# FedRoad: Secure and Efficient Road Network Queries over Traffic Data Federation

Shuai Huang
*Tsinghua University*, China
huang-s19@mails.tsinghua.edu.cn

Guoliang Li
*Tsinghua University*, China
liguoliang@tsinghua.edu.cn

Wei Zhou
*Xiamen University*, China
weizhou@stu.xmu.edu.cn

***Abstract*—Federated computing has emerged as a promising approach to address the data isolation problem, enabling multiple data owners to utilize secure multi-party computation (MPC) to collaboratively process queries while keeping the data decentralized, private, and secret. However, existing studies primarily focused on federated queries over structural data, which does not apply to non-structural road network queries prevalent in daily travel scenarios. To tackle this limitation, this paper proposes FedRoad, the first traffic data federation with secure and efficient road network shortest-path queries over it. In this context, the network topology is shared while each silo (*e.g.,* mobility services platform) holds an individual traffic observation of edge weights (*e.g.,* vehicle speeds), where we search the path with minimum joint weights (*e.g.,* the least traveling time). To ensure security, we implement a secret-sharing-based MPC operator to secretly compare joint path weights and achieve a secure federated shortest-path search based on it. To improve the efficiency over road network structures, we (1) first minimize the search iterations by proposing federated shortcut indices and effective federated lower-bound estimation methods, (2) then reduce the cost in each iteration by designing a priority queue structure dedicated to minimizing the expensive MPC comparison operations. Extensive experiments demonstrate that FedRoad significantly outperforms the baselines ($100\times$ faster) and is practical for usage (sub-second level running time).**

## I. Introduction

In the era of big data, numerous businesses rely on analytics over large volumes of data for decision-making. However, the issue of "data isolation" [1], [2] remains a constraint on data usage because sharing raw data among multiple owners is restricted by legal regulations [3], [4] and concerns regarding privacy and commercial competition.

To address the limitation of data isolation, "federated computing" [5], [6] has emerged. It aims to unite multiple data owners as a data federation, enabling joint queries to get the desired results collaboratively over the data individually held. These queries can be processed in the *use-without-disclosure* privacy, *i.e.,* the raw data is kept at each owner locally, and no owner can gets anything beyond the final query results, with the well-established secure multi-party computation [7] (MPC) techniques. Since MPC operations involve high encryption and communication overheads, previous research efforts [8]–[10] focused on reducing the usage of MPC operations to enhance the efficiency of MPC-based federated query processing.

This paper was supported by National Key R&D Program of China (2023YFB4503600), NSF of China (62525202, 62232009), Shenzhen Project (CJGJZD20230724093403007), Zhongguancun Lab, Huawei, and Beijing National Research Center for Information Science and Technology (BNRist).
*Guoliang Li is the corresponding author.

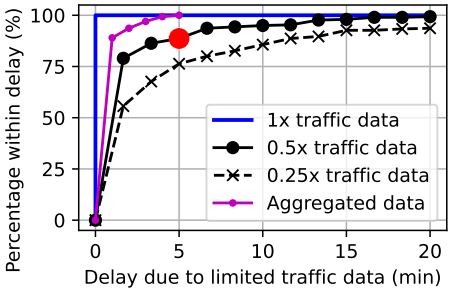


Fig. 1: The percentage delay under various traffic data, *e.g.,* the red point shows that: under $0.5\times$ traffic data, around $10\%$ routing results have a delay of more than 5 minutes.

However, these studies are primarily aimed at the structural queries (*e.g.,* over relational [8], [9] and spatial data [10]) and do not apply to the non-structural road network queries (*e.g.,* querying the shortest paths) prevalent in daily location-based services (LBS) such as map routing. In this scenario, real-time traffic data (*e.g.,* the congestion conditions and vehicle speeds of various roads) from various platforms is crucial for query quality (*e.g.,* identifying the path with the least travel time) [11]–[13]. To evaluate the impact of the volume of traffic data, we experimented with Beijing's road network. We use real taxi trajectories [14] to simulate a full ($1\times$) traffic data [15] when all the platforms form an ideal federation. Then, we sample subsets ($0.5\times$, $0.25\times$) of these trajectories to simulate individual platforms with less traffic data. For each query, we compute the "shortest" (*i.e.,* least traffic cost) paths on road networks under the three datasets. As shown in Figure 1, taking the travel time of roads under 1x traffic data as the ground truth, routes under reduced traffic data are less accurate and incur additional travel time compared to the ground-truth shortest paths. It indicates that *federation incorporating more traffic data can improve the effectiveness of road network shortest-path queries*.

To handle the limited real-time traffic data in each single LBS platform [15], several platforms such as Google Maps [16] and Uber [17] can form a **federation** (as shown in Figure 2) and collaborate on their respective traffic data to obtain a more accurate depiction of traffic conditions, thereby facilitating better routing recommendations (*e.g.,* the shortest-paths on the road network). A naive method to achieve this is uploading all the traffic data (*e.g.,* the traveling time of roads derived from the observed vehicle speeds) of various platforms to a third party to obtain a *joint road network* and processing

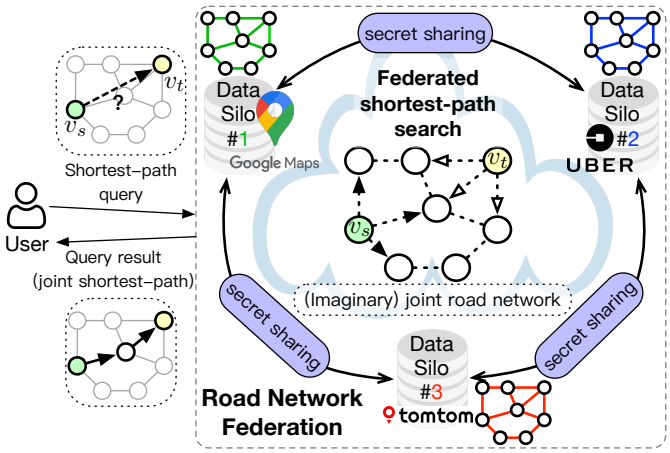


Fig. 2: An example of road network federation and the workflow of federated shortest-path queries.

the shortest-path queries locally on the third party. However, this method is infeasible due to privacy concerns (*e.g.,* even the aggregated vehicle speeds may lead to mobility leakage [18], [19]) and legal restrictions [3], [4]. These platforms will not expose the detailed road traffic data they hold (excluding the public road network topology). Instead, these platforms only *collaborate on queries* [5], [8], [10] (*e.g.,* for better routing services) and disclose the minimum insensitive information required for accessing the query results (*e.g.,* shortest-path).

Specifically, in Figure 2, three traffic data silos (*e.g.,* LBS platforms) share the same road network topology (common vertex and edge sets), with each silo individually holding a local edge weight set representing its observed traveling costs of roads. When a user wishes to route from $v_s$ to $v_t$, all silos collaboratively search the shortest path corresponding to the (imaginary) joint road network, where each edge takes the joint weight over the associating local weights from all silos (*e.g.,* by averaging the local weights from different silos to reduce the noise introduced by limited samples of vehicle speeds, as the curve "Aggregated data" in Figure 1 shows). In this process, these silos use MPC techniques, such as secret sharing [20], for essential communication required for the federated shortest-path search, ensuring that their respective edge weights remain local and confidential.

A **basic solution** for the secure federated shortest-path query could be as follows. First, we can implement a secret-sharing-based *secure comparison* operator, which secretly aggregates the associating local edge weights from all silos and compares the joint costs of any two paths. Then, we can enumerate all possible paths and utilize secure comparison to select the path with the minimum joint cost. To avoid a full enumeration over the large path space, we can employ the Dijkstra algorithm to explore paths judiciously. Specifically, a *federated Dijkstra search* iteratively expands the current nearest vertex, which is found by secure comparisons over the joint costs of all explored paths until reaching the target.

However, without considering the road network structure, the original federated Dijkstra search involves *numerous iterations* and incurs many expensive *secure comparisons in each iteration*. Thus, we must improve the search efficiency from two aspects: *(1) Reducing the number of search iterations*, *i.e.,* pruning the unpromising vertices to be explored. Although extensive methods are designed for this goal in local road networks (*i.e.,* shortcut-based indexing and heuristic-based pruning), they can not be directly applied to federated road networks due to inconsistency of federated indices and shifted bottleneck in federated queries. Firstly, although **road network indices** [21]–[24] consisting of pre-computed shortcuts can be utilized to skip vertices and reduce the search space, individually constructing shortcuts on each silo may lead to inconsistent shortcut sets, potentially resulting in wrong query results. Thus, how can we build consistent federated indices **(C1)**? Secondly, the A* algorithm can utilize an **estimated distance lower-bound** to prune more vertices, such as the landmark-based lower-bound [25], effectively balancing the trade-off between computational cost and estimation accuracy, to minimize the overall searching time. However, in federated queries, this method will incur heavy communication costs, which is an additional but crucial dimension in this trade-off. So how can we design methods to reduce communication costs and achieve a better balance in the new trade-off **(C2)**? *(2) Reducing the volume of secure comparisons in each iteration*, *i.e.,* required for finding the nearest vertex (at the end of the least-cost explored path) to expand next. A **priority queue** can be utilized to efficiently retrieve the least-cost path, by maintaining the partial order between path costs. The heap is the most commonly used structure in local shortest-path search due to its space efficiency. As the bottleneck shifts to secure comparisons in federated search, how can we design a priority queue structure that minimizes the secure comparisons **(C3)**?

In summary, we make the following contributions:

- We propose FedRoad, the first traffic data federation framework with secure road network queries. We implement a secret-sharing-based operator to collaboratively and securely compare joint path costs. FedRoad supports federated road network queries of two types, *i.e.,* single-source and single-pair shortest-path, while keeping the sensitive traffic data local and secret in each silo.
- For **C1**, we propose a method for constructing federated shortcut indices. The silos pre-compute shortcuts by collaboratively searching the corresponding joint shortest paths, ensuring consistent shortcut sets across all silos while preserving their respective shortcut weights, local and secret.
- For **C2**, we design two federated lower-bound estimation methods, aiming to improve the communication-accuracy-computation trade-off and minimize the query time with different choices: The first circumvents the heavy *communication* costs by slightly sacrificing *accuracy*. The second increases the *accuracy* by paying more *local computation*.
- For **C3**, we propose a priority queue structure, the *Tournament Merge Tree*, which minimizes secure comparisons in road network search. It adopts a *winner-tracking hierarchy* (Tournament-tree) to batch push vertices, minimizing the amortized comparisons needed for maintaining the partial orders among the vertices. It also employs *scale-balanced*

*merging* (Merge-tree) to maintain balance, thereby minimizing comparisons in pop operations.
- We have conducted extensive experiments on real-world datasets, which show that FedRoad outperforms all the baselines (100× faster) and is practical for usage.

