Arboretum: A Planner for Large-Scale Federated Analytics with Differential Privacy

Elizabeth Margolin*
University of Pennsylvania

Karan Newatia*
University of Pennsylvania

Tao Luo
University of Pennsylvania

Edo Roth+
University of Pennsylvania

Andreas Haeberlen
University of Pennsylvania / Roblox

Abstract

Federated analytics is a way to answer queries over sensitive data that is spread across multiple parties, without sharing the data or collecting it in a single place. Prior work has developed solutions that can scale to large deployments with millions of devices but, due to the distributed nature of federated analytics, these solutions can support only a limited class of queries – typically various forms of numerical queries, which can be answered with lightweight cryptographic primitives. Supporting richer queries, such as categorical queries, requires heavier cryptography, whose cost can quickly exceed even the resources of a powerful data center.

In this paper, we present Arboretum, a new federated analytics system that can efficiently answer a broader range of queries, including categorical queries, in deployments with millions or even billions of participants. Arboretum achieves this by 1) automatically optimizing query plans to find highly efficient ways to answer each query, and by 2) including the participant devices in the computation. Our evaluation shows that Arboretum can match the cost of earlier systems that have been hand-optimized for particular kinds of queries, and that it can additionally support a range of new queries for which no efficient solution exists today.

CCS Concepts: • **Security and privacy → Privacy-preserving protocols**; Distributed systems security.

Keywords: federated analytics, differential privacy, query planner

*These authors contributed equally to this work

+This work was done while at the University of Pennsylvania. The author is now at Google.

SOSP '23, October 23–26, 2023, Koblenz, Germany

ACM ISBN 979-8-4007-0229-7/23/10…$15.00
https://doi.org/10.1145/3600006.3624566

1 Introduction

The age of "big data" has brought a wealth of massive data sets that could be used to answer a wide spectrum of interesting questions. For instance, a mobile-device manufacturer might look for which apps cause a large battery drain [7], and a medical researcher might look for combinations of drugs, activities, and dietary choices that trigger rare side effects. The necessary data is already available on many personal devices, such as laptops and phones, so these questions could *in principle* be answered today. In practice, however, this kind of analysis would often raise serious privacy concerns: few users would feel comfortable sharing data about their medications or daily activities with a third party, even one they normally trust.

Recent work on federated analytics [48] has developed a safer way to perform this kind of analysis. This approach relies on a combination of two key ideas. First, rather than collecting the sensitive data in a central place, federated analytics answers queries by running a distributed, encrypted protocol between the participant devices; thus, the analyst never gains access to the users' raw data. And second, federated analytics typically provides a strong privacy guarantee, such as differential privacy [30], which limits how much the analyst could learn, in the worst case, about any individual user's data. A key challenge with this approach is scale: real deployments can have millions or even billions of devices. For instance, Apple currently has about 1.8 billion active devices, and similar deployments exist at Google [8], Microsoft [27], and Snap [47]. However, some systems, such as Honeycrisp [53] and Orchard [54], can work efficiently even at this scale.

Nevertheless, existing systems for large-scale federated analytics share an important limitation: they support only a fairly small range of queries – typically numerical queries that can be expressed in terms of counts and sums [54]. The reason is that these queries can be answered using relatively lightweight cryptographic primitives, so the computational cost for the analyst remains relatively low. Differential privacy itself could support a much wider range of queries, including the categorical queries we have mentioned above, and these queries can theoretically be answered with techniques such as fully homomorphic encryption (FHE) or multiparty

computation (MPC). However, at least with current technology, the computational cost at scale would be astronomical, and would far exceed even the resources of even a modern, well-equipped data center.

In some cases, the cost can be reduced with a carefully crafted protocol and/or with specialized cryptographic primitives. For instance, Böhler et al. [14] uses this approach successfully, and can answer categorical queries with roughly a million participants. However, designing such a protocol requires a lot of expertise, and the solutions tend to be highly query-specific, so this approach does not scale well. Moreover, even the carefully crafted solutions tend to be quite expensive, which limits the scale that can be achieved in practice. For instance, the million participants that Böhler's protocol reaches is still orders of magnitude smaller than some of the deployments that exist today.

In this paper, we present *Arboretum*, a query planner for federated analytics systems that can solve both of these problems. Arboretum is based on two key insights. The first is that, while the massive number of user devices is certainly a challenge because of the enormous amount of computation power it requires, *it is also an opportunity* because the devices can help with the computation. True, each individual device (e.g., a cellular phone or a laptop) has very limited resources, but, due to the sheer number of devices, even small contributions from a subset of the devices can add up to an enormous amount of computation power. For instance, a Dell PowerEdge R7525 server has a multi-core Geekbench score of 67,954, whereas a second-generation iPhone SE has only 3,027. However, a billion iPhones computing for one second each can still outperform 10,000 servers computing for an hour. Moreover, leveraging the user devices enables organic scaling: adding devices increases both the demand for resources *and* the supply. This approach has been used successfully, e.g., for CDNs [60] and decentralized systems [37, 51]. As we show here, it also holds promise for federated analytics.

Our second insight is that the design space has a fairly regular structure: queries contain high-level operators that can be instantiated in different ways (for instance, sums can be computed with sum trees of different fanouts, with a regular for loop, etc.), and computations can be carried out by different entities and/or with different cryptographic primitives. In our experience, the main sources of complexity are the sheer size of the design space, where even simple queries can be executed in millions of different ways, and the complicated dependencies. For instance, using a slightly slower implementation for one operator can massively speed up another, and using a particular cryptographic primitive might speed up additions but slow down comparisons, which can vastly increase or decrease the overall speed, depending on what else the query is doing. These dependencies are hard to track for a human developer, but they are easy to explore mechanically, using a type of query planner.

Arboretum accepts queries written in a simple high-level language and then automatically finds a way to execute them efficiently in a federated setting, spreading the computation across servers and/or clients as necessary. The queries can be formulated as if all the data existed in a central place, without regard to distribution or confidentiality; Arboretum automatically breaks the computation into smaller blocks, assigns the blocks to suitable entities for computation (possibly using homomorphic encryption or MPC), and chooses an efficient cryptosystem. We have implemented a prototype of Arboretum and applied it to ten differentially private queries, including six new queries that cannot be executed efficiently at scale by any current solution we are aware of. Our contributions are:

- Arboretum's query planner (Section 4);
- a generalized query runtime (Section 5);
- a prototype implementation (Section 6); and
- an experimental evaluation (Section 7).

2 Background

We begin by sketching three key technologies we use in Arboretum: differential privacy, multi-party computation, and homomorphic encryption.

