# Privacy Preserving Group Nearest Neighbor Search

Yuncheng Wu*
Renmin University of China
Beijing, China
yunchengwu@ruc.edu.cn

Ke Wang
Simon Fraser University
Burnaby, Canada
wangk@cs.sfu.ca

Zhilin Zhang
Simon Fraser University
Burnaby, Canada
zhilinz@sfu.ca

Weipeng Lin
Simon Fraser University
Burnaby, Canada
weipengl@sfu.ca

Hong Chen†
Renmin University of China
Beijing, China
chong@ruc.edu.cn

Cuiping Li
Renmin University of China
Beijing, China
cuipingli@ruc.edu.cn

## ABSTRACT

Group $k$-nearest neighbor ($k$GNN) search allows a group of $n$ mobile users to jointly retrieve $k$ points from a location-based service provider (LSP) that minimizes the aggregate distance to them. We identify four protection objectives in the privacy preserving $k$GNN search: (i) every user's location should be protected from LSP; (ii) the group's query and the query answer should be protected from LSP; (iii) LSP's private database information should be protected from users, i.e., the users cannot learn more information beyond the answer they requested; (iv) every user's location should be protected from the other users in the group.

We propose the first approach to meet the four privacy goals in the $k$GNN query. Our approach provides an accurate query answer and does not rely on heavy pre-computation on LSP like previous works. Our approach considers the most hostile environment that any $n-1$ users in the query group may collude to infer the location of the remaining user. Though we consider $k$GNN, the proposed privacy preserving approach can be easily adopted to any group query because it treats the query answering (i.e., $k$GNN) as a black box. Theoretical and experimental analysis suggest that our approach is highly efficient in both user computation and communication while incurring some reasonable overhead on LSP.

## 1 INTRODUCTION

The embedding of positioning capabilities (e.g., GPS) in mobile devices facilitates the emergence of location-based services (LBS), which allows the users to publish their location data for retrieving desired information from a database maintained by a location-based service provider (LSP). One typical application is the *group k-nearest neighbor* query ($k$GNN) proposed in [24], also known as aggregate nearest neighbor query [25] or aggregate similarity search [18, 19, 28]. The $k$GNN query allows a group of $n$ users to retrieve top-$k$ locations from the LSP's database to minimize some aggregate cost function $F$ over all $n$ users. Figure 1 shows that users $u_1, u_2, u_3$ ($n = 3$) jointly retrieve the top-2 meeting places $\{p_1, p_2\}$ ($k = 2$), where $p_1$ has the shortest total distance to the users, and $p_2$ has the second shortest total distance to the users. By generalizing the classic $k$-nearest neighbor ($k$NN) query from a single user in a query to multiple users in a query, the $k$GNN query offers richer semantics with broader applications in spatial databases [18, 19, 24, 25, 28].

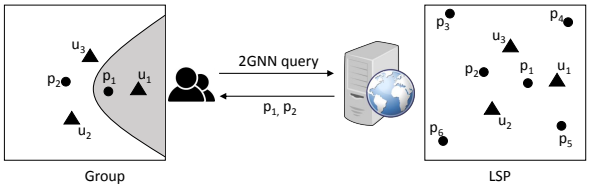



**Figure 1: A $k$GNN query ($k = 2$): $\{u_1, u_2, u_3\}$ retrieves $p_1, p_2$ as top-2 locations**

We consider four privacy concerns that arise in the $k$GNN query scenario. **Privacy I:** *every user's location privacy against LSP* since location can reveal the private information of users. **Privacy II:** *query privacy and answer privacy against LSP* because the query discloses the combination of users' locations and the relationship between users, and the query answer such as a meeting place may disclose the nature of the meeting or event. **Privacy III:** *LSP's data privacy against users.* LSP's database is the valuable and protected business asset [8, 12, 33] and the principle of least privilege [29] applies where no more information than the requested query answer should be returned to the users. Another reason for this privacy is the pay-per-result model [8, 33] where the users who pay for $k$ results should not receive more than $k$ results. **Privacy IV:** *every user's location privacy against other users* because users might not trust other users. For example, two business competitors like to query some meeting places but want to hide their own locations from each other. The most hostile environment is the *full user collusion* where $n-1$ users in the query collude to infer the location of the remaining user. In Figure 1, for example, colluding $u_2, u_3$ can infer that $u_1$ is located in the shaded area based on their own locations and the query answer $\{p_1, p_2\}$ received. If the shaded area is too small, $u_1$'s location privacy may be compromised. **Privacy IV** is required only in the case of $n > 1$.

Although many solutions, such as [1, 3, 12, 13, 17, 21, 26, 27, 30, 34, 36, 37], are proposed for the single user query (i.e., $n = 1$), only a few works addressed the group query (i.e., $n > 1$) [2, 14] but none of them achieves all four privacy concerns. Most existing work achieved **Privacy I** through returning candidate answers (e.g., [3, 13, 14, 17, 21, 26, 30]) or approximate answers (e.g., [1, 2, 34, 37]). However, returning candidate answers not only increases the communication cost but also violates **Privacy III**, while returning approximate answers degrades the answer utility as well as violates **Privacy II** since LSP knows the query answer that users obtained. [12, 27, 36] achieve **Privacy I-III** in

*This work was done when this author visited Simon Fraser University.
†Corresponding author.

the single user query case by heavily relying on pre-computing the query answers for all queries. These approaches are not applicable to the group query where the number of possible queries is large. A more detailed discussion of related work is presented in Section 9.

In this paper, we design a privacy preserving approach to the $k$GNN query for the general case of $n \geq 1$ while protecting **Privacy I-IV**. Our approach has the following novelties. *First*, it eliminates the need for pre-computing all query answers in order to address the privacy issues, while producing the exact answer. Hence, our approach can easily handle a dynamic database on LSP. *Second*, for **Privacy IV** we consider the most hostile environment, called *full user collusion*, where $n-1$ users may collude to infer the location of the remaining user. *Third*, we aim to reduce user computational cost and communication cost at a reasonable overhead on LSP, which particularly makes sense in the mobile user scenario. *Fourth*, though we consider $k$GNN, the proposed privacy preserving approach can be easily adopted to any group query because it treats the query answering (i.e., $k$GNN) as a black box. For example, our approach works with any choice of aggregate cost function $F$ and other location based group queries; to solve the privacy preserving meeting location determination (PPMLD) [5, 16, 31], we can replace the black box for $k$GNN in our approach with any existing (non-privacy preserving) meeting location determination algorithm.

The rest of the paper is structured as follows.

- Section 2 formulates the privacy preserving $k$GNN query problem, called **PPGNN**, where $n$ users jointly retrieve $k$-nearest neighbors from LSP while meeting the requirements of **Privacy I-IV**.
- Section 3 proposes a solution to PPGNN with $n = 1$. To our knowledge, this is the first work that eliminates the need for pre-computing all the query answers to achieve **Privacy I-III**.
- Section 4 proposes a solution to PPGNN with $n \geq 1$ while achieving **Privacy I-III**.
- Section 5 extends the solution to PPGNN with $n \geq 1$ to achieve **Privacy I-IV** under the full user collusion assumption. As far as we know, this is the first solution that achieves **Privacy I-IV**.
- Section 6 presents an optimization of the PPGNN solution, denoted PPGNN-OPT, to further reduce the communication cost and user computational cost.
- Section 7 theoretically analyzes the performance of the PPGNN solution and PPGNN-OPT solution.
- Section 8 presents empirical results on a real-world dataset, showing that the proposed approaches are highly communication efficient for the privacy guarantees achieved.
- Section 9 reviews the related work.

## 2 PROBLEM STATEMENT

We formally define the problem studied in this paper. Table 1 summarizes the frequently used notations. We assume that LSP owns a database of Points of Interest (POIs) where each POI has a location (e.g., latitude and longitude) and other associated information, and each user has a location. Both users and LSP are semi-honest: they follow the protocol exactly as specified, but may try to infer others' private information. Users can acquire their locations from satellites anonymously. A base station is responsible for the communication within users and between users and LSP (e.g., mobile communication provider). The base station will not collude with LSP, and there is a secure communication channel (e.g., Tor[1]) so that LSP cannot infer users' locations by IP addresses. We also assume that LSP does not collude with users. This assumption is reasonable because the penalty of collusion involving LSP is very high, including losing the trust of users and being prosecuted.

**Table 1: Summary of notations**

| Notation | Description |
|---|---|
| $n$ | the number of users in the group |
| $d$ | anonymity parameter for **Privacy I** |
| $\delta$ | anonymity parameter for **Privacy II** |
| $\theta_0$ | privacy parameter for **Privacy IV** |
| $l_{i,*}$ | real location of $u_i$ |
| $\mathbb{C}_*$ | real query $\{l_{1,*}, \cdots, l_{n,*}\}$ of the group |
| $\mathbb{L}_i$ | $u_i$'s location set |
| $l_{i,j}$ | $j$-th location in $\mathbb{L}_i$ |
| $\bar{n}, \bar{d}$ | partition parameters |
| $[\boldsymbol{v}]$ | encrypted indicator vector for $\mathbb{C}_*$ |
| $\boldsymbol{A}$ | answer matrix for all candidate queries |
| $N$ | the product of two large primes determined by $pk$ |

### 2.1 $k$GNN Query

Let $\mathbb{D} = \{p_1, \cdots, p_D\}$ be a database (owned by LSP) of $D$ POIs, and $\mathbb{C}_* = \{l_{1,*}, \cdots, l_{n,*}\}$ be the locations of $n$ users. Both $\mathbb{D}$ and $\mathbb{C}_*$ are in a metric space where a spatial distance function $dis$ is defined for any two locations, e.g., Euclidean distance [24, 25], road-network distance [38]. Let $F$ be a monotonically increasing aggregate function,

$$F(p, \mathbb{C}_*) = F(dis(p, l_{1,*}), \cdots, dis(p, l_{n,*})) \quad (1)$$

where $p$ can be any POI in $\mathbb{D}$. Commonly used $F$ includes *sum*, *max* and *min*. For example, with *sum* the query retrieves a meeting place with the minimum total distance to the users, and with *max* the query retrieves a collection place for the troops that leads to the earliest meet time (by minimizing the maximum distance for every troop to reach that place [25]), and with *min* the query retrieves a place that leads to the earliest time for any user to reach that place. In general, the query finds the $k$ best POIs $p$ from $\mathbb{D}$ in an ascending order of $F(p, \mathbb{C}_*)$, as defined below.

*Definition 2.1 (kGNN query [28]).* Given a spatial database $\mathbb{D}$, $n$ query locations $\mathbb{C}_*$, distance function $dis$, and aggregate function $F$, a $k$GNN query ($k \leq D$) retrieves a subset $\mathbb{P} = \{p_1, \cdots, p_k\}$ from $\mathbb{D}$, such that $\forall p_i \in \mathbb{P}$ and $\forall p \in \mathbb{D} - \mathbb{P}$, $F(p_i, \mathbb{C}_*) \leq F(p, \mathbb{C}_*)$, and for $1 \leq i < j \leq k$, $F(p_i, \mathbb{C}_*) \leq F(p_j, \mathbb{C}_*)$.□

### 2.2 Privacy Preserving $k$GNN Query

*Definition 2.2 (Privacy preserving kGNN query, or PPGNN).* Let $\mathbb{D}$ and $\mathbb{C}_*$ be those in Definition 2.1. A $k$GNN query is privacy preserving if the following conditions are satisfied:

(1) **Privacy I:** $\forall i \in [1, n]$, each user $u_i$'s location $l_{i,*}$ is indistinguishable from $d$ equally likely locations by LSP, $d > 1$;

(2) **Privacy II:** the query location $\mathbb{C}_*$ and its answer are indistinguishable from $\delta$ equally likely candidates by LSP, $\delta \geq d$;

(3) **Privacy III:** the users can learn nothing more than the requested answer to the query $\mathbb{C}_*$.

