

Moreover, as for the fully dynamic setting, the dynamic setting with lifetimes allows for a definition of *recourse*, defined as the number of changes (i.e., $|C_1 \triangle C_2|$) made to the cluster centers after each point insertion or deletion [27]. Both the algorithms we present here have trivial $O(k)$ recourse, and developing algorithms with $o(k)$ recourse is an interesting research direction. In the fully dynamic setting, Forster and Skarlatos [15] developed an algorithm that yields a 50-approximation with optimal recourse of 2 and $\widetilde{O}(\mathrm{poly}(|X|))$ update time, while Bhattacharya et al. [3] developed an algorithm that yields a 20-approximation with $O(1)$ recourse and $\widetilde{O}(k)$ update time. Understanding how these trade-offs can be improved in the dynamic setting with lifetimes would be a major development.

Finally, several extensions to the k-center problem can be tackled in the dynamic setting with lifetimes. For example, the algorithms could be extended to deal with variants of the problem, including dealing with outliers [4], diversity maximization [37], and fairness constraints [9]. Moreover, techniques for low-dimensional metric spaces could be used to improve the approximation guarantees under memory constraints [32] or to obtain $O(1)$ update times [17, 34].

C Additional pseudocodes

We provide here, for the sake of completeness, the pseudocodes for the query procedures. In particular, Algorithm 7 reports the query procedure for the algorithm presented in Section 3.

Algorithm 7: $\textsc{Query}_{2+\varepsilon}(t)$

$\gamma^* \leftarrow \min\{\gamma \in \Gamma \ : \ U^\gamma = \emptyset\}$;
return C^{γ^*}

Moreover, Algorithm 8 reports the query procedure for the algorithm presented in Section 4.

Algorithm 8: $\textsc{Query}_{6+\varepsilon}(t)$

foreach $\gamma \in \Gamma$ **do**
  Remove points x of A^γ and R^γ with $t_{\text{del}}(x) \leq t$
for *increasing values of* $\gamma \in \Gamma$ *such that* $|A^\gamma| \leq k$ **do**
  $C \leftarrow \emptyset$;
  foreach $q \in R^\gamma$ **do**
    if $C = \emptyset$ *or* $d(q, C) > 2\gamma$ **then** $C \leftarrow C \cup \{q\}$;
    if $|C| > k$ **then** go to next γ;
  return C

D Omitted Proofs

D.1 Proofs for Section 3

Proof of Remark 3.1. We describe an adversarial point set for a given guess $\gamma \in \Gamma$. At time $t = 1$, we insert a point p_1 with lifetime such that $t_{\text{del}}(p_1) = t_{\text{arr}}(p_1) + n + 1$.

At time $t = 2, \ldots, n$, we insert $n - 1$ points $p_2, \ldots, p_n$, each with lifetime such that $t_{\text{del}}(p) = t_{\text{arr}}(p) + 2n - 1$, and such that each p_i with $i > 1$ is at distance $3/2\gamma$ from p_1 (and, e.g., at distances $\gamma/10$ between each other). Clearly, all these points are assigned to cluster D_1, with the first point p_1 being the center. Then, at time $t = n + 1, \ldots, 2n$, we insert n additional points $p_{n+1}, \ldots, p_{2n}$, each with lifetime such that $t_{\text{del}}(p_i) = t_{\text{arr}}(p_i) + 2$ and such that each such p_i is at distance $5/2\gamma$ from p_1, at distance $5/2\gamma$ from the other points p_j with $n < j \leq 2n$, and at distance $3/2\gamma$ from points $p_2, \ldots, p_n$.