## II. Preliminary

### A. Problem Statement

**Road Networks Federation.** A *road network* is modeled as a graph $G = (V, E, W)$. A *vertex* $v_i \in V$ represents a road segment junction and an *edge* $e_{v_i v_j} \in E$ represents a road segment from $v_i$ to $v_j$. Each edge is associated with a positive *weight* $\omega(e_{v_i v_j}) \in W$, denoting the traveling time cost on $e_{v_i v_j}$, which periodically changes with the real-time traffic.

As shown in Figure 2, we consider a *road network traffic data federation* consisting of $P$ traffic silos, denoted as $F_1, \ldots, F_P$. (1) They share the *same graph topology*, *i.e.,* $(V, E)$, of the road network $G$; (2) They also share a set of public *static weights* $W_0$, which represent the traveling costs of edges under no traffic congestion (*e.g.,* derived from the road lengths and free-flow vehicle speeds). (3) Additionally, each silo $F_p, p \in \{1, ..., P\}$ holds a private *local weight set*, denoted as $W_p = \omega_p(*)$ for an edge $e \in E$, reflecting the silo's individual traffic observation of real-time vehicle speeds (shown by different edge colors in Figure 2).

We aim for the federation to collaborate on their traffic observation to obtain a more comprehensive traffic overview. Thus, imagine that in an "ideal world", the silos can upload all their private data (*i.e.,* the local weight sets) to a trusted third party and form a joint road network.

**Weighted Joint Road Network (WJRN).** The WJRN is a road network graph $\bar{G} = (V, E, \bar{W})$ with the shared graph topology $(V, E)$ and a joint edge weight set $\bar{W}$, which merges the $P$ local edge weight sets $W_1, \ldots, W_P$. Specifically, for each road $e \in E$, its *joint weight* is the average value of the $P$ associating local weights:

$$\bar{\omega}(e) = \frac{\sum_{p=1}^{P} \omega_p(e)}{P} \tag{1}$$

*Example 1 (WJRN):* Figure 3 shows a traffic data federation consisting of 2 silos $F_1, F_2$, holding $G_1, G_2$ respectively. The corresponding (imaginary) WJRN $\bar{G}$ is shown on the right. There are 8 vertices and 11 edges in each graph. Each silo holds an individual traffic observation (*i.e.,* the local edge weight sets $W_1$ and $W_2$) where each edge is associated with a positive weight in both directions. For example, the local weight of $e_{v_6 v_5}$ in $F_1$ is $\omega_1(e_{v_6 v_5}) = 5$. The joint weight of each edge is the average of associating local weights from all silos, *e.g.,* $\bar{\omega}(e_{v_6 v_5}) = (\omega_1(e_{v_6 v_5}) + \omega_2(e_{v_6 v_5}))/2 = 3$. Note that our methods can handle directed graphs regardless of the undirected graphs in the examples of this paper.

We consider the network queries on the (imaginary) WJRN, utilizing traffic data from all silos to enhance query results. One of the most common queries is the shortest-path query.

**Path.** A *path* is a sequence of adjacent edges between two vertices. We use $\rho = \langle v_0, v_1, \ldots, v_l \rangle$, where $v_0 = v_s, v_l = v_t$ and $e_{v_i v_{i+1}} \in E$ for $i \in [0, l)$, to denote a *path* from $v_s$ to $v_t$. Its *joint cost* $\bar{\phi}(\rho)$ on the WJRN is the sum of joint edge weights: $\bar{\phi}(\rho) = \sum_{i=0}^{l-1} \bar{\omega}(e_{v_i v_{i+1}})$. The joint path cost is also equal to the average of the associating *partial costs* $\phi_p(\rho)$ on all silos ($p = 1, \ldots, P$) for this path (according to Equation 1):

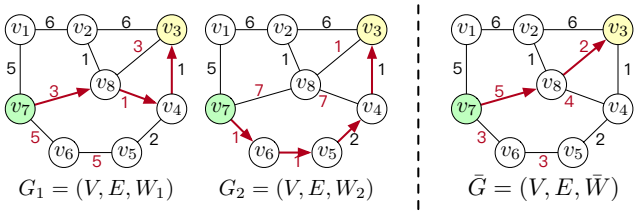


Fig. 3: Road networks and shortest-paths examples.

$$\bar{\phi}(\rho) = \sum_{i=0}^{l-1} \frac{\sum_{p=1}^{P} \omega_p(e_{v_i v_{i+1}})}{P} = \frac{\sum_{p=1}^{P} \phi_p(\rho)}{P} \tag{2}$$

**Shortest Path.** Among all the paths from $v_s$ to $v_t$, $\bar{\rho}^*(v_s, v_t)$ is the one with the least joint cost, which captures how to travel between these two vertices on the road network and is denoted as a *joint shortest-path*. We also use $\bar{\phi}$ to denote the joint shortest-path cost on the WJRN between two vertices: $\bar{\phi}(v_s, v_t) = \phi(\bar{\rho}^*(v_s, v_t))$. Similarly, we use $\rho_p^*(v_s, v_t)$ to denotes the *partial shortest-path* on a local road network $G_p$.

There are two basic types of shortest-path queries:

**Single-Pair Shortest-Path (SPSP).** Given a source vertex $v_s$ and a target vertex $v_t$ as an SPSP query, we answer the joint shortest-path from vertex $v_s$ to vertex $v_t$.

**Single-Source Shortest-Path (SSSP).** An SSSP query asks for the joint shortest-paths from source vertex $v_s$ to all other vertices $v \in V \setminus \{v_s\}$. We can also only query the $k$ (usually $k \ll |V|$) nearest vertices to vertex $v_s$, *a.k.a.* the $k$NN query.

*Example 2 (Shortest-path queries):* On the WJRN $\bar{G}$ in Figure 3, the SPSP from $v_7$ to $v_3$ is $\bar{\rho}^*(v_7, v_3) = \langle v_7, v_8, v_3 \rangle$, with joint cost $\bar{\phi}(v_7, v_3) = 7$. Note that the joint shortest-path is different from the partial shortest paths (*i.e.,* $\langle v_7, v_8, v_4, v_3 \rangle$ on $G_1$ and $\langle v_7, v_6, v_5, v_4, v_3 \rangle$ on $G_2$) and requires less traveling cost than these two paths. The SSSP from $v_2$ where $k = 3$ is $(v_2, \langle v_2 \rangle), (v_8, \langle v_2, v_8 \rangle)$ and $(v_3, \langle v_2\ , v_8, v_3 \rangle)$.

To answer these two types of joint queries, in the "ideal world", a trusted third party holds the WJRN with traffic data collected from all silos and can conduct the shortest-path search locally. However, in this way, the privacy and security of each silo's data are not preserved, typically prohibited by privacy regulations [3], [4] and commercial reasons. Therefore, FedRoad processes the shortest-path queries collaboratively under the following **classic security settings**:

1) *Autonomous Data Silos:* Each silo $F_p$ does not share its raw traffic data (*i.e.,* edge weights $W_p$) with other silos or any trusted third-party, which is aligned with the real-world data federations [8], [9], [26], [27]. Therefore, the WJRN shown in both Figure 2 and 3 does not exist in the real world, *e.g.,* on any silo or third-party. Moreover, the federation does not rely on a central honest broker [10].
2) *Security Guarantee:* During the query processing, any silo $F_p$ could not see or deduce sensitive information, *i.e.,* any edge weight $\omega(e)$ or path cost $\phi(\rho)$ on other silos $G_{p'} (p' \neq$

$p$) or on the WJRN $\bar{G}$. Only the insensitive comparison results, *e.g.,* whether $\bar{\phi}(\rho_A) < \bar{\phi}(\rho_B)$, between joint costs of paths $\rho_A, \rho_B$, are revealed among silos. This is similar to the classic problem where two millionaires compare who is richer without revealing their actual wealth [28].

3) *Semi-honest (Passive) Adversaries:* Each silo honestly executes queries and returns authentic results but may attempt to infer data from others during query execution. This assumption is common in federated query processing [8]–[10]. See Section II-B for the discussion on expanding to other adversary models such as with malicious silos.

To access query results on the imaginary WJRN while keeping raw traffic data local (Setting 1) and only revealing insensitive information (Setting 2) to each other, FedRoad follows the federated computing paradigm [5], [8], [9], using the techniques of secure multi-party computation.

### B. Secure Multi-Party Computation (MPC)

MPC algorithms enable parties (*e.g.,* silos) to jointly compute functions (*e.g.,* integer comparison) while keeping each party's inputs secret [7]. Currently, garbled circuits (GC) and secret sharing (SS) are two dominant MPC techniques. GC uses logic gates (*e.g.,* AND, XOR) to hide the computation program traces, where the final results can be revealed by secure evaluation [28]. While it can compute any arbitrary function (preferring binary computation), it becomes inefficient with more than two parties. SS splits each sensitive input into "secret shares", which in combination yield the original data, and then uses specific encoding to process the secret shares so that they cancel out into the result of cleartext value [20]. While SS supports fewer functions (preferring arithmetic computation), it is more efficient, particularly for settings with more than two parties [10], [29].

According to Equation 2, to securely compare the joint costs $\bar{\phi}(\rho)$ of paths, we need to secretly merge the associating partial costs $\phi_p(\rho)$ from silos. Therefore, we utilize MPC to implement a function named *federated sum-and-compare* (Fed-SAC) as a building block for the shortest-path search. Specifically, Fed-SAC secretly sums up the $P$ partial costs corresponding to each of two paths $\rho_A, \rho_B$ (obtaining two joint costs), and reveals only the comparison result between the two joint costs. Throughout this paper, we use function $[\ \phi_p(\rho_A)\ ] < [\ \phi_p(\rho_B)\ ]$ to denote the invocation of Fed-SAC from the perspective of a certain silo $F_p$, to securely upload its partial costs $\phi_p(\rho_A), \phi_p(\rho_B)$ and collaboratively compare the joint costs $\bar{\phi}(\rho_A), \bar{\phi}(\rho_B)$ with other silos. Under MPC protocol, this operation satisfies security (Setting 1 and Setting 2), as the local edge weights remain local and secret, and only the joint comparison results are exposed.

Since the Fed-SAC function primarily involves arithmetic operations (*i.e.,* integer addition and comparison), we adopt SS rather than GC to implement Fed-SAC as it is more efficient and can better scale to more than two silos. We leverage MP-SPDZ [29], an open-source library that implements multiple SS-based protocols and supports an arbitrary number of parties. Specifically, we use a protocol in MP-SPDZ with semi-honest security (Setting 3) to implement Fed-SAC. It is worth noting that we can also support other adversary models (*e.g.,* malicious and colluded) simply by switching to the corresponding underlying MPC protocol [30]. This paper focuses solely on the design of an efficient upper-layer algorithm, which is independent of the underlying protocol.