(4) **Privacy IV:** $\forall i \in [1, n]$, $u_i$'s location $l_{i,*}$ is hidden in a region from other users, where the size of the region is no less

[1] https://www.torproject.org/

than $\theta_0$ fraction of the size of the whole location space, $\theta_0 \in (0, 1]$. Under the *full user collusion assumption*, this property should hold even if any $n-1$ users collude, by sharing their locations, to infer the remaining user's location with the help of the query answer received. □

Condition (1) guarantees that the probability for LSP to infer a user's location is $\frac{1}{d}$. Condition (2) guarantees that the probability for LSP to infer the group's query, i.e., all users' locations, as well as the query answer, is $\frac{1}{\delta}$, where $\delta \geq d$. Condition (3) guarantees that the users learn only the query answer as defined by the query, in order to protect the LSP's private database. Condition (4) guarantees that any user's location can not be inferred by other users. The strength of these privacy guarantees depends on the setting of the privacy parameters $(d, \delta, \theta_0)$, which are specified by the users.

## 3 SINGLE USER QUERY

In this section, we consider PPGNN with single user, i.e., $n = 1$. In this case, there is no **Privacy IV**, and $\delta = d$. Section 3.1 gives some background knowledge about the generalized Paillier cryptosystem [10]. Section 3.2 presents our solution.

### 3.1 Generalized Paillier Cryptosystem

Generalized Paillier cryptosystem $\varepsilon_s$ $(s \geq 1)$ [10] is a probabilistic asymmetric encryption scheme that provides semantic security. Let $\mathbb{Z}_{N^s}$ and $\mathbb{Z}^*_{N^s}$ be the residue class ring module $N^s$ and the prime residue class group module $N^s$, respectively.

Generalized Paillier cryptosystem is composed of three algorithms (**Gen, Enc, Dec**): (1) given a security parameter *keysize*, the *key generation* algorithm $(sk, pk) = \textbf{Gen}(keysize)$ returns secret key $sk$ and public key $pk$. $N$ is the product of two large primes determined by $pk$. (2) The *encryption* algorithm $c = \textbf{Enc}(x, pk)$ maps a plaintext $x \in \mathbb{Z}_{N^s}$ to a ciphertext $c \in \mathbb{Z}^*_{N^{s+1}}$ using $pk$. (3) The *decryption* algorithm $x = \textbf{Dec}(c, sk)$ executes reverse operation of encryption by $sk$, obtaining plaintext $x$. The exact construction of **Enc** and **Dec**, which can be found in [10], is not important for this work, but the following homomorphism properties of the generalized Paillier cryptosystem are used by our solutions. For simplicity, we omit the public key $pk$ in the **Enc** algorithm. Let $x_1, x_2$ denote the plaintexts in $Z_{N^s}$.

**Homomorphic addition** $\oplus$**:** given two ciphertexts $\textbf{Enc}(x_1)$, $\textbf{Enc}(x_2)$, the ciphertext of the sum $x_1 + x_2$ can be obtained by multiplying the ciphertexts, i.e.,

$$\textbf{Enc}(x_1) \oplus \textbf{Enc}(x_2) : \ \textbf{Enc}(x_1) \cdot \textbf{Enc}(x_2) = \textbf{Enc}(x_1 + x_2) \tag{2}$$

**Homomorphic multiplication** $\otimes$**:** given a plaintext $x_1$ and a ciphertext $\textbf{Enc}(x_2)$, the ciphertext of the product $x_1 x_2$ can be obtained by raising $\textbf{Enc}(x_2)$ to the power $x_1$:

$$x_1 \otimes \textbf{Enc}(x_2) : \ \textbf{Enc}(x_2)^{x_1} = \textbf{Enc}(x_1 x_2) \tag{3}$$

**Homomorphic dot product** $\odot$**:** given a ciphertext vector $\textbf{Enc}(\boldsymbol{v}) = (\textbf{Enc}(v_1), \cdots, \textbf{Enc}(v_m))^T$ and a plaintext vector $\boldsymbol{x} = (x_1, \cdots, x_m)$, the ciphertext of the dot product $\boldsymbol{v} \cdot \boldsymbol{x}$ can be obtained by:

$$\begin{aligned} \boldsymbol{x} \odot \textbf{Enc}(\boldsymbol{v}) &: (x_1 \otimes \textbf{Enc}(v_1)) \oplus \cdots \oplus (x_m \otimes \textbf{Enc}(v_m)) \\ &= \textbf{Enc}(x_1 v_1 + \cdots + x_m v_m) \\ &= \textbf{Enc}(\boldsymbol{x} \cdot \boldsymbol{v}) \end{aligned} \tag{4}$$

For simplicity, we write $\textbf{Enc}(x)$ with $\varepsilon_1$ as $[x]$. One application of Eqn (4) is privately selecting the $i$-th element in a vector $\boldsymbol{x}$ without disclosing the value of $i$. Specifically, let $[\boldsymbol{v}] = ([v_1], \cdots, [v_m])^T$ where $v_i = 1$ and $v_j = 0$ for all $j \neq i$. Eqn (4) becomes $\boldsymbol{x} \odot [\boldsymbol{v}] = [\boldsymbol{x} \cdot \boldsymbol{v}] = [x_i]$, which returns $x_i$ in ciphertext. We will use this private selection to design our solution.

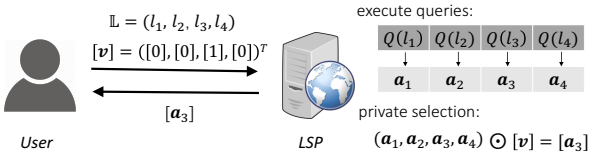



**Figure 2: Single user query example ($l_3$ is real)**

### 3.2 Proposed Solution

Figure 2 shows the idea of our solution through a running example. The user sends a location set $\{l_1, l_2, l_3, l_4\}$ to LSP, where $d = 4$, as well as an encrypted indicator vector $([0], [0], [1], [0])$ that implicitly specifies that $l_3$ is the real user location. LSP calculates the query for every location in the location set, producing the answers $\boldsymbol{a}_1, \boldsymbol{a}_2, \boldsymbol{a}_3, \boldsymbol{a}_4$, and privately selects $\boldsymbol{a}_3$ through the result $[\boldsymbol{a}_3]$ according to the homomorphism properties. This solution has three stages: query generation, query processing, and answer decryption. We discuss each stage in details. We denote the user's real location by $l_*$.

**Query Generation.** At first, the user randomly selects $d-1$ dummy locations from the location space and constructs a *location set* $\mathbb{L} = \{l_1, \cdots, l_d\}$ containing the real location $l_*$. Then the user creates an indicator vector $\boldsymbol{v} = (v_1, \cdots, v_d)^T$:

$$v_i = \begin{cases} 1, & l_i = l_* \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

After that, the user generates $(sk, pk)$ by calling the key generation algorithm with the security parameter *keysize* (e.g., the most commonly used is 1024 bits [23]), and executes element-wise encryption algorithm on $\boldsymbol{v}$, obtaining an encrypted $[\boldsymbol{v}] = ([v_1], \cdots, [v_d])$. Finally, the user sends $\{k, \mathbb{L}, pk, [\boldsymbol{v}]\}$ to LSP, where $k$ is the number of POIs to retrieve.

**Query Processing.** After receiving the user's query, LSP computes the $k$NN query for each location in the location set, resulting in $d$ query answers, $\boldsymbol{a}_1, \cdots, \boldsymbol{a}_d$, where each query answer is a list of $k$ POIs. The $k$NN query is computed on *plaintext*, so any plaintext solution for $k$NN query, such as [24], can be applied. We assume that each query answer $\boldsymbol{a}_i$ is represented by a vector of integers, $\boldsymbol{a}_i = (a_{i,1}, \cdots, a_{i,m})^T$, such that every element is less than $N$, where $m$ is the maximum number of integers required for any of the $d$ answers. If the number of integers encoded for a query answer is less than $m$, 0's are padded as placeholders.

Let $\boldsymbol{A}^{m \times d} = (\boldsymbol{a}_1, \cdots, \boldsymbol{a}_d)$ be the query answer matrix. The next theorem suggests that LSP can obtain the encrypted query answer for the real location $l_*$ by a private selection using $\boldsymbol{A}$ and $[\boldsymbol{v}]$. The encrypted query answer is returned to the user.

Theorem 3.1 (Private Selection). *Given an encrypted indicator vector $[\boldsymbol{v}] = ([v_1], \cdots, [v_d])^T$ such that $[v_i] = [1]$ and $[v_j] = [0]$ for all $j \neq i$, and the answer matrix $\boldsymbol{A}^{m \times d} = (\boldsymbol{a}_1, \cdots, \boldsymbol{a}_d)$, then*

$[\boldsymbol{a}_i]$ *is computed by* $\boldsymbol{A} \bigotimes [\boldsymbol{v}]$ *defined as follows:*

$$\boldsymbol{A} \bigotimes [\boldsymbol{v}] = \begin{pmatrix} a_{1,1} & \cdots & a_{d,1} \\ \vdots & \ddots & \vdots \\ a_{1,m} & \cdots & a_{d,m} \end{pmatrix} \bigotimes \begin{pmatrix} [v_1] \\ \vdots \\ [v_d] \end{pmatrix}$$
$$= \begin{pmatrix} [a_{1,1}v_1 + \cdots + a_{d,1}v_d] \\ \vdots \\ [a_{1,m}v_1 + \cdots + a_{d,m}v_d] \end{pmatrix}$$
$$= \begin{pmatrix} [0 + \cdots + a_{i,1} + \cdots + 0] \\ \vdots \\ [0 + \cdots + a_{i,m} + \cdots + 0] \end{pmatrix} = \begin{pmatrix} [a_{i,1}] \\ \vdots \\ [a_{i,m}] \end{pmatrix} = [\boldsymbol{a}_i]. \square$$

The notation $\bigotimes$ represents the homomorphic matrix multiplication, which executes homomorphic dot product operations in Eqn (4) between each row in $\boldsymbol{A}$ and $[\boldsymbol{v}]$.

**Answer Decryption.** After receiving $[\boldsymbol{a}_i]$, the user can execute the decryption algorithm on the ciphertext to get the exact query answer $\boldsymbol{a}_i$ for $l_*$.

In this solution, **Privacy I** is satisfied by anonymizing the user's real location among $d$ locations. **Privacy II** is satisfied because the real query answer is anonymized in $d$ query answers (note $\delta = d$ when $n = 1$) and the selection process is private. Also, LSP returns only the query answer for $l_*$, so **Privacy III** is satisfied.

# 4 GROUP QUERY

We now present the PPGNN solution for the group query where $n \geq 1$, which subsumes the solution in Section 3 as a special case. Section 4.1 presents a candidate query generation method that helps satisfy **Privacy II**. Section 4.2 describes the PPGNN solution. Section 4.3 proves the protection of **Privacy I-III**. The protection of **Privacy IV** is addressed in Section 5.