The first n calls of $\textsc{Update}_{2+\varepsilon}$ involve no deletions, so they take $O(k)$ time each. The call at time $n + 1$ inserts p_{n+1} as a new center, since it is at distance $> 2\gamma$ from p_1, in $O(k)$ time. However, the call at time $n + 2$ first involves the deletion of p_1 and moving $p_2, \ldots, p_n$ to the cluster with center p_{n+1}, and then inserts p_{n+2} as a new center. This takes $\Theta(n)$ time. Each call at time $t = n + 3, \ldots, 2n$ involves the deletion of point p_{t-2}, moving $p_2, \ldots, p_n$ to the cluster with center p_{t-1} and the insertion of p_t as a new center, which takes $\Theta(n)$. Therefore, the algorithm performs a total of $\Theta(n^2)$ operations. □

Proof of Lemma 3.2. At step $t = 0$, the property holds by vacuity. Suppose inductively that at step $t \geq 0$, the invariant holds for all points in X_t. We recall that the superscript (e.g. $U^{\gamma,t}$) denotes the state of a set *after* the t-th insert or delete operation. When the superscript is missing, it denotes a state during the execution of the operations. At time step $t + 1$:

- If INSERT(p, γ) is called, then all points in X_t are left untouched. Moreover, the invariant for p holds by construction.
- If DELETE(p, γ) is called and p is not a center, then all points in $X_t \setminus \{p\}$ are left untouched. Let instead p be a center $c_i^{\gamma,t}$. Then, for each $x \notin D_i^{\gamma,t} \cup U^{\gamma,t}$, note that the center and cluster index renaming does not affect the invariant. For each $x \in D_i^{\gamma,t}$, either x is inserted in D_h with $h = \operatorname{argmin}_{j=i+1,\dots\ell}\{c_j \in C\colon\ d(x, c_j) \leq 2\gamma\}$ (possibly by creating new clusters), or it is inserted in U if no center at distance $\leq 2\gamma$ is found. In the first case, note that $\min_{j=1,\dots,i-1} d(x, c_j) > 2\gamma$, otherwise by the inductive hypothesis x would not belong to D_i. Therefore, we have that $h = \operatorname{argmin}_{j=1,\dots\ell}\{c_j \in C\colon\ d(x, c_j) \leq 2\gamma\}$, thus fulfilling the invariant.

  Finally, let $U \neq \emptyset$ and consider a $x \in U$, either because it belonged in $U^{\gamma,t}$ or because it was inserted in U on line 13. In this case, one point u will be selected as a new center, and thus $|C| = k$. We have that if $d(x, C^{\gamma,t+1}) > 2\gamma$, then we have $d(x, u) > 2\gamma$ and therefore x remains in U, satisfying the first case of the invariant. Otherwise, since $x \in U$ implies that $d(x, c_i) > 2\gamma\ \forall i = 1, \dots, \ell - 1$, we have that $d(x, u) \leq 2\gamma$, and thus x is inserted in D_ℓ, thus satisfying the second case of the invariant.

Finally, after a call to INSERT or DELETE, the RECLUSTERING(γ) procedure might be called. In that case, we note that each $x \in \bigcup_{j=1}^{i-1} D_j^{\gamma,t}$ is left untouched, and the invariant is still valid. On the other hand, if $x \in U'$ after line 4, then it could not be inserted in any of the clusters D_j for $j = 1, \dots, i-1$, by the inductive hypothesis. Moreover, by construction it will be inserted in the cluster D_h with $h = \operatorname{argmin}_{j=i,\dots\ell}\{c_j \in C\colon\ d(x, c_j) \leq 2\gamma\}$, as a point can be inserted in a cluster only if it is not inserted in a cluster with a lower index. At the end of the loop (i.e., $|C| = k$), any point x such that $d(x, C) > 2\gamma$ is left in U.

We therefore have the claim by induction. □

Lemma D.1. *After a call of* RECLUSTERING(γ) *every point that was reclustered, i.e., every point belonging to the set U' in $\bigcup_{j=i}^{\ell^\gamma} D_j^\gamma \cup U^\gamma$ before the reclustering, becomes vanishing for γ.*

Proof of Lemma D.1. Since at each step j the new center c_j gets chosen from all remaining points as the one with maximal deletion time, any point x that gets inserted into D_j must have deletion time $t_{\text{del}}(x) < t_{\text{del}}(c_j)$, and is therefore vanishing. If a point is left in U, then it is vanishing by definition. □

Proof of Lemma 3.4. We will show that the vanishing property (for a fixed guess γ) is invariant under the call of INSERT, DELETE or RECLUSTERING algorithms, unless $t = t_{\text{del}}(x)$ and the point is thus removed. The property holds at $t = 0$ by vacuity and the proof follows by induction.