2.1 Differential Privacy

To protect the participants' sensitive data, our goal is to support *differential privacy* [30], the current gold-standard privacy definition, which has a rigorous mathematical foundation and a rich literature of applications. Informally, we can imagine a query as computing some randomized function f over a hypothetical database that contains a row of data for each user; f is differentially private if changing any single row changes the probability of any output, or set of outputs, by at most a negligible amount. Formally, if d_1 and d_2 are two databases that differ in a single row, we say that f is (ε, δ)-*differentially private* iff, for any set of possible outputs S,

$$Pr[f(d_1) \in S] \leq e^{\varepsilon} \cdot Pr[f(d_2) \in S] + \delta$$

Here, ε and δ are parameters that control the strength of the privacy guarantee. A common recommendation is that $\varepsilon < 1$ (say, $\varepsilon = 0.1$) and that δ should be smaller than $1/N$, where N is the total size of the database.

Mechanisms: A variety of mechanisms for achieving differential privacy have been proposed. For numerical queries, a common choice is the *Laplace mechanism* [30], which works by adding a bit of random noise to the result of the query. Suppose the user is asking to compute a function f, and suppose f has sensitivity s – that is, changing any single row in the database can change the value of f by at most s. Then $f' := f + Lap(s/\varepsilon)$ is $(\varepsilon, 0)$-differentially private. For categorical data, a common choice is the *exponential mechanism* [45]. Suppose each user's data is from some discrete range R (the possible "categories" to which the user can belong), and we

	FHE	All-to-all MPC	Böhler [14]	Orchard [54]	Arboretum
Aggregator computation	**O(N) → Years**	N/A	N/A	Hours	Hours
Participant bandwidth (typical)	MBs	**O(N) → PBs**	KBs	MBs	MBs
Participant bandwidth (worst-case)	MBs	**O(N) → PBs**	**O(N) → TBs**	GBs	~1 GB
Numerical queries	Yes	Yes	Yes	Yes	Yes
Categorical queries	Yes	Yes	Yes	**Limited**	Yes
Participants can contribute	**No**	Yes	**1 committee**	**1 committee**	Yes
Optimization	**No**	**No**	**No**	**No**	Automatic

Table 1. Approaches for supporting queries with hundreds of millions of participants.

have a quality score $q(r, d)$ that defines, for each $r \in R$, how "useful" output r would be if the database were d. If the quality score q is Δ-sensitive in its database argument, the exponential mechanism then outputs each possible $r \in R$ with probability proportional to $e^{\epsilon q(d,r)/(2\Delta)}$; this once again results in $(\varepsilon, 0)$-differential privacy. An alternative but equivalent implementation is to compute $q'(d, r) := q(d, r) + Gumbel(2\Delta/\varepsilon)$ and to then return $f'(d) := \text{argmax}_r\ q(d, r)$ [31, §3.4]. When releasing multiple outputs (e.g., the k most frequent items), one can either draw Gumbel noise k times to maintain $(\epsilon, 0)$-differential privacy or simply add noise once and release the outputs with the k highest scores, which yields $(\sqrt{k} \cdot \epsilon, 0)$-differential privacy [29].

Sampling: Another useful primitive is *secrecy of the sample* [56]. Let $\sigma(d, \phi)$ be a function that selects each element of some database d with probability ϕ, and suppose we have a query $f(d)$ that is $(\epsilon, 0)$-differentially private. Then $f(\sigma(d, \phi))$ is $(ln(1 + \phi(e^\epsilon - 1)), 0)$-differentially private, *as long as* nobody can observe which elements have been selected. When $\epsilon \leq 1$ and ϕ is sufficiently small, this is close to $(\frac{2\phi}{\epsilon}, 0)$-differentially private.

2.2 MPC and AHE/FHE

Next, we briefly introduce two cryptographic primitives. The first is homomorphic encryption. Its simplest form, *additively homomorphic encryption (AHE)*, is an encryption scheme with one extra property: if $E(a)$ and $E(b)$ are two ciphertexts that encode a and b, respectively, there is an operation $\boxplus$ such that $E(a) \boxplus E(b) = E(a + b)$ – in other words, $\boxplus$ can "add" two ciphertexts such that the result is a new ciphertext that encrypts the sum. More recently, *fully homomorphic encryption (FHE)* schemes have been introduced; these additionally support an operator $\boxtimes$ such that $E(a) \boxtimes E(b) = E(a \cdot b)$. With both additive and multiplicative homomorphisms, it is possible, at least in principle, to evaluate arbitrary functions on ciphertexts. There are several different FHE schemes with different tradeoffs: for instance, schemes like TFHE [24] are often used to compute Boolean circuits over encrypted bits, whereas others, such as BGV [18], are more commonly used for numeric operations. The former is more efficient for logical operations and comparisons, while the latter is more efficient for additions and multiplications.

The second cryptographic primitive is *multi-party computation (MPC)* [58], which is a way for $m \geq 2$ parties to jointly evaluate a function f on some secret inputs from each party without revealing anything about the inputs, except what the output of f already implies. There are a variety of different MPC frameworks that have been developed, but most relevant for Arboretum is the *honest majority* setting.

3 Overview

In this paper, we focus on a scenario with a large number of *participants*, a central *aggregator*, and at least one *analyst*. Each participant owns a small device – say, a laptop or a cellular phone – that contains some potentially sensitive data. The analyst would like to answer some questions about the combined data of all the users; the aggregator provides some resources and also acts as a coordinator for the computation. We have the following four goals:

- **Accuracy:** Queries should be answered correctly, and as precisely as possible.
- **Integrity:** Malicious participants should not have a disproportionate effect on accuracy.
- **Privacy:** Neither the aggregator nor other participants should learn much about an individual participant.
- **Efficiency:** The cost of answering queries should be reasonable, with up to a billion participants.

3.1 Threat model

Smaller-scale systems often assume that up to a third of the nodes can be Byzantine [41], but, for systems with millions of nodes, this seems overly pessimistic. Instead, we follow earlier systems in this space [52, 54] and make the *OB+MC assumption* [53], which consists two parts:

Occasionally Byzantine (OB): The aggregator is honest when the system is first started but could occasionally be Byzantine for short periods after that. We expect that an aggregator with a million-user deployment would typically be a large organization with a reputation to lose, so it would probably not be structurally evil, although it could briefly "misbehave" when it is hacked, or due to a malicious insider. Because of this, the aggregator should not be fully trusted. If a Byzantine period should occur, our only goal is to preserve the privacy of the participants; we do not attempt to prevent a Byzantine aggregator from damaging its own system.

Mostly Correct (MC): The participants are mostly correct, except for a small fraction (say, 3–5%), which can be Byzantine. Although a small fraction of participant devices could always be hacked or controlled by malware, even a large botnet with millions of devices would only represent a tiny fraction of a billion-device deployment.

3.2 Strawman solutions

We begin by discussing a few strawman solutions for answering the query *"Which zip code in the United States contains the most participants?"* with 10^8 participants. This query can be answered with the exponential mechanism. For reference, there are 41,683 zip codes in the United States. Table 1 shows a side-by-side comparison.