Recall that the real location of each user $u_i$ is denoted as $l_{i,*}$, $1 \leq i \leq n$, and $l_{i,*}$ is hidden in the location set $\mathbb{L}_i = \{l_{i,1}, \cdots, l_{i,d}\}$. From $\mathbb{L}_i$, $1 \leq i \leq n$, LSP can obtain a set of *candidate queries*, where each candidate query is a set of $n$ locations that contains exactly one location from every $\mathbb{L}_i$. One candidate query is the *real query*, denoted by $\mathbb{C}_* = \{l_{1,*}, \cdots, l_{n,*}\}$.

**Naive Solution**. With $\delta \geq d$, one straightforward method to satisfy both **Privacy I** and **Privacy II** is that every user $u_i$ generates a length $\delta$, instead of length $d$, location set $\mathbb{L}_i$, and all users arrange their real locations on the same position in $\mathbb{L}_i$, $1 \leq i \leq n$. Then LSP can extract one candidate query from the same position in the $n$ location sets, resulting in $\delta$ candidate queries. One of these candidate queries is the real query. However, this solution incurs the additional computational cost to generate $\delta - d$ extra dummy locations (e.g., using the dummy generation algorithm [20, 22]) for every user, and the additional communication cost to send the extra dummy locations to LSP. With users' computational power being limited (e.g., mobile devices) and the communication bandwidth being precious, this approach is not practical. To address this special requirement in the mobile user scenario, we propose a solution that aims to reduce the user computational cost and the communication cost at some overhead of the LSP computational cost.

## 4.1 Candidate Query Generation

Our solution keeps the location set $\mathbb{L}_i$ at the size $d$ but defines a novel protocol for LSP to generate at least $\delta$ candidate queries from the location sets $\mathbb{L}_i$, $1 \leq i \leq n$. With each $\mathbb{L}_i$ having $d$ elements, $d^n$ candidate queries can be generated by the cartesian product $\bigtimes_{i=1}^{n} \mathbb{L}_i$. We assume $\delta \leq d^n$, otherwise, a larger $d$ should be specified by the users.

Clearly, generating the maximum number of candidate queries, $d^n$, will satisfy **Privacy II**, but if $\delta \ll d^n$, this means that LSP will compute many unnecessary queries. Our method will generate a minimum number $\delta'$ of candidate queries such that $\delta' \geq \delta$, thus, satisfying **Privacy II**. To this end, we partition the user group into $\alpha$ *subgroups* of the size $\bar{\boldsymbol{n}} = (\bar{n}_1, \cdots, \bar{n}_\alpha)$, and partition every location set $\mathbb{L}_i$ into $\beta$ *segments* of the size $\bar{\boldsymbol{d}} = (\bar{d}_1, \cdots, \bar{d}_\beta)$. $\{\bar{\boldsymbol{n}}, \bar{\boldsymbol{d}}\}$ is called *partition parameters* and is known to both users and LSP. We will determine these parameters shortly. To ensure that the real query will be generated as one of the candidate queries, the following constraint must be satisfied by subgroups and segments: all users arrange their real locations in the *same segment*, and all users from the same subgroup arrange their real locations on the *same position* of that segment. Our query generation in Section 4.2 will enforce this constraint.

*Example 4.1.* In Figure 4a, the user group is partitioned into 2 subgroups by $\bar{\boldsymbol{n}} = (2, 2)$, and the location sets are partitioned into 2 segments by $\bar{\boldsymbol{d}} = (2, 2)$. All users arrange their real locations in $segment_2$, highlight in red. Also, the users in $subgroup_1$ arrange the real locations on the 2-nd position of $segment_2$, and the users in $subgroup_2$ arrange the real locations on the 1-st position of $segment_2$.$\square$

Given the location sets and the partition parameters, LSP generates the candidate queries as follows. Let $G_{i,j,t}$ be the subset of locations from the $t$-th position of the $i$-th segment and the $j$-th subgroup, and let $G_{i,j,:}$ be the subset of locations from the $i$-th segment and the $j$-th subgroup. Note that $G_{i,j,:}$ contains $\bar{d}_i$ locations. See Figure 4b for some examples. For $1 \leq i \leq \beta$, LSP computes the candidate queries for the $i$-th segment by the cartesian product

$$\bigtimes_{j=1}^{\alpha} G_{i,j,:} \tag{6}$$

This gives a total number of $\sum_{i=1}^{\beta} (\bar{d}_i)^\alpha$ candidate queries since there are $(\bar{d}_i)^\alpha$ combinations for the $i$-th segment. Each candidate query is uniquely identified by a sequence of the indexes $(i, j, t)$ for $G_{i,j,t}$, and all candidate queries are listed in the lexicographic order of such indexes. We call this list the *candidate query list*.

Figure 3c illustrates the cartesian product for each segment, generating a total of 8 candidate queries. Figure 3d shows the candidate query list, in which the real query $\mathbb{C}_*$ is at the position 7. This position of the real query can be calculated by the users based on the segment and the positions where the real locations are placed in all location sets. However, LSP does not know this position of the real query because the segment and positions for the real locations are confidentially chosen by the users.

**Determining the partition parameters $\{\bar{\boldsymbol{n}}, \bar{\boldsymbol{d}}\}$.** The partition parameters are determined by solving the problem

$$\underset{\alpha, \beta, \bar{\boldsymbol{d}}}{\text{minimize}} \quad \delta' = \sum_{i=1}^{\beta} (\bar{d}_i)^\alpha \tag{7}$$

$$\text{subject to} \quad \delta' \geq \delta \tag{8}$$

$$\sum_{i=1}^{\beta} \bar{d}_i = d \tag{9}$$

$$\alpha \in \mathbb{N}_{\leq n}, \beta \in \mathbb{N}_{\leq d}, \{\bar{d}_i\}_{i=1}^{\beta} \in \mathbb{N}_{\leq d} \tag{10}$$

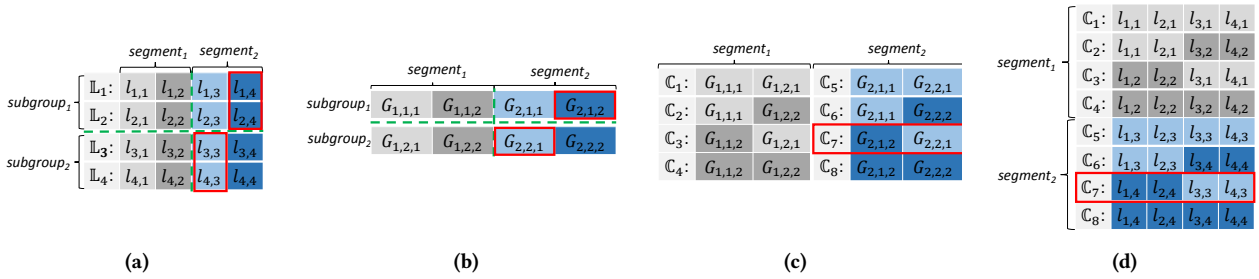



**Figure 3: Candidate query generation, where $n = 4$, $d = 4$, and $\delta = 8$: (a) the partition parameters are $\bar{n} = (2, 2)$ and $\bar{d} = (2, 2)$; (b) $G_{2,1,1}$ represents the set $\{l_{1,3}, l_{2,3}\}$ for $segment_2$, $subgroup_1$, and the first position in $segment_2$, while $G_{2,1,:} = \{G_{2,1,1}, G_{2,1,2}\}$; (c) candidate queries $\mathbb{C}_1 - \mathbb{C}_4$ are generated for $segment_1$, and candidate queries $\mathbb{C}_5 - \mathbb{C}_8$ are generated for $segment_2$; (d) the candidate query list contains the real query (in red).**

$\delta'$ in Eqn (7) is the total number of candidate queries generated above. Eqn (9) requires that the sum of all segment sizes should be $d$, and Eqn (10) requires that the parameters should be integers. The sizes of subgroups are not included in Eqn (7) because they are irrelevant. $d, \delta, n$ are constants and the rest are unknown variables. For the single user query, $n = 1$ and $\delta = d$, we can choose $\beta = d$ with each segment size equal to 1. The above is a nonlinear integer programming problem, which is NP-hard [4]. However, the results for frequently used $(n, d, \delta)$ can be pre-computed off line (e.g., using open-source solvers [6], Bonmin[2]). This only needs to be done once.

## 4.2 PPGNN Solution

The PPGNN solution has three stages: query generation, query processing, and answer decryption. Following [14], we assume that a coordinator user $u_c$ is selected randomly from the query group to assist query generation and answer decryption. Like any other user, no additional trust is assumed of $u_c$.

**Query Generation.** Algorithm 1 presents the query generation stage. $u_c$ first determines the partition parameters $\{\bar{n}, \bar{d}\}$ based on $\{n, d, \delta\}$ and calculates the number of candidate queries $\delta'$. As discussed above, the partition parameters could be pre-computed for frequently used $\{n, d, \delta\}$. $u_c$ randomly selects a segment $seg$ from $[1, \beta]$ according to the probability distribution based on the segment sizes, i.e.,

$$\mathcal{P} = (\frac{\bar{d}_1}{d}, \cdots, \frac{\bar{d}_\beta}{d}) \tag{11}$$

and $u_c$ randomly and uniformly selects a relative position $x_j$ of that segment for the $j$-th subgroup, and broadcasts $pos_j$ to all users in the $j$-th subgroup, $1 \le j \le \alpha$, where $pos_j$ is the absolute position (over all segments) corresponding to $x_j$. Then $u_c$ computes the position of the real query $\mathbb{C}_*$ in the candidate query list, called *query index*, denoted by $QI_{\mathbb{C}_*}$ as follows. Note that the candidate query list is arranged by the lexicographic order of the triples (segment index, subgroup index, position in the segment).

$$QI_{\mathbb{C}_*} = \sum_{i=1}^{seg-1} (\bar{d}_i)^\alpha + \sum_{j=1}^{\alpha} (x_j - 1)(\bar{d}_{seg})^{\alpha - j} + 1 \tag{12}$$

where the first term is the number of candidate queries before reaching the $seg$-th segment, and the second term is the number of candidate queries before reaching $\mathbb{C}_*$ in the $seg$-th segment.