**Algorithm 1:** Fed-SSSP (for each silo $F_p$)

**Input:** $(V, E)$: global road network topology,
$\omega_p(*)$: local weight set in $F_p$,
$v_s$: source vertex, $k$: query size

1 Initialize the result set and set of found paths $\mathcal{R} = \mathcal{Q} = \emptyset$
2 Initialize nearest vertex $v = v_s$ with $\rho = \langle v_s \rangle, \phi_p(\rho) = 0$
3 **while** $|\mathcal{R}| < k$ **do**
  /* **(Local step):** *Explore the nearest vertex* $v$ */
4  **if** $v \notin \mathcal{R}$ **then**
5  Add $(v, \rho)$ into $\mathcal{R}$
6  **foreach** *neighbor* $v'$ *where* $e_{vv'} \in E$ **do**
7  $\rho' = \rho \cup \langle v' \rangle,\ \phi_p(\rho') = \phi_p(\rho) + \omega_p(e_{vv'})$
8  Add $(v', \rho', \phi_p(\rho'))$ into $\mathcal{Q}$
  /* **(MPC step):** *Secretly find the shortest-path from* $\mathcal{Q}$ */
9  Initialize the shortest-path $v, \rho, \phi_p(\rho) = \mathcal{Q}[0]$
10  **foreach** $(v', \rho', \phi_p(\rho')) \in \mathcal{Q}[1:]$ **do**
11  **if** $[\ \phi_p(\rho')\ ] < [\ \phi_p(\rho)\ ]$ **then**
12  Update $v, \rho, \phi_p(\rho) = v', \rho', \phi_p(\rho')$
13  Remove the shortest-path $(v, \rho, \phi_p(\rho))$ from $\mathcal{Q}$
14 **return** $\mathcal{R}$

**Federated Query Bottleneck.** MPC computation incurs expensive network communication costs, making Fed-SAC the primary bottleneck in federated shortest-path queries. As the inherent cost of MPC can hardly be eliminated [9], the key to optimizing federated query processing lies in *minimizing the usage of* Fed-SAC.

### C. Federated SSSP Query

A naive method to find the shortest-path is enumerating all possible paths and then using Fed-SAC to compare their joint costs to identify the one with the least cost. To avoid enumerating in a large space, we can employ the Dijkstra algorithm, which efficiently searches for the shortest-paths by judiciously exploring the graph, *i.e.,* prioritizing expanding the nearest vertex. Algorithm 1 shows the search process executed by each silo $F_p$. It iteratively performs two key steps until the top-$k$ shortest-paths are found (line 3-13):

1) *Local Vertex Exploration Step.* Starting from the nearest vertex $v$ (initially set to $v_s$), the path $\rho$ from $v_s$ to $v$ is recorded as a shortest-path (line 5). Then we extend $\rho$ to $\rho'$ as a tentative shortest-path to $v'$ by each of $v$'s neighbors $v'$, and compute the partial cost $\phi_p(\rho')$ (line 6-7).
2) *Global MPC Comparing Step.* Among all the explored paths, we select $\rho$ with the minimum joint cost $\bar{\phi}(\rho)$ using Fed-SAC (line 9-12), which aggregates partial cost $\phi_p(\rho)$ from each silo $F_p$ to compare (line 11). Meanwhile, the corresponding target vertex $v$ becomes the new nearest vertex, to be expanded in the next local step.

In step 2), to avoid using $N-1$ MPC comparisons each time to find the minimum path cost among the $N = O(|V|)$

explored paths (the total time complexity for $k$ iterations is $O(k|V|)$, and the space complexity is $O(k\overline{|\mathcal{N}|})$ where $\overline{|\mathcal{N}|}$ represents the average number of neighbors because each iteration expands the neighbors of a vertex), we can use a *priority queue* such as the heap to store the partial order (*i.e.,* previous comparing results) between path costs, which is commonly utilized in local shortest-path search. Then, we can use $O(\log N)$ Fed-SACs to pop the shortest-path efficiently, reducing the time complexity to $O(k \log |V|)$.

Note that Fed-SAC is the only operation that needs to utilize the traffic data from other silos in Fed-SSSP, and Fed-SAC meets the security guarantees (mentioned in Section II-B). Thus, the entire algorithm meets the secure settings (see Section VII for more details).

### D. Federated SPSP Query

Fed-SPSP can be processed similarly, but it is more challenging to prune the unpromising vertices not in the final shortest-path. Intuitively, given a source vertex $v_s$ and a target vertex $v_t$, we can start a *bi-directional* federated Dijkstra search from both $v_s$ and $v_t$ simultaneously until they converge (Figure 4 (a)). To enhance efficiency, we can utilize the *A* algorithm*, with a heuristic lower-bound (*e.g.,* the straight-line distance calculated by the vertices' longitudes and latitudes), to "guide" the search toward the target and safely prune unpromising vertices. Specifically, each vertex is prioritized based not only on its current joint path cost away from $v_s$ but also on the estimated remaining distance to $v_t$, with their sum defined as the tentative cost. We also use a *priority queue* to maintain the partial relationships among vertices and efficiently retrieve the vertex with the minimum tentative cost.

However, this approach still requires exploring numerous vertices during the search, including those not in the final shortest-path. Therefore, we need to further optimize Fed-SPSP tailored for the road network.

## III. Federated Road Network SPSP Framework

We first introduce existing methods to reduce the search space in local road network (*i.e.,* without federation) SPSP queries and then propose how to redesign these techniques to apply them in FedRoad.

**Local Road Network SPSP.** There are extensive studies [31], [32] which can mainly be divided into two categories:

1) *Shortcut-based indices* use a pre-processing phase that adds shortcuts (*e.g.,* connecting "important" vertices) into $G$ and pre-computes their costs [22], [23], [33]. Then, during the query phase, the A* search algorithm can utilize these shortcuts to bypass many "unimportant" vertices and reach the target faster, particularly in queries of long-distance vertex pairs. For example, as shown in Figure 4 (c), the two added shortcuts (dashed red arrow) enable the A* search algorithm to prune more vertices and converge faster.
2) A more accurate *lower-bound estimation* also enables A* search to prune more vertices by prioritizing the exploration of vertices more likely to lead toward the target (colored by darker black in Figure 4 (e)). Among these, the landmark-based method ALT [25], is the most commonly used. It chooses a small set of landmarks and pre-computes their distances away from all the vertices. Then, for any two vertices, we can efficiently compute a lower-bound from their distances to each landmark and use the largest one as the most accurate estimation of the distance between them.

**Differences in Federated Road Networks.** However, these methods cannot be directly applied in FedRoad, mainly due to the following two differences about this problem:

- The pre-computed structures (*e.g.,* shortcut index and the vertex-landmark distances) in all silos need to be *globally consistent* to ensure correct federated query results. Therefore, we need a collaborative method for pre-processing.
- The *bottleneck* shifts from local computation and data access to the comminication in MPC operations. Therefore, our focus lies in minimizing the MPC comparisons (Fed-SAC), *e.g.,* in selecting lower-bounds, and we can pay more local computation in exchange for that.

Based on these differences, we redesign the shortcut indices and the lower-bound estimation for the federated A* search algorithm. These two methods *reduce the search iterations*, and we further *optimize the costs of MPC operations in each iteration* by designing a novel priority queue structure that minimizes the secret comparisons.

**Framework.** As shown in Figure 4, FedRoad optimizes the federated road network SPSP problem (*i.e.,* pruning unpromising vertices and reducing MPC comparisons) in three aspects.

**(1) Federated Shortcut Index.** To ensure accurate query processing, the indices in all silos need to be consistent with a (imaginary) global index built on the WJRN (*i.e.,* same shortcut set and correct joint shortcut weights). The key factor to address this issue lies in ensuring that: for each shortcut, the associating *witness paths* on all silos need to be consistent with the global witness path on WJRN, *i.e.,* the joint shortest-path. Therefore, we propose a framework for building federated shortcut indices, where all silos add shortcuts by *collaboratively searching the joint shortest-path*. As shown in Figure 4 (b), to add a shortcut, all silos use Fed-SPSP to search the corresponding joint shortest-path as the global witness path, and then each silo preserves the respective partial cost as the local shortcut weight. In this way, the shortcut indices on all silos are consistent, while only the insensitive witness path is synchronized across silos, and the sensitive shortcut weights keep local and secret.

In this way, we can utilize the federated shortcut indices to skip vertices and thus significantly reduce the search space.

**(2) Federated Lower-Bound Estimation.** The landmark-based method ALT can be extended to estimate a federated lower-bound (Fed-ALT). Specifically, the federation uses Fed-SPSP to collaboratively pre-compute the joint shortest-paths between all vertices and landmarks, which can provide a lower-bound by each landmark $l^i$ for any two vertices. Then, it can select the tightest joint lower-bound provided by the "farthest landmark" $l^*$ from $L$, equal to the lower-bound

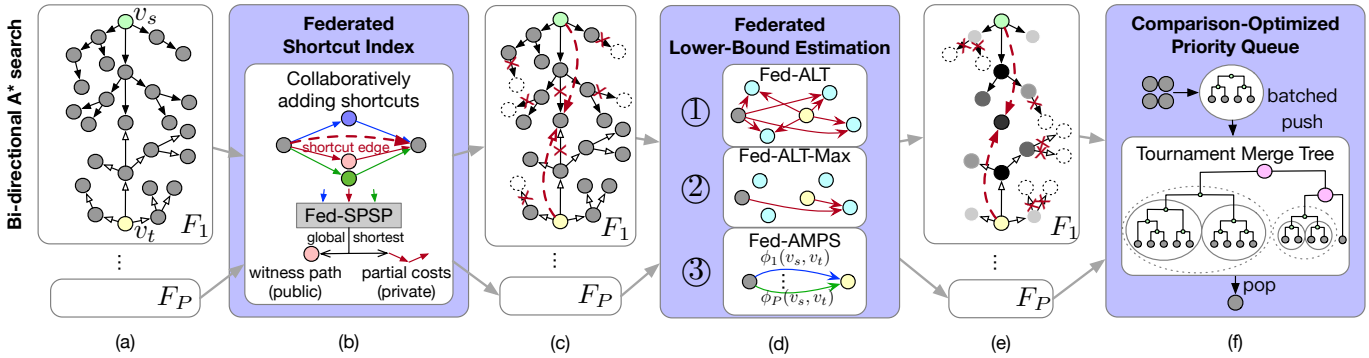


Fig. 4: Framework of the federated road network single-pair-shortest-path query processing

computed by ALT on the WJRN. However, Fed-ALT requires $|L|-1$ MPC comparisons to find the "farthest landmark", for each estimation (Figure 4 (d) ①), incurring heavy communication costs. To strike a better balance in the estimation *trade-off* involving *communication*, *computation*, and *accuracy*, for minimizing the end-to-end query processing time, we propose Fed-ALT-Max and Fed-AMPS as two choices.