Fix some $x \in X$ which is vanishing at time t.

- **Insertion**: If x is vanishing then it is already inserted, so any insertion operation will regard a new point and cannot modify either x or its center. Then, x will remain vanishing after any insertion operation.
- **Deletion**: Suppose that we call DELETE(p, γ). If $p = x$ the current time is $t_{\text{del}}(x)$ and the proposition holds. Furthermore, if $p \notin C^\gamma$ (i.e. p is not a center) its deletion will only alter p itself and no other point. Therefore, we can assume $p \neq x$ and $\exists i$ s.t. $p = c_i$. We then have 3 cases:
  1. $x \in D_i$. This would mean that the center of x has a deletion time smaller than that of x, which is in contradiction to the fact that x is vanishing.
  2. $x \in D_j$, for some $j \neq i$. In the case that $p = c_i$ the deletion algorithm affects clusters with index different from i by possibly adding more points to them (Line 8) and by relabeling their indexes (Line 16). Therefore, no changes are made to x or to its center c_j, and thus x remains vanishing.

3. $x \in U$. In this case, x remains in U or is inserted in D_l (Line 23). Either way, it remains vanishing, since we picked $c_l = u$ s.t. $t_{\text{del}}(u)$ is maximal among all points in U.

- **Reclustering**: Suppose that we call RECLUSTERING(i, γ). If $\exists j < i$ such that $x \in D_j$, then the reclustering operation does not affect D_j in any way, meaning that x remains vanishing. The other case is covered by Lemma D.1, and thus x remains vanishing.

□

Proof of Proposition 3.5. The main for loop of Algorithm 4 is repeated $O(k)$ times. In each iteration we need to scan U' linearly to first find the element with maximal t_{del} and then to check whether some other point in U' is close to it. So the main loop is executed in $O(k \cdot |U'|)$. The other parts of the algorithm can be executed in $O(|U'|)$, for a total running time of $O(k \cdot |U'|)$. Then:

$$k|U'| = k\left|U \cup \bigcup_{j=i}^{k} D_j\right| \leq k\left(|U| + \sum_{j=i}^{k}(|P_j| + |V_j|)\right) \leq 2k\sum_{j=i}^{k}|P_j|,$$

where the last inequality is guaranteed by the fact that Property (2) holds for i when calling the reclustering.

This means that the number of operations is $O\left(k \cdot \sum_{j=i}^{l}|P_j|\right)$ and thus bounded by the number of persistent points involved in the reclustering. Given that by Lemma D.1 all persistent points in $P_{j \geq i}$ will become vanishing and that by Lemma 3.4 every vanishing point remains vanishing, the number of persistent points in all reclusterings is bounded by n and thus the total number of operations made by all calls of RECLUSTERING(i, γ) is $O(n \cdot k)$ □

Proof of Lemma 3.7. We will show that for $c \neq x$ to satisfy the two properties implies the fact that when c was last picked as a center x was either not inserted or persistent. Suppose by contradiction that, when c was last picked as a center, x was vanishing. We have 3 possibilities:

1. c became a center for the last time at time t' during an insertion operation (Line 5) or during a re-assignment of persistent points after a deletion (Line 12). In both of these cases $c_{ID}^{\gamma,t'}(c) > c_{ID}^{\gamma,t'}(x)$ as c becomes the center of the cluster with the highest index. Given that we are assuming that t' is the last time that c was chosen as a center, there cannot be any reclustering involving c after time t'. Furthermore, if no reclustering is made, clusters stay in their relative order and point x will remain inside its cluster until its deletion. So the fact that $c_{ID}^{\gamma,t'}(c) \geq c_{ID}^{\gamma,t'}(x)$ is in contradiction with $c_{ID}^{\gamma,t}(c) \leq c_{ID}^{\gamma,t}(x)$.