FHE only: In principle, each participant could encrypt its data with FHE and upload it to the aggregator, who could evaluate the query on the ciphertexts, add the requisite amount of noise, and decrypt only the final result. This is more or less the classic setting for centralized differential privacy, plus encryption. However, at the scale we consider here, this approach requires a gigantic amount of computation, since the quality score $q(d, r)$ would have to be evaluated separately for each possible zip code. We estimate that, with 10^8 participants, this would require a 40-*trillion*-gate circuit that, with current technology, would take years to evaluate. In addition, the aggregator would hold the private key and would thus have to be trusted to decrypt only the final result and not the individual uploads.

All-to-all MPC: The participants could also input their zip code to a large MPC that evaluates the query and returns the final result. This solves the privacy issue from above, but it is even worse in terms of cost, since the per-participant bandwidth scales at least linearly with the number of participants. We are not aware of any practical MPC protocol that can scale beyond a few hundred parties.

MPC committee: Böhler and Kerschbaum [14] delegates the MPC to a small committee of participant devices. This scales better – at least to a few million participants – but the committee eventually does become a bottleneck: with 10^8 participants, even downloading the input data from each participant would already generate gigabytes of traffic, and evaluating a huge circuit in MPC would add hundreds of gigabytes more.

HE + MPC committee: Orchard [54] avoids this bottleneck by having the aggregator sum up the input data, using homomorphic encryption, and by using the MPC committee only for key generation, noising, and decryption. Orchard does scale to 10^8 participants, but it supports mostly Laplace-mechanism queries that can be expressed as sums plus some postprocessing. The exponential mechanism is supported, but only for tens of categories, which is nowhere near the number of possible zip codes. Anything larger would quickly exceed the computation power of the adversary and/or of the (single) committee of user devices.

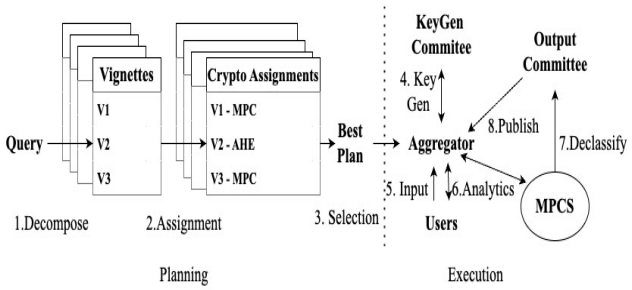

Figure 1. Arboretum Overview

3.3 Challenges

As the examples above show, one key problem is that the exponential mechanism tends to require far more computation than the Laplace mechanism. Our zip code problem requires evaluating the quality score $q(r, d)$ for *each* possible output r – i.e. the 41,683 possible zip codes. This alone increases the computation cost dramatically. In addition, the exponential mechanism also requires more expensive *kinds* of computation: for instance, choosing the highest quality score requires comparisons, which cannot be done in AHE alone and thus requires FHE.

Of course, optimizations can often bring the cost back down to a practical level – but there are lots of different optimizations that make sense in different parts of the design space, and these optimizations interact in complicated ways that are hard for a human developer to track. For a simple query with two parameters and six metrics, the full possibility space of feasible, let alone optimal, plans has eight dimensions (see Section 4.2). It is difficult for a human developer to pick the "correct" combination of optimizations for a particular scenario, and the cost of picking a suboptimal combination can be substantial.

3.4 Approach

Our proposed solution, Arboretum, is based on two key observations. The first is that *having a large number of participants is both a challenge and a blessing*: if each participant device helps just a little bit with the computation, this creates a massive pool of additional resources that can be used to process richer queries, even if they are beyond the reach of the aggregator alone. As previous systems in other domains, such as the NetSession CDN [60] or the PIER distributed query engine [37], have shown, this approach can create "organic scalability": adding more participants increases the resource demand, but also the resource supply, assuming that the users

can be incentivized to participate.[1] As with the earlier systems, the key question is how to efficiently distribute the work so that small devices, such as phones or laptops, can make a meaningful contribution. Arboretum achieves this by breaking query plans into small, bite-size pieces that are each within the means of a small device.

The second key observation is that, although large and full of complex dependencies, *the design space is regular enough to be explored mechanically*: high-level operators can be instantiated in different ways, the program can be segmented and transformed differently, and there are various parameters to be chosen. This is roughly analogous to query planning in a traditional DBMS, although of course the operators and transformations themselves are quite different. Thus, as we will show, it is possible to build a "query planner" for federated analytics that automatically finds a very good plan for most queries, without manual optimization or expert knowledge on the part of the analyst.

3.5 Roadmap

Figure 1 provides a high level overview of Arboretum. At a high level, there are two phases: a planning phase (§4) and an execution phase (§5). The planning phase begins when the analyst submits a query written in a simple query language (§4.1), along with a set of optimization goals, such as low bandwidth, and a set of constraints (§4.2). Arboretum then generates many possible query plans (§4.3), breaks them into small "vignettes" that can be assigned to either the aggregator or to small committees of user devices (§4.4), assigns suitable cryptographic primitives (§4.5), and then picks the plan that best achieves the chosen goal (§4.6). In the execution phase, Arboretum uses sortition to pick a set of committees (§5.1), including a special committee that generates a keypair and publishes its public key (§5.2). The user devices then encrypt their data with this key (§5.3), and Arboretum evaluates the various vignettes using FHE and/or MPC (§5.4), and finally decrypts the output of the last vignette (§5.5), which is the answer to the query.

4 Query planning

In this section, we describe how queries are formulated in Arboretum, and how Arboretum chooses a plan for each query.

4.1 Input language

Analysts formulate queries as if they ran on a single machine that has access to the entire data set, without considering distributed execution or encryption. Arboretum uses a simple imperative language for this that is loosely based on Fuzzi [59], although other languages could be used instead. Figure 2 shows the syntax, which includes loops, conditionals, arrays, and the standard arithmetic and logical operators. The participants' input data is available as a predefined two-dimensional array: `db[i][j]` contains the *j*.th input from participant *i*. The program's output(s) are returned by calling the `output` function.

stmt := *stmt*; *stmt* | *var* = *exp* | *exp* | *var*[*exp*] = *exp* |
for *var* = *exp* **to** *exp* **do** *stmt* **endfor** |
if *expr* **then** *stmt* **else** *stmt*
exp := *exp op exp* | *var* | *var*[*exp*] | *func*(*exp*,...) | *lit*
op := + | − | * | / | && | || | < | <= | > | >= | ! | ==

Figure 2. Arboretum's query language.

Arboretum supports several built-in functions: simple mathematical operations (`exp`, `clip`, etc.), aggregations over arrays (`sum`, `max`), a uniform sampling function (`sampleUniform`), the Laplace mechanism (`laplace`), and the exponential mechanism (`em`). As we will see in Section 4.3, some of these functions can be instantiated in several different ways. We use high-level operators in the input language because it helps with certifying differential privacy, and also because we do not expect the typical analyst to know, or care about, low-level implementation details.