Fed-ALT-Max. To alleviate the high *communication* costs for each estimation, we can sacrifice *accuracy* slightly. Specifically, all silos directly select an approximate "farthest landmark" $l_0^*$ by comparing the $|L|$ plain-text lower-bounds under the public static weights $W_0$ instead of collaboratively comparing the joint lower-bounds. Although the joint lower-bound provided by $l_0^*$ is slightly less accurate than $l^*$, this method is more cost-effective than Fed-ALT since it circumvents the large volume of MPC comparisons (Figure 4 (d) ②).

Fed-AMPS. We can pay more for *local computation* to increase the estimation *accuracy*, thus reducing search iterations. Specifically, if each silo searches a partial shortest-path locally, then the Mean of these Partial Shortest-path costs (MPS) is another lower-bound of the joint shortest-path cost (the derivation will be presented in Section V), which is much more accurate than Fed-ALT. Although Fed-AMPS requires each silo to perform an entire local shortest-path search for each estimation, this can be processed without communication.

Leveraging these effective and efficient lower-bounds, A* search algorithm can prune numerous unrelated vertices.

**(3) Comparison-Optimized Priority Queue.** The heap is space-efficient and has favorable memory access locality; thus, it is commonly employed to implement priority queues. However, when the bottleneck shifts to MPC comparison, it is no longer the optimal choice. In response, we design a novel priority queue structure, *i.e.,* the Tournament Merge-tree (TM-tree), which is dedicated to minimizing the number of comparisons in road network searches.

*Queue Operation Workload in Road Network Search.* During the A* search, when we explore a vertex, we pop it from the priority queue and push all its neighbors (may be more than 10) in. Therefore, the push operations usually arrive in groups and the total number of pushes is higher (*e.g.,* $5-10\times$).

*Batch Pushing.* Therefore, we propose to push the grouped $n$ vertices in batch (Figure 4 (f)), with $O(1)$ amortized comparisons instead of $O(\log|\mathcal{Q}|)$ in one-by-one insertion, where $|\mathcal{Q}|$ denotes the queue size. Specifically, we first batch-build a local queue for each insertion group and then merge it into the global queue. For the queue structure, we opt for a *winner-tracking hierarchy*, the Tournament-tree (T-tree), instead of the heap. This is because we can use the least comparisons $(n-1)$ to batch build a T-tree (while using up to $2n$ for a heap) and can merge two T-trees by only 1 comparison (while $O(\log n)$ for heaps). Although the T-tree is less efficient in terms of space usage and memory access than the heap, it significantly reduces MPC comparisons, making this trade-off worthwhile.

*Balance Keeping.* Moreover, we adopt a *scale-balanced merging* mechanism to keep the global queue balanced, which affects the popping costs. Specifically, we maintain multiple T-trees of different scales in the global queue. When inserting a local queue, we merge it with an existing T-tree only if their sizes are similar (*e.g.,* within $2\times$), and propagate the merging thereafter. This approach limits the height of TM-tree to $O(\log|\mathcal{Q}|)$.

## IV. Federated Shortcut Index

There are several road network indices [21]–[24], [34] designed for accelerating the SPSP searching. During a pre-processing phase, these indices commonly incorporate a set of shortcuts (*e.g.,* between important vertices), enabling a shortest-path search to skip over unpromising vertices. However, these indices can not be directly applied to federated road networks since they can not ensure consistent index structures (*i.e.,* shortcuts) across various silos. To address this, we propose a framework of building and maintaining *federated shortcut indices* based on various underlying algorithms.

A *shortcut* $e_{v_iv_j}$ preserves the shortest-path distance $\phi(\rho^*(v_i,v_j))$ from $v_i$ to $v_j$, and $\rho^*(v_i,v_j)$ is the *witness path* of this shortcut. For example, in the rightmost graph $\bar{G}$ in Figure 5, there are 3 shortcuts $e_{v_7v_2}, e_{v_7v_4}, e_{v_2v_4}$. The witness path of shortcut $e_{v_7v_2}$ is $\langle v_7, v_8, v_2\rangle$, and its weight $\bar{\omega}(e_{v_7v_2}) = \phi(v_7,v_2) = 6$. The number of shortcuts cannot be too large, as it would impact pre-processing time, space consumption, and query performance. Thus, we can only choose a limited set of "important" shortcuts.

For a road network federation, we build federated shortcut indices in all silos, which need to be *globally correct*. It means that the weighted joint shortcut set should be equivalent to a (imaginary) global shortcut set built on the WJRN, *i.e.,* globally correct shortcut weights and consistent shortcut sets. We first introduce how to compute the weight of each federated shortcut, then show how to collaboratively select appropriate shortcuts and ultimately construct a federated shortcut index.

**Algorithm 2:** Adding a shortcut (for each silo $F_p$)

**Input:** $(V, E)$: road network, $W_p = \omega_p(*)$: local weight set, $v_i, v_j$: the 2 end vertices

1 $\rho^*$ = Fed-SPSP( $V, E, W_p, v_i, v_j$ )
2 Computing $\phi_p(\rho^*) = \sum_{e \in \rho^*} \omega_p(e)$ locally
3 Add shortcut $e_{v_i v_j}$ with $\omega_p(e_{v_i v_j}) = \phi_p(\rho^*)$

**Algorithm 3:** Federated shortcut index construction by vertex contraction (for each silo $F_p$)

**Input:** $(V, E)$: road network, $W_p$: local weight set, $V_c$: set of vertices to be contracted

1 Initialize the shortcut set $\mathcal{S} = \emptyset$
2 **foreach** $v \in V_c$ *in ascending order of importance* **do**
3  **foreach** *neighbor* $v_i$ *where* $e_{v_i v} \in E$ **do**
4  $\mathcal{R}$ = Fed-SSSP( $V, E, W_p, v_i$ ) until all neighbors $v_j$ with $e_{v v_j} \in E$ are settled
5  **foreach** $v_j, \rho^* \in \mathcal{R}$ *where* $e_{v v_j} \in E$ **do**
6  **if** $\rho^*$ *passes through* $v$ **then**
7  Add $e_{v_i v_j}$ into $E$ and $\mathcal{S}$ with $\omega_p(e_{v_i v_j}) = \phi_p(\rho^*)$
8  Remove $v$ and the associating edges from $(V, E)$
9 **return** $\mathcal{S}$

**Federated Shortcut Computing.** For each shortcut $(v_i, v_j)$, we need to ensure its joint weight over all silos $\sum_{p=1}^{P} \omega_p(e_{v_i v_j})/P$ equal to corresponding weight in the WJRN $\bar{\omega}(e_{v_i v_j})$. All silos first use Fed-SAC to search a global shortest path as the witness path collaboratively. Then each silo can calculate and store the local shortcut weight based on the witness path individually. More specifically, as shown in Algorithm 2, we first search the global shortest path $\rho^*(v_i, v_j)$ between the two end vertices by federated SPSP (line 1). Then we preserve the associating partial cost $\phi_p(\rho^*)$ locally in each silo $F_p$ as the local shortcut weight $\omega_p(e_{v_i v_j})$ (line 2-3). Aggregating the local shortcut weights from all silos can yield the correct joint shortcut weight on WJRN.

As shown in Figure 5, for shortcut $e_{v_7 v_4}$, its local weights computed collaboratively in all silos are globally correct, *i.e.,* their joint weight $\bar{\omega}(e_{v_7 v_4}) = (12 + 4)/2$ is equal to $\bar{\omega}(e_{v_7 v_4}) = 8$ on the WJRN $\bar{G}$. This is because all these local shortcuts is associated with the same witness path (*i.e.,* the joint shortest-path) $\langle v_7, v_8, v_4 \rangle$. As a comparison, if each silo naively computes shortcuts individually, the local weight of $e_{v_7 v_4}$ on $G_1$ will be set to the local shortest-path cost $\phi_1(\langle v_7, v_8, v_4 \rangle) = 4$ and the joint shortcut weight $\bar{\omega}(e_{v_7 v_4}) = (4 + 4)/2 \neq 8$ is incorrect. This may result in wrong outputs during the federated shortest-path search.

**Federated Shortcut Selection.** Various algorithms [21], [23], [34] propose different strategies to select a set of "important" shortcuts, which depend on not only the road network topology but also the edge weights. The goal is to ensure all silos select the correct global shortcut sets in a federated road network. To achieve this, except for the weight-independent steps, all other steps need to be processed collaboratively. We will illustrate the vertex contraction-based algorithm [23] as an example, which incorporates "important" shortcuts by contracting a set of "unimportant" vertices (*e.g.,* with a low degree) successively. In this method, the vertex set to be contracted is selected at first (the selection is independent of the edge weights). Then, shortcuts are added to preserve shortest-paths in the remaining graph after a vertex $v$ is contracted, *i.e.,* when a shortest-path $\rho^*(v_i, v_j)$ passes through $v$, we add a shortcut $e_{v_i v_j}$ when contracting $v$. Since any shortest-path that passes through $v$ also passes through its neighbors, we only search for such $v_i, v_j$ from the neighbors of $v$. Algorithm 3 shows how all silos collaboratively add shortcuts: For each vertex to be contract (line 2), we perform a federated SSSP from each of its neighbors $v_i$ until the shortest-paths to all other neighbors are found (line 4). Then for each neighbor $v_j$ that the shortest-path $\rho^*(v_i, v_j)$ passes through $v$, we add a shortcut $(v_i, v_j)$ (line 5-7). In this way, all silos obtain consistent shortcut sets.

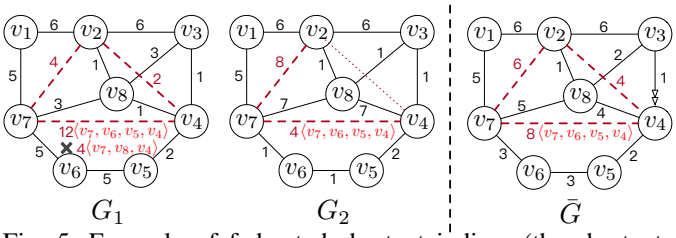

Fig. 5: Example of federated shortcut indices (the shortcuts are represented by red dashed lines).

As displayed in Figure 5, we perform federated SSSP when contracting $v_8$ and find that it is on the joint shortest-paths between $(v_2, v_7)$ and $(v_2, v_4)$, and we add shortcuts $e_{v_2 v_7}$ and $e_{v_2 v_4}$ in all silos. As a comparison, if each silo performs vertex contraction individually, $F_2$ will not add $e_{v_2 v_4}$ since the shortest-path $\langle v_2, v_3, v_4 \rangle$ on $G_2$ does not pass through $v_8$. Consequently, the resulting shortcut sets are inconsistent with the (imaginary) global shortcut set built on the WJRN.

Since contracting each of the $|V_c|$ vertices needs $|\mathcal{N}|$ Fed-SSSPs and each Fed-SSSP terminates as soon as the $|\mathcal{N}|$ neighbors are settled, the time complexity is $O(|V_c||\mathcal{N}|^2 \log |V|)$. The space complexities depends on the number of shortcuts added. In the worst case it is $O(|V|^2)$, but according to [23], it is typically only a constant multiple of $|E|$ and the time cost is also usually close to linear.