*Example 4.2.* In Figure 4a, with $seg = 2$, $\alpha = 2$, and $\bar{d}_1 = 2$, the first term in Eqn (12) is 4, and with $x_1 = 2$ and $x_2 = 1$, the second term is $(2 - 1) * 2^1 + (1 - 1) * 2^0 = 2$, thus $QI_{\mathbb{C}_*} = 7$. □

[2]https://neos-server.org/neos/solvers/

**Algorithm 1:** Query Generation

**Input:** $n$, $d$, $\delta$, $k$, $keysize$, $\{l_{i,*}\}_{i=1}^n$, $\theta_0$;
**Output:** Query $\{k, pk, \bar{n}, \bar{d}, [\boldsymbol{v}], \theta_0, \{(i, \mathbb{L}_i)\}_{i=1}^n\}$;
**Coordinator** $u_c$:
$\{\bar{n}, \bar{d}\} \leftarrow$ find the partition parameters; // Eqn (7)-(10);
$seg \leftarrow$ randomly select a segment by Eqn (11);
**for** $j \in [1, |\bar{n}|]$ **do**
  $x_j \leftarrow$ randomly and uniformly select from $[1, \bar{d}_{seg}]$;
  $pos_j \leftarrow \sum_{i=1}^{seg-1} \bar{d}_i + x_j$;
  **Send** $pos_j$ **to** all users in $subgroup_j$;
$\{sk, pk\} \leftarrow$ **Gen**($keysize$); // key generation
$\boldsymbol{v} \leftarrow$ construct indicator vector by the query index; // Eqn (12)
$[\boldsymbol{v}] \leftarrow$ element-wise encryption **Enc**($\boldsymbol{v}$, $pk$);
**Send** $\{k, pk, \bar{n}, \bar{d}, [\boldsymbol{v}], \theta_0\}$ **to** LSP;

**Every user** $u_i$ **in** $subgroup_j$:
receive $pos_j$ from $u_c$;
$\mathbb{L}_i \leftarrow$ generate location set, arranging $l_{i,*}$ on the $pos_j$-th position;
**Send** $(i, \mathbb{L}_i)$ **to** LSP;

**Algorithm 2:** Query Processing

**Input:** Query $\{k, pk, \bar{n}, \bar{d}, [\boldsymbol{v}], \theta_0, \{(i, \mathbb{L}_i)\}_{i=1}^n\}$, $\mathbb{D}$;
**Output:** $[\boldsymbol{a}_*]$;
$\{\mathbb{C}_t\}_{t=1}^{\delta'} \leftarrow$ generate the candidate query list by $\{\bar{n}, \bar{d}, \{(i, \mathbb{L}_i)\}_{i=1}^n\}$; // Section 4.1
**for** $t \in [1, \delta']$ **do**
  $\mathbb{P}_t \leftarrow$ compute $k$GNN query by $(k, \mathbb{C}_t, \mathbb{D})$;
  $\mathbb{P}'_t \leftarrow$ answerSanitation($\theta_0$, $\mathbb{P}_t$, $\mathbb{C}_t$); // Section 5.2
  $\boldsymbol{a}_t \leftarrow$ encode $\mathbb{P}'_t$ into integer vector by $N$ stated in $pk$;
$\boldsymbol{A} = (\boldsymbol{a}_1, \cdots, \boldsymbol{a}_{\delta'})$;
$[\boldsymbol{a}_*] = \boldsymbol{A} \bigotimes [\boldsymbol{v}]$; // Theorem 3.1
**Send** $[\boldsymbol{a}_*]$ **to** $u_c$

Then, $u_c$ constructs the encrypted indicator vector $[\boldsymbol{v}]$ of length $\delta'$, where $\boldsymbol{v}$ has 1 at the position $QI_{\mathbb{C}_*}$ and 0 everywhere else. Finally, $u_c$ sends the query $\{k, pk, \bar{n}, \bar{d}, [\boldsymbol{v}], \theta_0\}$ to LSP.

Meanwhile, every user $u_i (i \in [1, n])$ in the $j$-th subgroup arranges its real location $l_{i,*}$ at the received position $pos_j$ in $\mathbb{L}_i$ and dummy locations at the remaining positions, and sends $(i, \mathbb{L}_i)$ to LSP independently. With the user ID $i$, LSP can reconstruct $subgroup_1$ as the first $\bar{n}_1$ users, $subgroup_2$ as the next $\bar{n}_2$ users, and so on. Note that no user, including $u_c$, knows other users' real locations, because each user sends the location set to LSP directly.

**Query Processing.** Algorithm 2 presents the query processing stage. After receiving the query from users, LSP generates the candidate query list containing $\delta'$ candidate queries as described in Section 4.1, and executes the $k$GNN query for each candidate query. Line 4 calls the answerSanitation method to ensure that the query answer satisfies **Privacy IV**, which will be presented in Section 5.2. Let $\boldsymbol{A}^{m\times\delta'}$ be the answer matrix for the query answers $(\boldsymbol{a}_1,\cdots,\boldsymbol{a}_{\delta'})$. LSP privately selects the query answer $[\boldsymbol{a}_*]$ for $\mathbb{C}_*$ following Theorem 3.1, and sends $[\boldsymbol{a}_*]$ to $u_c$.

The answer decryption stage is the same as that in Section 3, except that $u_c$ will broadcast the answer $\boldsymbol{a}_*$ to all other users.

### 4.3 Privacy Guarantees

Theorem 4.3. *The PPGNN solution satisfies **Privacy I-III**.* .

Proof. **Privacy I**: The segment *seg* containing the real locations is selected following the probability distribution corresponding to the segment sizes, Eqn (11), thus, the probability that LSP infers this segment is $\bar{d}_{seg}/d$. The position for the real locations in the *seg*-th segment for each subgroup is selected randomly and uniformly, so given the segment *seg*, the probability that LSP can infer this position is $1/\bar{d}_{seg}$. Overall, the probability for LSP to identify each user's real location is $(\bar{d}_i/d)\cdot(1/\bar{d}_i) = 1/d$.

**Privacy II**: LSP generates $\delta'(\geq\delta)$ candidate queries and obtain $\delta'$ query answers before the private selection. So the probability for LSP to identify group's query $\mathbb{C}_*$ and the query answer is $1/\delta'$, which is no larger than $1/\delta$.

**Privacy III**: The private selection ensures that only the answer for $\mathbb{C}_*$ is returned, so the users learn no extra POI information beyond the query answer requested. □

## 5 FULL USER COLLUSION

So far, the PPGNN solution satisfies only **Privacy I-III**. We now consider answerSanitation on line 4 in Algorithm 2 to enforce **Privacy IV** under the full user collusion assumption. Our first observation is that the only communication within the user group is $\{pos_j\}_{j=1}^{\alpha}$ that are broadcast from $u_c$ to let all users arrange their real locations in their location sets. This information alone does not allow any $n-1$ colluding users to learn the real location of the remaining user. However, after receiving the query answer, $n-1$ colluding users can infer some possible region that contains the remaining user's real location, with the help of the ranking and location information of the POIs in the query answer. We first present this attack in Section 5.1, and devise a method to prevent this attack and satisfy **Privacy IV** in the rest of the section.

### 5.1 Inequality Attack

Suppose the users $\{u_1,\cdots,u_n\}$ located at the locations $\mathbb{C}_* = \{l_{1,*},\cdots,l_{n,*}\}$ respectively obtain the query answer in the form of $k$ ranked POIs $\mathbb{P} = \{p_1,\cdots,p_k\}$ such that

$$F(p_i,\mathbb{C}_*) \leq F(p_j,\mathbb{C}_*), \forall 1\leq i<j\leq k \tag{13}$$

Without loss of generality, let us assume that $u_1$ is the attack target and $\{u_i\}_{i=2}^{n}$ collude together to infer $u_1$'s location. Therefore, there is only one unknown variable $l_{1,*}$ in Eqn (13) because $l_{2,*},\cdots,l_{n,*}$ as well as the query answer $\mathbb{P}$ are known to the colluding users. Consequently, the colluding users can construct $k-1$ independent inequalities to infer the possible region of $l_{1,*}$:

$$F(p_i,\mathbb{C}_*) \leq F(p_{i+1},\mathbb{C}_*), \forall 1\leq i\leq k-1 \tag{14}$$

We refer this inference as the *inequality attack*. Suppose that the area of the solution region for Eqn (14) is $\theta$ (in percentage) of the area of the whole data space, **Privacy IV** is satisfied if and only if $\theta>\theta_0$ for *every target user* $u_1$; otherwise, i.e., $\theta\leq\theta_0$ for some target user $u_1$, **Privacy IV** is not satisfied, and we say the inequality attack succeeds. For instance, **Privacy IV** is not satisfied in Figure 1 if $\theta_0 = 0.5$, because the possible region for $u_1$'s location (in shaded) is less than half of the whole location space.

### 5.2 Answer Sanitation

One solution to prevent the inequality attack is that LSP randomly perturbs the order of POIs in $\mathbb{P}$ such that the inequalities in Eqn (14) cannot be correctly constructed. This solution will degrade the utility of the query answer because the users need to know the rank of returned locations. In addition, it is unclear how to ensure that the colluding users cannot reconstruct the original order or a partial order.

Instead, we design a sanitation method for LSP to return the longest prefix $\mathbb{P}' = \{p_1,\cdots,p_t\}$ of $\mathbb{P}$ while satisfying **Privacy IV**. In fact, LSP can simulate the inequality attack for every user using a prefix $\mathbb{P}'$ of $\mathbb{P}$ in Eqn (14). If $\theta>\theta_0$ holds on $\mathbb{P}'$ for *every target user*, where $\theta$ is the relative size of the solution region of Eqn (14), $\mathbb{P}'$ is safe of satisfying **Privacy IV**. We will consider how to test $\theta>\theta_0$ shortly. LSP can start with the shortest prefix $\mathbb{P}' = \{p_1\}$, which is always safe, and examine the length $t$ prefix only if the length $t-1$ prefix is safe. The query answer for a candidate query then is the prefix $\mathbb{P}' = \{p_1,\cdots,p_t\}$ that is safe, and if $t<k$, the next prefix $\{p_1,\cdots,p_{t+1}\}$ is not safe. Note that if the query answer for every candidate query is safe of satisfying **Privacy IV**, the returned answer for $\mathbb{C}_*$ after private selection satisfies **Privacy IV**.

### 5.3 Testing $\theta>\theta_0$

We now present how to test whether $\theta>\theta_0$ for a target user, where $\theta$ is the relative size of the solution region of Eqn (14). One approach is finding the exact solution region of Eqn (14). Unfortunately, finding this solution region is not straightforward, especially for any choice of the aggregation function $F$ and an arbitrary shape of the data space. On the other hand, it is easy to test whether a point $l_{1,*}$ satisfies all inequalities in Eqn (14), that is, falls into the solution region, without explicitly finding the solution region. This observation motivates the following statistic test.

Consider two hypothesis $H_0$ and $H_1$:

$$H_0:\theta\leq\theta_0, \qquad H_1:\theta>\theta_0 \tag{15}$$

There are two types of errors:

- **Type I Error**: Pr(reject $H_0|H_0$ is true) is the probability that a successful attack is not identified for a target user.
- **Type II Error**: Pr(not reject $H_0|H_1$ is true) is the probability that a non-attack is identified as a successful attack for a target user.

A small probability for Type I Error provides more confidence on privacy protection and a small probability for Type II Error provides more confidence on better utility of the returned answer. We want to bound these probabilities.

To test whether $H_0$ should be rejected, LSP can uniformly and independently sample $N_H$ points $X_1,\cdots,X_{N_H}$ from the data space. The outcome of each sample is a Bernoulli random variable

$$B = \begin{cases} 1, & \text{if } X_i \text{ satisfies the inequalities in Eqn (14)} \\ 0, & \text{otherwise} \end{cases}$$

The probability of $B = 1$ is equal to the relative size $\theta$ of the solution region of Eqn (14). $X = X_1 + \cdots + X_{N_H}$ follows a Binomial distribution. For a large $N_H$, this Binomial distribution can be approximated by the normal distribution. In this case, LSP can reject $H_0$ through the $Z$-test statistic [7]: reject $H_0$ if

$$X > N_H\theta_0 + z_\gamma\sqrt{N_H\theta_0(1-\theta_0)} \tag{16}$$

where $z_\gamma$ is the critical value of the normal distribution and $\gamma$ is a desired upper bound for the Type I Error probability, i.e., $\Pr(\text{reject } H_0 | H_0 \text{ is true}) \le \gamma$.