2. c became a center for the last time after a deletion. If $x \notin U$ the analysis is akin to the one in the first case. If $x \in U$ we know that $t_{\text{del}}(x) < t_{\text{del}}(c)$, thus at time t x will already be deleted and therefore cannot be vanishing.

3. c became a center for the last time during the execution of RECLUSTERING(i, γ). If $x \in D_j$ s.t. $j < i$ the analysis is akin to the one in the first case. Let instead $x \in U'$ (same U' as in Algorithm 4). If x ends up in a cluster with smaller index than the one of c, then we break the assumption that $c_{ID}^{\gamma,t}(c) \leq c_{ID}^{\gamma,t}(x)$. If instead it ends up in a cluster with index greater or equal to the one of c, then we must have $t_{\text{del}}(x) < t_{\text{del}}(c)$, and thus x cannot be vanishing at $t = t_{\text{del}}(c) - 1$ because it has been already deleted.

Given that by Lemma 3.4 a vanishing point remains vanishing, calling t' the time at which x becomes vanishing, every center c which satisfies the two properties of the Lemma must be picked at time $< t'$ and carried on up to a time $\geq t'$. Given that at most k centers can coexist simultaneously, the maximum number of centers $c \neq x$ satisfying the properties is k. Accounting for the possibility that $c = x$, we obtain at most $k + 1$ centers. □

Proof of Theorem 3.9. By implementing the priority queue Q as a heap, each of the $\Theta(n)$ insertions and $O(n)$ remove-min operations in Q costs $O(\log |X_t|)$, so their overall cost is $O(n \cdot \max_t \log |X_t|)$.

By Propositions 3.6 and 3.8, for each $\gamma \in \Gamma$, we have that the overall costs of all calls INSERT(p, γ) and DELETE(p, γ) is $O(nk)$. Since $|\Gamma| \in O\big(\log_{1+\varepsilon} \Delta\big) = O(\log \Delta/\varepsilon)$, the theorem follows. □

Proof of Theorem 3.10. Note that we iterate through the guesses $\gamma \in \Gamma$, and for each guess we check in constant time the size of C^γ. This takes $O(|\Gamma|) = O(\log \Delta/\varepsilon)$ time. Finally, returning the solution takes $O(|C^{\gamma^*}|) = O(k)$ time. □

Proof of Theorem 3.11. Note that, by construction, after the call of UPDATE(p, t), only points of X_t are maintained in memory. Moreover, for a fixed guess γ, any such point $p \in X_t$ can belong, by construction, to only one of the sets $D_i^\gamma, i = 1, \ldots, \ell^\gamma$ or U^γ. We therefore have that $\sum_{i=1}^{\ell^\gamma} |D_i^\gamma| + |U^\gamma| \leq |X_t|$. Moreover, clearly $|C^\gamma| \leq |X_t|$ as the centers are all distinct. Since we have $|\Gamma| \in O(\log \Delta/\varepsilon)$, the data structure maintains at most $O((\log \Delta/\varepsilon)|X_t|)$ points. Moreover, the priority queue Q contains each point of the active set exactly once, for a total of $|X_t|$ points. Finally, until the following call of UPDATE, the number of points stored in the data structures can only decrease (i.e., if QUERY calls DELETE), thus concluding the proof. □

D.2 Proofs for Section 4

Proof of Lemma 4.1. Let us focus on an arbitrary guess γ. We say that a point $q \in X_t$ is *relevant for γ and t* if $|A^{\gamma,t}| \leq k$ or $|A^{\gamma,t}| > k$ and $t_{\mathrm{del}}(q) \geq \min_{a \in A^{\gamma,t}} t_{\mathrm{del}}(a)$. Clearly, the proof concerns only relevant points. Also, for every point $q \in X_t$, we let $\psi^\gamma(q)$ denote the attractor that q is attracted to. If q is selected as a representative by UPDATE$_{6+\varepsilon}(q)$, we let $\psi^\gamma(q)$ be such that $rep^\gamma(\psi^\gamma(q)) = q$. Otherwise, we let $\psi^\gamma(q)$ be any attractor belonging to the set E during the execution of UPDATE$_{6+\varepsilon}(q)$. Clearly, $d(q, \psi^\gamma(q)) \leq 2\gamma$. We will argue that for every point $q \in X_t$ which is relevant for γ and t, there exists a point $r \in R^{\gamma,t}$ such that $\psi^\gamma(r) = \psi^\gamma(q)$.