Figure 3 shows a simple example program (`top1`) that we will use as a running example. Each participant *i* belongs to one of several categories (say, hair color) and sets `db[i][k]` to 1 for the relevant category and to 0 otherwise. The program sums up `db` to obtain a vector of quality scores, which are simply the number of participants that belong to a given category, and then invokes the exponential mechanism to select a category, which it then returns. Notice that the program is written as if `db` existed on a single machine, and that no cryptography is being used. Distribution and encryption are handled transparently by Arboretum.

4.2 Constraints and goals

Along with the program, the analyst specifies an optimization goal and, optionally, a set of limits on the costs of solutions they are willing to accept. Our prototype supports six metrics that can be used to express these: two consider the aggregator (computation time and bytes sent), and the remaining four consider participant devices (expected and maximum computation time, and expected and maximum bytes sent). For participant devices, the expected and maximum values differ because only a few devices are selected to serve on a committee, but these devices will have a higher cost. Other metrics, such as energy, should not be difficult to add if desired.

Arboretum discards any query plans that exceed the specified limits, and, among the remaining plans, returns the "best" one according to the chosen goal. For instance, an analyst could specify that the aggregator must not spend more than 1,000 core-hours and that user devices must not be asked to

[1] We do not focus on specific incentives here because this question is not specific to Arboretum: it is common to all systems that rely on user participation. Standard solutions should apply: if Arboretum is part of a larger system that users enjoy, they could be asked to opt in; in a research study, standard incentive methods (raffles, lotteries, ...) could be used; etc.

```
aggr = sum(db);
result = em(aggr);
output(result);
```

Figure 3. A simple example query (`top1`).

send more than 500 MB, and they could ask for the plan with the lowest expected computation time on participant devices.

Once a query has been submitted, Arboretum attempts to certify that the query is differentially private, and to determine a sensitivity bound. Since this step is not the focus of our paper, we simply adopt a method from prior work – specifically the approach from Fuzzi [59], which our query language is based on. Fuzzi handles both explicit and implicit flows, and it can certify many kinds of queries automatically, without help from the analyst. However, other approaches could be used instead; for instance, CertiPriv [10] would enable analysts to supply their own proofs of privacy, and thus allow Arboretum to accept queries where automatic certification fails. If the goal is to shield the analyst from additional details, such as the choice of specific mechanisms for differential privacy, a declarative input language could be used instead.

4.3 Program transformations

Once a query has been certified as differentially private, Arboretum transforms it into a query plan it can actually execute. This involves three steps: 1) replacing each abstract high-level operation, such as `sum` or `em`, with a concrete implementation; 2) deciding whether each step of the query should be performed by the aggregator, by a committee of devices, or by the participant devices themselves; and 3) adding suitable encryption to maintain confidentiality. Each of these steps can be done in several different ways, so, by trying all combinations, Arboretum can usually generate a large number of candidate plans for a given query.

We begin by discussing the first step. Many of Arboretum's abstract operations can be implemented in several different ways. One simple example is the `sum` operator that sums up the contents of an array: when the sum is computed by the aggregator, a simple for loop will do – but if the sum is computed by committees, a sum tree is better, because it can be spread across several committees. However, there is no single "best" degree for this tree! On the one hand, larger degrees will require fewer committees, so the cost of starting a committee can be amortized better and the *expected* cost is lower; on the other hand, lower degrees require each committee to do less work and thus lead to a lower *maximum* cost. A similar tradeoff exists with the `max` and `argmax` operators.

A more complicated tradeoff exists for the exponential mechanism. Figure 4 shows two possible instantiations of the `em` operator. On the left is a straightforward implementation of the textbook approach from [31, §3.4], which exponentiates the scores `s` to form an array `es`, then draws a random value `r` between 1 and `sum(es)`, and then returns the first category

```
function em(s)
 L=max(s)-11; // 16 bits
 for i=0 to len(s)-1 do
  if (s[i]>=L) then
   es[i]=exp((s[i]-L)*ϵ
   /(2*sens));
  else
   es[i]=0;
 r=random(sum(es));
 s[0]=0;
 for i=0 to len(s)-1 do
  s[i+1]=s[i]+es[i];
  if (r>=s[i]&&r<s[i+1])
   return declassify(i)
```

```
function em(s)
 for i=0 to len(s)-1
  ns[i]=s[i]+
   Gumbel(2*sens/ϵ);
 x = 0;
 for i=1 to len(s)-1
  if (ns[i]>ns[x])
   then x = i;
 return declassify(x)
```

Figure 4. Two instantiations of the `em` operator, based on exponentiation (left) and on Gumbel noise (right), respectively.

`i` such that the sum of the preceding elements of `es` is at most equal to `r`. Our only modification is that, since we have to work with finite-precision numbers, we normalize `es` to the range $[1, e^L]$ and ignore any elements with smaller scores; this results in (ϵ, δ)-differential privacy (see [44, §A]). On the right is a variant, based on [29], that adds Gumbel noise to each score and then returns the element with the largest noised score. The tradeoff between these variants is complicated and depends on whether they are executed in FHE or using MPC. Notice that both variants invoke a `declassify` function to indicate that their result is safe to release in unencrypted form (see Section 4.5).

4.4 Basic type inference; Vignettes

Once all high-level operators have been instantiated, Arboretum performs type inference to assign to each variable and each expression 1) a basic type (`int`, `fix`, or `bool`), and 2) a value range. The latter is important for deciding the parameters of the cryptosystems to be used (e.g., the plaintext modulus). The range bounds we infer are conservative; for instance, the lower and upper bounds for `a*b` are simply the products of the lower and upper bounds of `a` and `b`, respectively. However, the analyst can, if necessary, use the `clip` function to clip a variable to a smaller range.

Next, Arboretum decides which entity should execute each of the steps of the resulting program: the aggregator, a committee of participant devices, or a specific participant device. To this end, Arboretum breaks the program into short sequences of consecutive statements, which we call *vignettes*. The program thus becomes a sequence of vignettes, each of which is assigned to a particular entity. As a special case, a vignette that consists entirely of a data-parallel `for` loop can be *parallelized*, that is, its iterations can be assigned to different entities. For instance, a vignette that uses committees to compute a level of a `sum` tree can be parallelized, so that different committees compute the sum for different vertexes, and a vignette that encrypts the initial input can be parallelized so that each device encrypts its own data.

As a first approximation, Arboretum tries all possible combinations of vignette boundaries and locations. This seems

fine because we expect queries to be relatively short. However, we do implement a few simple heuristics to cut down the search space. We use branch-and-bound, by scoring the vignettes along the way (Section 4.6), and we discard partial solutions as soon as they exceed one of the analyst's limits or become worse than the best known solution. We do not allow constant assignments (such as `x = 0`) to run in a vignette by themselves, and we do not allow consecutive vignettes to run in the same location, since in that case they might as well be merged. The only exception is if both run on committees, which can make sense when there is a limit on the amount of computation a committee member may do.

4.5 Encryption-type inference

At this point, the program represents a distributed computation that returns the correct result; however, it does not yet ensure confidentiality, since the values are not yet encrypted. Arboretum's next step is to determine what needs to be encrypted, and how. This is done in three steps.