To test if $\theta > \theta_0$ in the answer sanitation, LSP tests the inequality in Eqn (16) instead. From the above discussion, the probability of capturing a successful attack, i.e., $\Pr(\text{not reject } H_0 | H_0 \text{ is true})$, is at least $1 - \gamma$. Since this approach only requires testing if a point satisfies the inequalities in Eqn (14) (for computing $X$), it is applicable to any choice of the aggregation function $F$ and any shape of the data space.

To determine the sample size $N_H$, we need to take the probability of Type II Error, i.e., $\Pr(\text{not reject } H_0 | H_1 \text{ is true})$, into consideration. Let $\theta_1$ denote the minimum $\theta$ value in $H_1$ that we want to significantly differentiate from $\theta_0$. [15] suggests $\frac{\theta_1}{\theta_0} = (1 + \phi)$, where $\phi$ is the ratio difference between $\theta_1$ and $\theta_0$. The two types of errors can be bounded given enough samples, as stated below.

Theorem 5.1 (Sample Size [11]). *In the one-tailed hypothesis testing, the sample size $N_H$ required for $\Pr(\textit{Type I Error}) \le \gamma$ and $\Pr(\textit{Type II Error}) \le \eta$ is given by*

$$N_H \ge \left\lceil \frac{z_\gamma\sqrt{\theta_0(1-\theta_0)} + z_\eta\sqrt{\theta_1(1-\theta_1)}}{\theta_1 - \theta_0} \right\rceil^2 . \square \tag{17}$$

The commonly used $\gamma, \eta, \phi$ are $\gamma = 0.05$, $\eta = 0.2$, and $\phi = 0.1$. Once the users specify $\theta_0$, LSP can determine the sample size $N_H$ using Eqn (17).

## 5.4 Privacy Guarantees

Theorem 5.2. *The PPGNN solution with the answer sanitation satisfies **Privacy I-IV** under the full user collusion assumption.*

Proof. **Privacy I-III** follows from Theorem 4.3. Suppose the returned query answer for $\mathbb{C}_*$ is $\mathbb{P}_*$, which passes the answer sanitation. Therefore, for every target user $u_i$ in the group query, the Type I Error probability, $\Pr(\text{reject } H_0 | H_0 \text{ is true})$, is bounded by $\gamma$. In other words, $u_i$'s real location is guaranteed to hide in a region that is at least $\theta_0$ (in percentage) of the whole space with the confidence $1 - \gamma$, i.e., **Privacy IV** is satisfied.

A noteworthy point is that the answer sanitation does not affect the protection for **Privacy I-III** because it only reduces the number of POIs in the query answer returned to the users (by returning a prefix of the original top-$k$ answer). The users learn from the reduced list of POIs that the remaining POIs in the answer are not returned because there is an inequality attack. However, without these remaining POIs, the users cannot perform such attacks. $\square$

# 6 OPTIMIZED PPGNN

The indicator vector $\boldsymbol{v}$ with length $\delta'$ has only a single 1 (specifying $\mathbb{C}_*$) and $(\delta' - 1)$ 0's. If $\delta'$ is large, additional user computational cost and communication cost are spent on encrypting and transmitting many 0's. In this section, we present an optimization of PPGNN solution, called PPGNN-OPT, to reduce these costs. Let $[\![\cdot]\!]$ denote the ciphertext generated by the generalized Paillier cryptosystem $\varepsilon_s$ with $s = 2$ (see Section 3.1), and as before, let $[\cdot]$ denote the ciphertext generated by $\varepsilon_s$ with $s = 1$. The encryption and decryption with $\varepsilon_2$ can use the same public key and secret key as those with $\varepsilon_1$ [10].

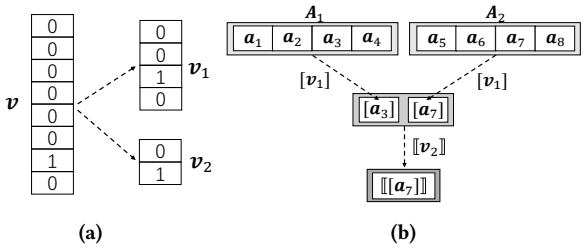



**Figure 4: Optimization example: (a) changes on the user side; (b) two phases private selection on the LSP side**

To explain our optimization, Figure 4 illustrates the changes made on the user side and the LSP side for the example in Figure 3 where $\delta' = \delta = 8$. Instead of using the original indicator vector $\boldsymbol{v} = (0, 0, 0, 0, 0, 0, 1, 0)^T$ to indicate the position of the real query, $u_c$ uses two small vectors $\boldsymbol{v}_1 = (0, 0, 1, 0)^T$ and $\boldsymbol{v}_2 = (0, 1)^T$. On the user side, $u_c$ executes element-wise encryption on $\boldsymbol{v}_1$ with $\varepsilon_1$ and executes element-wise encryption on $\boldsymbol{v}_2$ with $\varepsilon_2$, obtaining $[\boldsymbol{v}_1] = ([0], [0], [1], [0])^T$, $[\![\boldsymbol{v}_2]\!] = ([\![0]\!], [\![1]\!])^T$. These vectors are sent to LSP. In this case, the user computational cost and the communication cost are that for computing and transmitting $[\boldsymbol{v}_1]$ and $[\![\boldsymbol{v}_2]\!]$, instead of that for $[\boldsymbol{v}]$.

On the LSP side, after obtaining the answer matrix $\boldsymbol{A}$, LSP privately selects the desired answer in two phases. Firstly, LSP partitions $\boldsymbol{A}$ into two sub-matrices $(\boldsymbol{A}_1, \boldsymbol{A}_2)$. For each sub-matrix, LSP executes a private selection using $[\boldsymbol{v}_1]$ (Theorem 3.1), resulting in the vector $([\boldsymbol{a}_3], [\boldsymbol{a}_7])$. Secondly, LSP selects the final answer $[\![[\boldsymbol{a}_7]]\!]$ from $([\boldsymbol{a}_3], [\boldsymbol{a}_7])$ using $[\![\boldsymbol{v}_2]\!]$, by treating the ciphertext of $\varepsilon_1$ as a plaintext of $\varepsilon_2$ in Theorem 3.1. $u_c$ can decrypt $[\![[\boldsymbol{a}_7]]\!]$ two times to obtain the plaintext query answer $\boldsymbol{a}_7$.

In general, if the length of $\boldsymbol{v}_2$ is $\omega$, the length of $\boldsymbol{v}_1$ is $\delta'/\omega$. We assume that $\delta'/\omega$ is an integer by padding 0's at the end of $\boldsymbol{v}$ if necessary. We want to choose $\omega$ that minimizes the total communication cost between users and LSP. We focus on the ciphertexts transmitted between $u_c$ and LSP because the plaintext location sets transmitted between users and LSP remain unchanged. The length of a ciphertext of $\varepsilon_2$, which is in $\mathbb{Z}_{N^3}$, is about twice the length of a ciphertext of $\varepsilon_1$, which is in $\mathbb{Z}_{N^2}$. Let $L_e$ denote the length of a ciphertext with $\varepsilon_1$. We want the integer $\omega$ such that

$$\underset{\omega \in \mathbb{N}_{\le d}}{\text{minimize}} \qquad cost(\omega) = (2\omega + \delta'/\omega + 2m) \cdot L_e \tag{18}$$

where the first term accounts for the length of $[\![\boldsymbol{v}_2]\!]$, the second term accounts for the length of $[\boldsymbol{v}_1]$, and the third term accounts for the length of the returned answer ($m$ is the number of ciphertexts required for storing an answer). The optimal $\omega$ for Eqn (18) is the nearest integer to $\sqrt{\delta'/2}$, with the minimum communication cost $cost \approx 2(\sqrt{2\delta'} + m) \cdot L_e$.

In comparison, the communication cost of using the original encrypted indicator vector $[\boldsymbol{v}]$ is $cost' = (\delta' + m) \cdot L_e$, where $\delta'$ accounts for the length of $[\boldsymbol{v}]$ and $m$ accounts for the length of the returned answer. The above optimization reduces the cost when $cost < cost'$, or $2\sqrt{2\delta'} < \delta' - m$. Since $2\sqrt{2\delta'}$ is positive, $2\sqrt{2\delta'} < \delta' - m$ holds only if $\delta' > m$, and in this case, we have $\delta'^2 + b\delta' + c > 0$, where $b = -(2m + 8)$ and $c = m^2$. The solutions for $\delta'^2 + b\delta' + c > 0$ are $\delta' > r_1$ or $\delta' < r_2$, where

**Table 2: Performance analysis**

| | PPGNN | PPGNN-OPT |
|---|---|---|
| Total Communication Cost | $O(nd)L_l + O(\delta')L_e + O(k)L_e$ | $O(nd)L_l + O(\sqrt{\delta'})L_e + O(k)L_e$ |
| User Computational Cost | $O(nd)C_l + O(\delta')C_e + O(k)C_e$ | $O(nd)C_l + O(\sqrt{\delta'})C_e + O(k)C_e$ |
| LSP Computational Cost | $O(\delta')(C_q + C_s) + O(\delta'k)C_e$ | $O(\delta')(C_q + C_s) + O(\delta'k)C_e + O(\sqrt{\delta'k})C_e$ |

$r_1 = m + 4 + 2\sqrt{2m+4}$ and $r_2 = m + 4 - 2\sqrt{2m+4}$. However, since $\delta' > m$ and $r_2 < m$, only $\delta' > r_1$ can be the solution.

In conclusion, on the communication cost, PPGNN-OPT outperforms PPGNN if and only if $\delta' > r_1$ holds. Usually $k$ is not very large and several POIs' information can be encoded into one big integer, therefore $m$ is small and PPGNN-OPT can reduce the cost.

## 7 PERFORMANCE ANALYSIS

Table 2 summarizes the performance analysis of the PPGNN and PPGNN-OPT solutions in terms of communication cost, user computational cost, and LSP computational cost.

**Communication cost.** Let $L_l$ and $L_e$ denote the length of a location and the length of a ciphertext of $\varepsilon_1$, respectively. The communication cost of PPGNN includes: $n$ location sets with size $d$ each, i.e., $O(nd)L_l$, $[\boldsymbol{v}]$ with size $\delta'$, i.e., $O(\delta')L_e$, the returned answer with $m$ encrypted integers that is proportional to the number of POI to be retrieved $k$, i.e., $O(k)L_e$. The total cost is $O(nd)L_l + O(\delta')L_e + O(k)L_e$. With PPGNN-OPT, the cost for location sets does not change, but the cost for ciphertexts is $O(\sqrt{\delta'})L_e + O(k)L_e$ (see Section 6). Therefore, the total cost is $O(nd)L_l + O(\sqrt{\delta'})L_e + O(k)L_e$.

**User computational cost.** Let $C_l$ denote the cost for generating a dummy location, and $C_e$ denote the cost for execution on a ciphertext of $\varepsilon_1$ (e.g., encryption, decryption). The user computational cost of PPGNN includes: location sets with size $d$ generated by all the users, i.e., $O(nd)C_l$, encryption of $[\boldsymbol{v}]$ with size $\delta'$ computed by $u_c$, i.e., $O(\delta')C_e$, and decryption of the returned answer, i.e., $O(k)C_e$. The total cost is $O(nd)C_l + O(\delta')C_e + O(k)C_e$. Similar to the analysis of communication cost, for PPGNN-OPT, the total cost is $O(nd)C_l + O(\sqrt{\delta'})C_e + O(k)C_e$.