If q is the point arrived at time t (i.e., $q = p$ and $t_{\mathrm{arr}}(q) = t$) then we set $r = rep^{\gamma,t}(\psi^\gamma(q))$, which is possibly q itself. If q arrived at an earlier time, it is immediate to argue that since q is relevant at time t, it has also been relevant at all times t', with $t_{\mathrm{arr}}(q) \leq t' < t$, when a point arrived or a query was invoked. Set $r_0 = rep^{\gamma,t_{\mathrm{arr}}(q)}(\psi^\gamma(q))$. We have that $\psi^\gamma(r_0) = \psi^\gamma(q)$. Also, it holds that $t_{\mathrm{del}}(r_0) \geq t_{\mathrm{del}}(q)$, therefore r_0 is also relevant at all times t', with $t_{\mathrm{arr}}(q) \leq t' < t$. Being relevant, r_0 has never been removed from $R^{\gamma,t'}$ during some invocation of CLEANUP, which only removes non-relevant points. Consequently, since r_0 is in $R^{\gamma,t_{\mathrm{arr}}(q)}$, either r_0 is still in $R^{\gamma,t}$ (and we set $r = r_0$), or it has been expunged from some set $R^{\gamma,t_{\mathrm{arr}}(r_1)}$, due to the arrival of some longer-lived representative r_1 at time $t_{\mathrm{arr}}(r_1) > t_{\mathrm{arr}}(q)$. Observe that $\psi^\gamma(r_1) = \psi^\gamma(r_0)$, and that r_1 is also relevant at time t. If $r_1 \in R^{\gamma,t}$, we set $r = r_1$, otherwise, we apply the same reasoning to r_1. Iterating the argument, we determine a sequence of relevant points $r_0, r_1, \ldots, r_h$, with $t_{\mathrm{arr}}(r_0) \leq t_{\mathrm{arr}}(q) < t_{\mathrm{arr}}(r_1) \ldots < t_{\mathrm{arr}}(r_h)$ and $\psi^\gamma(q) = \psi^\gamma(r_0) = \ldots = \psi^\gamma(r_h)$, with $r_h \in R^{\gamma,t}$. We then set $r = r_h$. The lemma follows immediately since $\psi^\gamma(r) = \psi^\gamma(q)$ implies $d(q, r) \leq 4\gamma$. □

Proof of Theorem 4.2. Let $\gamma = (1+\beta)^i \in \Gamma$ be the guess such that the returned solution C is computed from $R^{\gamma,t}$, for some $\lfloor \log_{1+\beta} d_{\min} \rfloor \leq i \leq \lceil \log_{1+\beta} d_{\max} \rceil$. By construction, $|A^{\gamma,t}| \leq k$, and, for each $p \in R^{\gamma,t}$, $d(p, C) \leq 2\gamma$. Using Lemma 4.1 and the triangle inequality, we have that for each $p \in X_t$, $d(p, C) \leq 6\gamma$. Now, if $i = \lfloor \log_{1+\beta} d_{\min} \rfloor$ then $\gamma \leq r_k^*(X_t)$ and the theorem holds. Instead, if $i > \lfloor \log_{1+\beta} d_{\min} \rfloor$, we have, by construction, that for $\gamma' = (1+\beta)^{i-1} = \gamma/(1+\beta)$ either $|A^{\gamma',t}| = k+1$ or $k+1$ points of $R^{\gamma',t}$ have been found at distance $> 2\gamma'$ from one another. From what we observed earlier, in either case we have $\gamma' < r_k^*(X_t)$, hence $\gamma < (1+\beta) r_k^*(X_t)$. Thus, for each $p \in X_t$, $d(p, C) \leq 6\gamma < 6(1+\beta) r_k^*(X_t) \leq (6+\varepsilon) r_k^*(X_t)$, and the theorem follows. □