First, Arboretum identifies all values that need to be kept confidential. This includes anything that a) is derived directly or indirectly from the input database `db`, b) has not been passed through the `declassify` function, and c) is used in a vignette that runs on the aggregator or on individual participant devices. (Committees execute their vignettes using MPC, which already ensures confidentiality.) The only exception is that each participant device i is allowed to see its own input data `db[i]`. We use conservative taint tracking to find these values, starting from `db`.

Next, Arboretum adds encryption and decryption statements – initially without a specific cryptosystem. When a confidential value v is used in a participant or aggregator vignette, Arboretum inserts, right after the statement that creates v, a statement `v'=enc(v)` that creates an encrypted clone v'. It then replaces any instances of b in participant or aggregator vignettes with v'. When an encrypted value v' is passed to a committee vignette, Arboretum adds a statement `v''=dec(v')` at the beginning of that vignette and replaces any instances of v' with v''.

Third, Arboretum decides which cryptosystem to use for each value. If an encrypted value is only used in additions, it uses AHE, otherwise FHE. Whenever a cryptosystem is used for the first time, Arboretum inserts a key generation vignette at the beginning of the program and assigns it to a committee, to prevent any single entity from obtaining the private key.

Figure 5 shows, as an example, one of the candidates that are generated from the query in Figure 3 when there are 2^{30} participants and 10 possible outputs. Here, the `sum` operator has been instantiated with a simple AHE-based sum and the `em` operator has been instantiated with a version that uses Gumbel noise. Notice that the Gumbel noise for each possible output is generated in a separate committee, and that a vignette has been added at the beginning to generate the AHE keypair.

```
vignette (committee)
  ahePriv = aheKeygen();
  ahePub = pubkey(ahePriv);
parallel vignette (participant i)
  encdb[i] = aheEnc(db[i], ahePub);
vignette (aggregator)
  s = 0;
  for i=1 to 2^30 do
    s = s + encdb[i];
parallel vignette (committee i)
  ds[i] = aheDec(s, ahePriv)[i];
parallel vignette (committee i)
  ns[i] = ds[i]+Gumbel(2*sens/ε);
vignette (committee)
  x = 0;
  for i=1 to 10 do
    if (ns[i]>ns[x])
      then x = i;
  choice = declassify(x);
vignette (aggregator)
  output(choice);
```

Figure 5. One of the candidate programs that are generated from the query in Figure 3.

4.6 Scoring

At this point, the candidate is complete. Arboretum now estimates the cost of running the candidate, to see whether it meets the analyst's constraints (Section 4.2), and to see whether it is better than the best known candidate so far. Scoring is based on a simple cost model, which we have built by benchmarking each building block – such as FHE operations, MPC start-up cost, incremental MPC costs for computations, etc. – on a reference platform. The model needs to be generated only once; after that, we can score a given query simply by adding up all the costs for the operations it performs. If necessary, the manual benchmarking step could be avoided by using an automated cost modeling framework, such as CostCO [33].

This approach obviously does not yield the *exact* costs of running a query, for many reasons. For instance, the devices that are used by the actual participants could be different from our reference platform (or it could be a heterogeneous mix of devices) and the building blocks we use, such as the MPC frameworks, could apply their own internal optimizations that could cause the total cost of a computation to differ from the costs of the individual operations. However, recall that we do not use scoring to predict the actual cost, but rather to weed out expensive candidates. Even a rough cost model should suffice for this purpose, although of course any inaccuracies could cause the chosen candidate to be somewhat more expensive than the 'true' optimal candidate.

5 Execution

Next, we show how Arboretum executes its chosen plan.

5.1 Setup

When query execution begins, Arboretum chooses the devices that will serve on each committee. Arboretum leverages

MPCs for the honest-majority setting. Thus, the committee members must be chosen uniformly at random to prevent an adversary (either the aggregator or subset of the users) from biasing membership towards their confederates.

We generalize the sortition mechanism from Honeycrisp [53, §3.2] for this. The system starts with a random "block" of bits B_0, which is chosen during a trusted setup phase (recall from Section 3.1 that the aggregator is trusted at the beginning), as well as a Merkle tree M_0 that contains the registered devices. When the i.th query is submitted, each device signs a message $(B_i, i, 0)$ using a deterministic signature scheme (such as RSA with deterministic padding) and hashes the signature. The $c \cdot m$ devices with the lowest hashes then form the committees: the device with the x.th-lowest hash joins committee $\lfloor x/m \rfloor$. Thus, each device serves on at most one committee.

Notice that the minimum committee size m depends on the number of committees c: we need an honest majority in *all* c committees with high probability, even if some devices go offline. Suppose we want to tolerate up to a fraction g of the m members of each committee going offline, without incurring any additional costs. Since the malicious members could all conspire to remain online, we need to ensure that there is still an honest majority among the remaining $(1-g)\cdot m$ members. So we choose m to be the smallest number such that $1 - \left(\sum_{i=\lfloor 0..\frac{(1-g)\cdot m}{2} \rfloor} \binom{m}{i} f^i (1-f)^{m-i}\right)^c \leq p_1$, where p_1 is the desired upper bound on the probability of a privacy failure in one round. (If the system is expected to run up to R rounds and the desired probability of *ever* experiencing a privacy failure is p, we can choose p_1 such that $p = 1 - (1-p_1)^R$.) Since the number of committees can vary between query plans, Arboretum calculates the minimum committee size for a given query plan before scoring it, so it can estimate the cost of the MPCs correctly. If more than $g \cdot m$ members of some committee i go offline at the same time, Arboretum can reassign i's tasks to committe $i + 1 \mod c$.

5.2 Key generation

The first committee serves as the key generation committee, which is responsible for generating the public and private AHE/FHE keys. This committee has two functions. First, it uses the sensitivity bound from Section 4.2 to check whether the balance in the analyst's "privacy budget" is sufficient to run the query. If it is not, the query fails. If it is, the committee jointly signs a *query authorization certificate* [53, §3.4], which contains the public AHE and/or FHE keys, the query sequence number, the query plan, the remaining budget balance for use by the next query's key generation committee, a fresh Merkle tree M_i with the currently registered devices, and a new block B_{i+1} of random bits, which the committee jointly generates in an MPC as $\oplus_j x_j$, where the x_j are random bits input by committee member j. This certificate is sent to the aggregator, which publishes it. (The inclusion of M_i prevents "computational grinding" attacks in which a Byzantine aggregator, knowing the value of B_{i+1}, tries lots of new keypairs to find ones that will be chosen for committees.)

The private key is securely transferred via secret shares to the committee which will serve as the decryption committee using a Verifiable Secret Redistribution (VSR) scheme [35], similar to protocols such as Mycelium [52]. As long as both the encryption and decryption committee have an honest majority, VSR ensures that the decryption committee can reliably reconstruct the private key. Additionally, VSR prevents malicious members of different committees from colluding to recover the private key.