**Federated Index Updating.** The traffic conditions change in real-time. Thus, we need to update the federated shortcut indices periodically. Similar to federated index building, the federation performs Algorithm 2 to collaboratively recompute shortcut weights. The difference is that in each update, usually, only the weights of a small subset of edges change, affecting only a small portion of shortcuts. We only need to recompute this subset of shortcuts. We can leverage some methods such as [11] to efficiently identify the affected shortcuts.

## V. Federated Lower-Bound Estimation

The pruning performance of A* search depends on the effectiveness of lower-bound estimation, *i.e.,* how close the estimated value $\pi(v_s, v_t)$ is to the real distance $\phi(v_s, v_t)$. However, the estimation cost should be reasonable to deserve its benefits. We first introduce a straightforward method which

can provide joint lower-bounds for federated road networks based on Landmarks and the Triangle inequality (Fed-ALT). Then, we point out its high communication costs in estimation, and introduce Fed-ALT-Max which reduce the communication required. We also propose another method, Fed-AMPS, aiming at improving the estimation accuracy. These algorithms are also shown in Algorithm 4.

**Fed-ALT.** First, we select a landmark set $L \subset V$ using an existing algorithm [25], [35], where the chosen landmarks are public and static regardless of the changes of the edge weights. Then in the *pre-processing phase*, all silos use federated SSSP to collaboratively search all the vertex-landmark shortest-paths $\rho^*(v, l_i)$ and record the associated partial costs $\phi_p(\rho^*)$ in each silo, which can later provide lower-bounds for any two vertex. We use $\Phi_p[|V|][|L|]$ to denote the pre-computed vertex-landmark distance matrix of $F_p$. Note that if each silo individually computes the landmark-vertex distances, the associated shortest-paths may be inconsistent, leading to incorrect joint costs (similar to the error case of shortcut weights in Section IV). In the *querying phase*, for any $(v_s, v_t)$, each landmark $l_i$ can provide a lower-bound $\bar{\phi}(v_s, l_i) - \bar{\phi}(v_t, l_i)$ of their joint shortest-path distance $\bar{\phi}(v_s, v_t)$, according to the triangle inequality. To get the tightest joint lower-bound provided by $L$, each silo computes a partial lower-bound $\Phi_p[v_s][l_i] - \Phi_p[v_t][l_i]$ by each landmark $l_i$, then all silos use Fed-SAC to aggregate and compare the joint costs of all $|L|$ lower-bounds and choose the maximum one as the tightest lower-bound estimation $\max_{l_i \in L}\{\bar{\phi}(v_s, l_i) - \bar{\phi}(v_t, l_i)\}$.

However, $|L|-1$ secret comparisons for each cost estimation are too expensive. Moreover, the collaborative pre-computing processes also incur heavy communication costs.

**Fed-ALT-Max.** To avoid the $|L|-1$ Fed-SACs in each lower-bound estimation $\phi(v_s, v_t)$, we use plain-text computation to select the landmark $l^*$ that can most likely give the maximum lower-bound. Intuitively, for a vertex-target pair, the farthest landmark $l_0^*$ in the static road network $G_0$, *i.e.,* $l_0^* = \arg\max_{l_i \in L}\{\phi_0(v_s, l_i) - \phi_0(v_t, l_i)\}$, is also likely to be the farthest in the WJRN. Thus, in advance, we compute the plain-text distance matrix $\Phi_0$ on the public $G_0$. Then each silo can locally select the common $l_0^*$, and then use $\Phi_p[v_s][l_0^*] - \Phi_p[v_t][l_0^*]$ as the partial cost estimation. This way, although the estimated lower bounds are slightly looser than Fed-ALT, the estimation cost is much cheaper.

**Fed-AMPS.** We also propose another federated potential function whose accuracy is higher than ALT, *i.e.,* the *joint partial shortest-path cost* in different silos is also a lower-bound of the joint shortest-path cost. Specifically, since $\bar{\phi}_P(v_s, v_t)$ is the average of partial costs $\sum_{p=1}^{P} \phi_p(\rho^*(v_s, v_t))/P$ (denoting $\rho^*(v_s, v_t)$ by $\rho^*$ for short), where each $\phi_p(\rho^*)$ is bounded by the local shortest-path cost $\phi_p(\rho_p^*)$. Therefore, to estimate $\bar{\phi}_P(v_s, v_t)$, each silo $F_p$ only needs to search the local shortest-path $\rho_p^*$, and the aggregation of all the partial shortest-path costs $\phi_p(\rho_p^*)$ provides a joint lower-bound:

$$\frac{\sum_{p=1}^{P} \phi_p(\rho_p^*)}{P} \leq \frac{\sum_{p=1}^{P} \phi_p(\rho^*)}{P} = \bar{\phi}_P(\rho^*) = \bar{\phi}_P(v_s, v_t) \quad (3)$$

**Algorithm 4:** Federated lower-bound estimation (for each silo $F_p$)

*/* The vertex-landmark cost matrices $\Phi_1, \ldots, \Phi_P$ have been pre-computed collaboratively, $\Phi_0$ pre-computed publicly */*

**Fed-ALT** $\pi_p(v_s, v_t)$**:**
1. Initialize $\pi_p^{max} = \Phi_p[v_s][l_0] - \Phi_p[v_t][l_0]$
2. **for** $i = 1, \ldots, |L|$ **do**
3. **if** [ $\pi_p^{max}$ ] < [ $\Phi_p[v_s][l_i] - \Phi_p[v_t][l_i]$ ] **then**
4. Update $\pi_p^{max} = \Phi_p[v_s][l_i] - \Phi_p[v_t][l_i]$
5. **return** $\pi_p^{max}$

**Fed-ALT-Max** $\pi_p(v_s, v_t)$**:**
6. Select $l_0^* \in L$ with the max $\Phi_0[v_s][l_i] - \Phi_0[v_t][l_i]$
7. **return** $\Phi_p[v_s][l_0^*] - \Phi_p[v_t][l_0^*]$

**Fed-AMPS** $\pi_p(v_s, v_t)$**:**
8. Search the local shortest-path $\rho_p^*(v_s, v_t)$ on $G_p$
9. **return** its partial cost $\phi_p(\rho_p^*)$

Fed-AMPS (*i.e.,* A* + Mean Partial Shortest-path cost) has better estimation accuracy than Fed-ALT (will be shown by experiments in Section VIII). This is based on the cost of local SPSP searching in each silo. To accelerate this, we can build local indices for each $G_p$. After all, since the cost of local SPSP searching is minor compared to the cost of Fed-SAC, it should have better end-to-end performance than Fed-ALT.

In summary, the communication overheads (each joint cost comparison needs only 1 secret aggregation) of Fed-ALT-Max and Fed-AMPS are significantly lower than that of Fed-ALT(requiring $|L| - 1$ more secret comparisons in each estimation), making both more efficient than Fed-ALT (both Fed-ALT and Fed-ALT-Max need $O(|V||L|)$ space to store the vertex-landmark cost matrices while Fed-AMPS need to store indices for local SPSP searching such as the shortcut indices of $O(|E|)$ space). Fed-AMPS requires more local computation than Fed-ALT-Max but achieves higher accuracy, making it likely to outperform Fed-ALT-Max if accuracy is more critical in the end-to-end query time (*i.e.,* to find the shortest-path).

The federated shortcut index and the federated A* pruning significantly reduce the number of explored vertices. We next discuss reducing the cost of exploring each vertex by designing a novel priority queue structure that minimizes the comparisons for the SPSP searching.

## VI. Tournament Merge Tree

In the Dijkstra search, we alternately explore vertices and select the next vertex $v$ to extend with the highest priority (*i.e.,* the minimum cost). To avoid comparing all $N$ visited vertices to find $v$ (using $N-1$ secure comparisons), we use a priority queue structure $\mathcal{Q}$ (*e.g.,* heap) to maintain a partial order of the visited vertices so that the number of secure comparisons in each push and pop operation is acceptable.

The heap is a popular choice for implementing priority queues because it has favorable computational complexity ($O(\log|\mathcal{Q}|)$), space usage ($|\mathcal{Q}|$), and memory access locality. However, in the federated query processing, the bottleneck shifts to the secure comparison operation. Thus, the heap is no longer the optimal choice. Therefore, we propose a

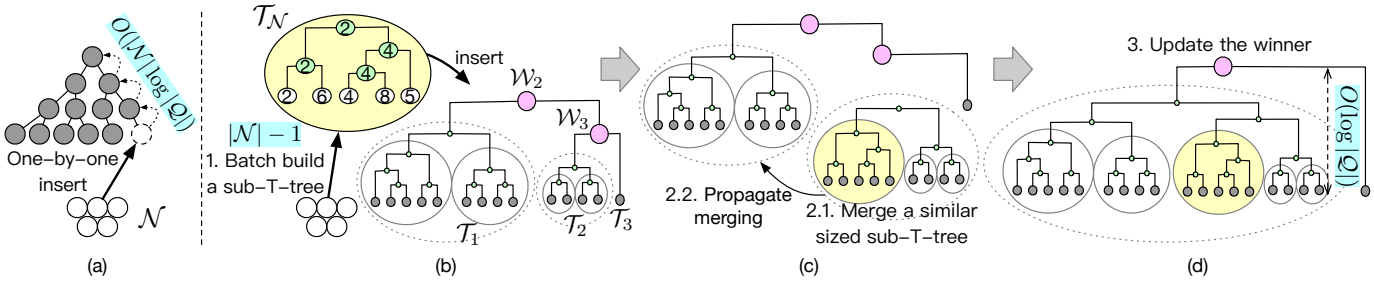


Fig. 6: The insertion on a heap (left) and on a Tournament Merge Tree (right)

comparison-optimized priority queue structure, *i.e.,* the Tournament Merge Tree (TM-tree). We will first introduce the design choices of the TM-tree to optimize the number of comparison operations inside road network search, then introduce its insertion and deletion processes.

### A. Design Choices

**Batch Insertion.** In road network search, when we explore a vertex, we pop it from the priority queue $\mathcal{Q}$ and push all its neighbors $\mathcal{N}$ in. The number of push operations that come consecutively varies from 1 to several tens and accounts for the vast majority of all queue operations. If we insert these $|\mathcal{N}|$ items one by one into a heap (Figure 6(a)), since each item needs to be sifted up from the bottom, the total number of comparisons is $O(|\mathcal{N}|\log|\mathcal{Q}|)$. To reduce the amortized cost for each insertion, we can batch the insertions by a 2-step pushing process: first, building a sub-queue of the $|\mathcal{N}|$ items (*e.g.,* using $O(|\mathcal{N}|)$ comparisons by the bottom-up heapify), then merging it into a global queue.