Proof of Lemma 4.4. Let us fix arbitrary γ and t, and observe that, straightforwardly, $|A^{\gamma,t}| + |R^{\gamma,t}| \leq 2|X_t|$. We will now show that $|A^{\gamma,t}| + |R^{\gamma,t}| = O(k + H)$. By construction, we know $|A^{\gamma,t}| \leq k+1$, so we only have to show that $|R^{\gamma,t}| = O(k + H)$. Let $A^\gamma_{\mathrm{all}}(t)$ denote the set of all attractors ever arrived up to time t. If $A^\gamma_{\mathrm{all}}(t) \setminus A^{\gamma,t} = \emptyset$, then no attractor has been removed, and hence $R^{\gamma,t}_{\mathrm{orph}} = \emptyset$. Therefore, $|R^{\gamma,t}| = |A^{\gamma,t}| \leq k+1$, and the claim follows. Otherwise, let q be the attractor with largest arrival time among those not in $A^{\gamma,t}$, namely, $q = \mathrm{argmax}_{a \in A^\gamma_{\mathrm{all}}(t) \setminus A^{\gamma,t}} t_{\mathrm{arr}}(a)$. Hence, q arrived at some time $t_{\mathrm{arr}}(q) < t$ and was removed from A^γ at some time t', with $t_{\mathrm{arr}}(q) \leq t' < t$,

either by Line 2 of the UPDATE procedure, because it expired, or by Line 3 of the CLEANUP procedure. In the time interval $(t_{\rm arr}(q), t]$ no more than $k+1$ new attractors arrived since, otherwise, one of the attractors arrived in $(t_{\rm arr}(q), t]$ would not belong to $A^{\gamma,t}$, because A^γ can never contain more than $k+1$ points, and this would contradict the choice of q. For a point $p \in R^{\gamma,t}$, let $\psi^\gamma(p)$ denote the attractor for which p is representative. We partition $R^{\gamma,t}$ into 3 subsets:

- $R^{\gamma,t}(1) = \{p \in R^{\gamma,t} : t_{\rm arr}(\psi^\gamma(p)) \in (t_{\rm arr}(q), t]\}$
- $R^{\gamma,t}(2) = \{p \in R^{\gamma,t} : \psi^\gamma(p) \in A^{\gamma,t_{\rm arr}(q)}\}$
- $R^{\gamma,t}(3) = \{p \in R^{\gamma,t} : \psi^\gamma(p) \notin A^{\gamma,t_{\rm arr}(q)} \text{ and } t_{\rm arr}(\psi^\gamma(p)) \leq t_{\rm arr}(q)\}$

It is immediate to see that $R^{\gamma,t} = R^{\gamma,t}(1) \cup R^{\gamma,t}(2) \cup R^{\gamma,t}(3)$. Moreover, $|R^{\gamma,t}(1)| \leq k+1$ from what we observed before, and $|R^{\gamma,t}(2)| \leq k+1$ since $|A^{\gamma,t_{\rm arr}(q)}| \leq k+1$ by construction. We now argue that $|R^{\gamma,t}(3)| = O(H)$. First note that, since arrival times are distinct, at most one point of $R^{\gamma,t}(3)$ can have $t_{arr}(\psi^\gamma(p)) = t_{arr}(q)$; all remaining points have attractors that entered and left A^γ before $t_{arr}(q)$, and thus arrived prior to $t_{arr}(q)$. Let p be the point of $R^{\gamma,t}(3)$ with earliest arrival time. So, all other points of $R^{\gamma,t}(3)$ must have arrived in the open interval $(t_{\rm arr}(p), t_{\rm arr}(q))$. Thus, to show that $|R^{\gamma,t}(3)| = O(H)$ it is sufficient to argue that less than H points arrived in $(t_{\rm arr}(p), t_{\rm arr}(q))$. By contradiction, suppose that this is not the case. Then, the assumption that X is H-ordered implies that $t_{\rm del}(p) < t_{\rm del}(q)$. Since q was removed from A^γ at time $t' < t$, we have two cases:

- q expired at t' and was removed at Line 2 of the UPDATE procedure. Hence, $t_{\rm del}(p) < t_{\rm del}(q) = t' < t$, and p cannot be in $R^{\gamma,t}$.
- q was removed from A^γ at time t' at Line 3 of the CLEANUP procedure, since q had minimum deletion time among $k+2$ attractors. Then, right after the removal of q from A^γ, $t_{\rm min}$ was set to a value $\geq t_{\rm del}(q)$ and all representatives r with $t_{\rm del}(r) < t_{\rm del}(q) \leq t_{\rm min}$ were removed. Therefore, since $t_{\rm del}(p) < t_{\rm del}(q)$, then p would have been removed as well.

In either case, p could not be in $R^{\gamma,t}$, which contradicts the choice of p. □

Proof of Theorem 4.5. Consider a $\gamma \in \Gamma$. The removal of expired points performed in Line 2, can be done with a linear scan of in $O(|A^{\gamma,{\rm pr}(t_{\rm arr}(p))}| + |R^{\gamma,{\rm pr}(t_{\rm arr}(p))}|)$, as well as creating the set E, and potentially the set E'. Adding points to A^γ and R^γ, setting $rep^\gamma(p)$ to p, and selecting an arbitrary point of E' can be done in constant time. Finally, procedure CLEANUP can be executed in $O(|A^{\gamma,{\rm pr}(t_{\rm arr}(p))}| + |R^{\gamma,{\rm pr}(t_{\rm arr}(p))}|)$ time. Recall that, from Lemma 4.4, we have that $|A^{\gamma,{\rm pr}(t_{\rm arr}(p))}| + |R^{\gamma,{\rm pr}(t_{\rm arr}(p))}| \in O(\min\{(k+H), |X_{{\rm pr}(t_{\rm arr}(p))}|)\}$. Since, $|\Gamma| \in O(\log \Delta/\varepsilon)$, we have the claim. □

Proof of Theorem 4.6. The removal of expired points at the beginning of the procedure $O(|A^{\gamma,{\rm pr}(t)}| + |R^{\gamma,{\rm pr}(t)}|)$, for each of the $|\Gamma| \in O(\log \Delta/\varepsilon)$ guesses. Then, for each γ, running the procedure of Hochbaum and Shmoys [23] on R^γ takes $O(k \cdot |R^\gamma|) = O\big(k \cdot |R^{\gamma,{\rm pr}(t)}|\big)$ time. Recalling that $|A^{\gamma,{\rm pr}(t)}| + |R^{\gamma,{\rm pr}(t)}| \in O\big(\min\{(k+H), |X_{{\rm pr}(t)}|\}\big)$ by Lemma 4.4 and that $|\Gamma| \in O(\log \Delta/\varepsilon)$, we have the claim. □

Proof of Theorem 4.7. The claim follows by bounding the number of points maintained for each guess γ via Lemma 4.4, and by recalling that $|\Gamma| \in O(\log \Delta/\varepsilon)$. □

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract mentions the two algorithms we propose, their complexities and (briefly) also their assumptions.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Limitations are mentioned in the conclusions and in Appendix B.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

Justification: All theoretical claims are provided with proofs (potentially in the appendix).

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [N/A]

   Justification: The paper does not include experiments.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [N/A]

Justification: The paper does not include experiments.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [N/A]

   Justification: The paper does not include experiments.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [N/A]

   Justification: The paper does not include experiments.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.

- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [N/A]

   Justification: The paper does not include experiments.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The paper conforms with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: There is no societal impact of the work performed.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper poses no such risks.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [N/A]

    Justification: The paper does not use existing assets.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.

- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: The paper does not release new assets.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: The core method development in this research does not involve LLMs as any important, original, or non-standard components.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.