**LSP computational cost.** Let $C_q$ denote the cost for executing a $k$GNN query (e.g., MBM algorithm [24]), and $C_s$ denote the cost for answer sanitation for one candidate query. The LSP computational cost of PPGNN includes: $k$GNN queries and answer sanitation for $O(\delta')$ candidate queries, i.e., $O(\delta')(C_q + C_s)$, and the private selection on $\delta'$ answers with size $m$, i.e., $O(\delta'k)C_e$. Hence, the total cost is $O(\delta')(C_q + C_s) + O(\delta'k)C_e$. For PPGNN-OPT, the costs for $k$GNN query and answer sanitation remain unchanged, but the cost for private selection is $O(\delta'k)C_e + O(\sqrt{\delta'k})C_e$ because of the two phases private selection, where the first phase operates on $O(\delta')$ answers and the second phase on $O(\sqrt{\delta'})$ answers. The total cost is $O(\delta')(C_q + C_s) + O(\delta'k)C_e + O(\sqrt{\delta'k})C_e$.

In summary, the communication cost of PPGNN-OPT is asymptotically better than that of PPGNN. However, since the above analysis ignores constant coefficients, in practice, whether PPGNN-OPT is better depends on $\delta'$ and $m$, as discussed at the end of Section 6 (note that $m = xk$, where $x$ is the number of big integers needed to encode one POI). The comparison on user computational cost is similar. The LSP computational cost of PPGNN-OPT is always larger than PPGNN because the second private selection is an extra cost comparing to PPGNN. We will experimentally compare the two solutions in Section 8.

## 8 EXPERIMENTS

We evaluated the performance of PPGNN (Section 4.2), PPGNN-OPT (Section 6), and the Naive solution (the beginning of Section 4). Since the baseline methods for $n = 1$ and $n > 1$ are different, we consider the single user query scenario ($n = 1$) in Section 8.2, and the group query scenario ($n > 1$) in Section 8.3.

### 8.1 Experimental Setup

We conducted experiments on a machine with Intel (R) Core (TM) i7-3770 CPU @ 3.40 GHz×8 machine with 15.6G of RAM, running Ubuntu 16.04.1 LTS. All algorithms were implemented in C++. The classic Minimum Bounding Method (MBM) [24] is applied as our plaintext $k$GNN algorithm in PPGNN, PPGNN-OPT, and the Naive solution, and the aggregation function $F$ is *sum*. We employed the *GMP*[3] library for big integer computation and *libhcs*[4] library for operations of the generalized Paillier scheme. The *keysize* for $\varepsilon_1$ is 1024 bits and the *keysize* for $\varepsilon_2$ is 2048. Table 3 summarizes all parameters and their ranges.

**Dataset.** We used a real-world dataset Sequoia[5], which is widely used in previous studies [12–14, 27, 36]. The dataset contains 62556 POIs from California, including the coordinate and name. As in these previous works, the location space is normalized into a square space, and the real location for every user in a group query was randomly generated as a point in this space. The coordinates of POIs (8 bytes per POI) are returned as the query answer.

**Baselines.** For $n = 1$, we choose the approximate private $k$NN query approach, APNN, in [36] as the baseline. In APNN, LSP partitions the data space into grid cells and pre-computes $k$NN results with respect to the center of each cell and encrypts them. At the query time, the user chooses a square cloak-region containing her location which consists of $b^2$ cells, and initiates a two-stages cryptographic protocol to retrieve the desired answer. The protocol ensures that LSP does not know which cell the user is located in, nor which answer is retrieved, which ensures **Privacy I-II** with the privacy level $b^2$, and the user can only decrypt the requested answer, which ensures **Privacy III**. In our experiments, APNN has a query cloak-region consisting of $5^2$ grid cells, which is equivalent to our default setting $d = 25$ for **Privacy I**. Note that APNN produces only approximate answers and relies on pre-computation of $k$NN results with respect to the center of every cell. This method is not suitable if the exact answer is required or if the database is dynamic. Also, it cannot be extended to the group query scenario because the number of possible queries is significantly large. We did not consider other approaches such as [12, 27] because they are less efficient than APNN according to [36].

For $n > 1$, the first baseline is the incremental pruning private filter (IPPF) algorithm in [14], which is the first work considering users' location privacy in the $k$GNN query. With the cloak-region technique, IPPF ensures **Privacy I-II** but not **Privacy III-IV**. The

[3] http://gmplib.org
[4] https://github.com/tiehuis/libhcs
[5] http://chorochronos.datastories.org/?q=node/58

second baseline is the group location privacy (GLP) algorithm in [2], which applies a secure multiparty computation technique for the users to compute their centroid, and LSP returns the $k$NN query answer with respect to the centroid. GLP ensures **Privacy I** and **III**, but not **Privacy II** and **IV**. A more detailed discussion for IPPF and GLP solutions can be found in Section 9.

**Metrics.** We measured three dominating costs for the queries: the total *communication cost* (including communications between the user group and LSP, as well as the communications within the user group), the total *user cost* (the sum of all users' computational cost), and the *LSP cost* (all the computations execute on LSP during the query process). In the group query scenario, we also evaluated the number of POIs returned to the users, an indicator of the quality of the answer while resisting the full user collusion inequality attack. We executed 500 queries and reported the average cost.

For the discussion below, the reader is referred to Table 3 for the ranges and default settings of all parameters.

**Table 3: Parameters evaluated**

| | Parameter | Range | Default |
|---|---|---|---|
| $n = 1$ | **Privacy I** parameter ($d$) | $[5, 50]$ | 25 |
| | POI to be retrieved ($k$) | $[2, 32]$ | 8 |
| $n > 1$ | **Privacy II** parameter ($\delta$) | $[25, 200]$ | 100 |
| | POI to be retrieved ($k$) | $[2, 32]$ | 8 |
| | user number ($n$) | $[2, 32]$ | 8 |
| | **Privacy IV** parameter ($\theta_0$) | $[0.01, 0.1]$ | 0.05 |

## 8.2 Evaluation for Single User Query ($n = 1$)

For $n = 1$, we evaluated PPGNN and PPGNN-OPT by varying $d$ and $k$. We did not evaluate Naive that is designed for $n > 1$.

**Varing $d$:** Figure 5a-5c compares the three costs of PPGNN and PPGNN-OPT for varying $d$. Note that APNN does not depend on $d$ and is not included. All three costs increase as $d$ increases because each user needs to generate more dummy locations and executes more encryption on the indicator vector, and LSP needs to compute more candidate queries and select the final answer from more query answers. Figure 5a shows that $d = 15$ is the threshold for PPGNN-OPT to outperform PPGNN regarding the communication cost. When $d \geq 15$ (note that $\delta' = d$ when $n = 1$), the communication cost reduction of PPGNN-OPT starts to take effect, which is consistent with the analysis in Section 7. Figure 5b shows a similar trend for the user cost, but with a different threshold $d = 25$ for PPGNN-OPT to beat PPGNN, since the coefficients in Eqn (18) are different with respect to user cost. Usually the cost of operation using $\varepsilon_2$ consumes more than 2 times (≈3 times in our experiments) than that using $\varepsilon_1$, leading to a larger threshold required. For the LSP cost (Figure 5c), PPGNN always performs better because LSP needs to execute two-phases private selection in PPGNN-OPT. In the mobile user scenario, reducing the communication cost and the user cost has a priority over reducing the LSP cost.

**Varying $k$:** Figure 5d-5f reports the performance of PPGNN, PPGNN-OPT, and APNN for varying $k$. The communication costs (Figure 5d) of the three solutions have a staged growth when $k$ goes up because 15 POIs information can be encoded by a big integer in our settings. Figure 5e shows a similar trend on the user cost for the three solutions. For PPGNN and PPGNN-OPT, the LSP cost (Figure 5f) increases when $k$ becomes larger because the $k$NN query time and private selection time increase. APNN has

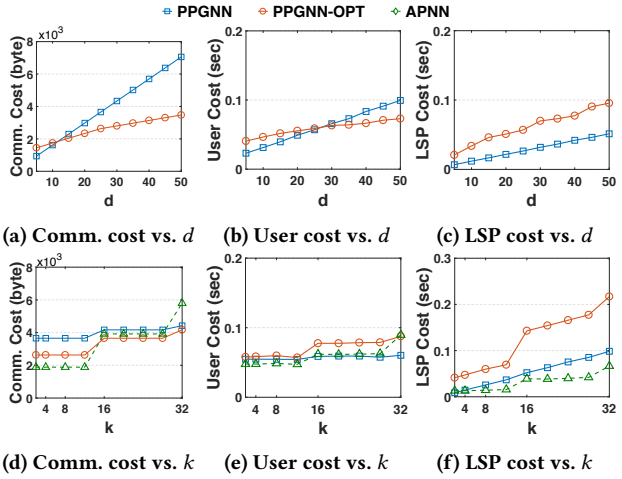



**Figure 5: Effect of parameters when $n = 1$**

the lowest LSP computational cost because of pre-computed $k$NN query answers for all grid cells. This computational gain is paid by returning only approximate $k$NN answers and the potentially expensive update cost for recomputing all $k$NN answers for a dynamic database.

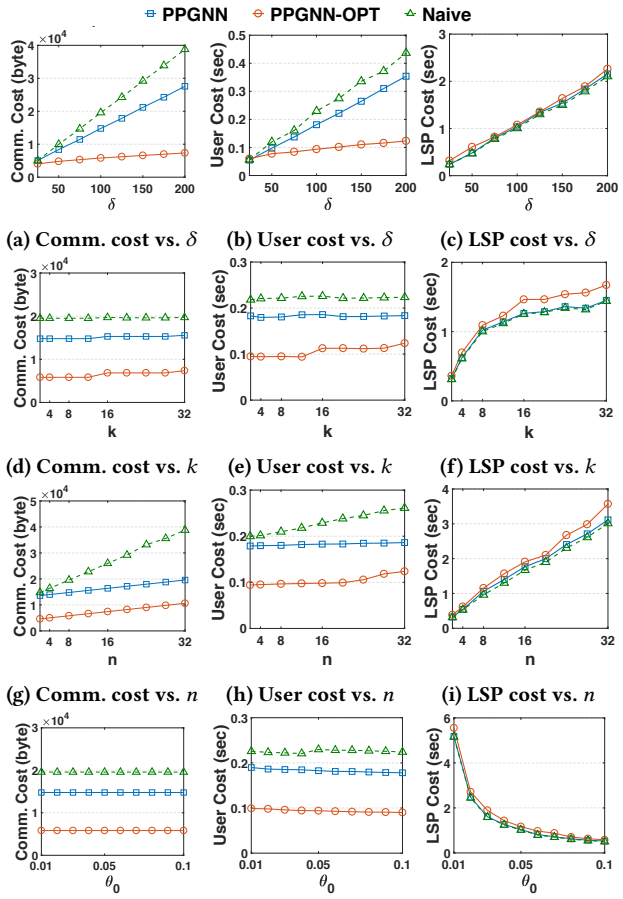
## 8.3 Evaluation for Group Query ($n > 1$)

When $n > 1$, we evaluated PPGNN, PPGNN-OPT and Naive in Section 8.3.1 by varying $\delta$, $k$, $n$, and $\theta_0$. We set $d = 25$ by default and did not show its effect because of the relatively stable performance. We experimentally tested for every $(n, d, \delta)$ where $n \in [2, 32]$, $d \in [5, 50]$, $\delta \in [50, 200]$ and the average difference between $\delta'$ and $\delta$ is approximately 1, i.e., $\delta' \approx \delta$. We used the commonly used confidence levels $\gamma = 0.05$, $\eta = 0.2$ and $\phi = 0.1$ in all experiments in the hypothesis testing. The comparison with the baselines IPPF and GLP, which have only two parameters same as ours, $k$ and $n$, is reported in Section 8.3.2.