**Winner-Tracking Hierarchy.** To support batch insertion, we use a priority queue structure which keeps track of the "winner" of a series of "competitions" among items, *e.g.,* the yellow part in Figure 6 (b), named tournament tree (T-tree for short), *a.k.a.* winner tree. The reason why the T-tree is a better choice than the heap is: we can batch build a T-tree (step 1) by $|\mathcal{N}|-1$ comparisons (which reaches the lower-bound since we can not use fewer comparisons to find the minimum one among $\mathcal{N}$), and can simply merge two T-trees (step 2) by one comparison. Although the space usage (extra internal nodes to store the winners) and memory access continuity (using pointers) of a T-tree is less efficient than a heap, it is not crucial in the federated query processing where the primary bottleneck is the MPC comparisons.

**Scale-Balanced Merging.** In the second step of the batch insertion, if we naively merge the newly added sub-T-tree with the existing T-tree, *i.e.,* adding a parent node representing the "competition" between their respective "winners", the tree height will grow linearly with the number of batch insertions, which will increase the costs of pop operations. Therefore, we maintain a series of sub-T-trees of various scales (*e.g.,* the right part of Figure 6(b)) and merge two sub-T-trees only when they have similar sizes. In this way, we limit the tree height to $O(\log|\mathcal{Q}|)$ and thus have a better popping performance.

### B. TM-Tree: Pushing and Popping Processes

In a TM-tree (the right part of Figure 6 (b)), there are a series of $m$ sub-T-trees $\mathcal{T}_m,\dots,\mathcal{T}_1$ (surrounded by dashed circles) of various scales, *i.e.,* $|\mathcal{T}_i|>\alpha|\mathcal{T}_{i-1}|$ for $i\in\{2,\dots,m\}$, where $\alpha>1$ is a parameter controlling the tree balance. There are $m-1$ *winner nodes* $\mathcal{W}_m,\dots,\mathcal{W}_2$ to store the comparison results between the priorities of subtrees (the pink nodes). Larger sub-T-trees are located at higher levels meaning they "compete" in a later order, *i.e.,* $\mathcal{W}_2=\mathsf{Winner}(\mathcal{T}_2,\mathcal{T}_1),\mathcal{W}_3=\mathsf{Winner}(\mathcal{T}_3,\mathcal{W}_2),\dots$, where $\mathsf{Winner}()$ return the one with higher priority (*i.e.,* lower cost).

**Batch Push Process.** We use three steps to push a batch of items $\mathcal{N}$. (1) We build a sub-T-tree $\mathcal{T}_\mathcal{N}$ on $\mathcal{N}$, by $|\mathcal{N}|-1$ comparisons (the results are stored in the $|\mathcal{N}|-1$ winner nodes, see the upper left of Figure 6 (b)). (2) To insert $\mathcal{T}_\mathcal{N}$ into the global TM-tree, we first search whether there is any sub-T-tree $\mathcal{T}_i$ with a similar size, *i.e.,* $|\mathcal{T}_\mathcal{N}|\le\alpha|\mathcal{T}_i|$ and $|\mathcal{T}_i|\le\alpha|\mathcal{T}_\mathcal{N}|$. If more than one sub-T-tree meet this condition, intuitively, we choose the one with the closest size. We merge $\mathcal{T}_\mathcal{N}$ with this $\mathcal{T}_i$ by performing a comparison between them and adding a winner node above them. If the new sub-T-tree $\mathcal{T}_i'$ can be further merged, *e.g.,* $|\mathcal{T}_{i+1}|\le\alpha|\mathcal{T}_i'|$, we continue merging to the left until there are no more similarly sized sub-T-trees. This process is shown in Figure 6 (c). In turn, if there is no sub-T-tree with a similar size with $\mathcal{T}_\mathcal{N}$, we simply insert $\mathcal{T}_\mathcal{N}$ into the right place among other sub-T-trees. (3) Assume that $\mathcal{T}_\mathcal{N}$ is inserted at the place of $\mathcal{T}_i$ (*i.e.,* the last merged sub-T-tree or the inserted place), we need to update all the affected winner nodes (Figure 6 (d)). Specifically, we first update the associating $\mathcal{W}_i$ (or $\mathcal{W}_2$ for the rightmost sub-T-tree $\mathcal{T}_1$), then propagate to the left $\mathcal{W}_{i+1},\mathcal{W}_{i+2},\dots$ and stop when the winner is not changed in the last competition.

The batch pushing of $|\mathcal{N}|$ items takes $|\mathcal{N}|-1$ comparisons in the sub-T-tree building and $O(m)$ comparisons for inserting this sub-T-tree into the TM-tree. Since $m$ is limited by $O(\log|\mathcal{Q}|)$, the amortized number of comparisons for each item is $1+O(\log|\mathcal{Q}|)/|\mathcal{N}|$.

**Pop Process.** The pop process of the TM-tree is similar to the T-tree. Each time an item pops, it first replaces the parent winner node with the sibling node of this item. Then, it updates all the winner nodes along the path to the root to find the new "champion". The number of comparisons is equal to the path length, which is limited by the tree height $O(\log|\mathcal{Q}|)$.

## VII. Security Analysis

We use a simulation-based method [36] to prove the security of Algorithm 1, under semi-honest adversaries. In this process, each silo $F_p$ can be replaced by a simulator $\mathcal{S}$ which only knows the comparison results between the joint path costs, not having any edge weight on $G_p$. Then in each search iteration, in the local step, $\mathcal{S}$ can extend the path $\rho$ to $\rho'$ by each neighbor $v'$ of the given vertex $v$, without knowing the local cost $\phi_p(\rho')$.

In the MPC step, $\mathcal{S}$ can select the next nearest vertex $v$ with a path of the minimum joint cost. Finally, $\mathcal{S}$ terminates after finding $k$ shortest-paths. This shows that each silo can simulate the execution of Algorithm 1, which means the Fed-SSSP searching is secure. Fed-SPSP can be proved similarly.

Regardless of the query pattern, one can only learn the relative order of joint costs between paths but not the exact edge weights of each silo.

## VIII. Experiments

### A. Experiment Setting

**Datasets.** We use three real-world road networks CAL, BJ, and FLA. Table I provides details. For each dataset, we generate the edge weight sets for $P$ silos by randomly increasing the edge weights to simulate traffic congestion. Specifically, we randomly choose a subset of congested road segments $E_c$ from $E$ by a congestion ratio $\beta = E_c/E$. We then increase the weight $\omega'(e) = \omega(e) * (1 + \theta)$ of each edge $e \in E_c$ by a proportion $\theta$ sampled from the uniform distribution $U(0, \theta_{max})$. We generate the edge weight sets $W_1, \ldots, W_P$ of all $P$ silos through $P * E_c$ times sampling. We vary $\beta$ and $\theta_{max}$ to simulate different congestion levels:

- Free traffic: $\beta = \theta_{max} = 0$ (using original road lengths)
- Slight congestion: $\beta = 10\%, \theta_{max} = 30\%$
- Moderate congestion: $\beta = 20\%, \theta_{max} = 50\%$
- Heavy congestion: $\beta = 50\%, \theta_{max} = 100\%$

By default, we simulate moderate traffic congestion on the road networks in a data federation of 3 silos.

**Evaluation.** We generate SPSP queries by randomly sampling vertex pairs $(v_s, v_t)$ from $V$. We divide these queries into five groups by the number of road segments (*i.e.,* hops) in the shortest-path of the original graph $G_0$. For example, on CAL, we generate five groups of queries with their hop counts in 5 intervals with boundaries at 0, 50, 100, 150, 200, 250.

We evaluate query efficiency by reporting the average *running time*, *network communication*, and *communication rounds* of all silos in 20 consecutive queries in each group.

**Implementation.** We implemented federated query processing in the MP-SPDZ [29] framework. We choose the "Temi" protocol [37] with the "edaBits" optimization (both can be found in [30]) to implement the Fed-SAC operator.

**Experiment environment.** The experiments are conducted on up to 5 machines. Each machine has 2 Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20GHz processors, with 128 GB RAM, running Ubuntu 22.04.2 LTS. The network bandwidth between machines is up to 1GB/s.

### B. Comparative Analysis

**Baselines.** (1) Naive-Dijk: Bidirectional Dijkstra searching with all the distance comparisons replaced by Fed-SAC; (2) +Fed-Shortcut: Utilizing a federated shortcut index (Section IV). Specifically, we use CH [23] in the experiments; (3) +Fed-AMPS: Using Fed-AMPS for pruning (Section V) over Fed-Shortcut; (4) + Fed-ALT-Max: Using Fed-AMPS for pruning over Fed-Shortcut; (5) +TM-tree: Using TM-trees (Section VI) as the priority queues instead of min-heaps, with the balance factor $\alpha = 4$, over Fed-Shortcut and Fed-AMPS; (6) Naive-Dijk +TM-tree: Using TM-trees as the priority queues instead of min-heaps over standard Naive-Dijk, as it is a standalone component.

TABLE I: Details of datasets

| Dataset | Region | #Vertices | #Edges |
|---|---|---|---|
| CAL | California | 21,048 | 43,386 |
| BJ | Beijing | 338,024 | 881,050 |
| FLA | Florida | 1,070,376 | 2,687,902 |

**Query performance.** We evaluate the query costs of 4 methods in all three datasets. Figure 7 and Figure 8 respectively show the processing times and communication sizes varying query scale (*i.e.,* number of hops in the final shortest-path). In summary, we have the following observations:

(1) The curves of running time and communication cost are similar. This indicates that MPC communication is the bottleneck of federated SPSP query processing.

(2) Both the running time and communication cost of the query increase with larger query scales for all four methods. This is because for vertex pairs that are farther away from each other, the Dijkstra search space is larger, and we need more search iterations to find the shortest path.

(3) When utilizing a federated shortcut index, the query performance is much better (10x-100x) than Naive-Dijk. The costs increase more slowly with the growing query scale. This is because, with a federated shortcut index, we perform a hierarchical bi-directional search, where the search from the two vertices that are far away can switch to higher levels and meet sooner, skipping many lower-level vertices. This significantly reduces search iterations (*e.g.,* from 10K to 10).

(4) When using Fed-AMPS for pruning, the query performance improves (10x-80x), which is better than Fed-ALT-Max. This is because Fed-AMPS provide a very accurate federated lower-bound (*e.g.,* under 1% relative error) in a relatively cheap cost (*i.e.,* local partial shortest-path computation). Therefore, the A* search can avoid scanning most vertices not in the final shortest-path and terminate sooner.

(5) TM-tree further improves the performance by around 50%. This is because it reduces the MPC comparisons in queue operations by batching the pushing of items (neighbors of the vertex being explored), reducing the amortized MPC comparisons of each item from $O(\log |\mathcal{Q}|)$ to $O(1)$ while keeping the $O(\log |\mathcal{Q}|)$ complexity of each pop operation. Note that TM-tree also works on standard Dijk, but the gain is not as significant as when integrating it over the shortcut index. This is because, after adding shortcuts, the average number of neighbors increases, making the batch enqueue mechanism of TM-tree more beneficial.