5.3 Input

Using the public key generated in the previous step, users encrypt their local data and send both their encrypted data and a zero-knowledge proof (ZKP) validating correct formatting to the aggregator. The ZKPs guarantee that malicious participants cannot corrupt the results by submitting malformed inputs – say, by pretending that their user is 1,000 years old, or by submitting an input which is not an one-hot encoding of the participant's local value. The aggregator then verifies the ZKPs uploaded by participants. Input data never leaves any individual device unless it has been encrypted under an HE scheme. Although malicious users can skew the results by providing well-formed but incorrect inputs, the effect should be roughly proportional to the fraction of users that are malicious, which we have assumed to be small (Section 3.1).

To protect against a malicious aggregator, Arboretum also requires the aggregator to build a Merkle hash tree (MHT) with the results of the individual steps in the leaves (excluding only the final output step). Each participant device then picks some leaves at random and challenges the aggregator to return a) the contents of this leaf, and b) an inclusion proof. The devices then verify the steps they have audited. The number of leaves each device audits is chosen such that the probability of missing an incorrect step is smaller than some system parameter $p_{\max}$.

5.4 Processing

Next, the query plan is executed step by step, one vignette at a time. For vignettes involving individual users or the aggregator, integrity is maintained using ZKPs for malicious users and via MHT verification for a malicious aggregator, similar to the previous step. For vignettes executed by individual users, the aggregator checks the ZKPs and ignores inputs from devices that fail to provide a correct proof.

For vignettes involving MPCs, the committees are chosen from the sortition phase discussed earlier. Assuming an honest majority, the privacy of each individual MPC vignette is protected by the security of the MPC protocol, and malicious members in an honest-majority committee cannot reconstruct the secret data. To guarantee privacy when data is sent between MPC vignettes, we generalize the idea from the key

generation step: the intermediate data generated at the end of an MPC vignette is securely transferred via secret shares to the next vignette involving an MPC using VSR [35]. Assuming each committee has an honest majority, VSR guarantees that an MPC can reliably reconstruct the secret data sent to it from the previous MPC via secret shares while preventing malicious members of different committees from colluding to recover the data in the clear.

When MPC vignettes are completed, the aggregator serves as a "mailbox" by accepting these outputs (secret shares) and making them available to the next vignette. Outputs are signed with the sender's key and encrypted with the recipient's key, so the aggregator cannot view these messages' contents. It can corrupt or drop them, but it could only harm itself by doing this (by causing the query to abort and not return any results, which we assume the aggregator is interested in); it would not impact privacy.

5.5 Output

The final vignette involves the output committee. This committee combines the secret shares of the individual members to obtain the final result, which it then releases to the analyst. Recall from Section 4.2 that the input program is certified as differentially private; since the transformations preserve this property (they do not affect *what* the query does, only *how* it does it), this output is safe to release. We have included a proof of correctness in [44, §B].

6 Implementation

For our experiments, we implemented a prototype of Arboretum's query planner in C++; this prototype has 11,787 lines of code. We provide some key details below.

Cryptosystems: Our prototype uses the BGV cryptosystem [18] for FHE. The specific parameters depend on the encryption types the query planner infers (Section 4.5), but a typical query with one-hot encoding uses a plaintext modulus to 2^{30} (enough to sum binary values across one billion users), a 135-bit prime for the ciphertext modulus, and a polynomial degree of 2^{15}. This results in over 256 bits of security [6].

MPCs, ZKPs, and VSR: For multi-party computation, we use the MP-SPDZ framework [39] – specifically, the SPDZ-wise Shamir program [23], where operations are performed in a finite field with a configurable prime modulus. This is a natural fit for BGV key generation and decryption, and it provides security against malicious parties as long as there is an honest majority. The SPDZ protocol is UC-secure [26], so, since Arboretum is using it in a black-box way, our MPCs remain secure under general composition [40]. For efficiency, the encryption, decryption, and key generation MPCs set the prime modulus to BGV's ciphertext modulus. For the zero-knowledge proofs, we use the ZoKrates toolbox [2], with the `bellman` [1] backend and the G16 scheme [36]. Since ZoKrates requires a trusted setup, we use the first committee to perform the necessary trusted setup (the same is done in Mycelium [52]). We use signed proofs to prevent replay proofs due to G16's malleability. For Verifiable Secret Redistribution, we obtained the source code from the authors of Mycelium [52], which implemented Extended VSR [35].

Cost model: Our prototype includes a cost model that is based on benchmarks of the various cryptographic primitives on a Dell PowerEdge R430 server with two 2.4 GHz E5-2620 CPUs and 64 GB of RAM. This model cannot exactly predict the costs of a query on a heterogeneous mix of devices, but it should be sufficient for ordering the solutions. For primitives with many settings, we benchmarked some representative settings and interpolated the others. Our model does account for some simple MPC optimizations, such as the fact that the first comparison is more expensive than subsequent comparisons because it requires the generation of multiplication triples. We include validation data for our model in [44, §C].

Precision: In MP-SPDZ, we use the `cfix` and `sfix` fixpoint types for operations with non-integer values. We set the fixpoint length to be 30 bits for the integer part and 16 bits of precision for the decimal part, which gives 40 bits of statistical security in the MPC programs. The use of fixpoint types avoids some of the complications with implementing differential privacy, such as irregularities due to floating-point implementations [46]; we additionally use base-2 for the exponential mechanism, as suggested by Ilvento [38], which also has better support in MP-SPDZ. As with most other implementations, the use of finite-range data types adds a small δ to the guarantee, since the tails of the Laplace and Gumbel distributions are cut to the representable value range.

Secrecy of the sample: We implement secrecy of the sample (Section 2.1) as follows. Let $\frac{x}{b}$ be a fractional approximation of the sample size, where b is the total number of bins in a standard ciphertext – for instance, setting $x = b/2$ would sample 50% of the participants. First, a committee samples a value j uniformly at random from 1 to b. Each participant device also randomly chooses an index i from 1 to b, and places their encrypted local input only in that i-th bin, setting all other bins to 0. They upload the result to the aggregator as usual. To sample from the desired range, the committee only decrypts bins from j to $j + x - 1$ (modulo b) – they can do this by summing over all bins in the aggregate ciphertext, but replacing the bin values with 0 when they fall outside this range. We discuss security of this protocol in [44, §D], but in essence, participant devices do not know which random value has been sampled by the committee (so they cannot force themselves to be included or excluded, or even know if they were sampled), and neither the committee nor the aggregator knows which bins the participant devices selected, so the secrecy of the sample is preserved.

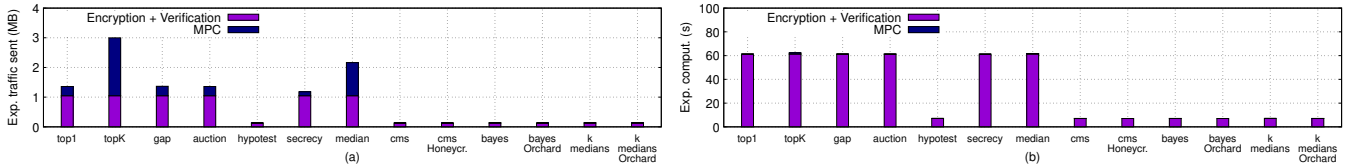

Figure 6. Expected bandwidth (a) and computation (b) required of each participant in a run of each algorithm.