*8.3.1 Evaluation of Our Approaches.* **Varying $\delta$:** Figure 6a-6c shows the comparison for varying $\delta$. Unlike the comparison for $n = 1$ in Section 8.2, the communication cost and user cost of PPGNN-OPT are much smaller than those of PPGNN and this advantage increases as $\delta$ increases. In fact, the size of the encrypted indicator vectors ($[\![\boldsymbol{v}_1]\!]$, $[\![\boldsymbol{v}_2]\!]$) in PPGNN-OPT is proportional to $O(\sqrt{\delta'})$, whereas the size of the encrypted indicator vector ($[\![\boldsymbol{v}]\!]$) in PPGNN is proportional to $O(\delta')$. Therefore, for a large enough $\delta$, although the encryption using $\varepsilon_2$ in PPGNN-OPT consumes about three times the cost of the encryption using $\varepsilon_1$, the user cost is still much lower.

The Naive solution incurs the most communication cost because every user in the group needs to send extra $\delta - d$ dummy locations. The LSP costs are almost the same for the three solutions, and are much larger than that for the single user query. Because when $n > 1$, the answer sanitation in Section 5.1 is activated to ensure **Privacy IV**, and this operation dominates the LSP cost. We will discuss more about the LSP cost for answer sanitation in Section 8.3.2.

**Varing $k$:** 6d-6f shows the comparison for varying $k$. The relative comparison of the three solutions is similar to Figure 6a-6c, except that the communication cost and user cost are relatively stable as $k$ increases, with PPGNN-OPT being the best performer. Figure 6f shows that, when $k$ increases, the LSP costs for the




(a) Comm. cost vs. $\delta$ (b) User cost vs. $\delta$ (c) LSP cost vs. $\delta$

(d) Comm. cost vs. $k$ (e) User cost vs. $k$ (f) LSP cost vs. $k$

(g) Comm. cost vs. $n$ (h) User cost vs. $n$ (i) LSP cost vs. $n$

(j) Comm. cost vs. $\theta_0$ (k) User cost vs. $\theta_0$ (l) LSP cost vs. $\theta_0$

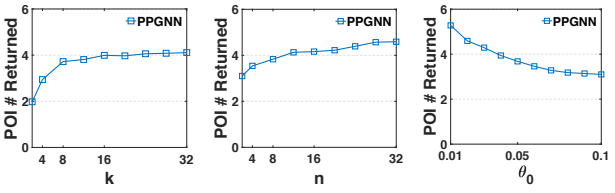
**Figure 6: Effect of parameters when $n > 1$**

three solutions first go up because the number of inequalities in answer sanitation increases, and become stable when $k$ reaches a number since the number of safe POIs becomes stable as $k$ increases (see Figure 7a).

**Varing $n$:** Figure 6g-6i shows the comparison for varying $n$. The trend is similar to Figure 6a-6c, with PPGNN-OPT being the best performer for the communication cost and user cost. For PPGNN and PPGNN-OPT, $n$ does not affect the size of encrypted indicator vector(s), but for the Naive solution, every user needs to generate and send extra $\delta - d$ dummy locations, which leads to the faster increase in the communication cost and user cost. The LSP cost of three solutions increases linearly because the total number of inequalities considered in the answer sanitation grows linearly with $n$.

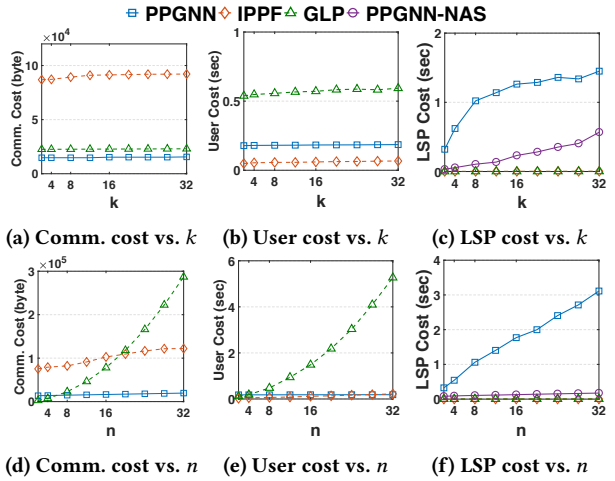
**Varying $\theta_0$:** Figure 6j-6l shows the comparison for varying $\theta_0$. The communication cost and user cost are stable since $\theta_0$ only affects the computation on LSP through the sample size $N_H$ required as in Eqn (17). The LSP cost first decreases greatly and becomes stable as $\theta_0$ increases because the sample size in Eqn (17) behaves this way. In other words, a stronger Privacy IV level means a faster answer sanitation because fewer samples are required in the hypothesis testing.

**Number of POIs returned:** Recall that the answer sanitation will remove some lower ranked POIs in the top-$k$ answer to ensure **Privacy IV** under the full user collusion assumption. Thus, the number of POIs actually returned to the users could




(a) POI number vs. $k$ (b) POI number vs. $n$ (c) POI number vs. $\theta_0$

**Figure 7: The number of POIs returned per answer**




(a) Comm. cost vs. $k$ (b) User cost vs. $k$ (c) LSP cost vs. $k$

(d) Comm. cost vs. $n$ (e) User cost vs. $n$ (f) LSP cost vs. $n$

**Figure 8: Comparison with other approaches.**

be smaller than $k$. This experiment will study how the answer sanitation affects this number. We only consider PPGNN because PPGNN-OPT and Naive will return the same answer as PPGNN. Note that $k$, $n$, and $\theta_0$ can affect this number, but $\delta$ cannot.

Figure 7a-7c shows the number of POIs returned per answer for varying $k$, $n$, and $\theta_0$. The default settings of $k$, $n$, and $\theta_0$ are 8, 8, and 0.01, respectively. In Figure 7a, as $k$ increases, the number of POIs returned first increases and then becomes stable around 4. This is because, with $n = 8$ and $\theta_0 = 0.01$, 4 inequalities usually lead to a successful inequality attack and a larger $k$ has no additional impact. In Figure 7b, the number of returned POIs rises slightly as $n$ increases. At first glance, this seems counter-intuitive because ensuring no attack on more users is more restrictive. At a closer look, with more users involved in a query, the target user's location $l_{1,*}$ weights less in determining if the inequalities in Eqn (14) hold or not, therefore, there are more choices for $l_{1,*}$, thus, a larger solution region for $l_{1,*}$ and it is easier to add the next POI in the answer. The trend in Figure 7c is expected because a larger $\theta_0$ leads to a stronger **Privacy IV**, consequently, fewer POIs can be returned.

To conclude, the top 2 to 5 POIs are still returned for a query even after the answer sanitation. In practice, such numbers are usually sufficient because the users might only need to select one from them.

*8.3.2 Comparison with Baseline Approaches.* We compared PPGNN with the baselines IPPF and GLP for varying $k$ and $n$. $d$, $\delta$, and $\theta_0$ specific to PPGNN are set to their default values. We only consider PPGNN because the experiments above have shown that PPGNN-OPT is better than PPGNN. While PPGNN satisfies **Privacy IV** assuming full user collusion, let PPGNN-NAS denote

**Table 4: Comparison with existing work**

| Group Size | Approaches | Technique | Privacy I | Privacy II | Privacy III | Privacy IV |
|---|---|---|---|---|---|---|
| $n = 1$ | [3, 9, 21] | Cloak-Region | ✓ | ✓ | × | - |
| | [17, 30] | Dummy | ✓ | ✓ | × | - |
| | [13, 26] | Private Information Retrieval | ✓ | ✓ | × | - |
| | [1, 34, 37] | Perturbation | ✓ | × | ✓ | - |
| | [12, 27, 36] | Hybrid Techniques | ✓ | ✓ | ✓ | - |
| | Our approach | Dummy+Paillier | ✓ | ✓ | ✓ | - |
| $n > 1$ | [14] | Cloak-Region | ✓ | ✓ | × | × |
| | [2] | Secure Multiparty Computation | ✓ | × | ✓ | × |
| | Our approach | Dummy+Paillier | ✓ | ✓ | ✓ | ✓ |

the relaxed PPGNN that satisfies **Privacy IV** assuming no user collusion. So PPGNN-NAS does not run the answer sanitation. For IPPF, we chose the query rectangle area (for each user) to be 0.0005% of the data space, which is comparable to choosing $d = 25$ locations (our default setting) as the location set from 5000,000 addresses in California[6]. For GLP, we chose the same *keysize* as PPGNN.

**Varying $k$:** Figure 8a-8c shows the comparison for varying $k$. The communication cost of IPPF is much larger than PPGNN and GLP. In fact, IPPF returns all the candidate POIs, which can be several thousands per query on average, to the users, and the users have to filter the candidate POIs. GLP consumes more user cost than PPGNN and IPPF because there are $O(n^2)$ cryptographic operations and every user has to transmit encrypted values to all other users. In Figure 8c, the gap between PPGNN and PPGNN-NAS is the LSP time spent on the answer sanitation, which dominates the LSP cost. IPPF and GLP consume less LSP cost, however, IPPF cannot satisfy **Privacy III** and **IV**, and GLP cannot satisfy **Privacy II** and **IV** and provide only an approximate answer.

**Varying $n$:** Figure 8d-8f shows the comparison for varying $n$. PPGNN is significantly communication efficient than IPPF and GLP. The communication cost for GLP increases quickly with $n$ because the number of transmitted encrypted values is $O(n^2)$. There is a similar trend on the user cost. Again, Figure 8f shows that PPGNN spent significant LSP time on the answer sanitation to deal with the full user collusion, whereas PPGNN-NAS has almost the same LSP time as IPPF and GLP.

## 8.4 Summary

For $n = 1$, our solutions, PPGNN and PPGNN-OPT, are comparable with or better than the existing solution APNN that heavily relies on pre-computing the answers for all possible queries to reduce the run-time cost. However, APNN produces only approximate answers and the pre-computation means an expensive update cost. For $n > 1$, which is the main focus of this paper, PPGNN and PPGNN-OPT, have significantly smaller communication cost and user cost than existing solutions IPPF and GLP while providing stronger privacy guarantees, i.e., **Privacy I-IV**, and PPGNN-OPT performs better than PPGNN in most cases. We believe that reducing the communication cost and user communication cost is the priority in the mobile user scenario as considered here. The pay for achieving the stronger privacy guarantee is some increase in the LSP cost, especially when dealing with the full user collusion assumption for **Privacy IV**. To our knowledge, this is the first work considering this assumption.