(6) The final processing time of each federated SPSP query in various distance scales is under 1 second on average, which is practical for real-time query response.

**Scalability.** We evaluate the query costs varying the number of data silos from 2 to 8 in all three datasets. We only test the first query group of each dataset (*i.e.,* 0-50 hops in CAL, 0-100 hops in BJ, 0-150 hops in FLA). Based on Figure 9, we have the following observations: (1) All four proposed

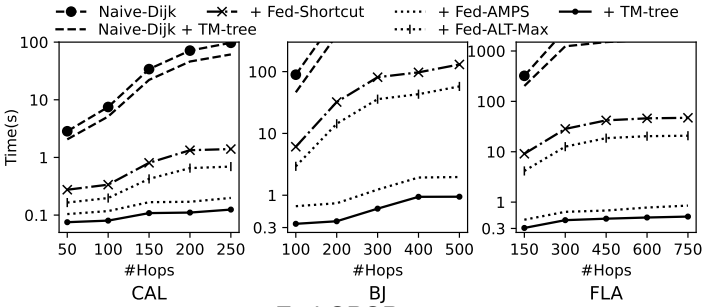


Fig. 7: Running time of Fed-SPSP vs. varying query scale.

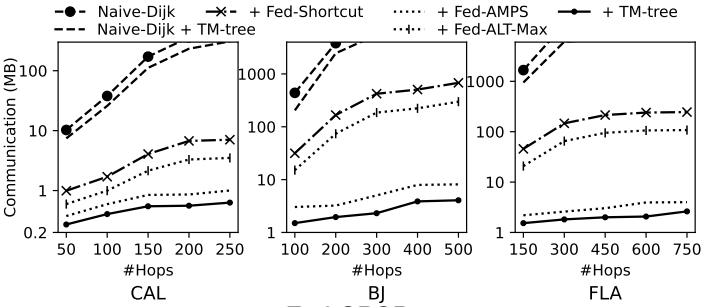


Fig. 8: Communication of Fed-SPSP vs. varying query scale.

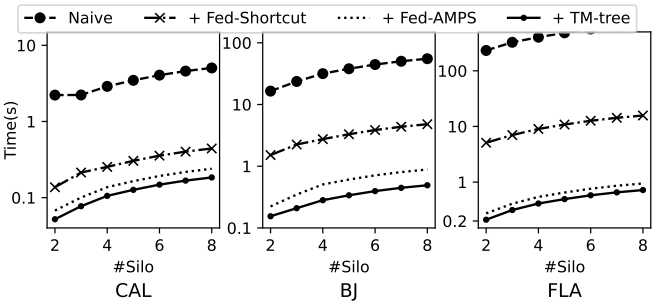


Fig. 9: Time costs of federated SPSP queries vs. varying #silo.

TABLE II: Construction and update time vs. edge percentage with changed weights of federated shortcut index (minutes).

| Dataset | 0.1% | 1% | 10% | Construction |
|---|---|---|---|---|
| CAL | 0.06 | 0.1 | 0.5 | 1.3 |
| BJ | 1.2 | 3.7 | 12 | 41 |
| FLA | 3.5 | 7.4 | 37 | 174 |

methods significantly improve the query performance in all datasets, which is consistent with the previous experiments; (2) For each method, the query time increases nearly linearly with the number of data silos. This is because the communication cost for performing a secret comparison can be expressed as $R \cdot (L + S/B)$, where $R$ is the number of communication rounds, $S$ is the communication size each round, $L$ is the network latency, and $B$ is the network bandwidth. The total number of secret comparisons only depends on the query scale and the Fed-SPSP algorithm, which is the major focus of this paper, and is independent of silo scale $P$. On the other hand, the underlying secret-sharing protocol determines both the values of $R$ as well as how $S$ scales with $P$. Since the "Temi" protocol [30], [37], [38] (mentioned in Section VIII-A) has a good scalability with $P$, the communication cost grows at a relatively modest rate with the number of silos and our proposed method is practical with the support of this protocol.

**Index Construction & Dynamic Update.** We evaluate the efficiency of construction and dynamic update of federated shortcut index with a partial update method [11]. As shown in the Table II, with the support of an efficient index update algorithm, the federated shortcut index can be updated within minutes when only a small portion of edge weights change. In most cases, this update speed is sufficient to accommodate the frequency of real-world traffic changes.

### C. Ablation Study

We perform ablation experiments over several facets of our federated road network query algorithms on the CAL dataset.

**Effect of Fed-SAC.** To evaluate the correlation between the usages of secure comparison operator (Fed-SAC) and the query costs, we process federated SPSP queries of various scales (*i.e.,* hops) on CAL by different methods. Note that all costs are normalized into 0-1 by dividing the max values. Figure 10 shows that all the query costs are linearly proportional to the usage of Fed-SAC in the query processing. This observation validates that the MPC operation is the main bottleneck in federated query processing. Its effects are much more significant than plain-text computation, memory access, *etc.*, which are important in single-machine query processing.

**Effect of the federated A* lower-bound estimation.** To evaluate different lower-bound estimation methods for federated A* pruning (Section V), we test their accuracy under various traffic congestion. For Fed-ALT and Fed-ALT-Max, we randomly select various numbers of landmarks (*i.e.,* 16, 32 and 64). We also evaluate the accuracy of ALT with 64 landmarks, which use the vertex-landmark distance matrix $\Phi_0$ on the static road network $G_0$ (*i.e.,* the traffic congestion level of "Free") to estimate a shortest-path distance. Note that the estimation results of this method can not guarantee to be a lower-bound if we allow edge weights of silos in the traffic federation smaller than $G_0$. We report the mean relative errors of 100 queries of various scales on CAL in Figure 11:

(1) The accuracy of ALT decreases as the edge weights increase, while the accuracy of all other federated lower-bound estimation methods does not change significantly. This is because the joint shortest-path costs increase with edge weights, but the estimation of ALT remains unchanged, so the gap between them gets larger. On the other side, the federated methods are effective in various traffic congestion.

(2) With a larger landmark set, both Fed-ALT and Fed-ALT-Max have a lower mean estimation error. This is because a landmark is more likely to provide a tighter lower-bound, so the maximum among them is more likely to become larger. However, we also need more space to store the pre-computed distance matrix, and ALT needs more MPC operations to compare the joint lower-bounds.

(3) The estimation accuracy of Fed-ALT-Max is almost the same as Fed-ALT. This validates that for a vertex pair, the "farthest landmark" with the tightest static lower-bound on $G_0$ can provide an approximate tightest lower-bound on the graph with dynamic traffic. Since Fed-ALT-Max avoids using MPC operations to compare joint costs, it is better than ALT.

(4) Fed-AMPS achieves better estimation accuracy than Fed-ALT and Fed-ALT-Max. This indicates that the mean partial shortest-path cost provides a very tight lower-bound than landmark-based methods. Since the cost to search local

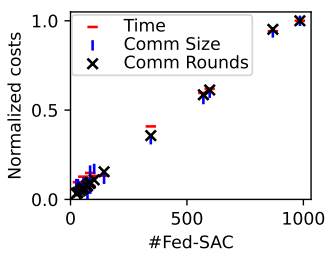


Fig. 10: Query costs are proportional to the usage of Fed-SAC.

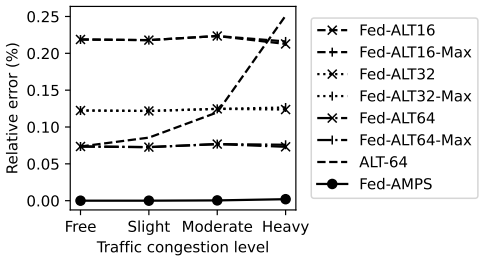


Fig. 11: Accuracy of various federated lower-bound estimation methods.

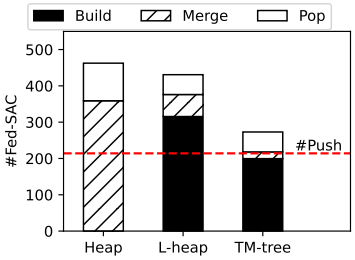


Fig. 12: The comparison numbers in different priority queues.

shortest-paths is minor compared to MPC operations and the benefits of significantly reducing searching iterations, Fed-AMPS is the better point in the federated lower-bound trade-off for improving the overall query performance.

**Effect of the TM-tree.** In Section VI, we introduce the design choices of the TM-tree to reduce comparisons. To evaluate the effects of these designs, we investigate three priority queues:

- Heap. The binary heap without batch insertion.
- L-heap. Using the Leftist heap [39] for batch insertion: first build a sub-heap from bottom to up in $O(|\mathcal{N}|)$, then merge it into the global leftist heap in $O(\log|\mathcal{Q}|)$.
- TM-tree. The TM-tree with $\alpha = 4$.

We process 100 queries of various scales on BJ and report the average usage of Fed-SAC in building sub-queue, merging to the global queue (considering every push in the Heap as a merge), and popping. All these priority queues are used under Fed-Shortcut and Fed-AMPS. We also report the total number of push operations in the bi-directional A* searching, which is a lower-bound of the total comparisons. Note that all these numbers are the sum of the corresponding priority queues in the two search directions. Figure 12 shows that:
(1) In road network search, the number of push operations (building + merging) and the associating comparisons are greater than those of pop operations.
(2) The number of comparisons in sub-heap building in L-heap is slightly fewer than that in the push of Heap but becomes greater than it after adding the merge. This indicates that although the leftist heap batches the insertion of items $\mathcal{N}$ and reduces the heapifying complexity from $O(|\mathcal{N}|\log|\mathcal{Q}|)$ to $O(|\mathcal{N}|)$, since the constant (can be up to 2) is not small enough, the improvement is minor. Moreover, L-heap introduces extra $O(\log|\mathcal{Q}|)$ for merging, so the total comparisons in push operations become more than Heap. On the other hand, the comparisons in pop operations of L-heap are fewer than Heap because the skewed structure of the leftist heap reduces the length of the updating path after popping an item.
(3) TM-tree reduces the average comparisons of each push operation to almost 1 (very close to the dashed line of #push). This is because a Tournament-tree uses the least comparisons $|\mathcal{N}|-1$ in batch building and $O(\log_\alpha|\mathcal{Q}|)$ for merging, both of which are significantly lower than those of a leftist heap.

## IX. Related Work

### A. Federated Computing

To solve the "data isolation" problem, federated computing has emerged as a hot research topic in the database field [5], [6]. To satisfy the security constraints, data federation usually uses MPC techniques [20], [28] to secretly aggregate the partial results from each owner. SMCQL [8] utilizes MPC to secretly compute SQL results from multiple private relational databases. Conclave [9] further improves the performance of joins and aggregations of federated SQL. Hu-Fu [10] proposes a federated spatial database. These systems focus on structural data, where the final results can be computed by secretly merging the local processing results. Besides, for graph data, FedGraph [40] proposes a distributed subgraph matching approach, and Fed-LTD [41] studies distributed bipartite graph matching. However, these approaches can neither address the shortest-path problem nor ensure data security. We focus on secure federated shortest-path query, requiring multiple rounds of MPC communication. We redesign the algorithms from road network shortest-path searching to optimize the query cost.