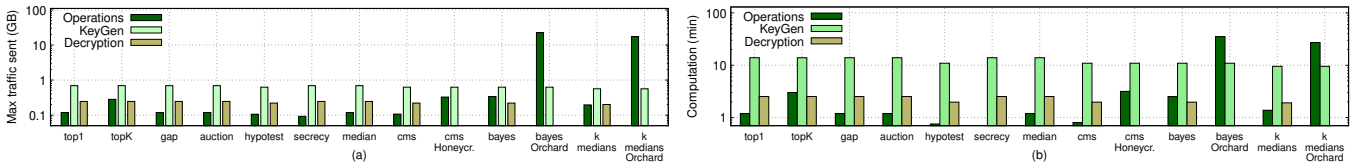

Figure 7. Actual bandwidth (a) and computation (b) required of participants serving on a committee, by committee type. The percentage of total participants which serve on a committee of any type ranges from 0.00022% (`k-medians`) to 0.49% (`topK`).

7 Evaluation

Arboretum extends the range of queries that can be answered at scale, but it also generalizes some existing solutions. To show this, we have chosen a mix of new and old queries for our evaluation, which is shown in Table 2. The first six queries are new: the first five use the exponential mechanism, and we are not aware of any other system that can answer them efficiently with billions of users and without a trusted aggregator. The remaining queries are adapted from earlier work: `cms` is the query from Honeycrisp [53], `bayes` and `k-medians` are two queries from Orchard [54], and `median` is the query from Böhler et al. [14], which can be easily extended to support quantiles. Since Orchard's query language is functional, we rewrote these queries in Arboretum's language; the other two systems are for specific queries and do not have a query language, so we implemented their queries in Arboretum. Table 2 also shows the number of lines for each query, to show that they can be formulated concisely in our language. For categorical queries (the first five in Table 2), we use a one-hot encoding for the categories. Our implementation of the `median` query also uses one-hot encoding and differs from the one in [14] in a few other details; we provide more information in [44, §E].

7.1 Experimental setup

We compare Arboretum to Orchard [54] and Honeycrisp [53]. We used the source code from [53, 54], and we closely follow the methodology in these papers, by benchmarking the individual building blocks and then extrapolating the overall costs of deployments with millions of nodes. For benchmarking, we used five PowerEdge R430 servers, each with twelve cores (2xE5-2620 at 2.4 GHz), 64 GB of RAM, and Fedora Server 34; to reduce interference, we pinned each participant process to a separate core. To further simplify the comparison, we used the same key parameters as [53, 54], unless stated otherwise: $N = 10^9$ participants, up to $f = 3\%$ malicious participants, and a 10^{-8} probability of privacy failure after running 1,000 queries. With $g = 0.15$ to tolerate a 15% churn, this leads to committee sizes of about 40 members (depending on the number of committees), which is the setting Orchard uses. We used C=1 categories for the `hypotest` and `cms` queries, C=10 for the `kmedian` query, C=115 for the `bayes` query (as in [54]), and C=2^{15} for the other queries. For `topK`, we used $k = 5$ to return the five most common items. To achieve a fair comparison to Orchard and Honeycrisp, which use SCALE-MAMBA for MPC, we reimplemented the MPCs for `cms`, `bayes`, and `k-medians` in the more efficient MP-SPDZ framework.

Query	Action	From	Lines
`top1`	Most frequent item	[31]	3
`topK`	Top-K selection	[29]	8
`gap`	Exp. mechanism with gap	[28]	8
`auction`	Unbounded auction	[45]	7
`hypotest`	Hypothesis testing	[20]	12
`secrecy`	Secrecy of sample	[9]	16
`median`	Median	[14]	39
`cms`	Count-mean sketch	[53]	5
`bayes`	Naïve Bayes	[54]	16
`k-medians`	K-Medians	[54]	30

Table 2. Supported queries.

We were not able to compare our `median` results to the original algorithm from Böhler and Kerschbaum [14] because the source code was not available and it was not clear from the paper how to calculate the ranks. However, based on [14, §E], a committee with $m = 10$ members required 1.41 GB of traffic for $N = 10^6$ participants; if we assume at least linear scaling in N and m, $m = 40$ and $N = 1.3 \cdot 10^9$ would require more than 7.3 TB of traffic, which is beyond the means of a typical participant.

7.2 Cost of running queries

We begin by examining the cost of running the queries in Table 2. We allow participant devices to send up to 4 GB of traffic and spend up to 20 minutes of computation time, and we limit the aggregator's computation to 1,000 core hours.

Figures 6(a) and 6(b) show the *expected* bandwidth and computation required of each participant, respectively; there is one bar for each query, which includes both the cost for normal participant-side computations and the expected cost

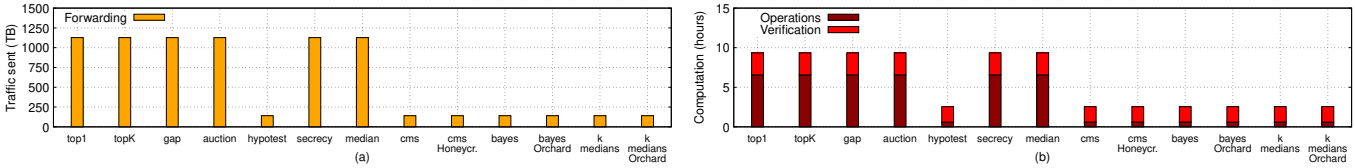

Figure 8. Bandwidth (a) and computation (b) required of the aggregator. (b) assumes that the aggregator has 1,000 cores.

of serving on a committee (that is, the actual cost of each committee type multiplied by the probability of being selected for a committee of that type). As expected, the figures show that the exponential mechanism has a much higher cost than the Laplace mechanism; the cost is particularly high for the `topK` query, which has to find the highest quality score k times. Nevertheless, the expected costs are low in absolute terms: each participant sends between 132 kB and 3 MB and spends between 7.1 s and 62.4 s of computation time.

Of course, if a participant is actually selected to serve on a committee, its costs are much higher than the expected costs; the precise amount depends both on the query and on the committee type. Figures 7(a) and 7(b) show these costs, with separate bars for each committee type. The key-generation committee is the most expensive; it consumes roughly 700 MB of traffic and 14 minutes of computation time. Although these costs are higher, they are still well within the means of a typical device (especially if the computation is done at night, while plugged in), and the odds of being selected for a committee are very low: for instance, with 10^9 participants, the `topK` query has one 42-member committee for key generation, 328 for decryption, and 115,334 for operations such as noising and computing the argmax. Thus, in a given run of `topK`, only 0.49% of the participants are serving on a committee of any type.