[6]https://openaddresses.io/

# 9 RELATED WORK

Most existing work focus on the single user query case, i.e., $n = 1$. To protect **Privacy I**, some techniques obfuscate user's exact location in a cloak-region (CR) [3, 9, 21] or use dummy query locations [17, 30]. Another technique [13, 26] is based on private information retrieval (PIR), allowing the user to retrieve a particular record from LSP without revealing which record is retrieving. In these techniques, LSP needs to return a super-set of the exact query answer, which not only increases the communication cost but also sacrifices **Privacy III**. Data transformation [37] and differential privacy approach [1, 34] perturb user's exact location to a false location, so the query answer is approximate, and meanwhile, **Privacy II** is sacrificed because LSP knows the query answer. The approaches [12, 27, 36] that based on a hybrid of PIR and cryptography technique can protect **Privacy I-III**, but LSP needs to pre-compute the answers to all possible queries. This technique does not work for $n > 1$ because of too many group queries. Also, if a POI information is updated, LSP needs to re-compute all answers, which is too expensive.

For the group query scenario, i.e., $n > 1$, Hashem *et al.* [14] obfuscates each user's exact location into a region, and LSP executes the $k$GNN query *w.r.t.* these regions, returning a super-set of the query answer. Beside the extra work of filtering the answer set by the users, this approach sacrifices **Privacy III** since extra POI information is returned to the users. **Privacy IV** is also violated because a user's exact location is compromised if its predecessor and successor collude in the filtering phase [2]. In the work by Talouki *et al.* [2], users compute their centroid by secure multiparty computation (SMC) [35] and LSP returns the $k$NN answer for that centroid. This approach cannot protect **Privacy II** and **Privacy IV** because LSP knows the query answer and $n - 1$ users can recover the last user's exact location by their own locations and the centroid.

Table 4 summarizes and compares the above works with our work.

In the privacy preserving meeting location determination (PPMLD) [5, 16, 31], a group of users each chooses a preferred meeting location and send the encrypted locations to the server, who then selects the one to minimize the aggregate distance to all preferred locations. In our work, the query answer is selected from the POI database of LSP and the users specify their current locations, instead of preferred meeting locations. The PPMLD method cannot be adopted to our privacy preserving $k$GNN problem because their cryptographic selection is specific to PPMLD. On the other hand, our approach can be adopted to PPMLD by replacing the $k$GNN building block with any existing non-privacy preserving meeting location determination solution.

The data ownership and privacy implication in our problem are different from most works on secure query processing in the outsourcing database (ODB) model [32]. In ODB, the users and the data owner are trusted and the server is not trusted. The data owner outsources the encrypted database to the server and the user retrieves the query answer from the server. The privacy objective is to prevent the server from learning anything about the database and the user query. In our problem, LSP owns the data and multiple users jointly specify a query, and no party trusts anyone except herself.

## 10 CONCLUSION

This work identifies four privacy objectives for the group $k$-nearest neighbor query, $k$GNN, and designs a privacy preserving $k$GNN solution, PPGNN. To our knowledge, this is the first work that address all of these privacy objectives. Though we consider $k$GNN, the proposed privacy preserving approach can be easily adopted to any group query because it treats the query answering (i.e., $k$GNN) as a black box.

## ACKNOWLEDGMENT

This work is supported by National Science Foundation of China (No.61532021, 61772537), National High Technology Research and Development Program of China (863) (No.2014AA015204), National Key R & D program of China (No.2016YFB1000702), National Basic Research Program of China (973) (No.2014CB340403). Ke Wang's work was partially supported by a discovery grant from The Natural Sciences and Engineering Research Council of Canada (NSERC). This work was partially done when some authors worked in SA Center for Big Data Research in Renmin University of China. The SA Center is funded by Chinese National 111 Project Attracting International Talents in Data Engineering and Knowledge Engineering Research.

## REFERENCES

[1] Miguel E. Andrés, Nicolás Emilio Bordenabe, Konstantinos Chatzikokolakis, and Catuscia Palamidessi. 2013. Geo-indistinguishability: differential privacy for location-based systems. In *CCS*. 901–914.
[2] Maede Ashouri-Talouki, Ahmad Baraani-Dastjerdi, and Ali Aydin Selçuk. 2012. GLP: A cryptographic approach for group location privacy. *Computer Communications* 35, 12 (2012), 1527–1533.
[3] Bhuvan Bamba, Ling Liu, Péter Pesti, and Ting Wang. 2008. Supporting anonymous location queries in mobile environments with privacygrid. In *WWW*. 237–246.
[4] Pietro Belotti, Christian Kirches, Sven Leyffer, Jeff Linderoth, James Luedtke, and Ashutosh Mahajan. 2013. Mixed-integer nonlinear optimization. *Acta Numerica* 22, 1–131.
[5] Igor Bilogrevic, Murtuza Jadliwala, Vishal Joneja, Kübra Kalkan, Jean-Pierre Hubaux, and Imad Aad. 2014. Privacy-Preserving Optimal Meeting Location Determination on Mobile Devices. *IEEE Trans. Information Forensics and Security* 9, 7 (2014), 1141–1156.
[6] Pierre Bonami, Mustafa Kilinç, and Jeff Linderoth. 2012. *Algorithms and Software for Convex Mixed Integer Nonlinear Programs.* Springer New York, 1–39.
[7] Sprinthall R. C. *Basic Statistical Analysis (9th ed.).* Pearson Education.
[8] Sunoh Choi, Gabriel Ghinita, Hyo-Sang Lim, and Elisa Bertino. 2014. Secure kNN Query Processing in Untrusted Cloud Environments. *TKDE* 26, 11 (2014), 2818–2831.
[9] Chi-Yin Chow, Mohamed F. Mokbel, and Xuan Liu. 2006. A peer-to-peer spatial cloaking algorithm for anonymous location-based service. In *ACM-GIS*. 171–178.
[10] Ivan Damgård and Mads Jurik. 2001. A Generalisation, a Simplification and Some Applications of Paillier's Probabilistic Public-Key System. In *Public Key Cryptography*. 119–136.
[11] Joseph L Fleiss, Bruce Levin, and Myunghee Cho Paik. 2003. *Statistical methods for rates and proportions; 3rd ed.* Wiley, Hoboken, NJ.
[12] Gabriel Ghinita, Panos Kalnis, Murat Kantarcioglu, and Elisa Bertino. 2011. Approximate and exact hybrid algorithms for private nearest-neighbor queries with database protection. *GeoInformatica* 15, 4 (2011), 699–726.
[13] Gabriel Ghinita, Panos Kalnis, Ali Khoshgozaran, Cyrus Shahabi, and Kian-Lee Tan. 2008. Private queries in location based services: anonymizers are not necessary. In *SIGMOD*. 121–132.
[14] Tanzima Hashem, Lars Kulik, and Rui Zhang. 2010. Privacy preserving group nearest neighbor queries. In *EDBT*. 489–500.
[15] D.C. Howell. 2013. *Statistical Methods for Psychology.* Wadsworth Cengage Learning.
[16] Yan Huang and Roopa Vishwanathan. 2010. Privacy Preserving Group Nearest Neighbour Queries in Location-Based Services Using Cryptographic Techniques. In *GLOBECOM*. 1–5.
[17] Hidetoshi Kido, Yutaka Yanagisawa, and Tetsuji Satoh. 2005. An anonymous communication technique using dummies for location-based services. In *ICPS*. 88–97.
[18] Feifei Li, Ke Yi, Yufei Tao, Bin Yao, Yang Li, Dong Xie, and Min Wang. 2016. Exact and approximate flexible aggregate similarity search. *VLDB J.* 25, 3 (2016), 317–338.
[19] Yang Li, Feifei Li, Ke Yi, Bin Yao, and Min Wang. 2011. Flexible aggregate similarity search. In *SIGMOD*. 1009–1020.
[20] Hua Lu, Christian S. Jensen, and Man Lung Yiu. 2008. PAD: privacy-area aware, dummy-based location privacy in mobile services. In *Seventh ACM International Workshop on Data Engineering for Wireless and Mobile Access*. 16–23.
[21] Mohamed F. Mokbel, Chi-Yin Chow, and Walid G. Aref. 2006. The New Casper: Query Processing for Location Services without Compromising Privacy. In *PVLDB*. 763–774.
[22] Ben Niu, Qinghua Li, Xiaoyan Zhu, Guohong Cao, and Hui Li. 2014. Achieving k-anonymity in privacy-aware location-based services. In *INFOCOM*. 754–762.
[23] Pascal Paillier. 1999. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *EUROCRYPT*. 223–238.
[24] Dimitris Papadias, Qiongmao Shen, Yufei Tao, and Kyriakos Mouratidis. 2004. Group Nearest Neighbor Queries. In *ICDE*. 301–312.
[25] Dimitris Papadias, Yufei Tao, Kyriakos Mouratidis, and Chun Kit Hui. 2005. Aggregate nearest neighbor queries in spatial databases. *ACM Trans. Database Syst.* 30, 2 (2005), 529–576.
[26] Stavros Papadopoulos, Spiridon Bakiras, and Dimitris Papadias. 2010. Nearest Neighbor Search with Strong Location Privacy. *PVLDB* 3, 1 (2010), 619–629.
[27] Russell Paulet, Md. Golam Kaosar, Xun Yi, and Elisa Bertino. 2014. Privacy-Preserving and Content-Protecting Location Based Queries. *TKDE* 26, 5 (2014), 1200–1210.
[28] Humberto Luiz Razente, Maria Camila Nardini Barioni, Agma J. M. Traina, Christos Faloutsos, and Caetano Traina Jr. 2008. A novel optimization approach to efficiently process aggregate similarity queries in metric access methods. In *CIKM*. 193–202.
[29] Jerome H. Saltzer and Michael D. Schroeder. 1975. The protection of information in computer systems. *Proc. IEEE* 63, 9 (1975), 1278–1308.
[30] Pravin Shankar, Vinod Ganapathy, and Liviu Iftode. 2009. Privately querying location-based services with SybilQuery. In *UbiComp*. 31–40.
[31] Xiaofen Wang, Yi Mu, and Rongmao Chen. 2016. One-Round Privacy-Preserving Meeting Location Determination for Smartphone Applications. *IEEE Trans. Information Forensics and Security* 11, 8 (2016), 1712–1721.
[32] Wai Kit Wong, David Wai-Lok Cheung, Ben Kao, and Nikos Mamoulis. 2009. Secure kNN computation on encrypted databases. In *SIGMOD*. 139–152.
[33] David J. Wu, Joe Zimmerman, Jérémy Planul, and John C. Mitchell. 2016. Privacy-Preserving Shortest Path Computation. In *NDSS*.
[34] Yonghui Xiao and Li Xiong. 2015. Protecting Locations with Differential Privacy under Temporal Correlations. In *CCS*. 1298–1309.
[35] Andrew Chi-Chih Yao. 1982. Protocols for Secure Computations (Extended Abstract). In *23rd Annual Symposium on Foundations of Computer Science*. 160–164.
[36] Xun Yi, Russell Paulet, Elisa Bertino, and Vijay Varadharajan. 2016. Practical Approximate k Nearest Neighbor Queries with Location and Query Privacy. *TKDE* 28, 6 (2016), 1546–1559.
[37] Man Lung Yiu, Christian S. Jensen, Xuegang Huang, and Hua Lu. 2008. SpaceTwist: Managing the Trade-Offs Among Location Privacy, Query Performance, and Query Accuracy in Mobile Services. In *ICDE*. 366–375.
[38] Man Lung Yiu, Nikos Mamoulis, and Dimitris Papadias. 2005. Aggregate Nearest Neighbor Queries in Road Networks. *TKDE* 17, 6 (2005), 820–833.