Mechanisms like contribution-based rewards [42] and industry consoria [43] can encourage participation in data federation. Furthermore, collaboration can be applied in application within single organizations with distributed branches with better reliance. Data federation can enhance availability through strategies like redundant servers and timeout retries.

### B. Shortest Path Searching on Road Networks

**Road Network Index.** These methods use a pre-processing phase to build indexes. [23] constructs a set of pre-computed shortcuts so that the search can skip over "unimportant" vertices. [24] builds a hierarchical balance graph-tree index and searches for the shortest distance by traversing the corresponding tree nodes. [44] proposes a hierarchical index structure based on tree decomposition and achieves high performance.

**Goal-Directed Pruning.** These methods aim to "guide" the search toward the target and prune vertices that are not in the direction of it. [25] utilizes a set of landmarks to provide a lower-bound, which can be utilized in A* search to prune unpromising vertices safely. Besides, there are approximate algorithms [45], [46] that answer inexact shortest-path distances.

## X. Conclusion

In this paper, we propose the first secure and efficient framework FedRoad for road network queries over a traffic data federation. We first implement a secret-sharing-based operator to securely compare joint path costs, based on which we can perform a secure federated Dijkstra search. To minimize the search iterations, we propose a federated shortcut index to reduce the search space. We also design effective and efficient federated lower-bound estimation methods. To reduce the MPC comparisons in each iteration, we further design a comparison-optimized priority queue TM-tree. Experiments demonstrate the superior performance of FedRoad.

## REFERENCES


[1] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 10, no. 2, pp. 1–19, 2019.

[2] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE signal processing magazine*, vol. 37, no. 3, pp. 50–60, 2020.

[3] E. Parliament and T. C. of the European Union, "The general data protection regulation (gdpr)," 2016. [Online]. Available: https://gdpr.eu/

[4] S. of California Department of Justice, "California consumer privacy act (ccpa)," 2018. [Online]. Available: https://oag.ca.gov/privacy/ccpa

[5] A. Bharadwaj and G. Cormode, "An introduction to federated computation," in *Proceedings of the 2022 International Conference on Management of Data*, 2022, pp. 2448–2451.

[6] Y. Tong, Y. Zeng, Z. Zhou, B. Liu, Y. Shi, S. Li, K. Xu, and W. Lv, "Federated computing: Query, learning, and beyond," 2023.

[7] R. Cramer, I. B. Damgård *et al.*, *Secure multiparty computation*. Cambridge University Press, 2015.

[8] J. Bater, G. Elliott, C. Eggen, S. Goel, A. N. Kho, and J. Rogers, "Smcql: Secure query processing for private data networks." *Proc. VLDB Endow.*, vol. 10, no. 6, pp. 673–684, 2017.

[9] N. Volgushev, M. Schwarzkopf, B. Getchell, M. Varia, A. Lapets, and A. Bestavros, "Conclave: secure multi-party computation on big data," in *Proceedings of the Fourteenth EuroSys Conference 2019*, 2019, pp. 1–18.

[10] Y. Tong, X. Pan, Y. Zeng, Y. Shi, C. Xue, Z. Zhou, X. Zhang, L. Chen, Y. Xu, K. Xu *et al.*, "Hu-fu: Efficient and secure spatial queries over data federation," *Proceedings of the VLDB Endowment*, vol. 15, no. 6, p. 1159, 2022.

[11] D. Ouyang, L. Yuan, L. Qin, L. Chang, Y. Zhang, and X. Lin, "Efficient shortest path index maintenance on dynamic road networks with theoretical guarantees," *Proceedings of the VLDB Endowment*, vol. 13, no. 5, pp. 602–615, 2020.

[12] A. Sharma, V. Ahsani, and S. Rawat, "Evaluation of opportunities and challenges of using inrix data for real-time performance monitoring and historical trend assessment," 2017.

[13] "Tomtom white paper." [Online]. Available: https://download.tomtom.com/open/crm/lib/docs/download/HDT_White_Paper.pdf

[14] Z. Shang, G. Li, and Z. Bao, "Dita: Distributed in-memory trajectory analytics," in *Proceedings of the 2018 International Conference on Management of Data*, 2018, pp. 725–740.

[15] H. Hu, G. Li, Z. Bao, Y. Cui, and J. Feng, "Crowdsourcing-based real-time urban traffic speed estimation: From trends to speeds," in *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*. IEEE, 2016, pp. 883–894.

[16] "Google map." [Online]. Available: https://www.google.com/maps

[17] "Uber." [Online]. Available: https://www.uber.com/

[18] V. Primault, A. Boutet, S. B. Mokhtar, and L. Brunie, "The long road to computational location privacy: A survey," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 3, pp. 2772–2793, 2018.

[19] A. Sealfon, "Shortest paths and distances with differential privacy," in *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, 2016, pp. 29–41.

[20] I. Damgård, V. Pastro, N. Smart, and S. Zakarias, "Multiparty computation from somewhat homomorphic encryption," in *Annual Cryptology Conference*. Springer, 2012, pp. 643–662.

[21] H. Bast, S. Funke, P. Sanders, and D. Schultes, "Fast routing in road networks with transit nodes," *Science*, vol. 316, no. 5824, pp. 566–566, 2007.

[22] P. Sanders and D. Schultes, "Highway hierarchies hasten exact shortest path queries," in *European Symposium on Algorithms*. Springer, 2005, pp. 568–579.

[23] R. Geisberger, P. Sanders, D. Schultes, and D. Delling, "Contraction hierarchies: Faster and simpler hierarchical routing in road networks," in *Experimental Algorithms: 7th International Workshop, WEA 2008 Provincetown, MA, USA, May 30-June 1, 2008 Proceedings 7*. Springer, 2008, pp. 319–333.

[24] R. Zhong, G. Li, K.-L. Tan, L. Zhou, and Z. Gong, "G-tree: An efficient and scalable index for spatial search on road networks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 8, pp. 2175–2189, 2015.

[25] A. V. Goldberg and C. Harrelson, "Computing the shortest path: A search meets graph theory." in *SODA*, vol. 5, 2005, pp. 156–165.

[26] J. Bater, X. He, W. Ehrich, A. Machanavajjhala, and J. Rogers, "Shrinkwrap: efficient sql query processing in differentially private data federations," *Proceedings of the VLDB Endowment*, vol. 12, no. 3, 2018.

[27] J. Bater, Y. Park, X. He, X. Wang, and J. Rogers, "Saqe: practical privacy-preserving approximate query processing for data federations," *Proceedings of the VLDB Endowment*, vol. 13, no. 12, pp. 2691–2705, 2020.

[28] A. C. Yao, "Protocols for secure computations," in *23rd annual symposium on foundations of computer science (sfcs 1982)*. IEEE, 1982, pp. 160–164.

[29] M. Keller, "MP-SPDZ: A versatile framework for multi-party computation," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020. [Online]. Available: https://doi.org/10.1145/3372297.3417872

[30] "Multi-protocol spdz." [Online]. Available: https://github.com/data61/MP-SPDZ/

[31] H. Bast, D. Delling, A. Goldberg, M. Müller-Hannemann, T. Pajor, P. Sanders, D. Wagner, and R. F. Werneck, "Route planning in transportation networks," *Algorithm engineering: Selected results and surveys*, pp. 19–80, 2016.

[32] L. Wu, X. Xiao, D. Deng, G. Cong, A. D. Zhu, and S. Zhou, "Shortest path and distance queries on road networks: An experimental evaluation," *arXiv preprint arXiv:1201.6564*, 2012.

[33] S. Jung and S. Pramanik, "An efficient path computation model for hierarchically structured topographical road maps," *IEEE Transactions on Knowledge and Data Engineering*, vol. 14, no. 5, pp. 1029–1046, 2002.

[34] M. Holzer, F. Schulz, and D. Wagner, "Engineering multilevel overlay graphs for shortest-path queries," *Journal of Experimental Algorithmics (JEA)*, vol. 13, pp. 2–5, 2009.

[35] A. V. Goldberg and R. F. F. Werneck, "Computing point-to-point shortest paths from external memory." *ALENEX/ANALCO*, vol. 2, 2005.

[36] O. Goldreich, *Foundations of cryptography: volume 2, basic applications*. Cambridge university press, 2009.

[37] R. Cramer, I. Damgård, and J. B. Nielsen, "Multiparty computation from threshold homomorphic encryption," in *Advances in Cryptology—EUROCRYPT 2001: International Conference on the Theory and Application of Cryptographic Techniques Innsbruck, Austria, May 6–10, 2001 Proceedings 20*. Springer, 2001, pp. 280–300.

[38] M. Keller and K. Sun, "Secure quantized training for deep learning," in *International Conference on Machine Learning*. PMLR, 2022, pp. 10 912–10 938.

[39] C. A. Crane, *Linear lists and priority queues as balanced binary trees*. Stanford University, 1972.

[40] Y. Yuan, D. Ma, Z. Wen, Z. Zhang, and G. Wang, "Subgraph matching over graph federation," *Proceedings of the VLDB Endowment*, vol. 15, no. 3, pp. 437–450, 2021.

[41] Y. Wang, Y. Tong, Z. Zhou, Z. Ren, Y. Xu, G. Wu, and W. Lv, "Fed-ltd: Towards cross-platform ride hailing via federated learning to dispatch," in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022, pp. 4079–4089.

[42] Y. Wang, K. Li, Y. Luo, G. Li, Y. Guo, and Z. Wang, "Fast, robust and interpretable participant contribution estimation for federated learning," in *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE, 2024, pp. 2298–2311.

[43] I. I. Consortium, "Design considerations and guidelines for implementing federated learning in smart manufacturing applications," 2022. [Online]. Available: https://www.iiconsortium.org/news-pdf/joi-articles/2022-March-JoI-Design-Considerations-and-Guidelines-for-Implementing-Federated-Learning-in-Smart-Manufacturing-Applications.pdf

[44] D. Ouyang, L. Qin, L. Chang, X. Lin, Y. Zhang, and Q. Zhu, "When hierarchy meets 2-hop-labeling: Efficient shortest distance queries on road networks," in *Proceedings of the 2018 International Conference on Management of Data*, 2018, pp. 709–724.

[45] J. Sankaranarayanan and H. Samet, "Query processing using distance oracles for spatial networks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 8, pp. 1158–1175, 2010.

[46] S. Huang, Y. Wang, T. Zhao, and G. Li, "A learning-based method for computing shortest path distances on road networks," in *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 2021, pp. 360–371.