Figures 8(a) and 8(b) show the cost for the aggregator. Once again there is a clear difference between the exponential and Laplace mechanisms: the former involves more committees, so more bandwidth is spent on forwarding. (`hypotest` is an exception here because it has only a single category.) The bandwidth costs are high in absolute terms, but, on average, each of the 1 billion participants just receives about 1.1 MB, which is the size of a small image file. The computation time is below 10 hours when 1,000 cores (about 10 powerful servers) are used; most of the tasks are trivially parallelizable, so the time could be reduced by using more cores.

In the case of queries we took from Honeycrisp and Orchard, the figures also show the costs of the original system. These costs are almost identical to Arboretum's in expectation, however, the cost for committee members was much higher because Orchard does not leverage multiple committees. Notice that the original systems were custom-designed for these queries, whereas Arboretum was able to find these query plans independently, without human intervention.

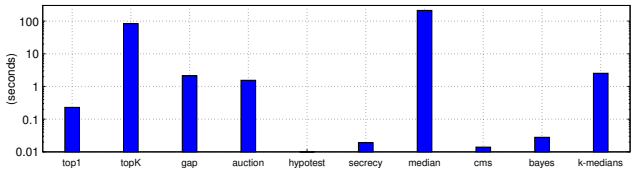

Figure 9. Runtime of the query planner.

7.3 Cost of query planning

Before Arboretum can run a query, it first needs to choose a good plan. As discussed in Section 4, this involves generating and scoring many different candidate plans. To see how expensive this step is, we ran the query planner on a PowerEdge R430 server for each of the queries from Table 2, and we measured the runtime until a plan was chosen.

Figure 9 shows our results. The runtime varies widely, from 10 ms (`hypotest`) to 212 s (`median`); this is because the more complex queries have a larger design space, with more possible combinations of operator expansions, more ways to divide them into vignettes, and more combinations of placement decisions. For instance, in the case of `median`, the query planner considers 1,251,001 different plan prefixes and 16 full candidate plans before it makes a decision. Since the queries themselves take hours to run (Section 7.2), it seems fine to spend a few minutes on planning. Ultimately, the end-to-end performance cost of Arboretum is dominated by the cost of execution, as shown in Figures 6–8.

To test whether our branch-and-bound heuristics are effective, we also ran the query planner with these heuristics disabled. This caused the planner to run out of memory for half of the queries; in the cases where it did terminate, it took between one and three orders of magnitude more time.

7.4 Effect on battery-powered devices

Although the probability of serving on a committee for any given query is very low, Figure 7 shows that the costs can be substantial if a device is selected. This is an important concern, particularly for mobile devices, which have limited battery capacity. To estimate the effect, we ran the most expensive MPC for each query with a party on a Raspberry Pi 4, and we used a USB power meter (TOL-15571) to measure the power consumption. The Raspberry Pi 4 is a reasonable proxy for a mobile device: it has a 1.8GHz Cortex-A72 CPU, which was used in mobile phones around 2015. To get only the power used for the computation, we also measured, and subtracted, the baseline power draw when the CPU was idle. We measured the overall power draw of the MPCs, which includes both computation and communication.

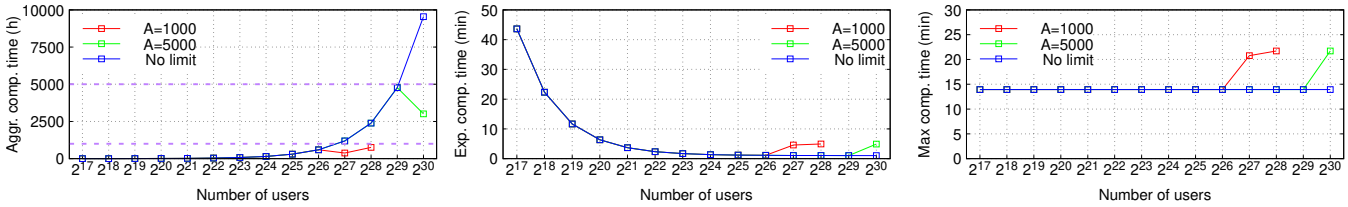

Figure 10. Scalability of computation cost for the aggregator (a), and expected (b) and maximum (c) cost for participants.

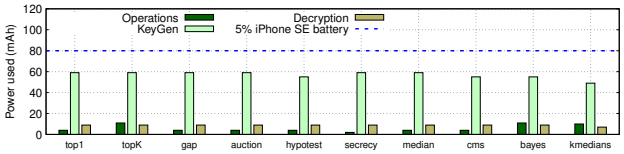

Figure 11. Power consumption on a Raspberry Pi 4.

Figure 11 shows our results; for comparison, we also show 5% of a current mobile-phone battery capacity (the 1,624 mAh of a 2022 iPhone SE). The power consumption varies with the query, but was below 5% for all of the queries we tried. Recall that the chances of being selected for the worst-case MPC are minuscule (on the order of 10^{-6}). If a given device does get selected, the power consumption is certainly nontrivial, but should be manageable. The basic cost without committee service, for the ZK proof and the encryption, was 6 mAh.

7.5 Effects of heterogeneity

Most of our experiments so far were run on server-class machines in a cluster, whereas a practical deployment would contain a mix of different devices in a variety of geographic locations. Our next set of experiments is meant to estimate the effects of this heterogeneity.

Geo-distribution: If the devices are distributed over a large geographic area, the propagation delays between the devices would increase. This would primarily affect the MPC-based vignettes, since they perform many communication rounds. To estimate the effect, we ran one of our most complex MPCs (Gumbel noise) with 42 parties across four servers, and we used `tc` to change the latencies between the servers as if they were located in Mumbai, New York, Paris, and Sydney. This increased the MP-SPDZ time from 73.8s to 521.2s (+606%). The exact effect on other queries would depend on the number and complexity of its MPC vignettes, as well as on the distribution of the user devices.

Slower devices: If the devices contain mobile or embedded devices, Arboretum's computations would take more time. For instance, an RSA-2048 signature takes 767μs on our servers, but 6 ms on a Raspberry Pi 4. Once again, the compute-heavy MPCs are the most likely to be affected. We ran the Gumbel-noise MPC twice, once with 42 servers and once with 38 servers and 4 Raspberry Pi 4s; the computation time increased from 73.8s to 111.7s (+51%). The MPC rounds are bottlenecked by the slowest device, so the exact number of slow devices should not matter (much).

If we consider both effects, the turnaround times in a heterogeneous deployment would likely be about an order of magnitude higher than the ones in Figure 6(b) and 7(b). However, recall that Arboretum is not meant to be used for interactive queries; for a medical study or a diagnostic query, an increase from a few minutes to a few hours seems unproblematic.

7.6 Scalability

A final question is how well Arboretum scales to large numbers of participants, and whether the scaling behavior is qualitatively different from that of earlier systems. To examine this, we used the `top1` query as an example and generated query plans for a wide range of system sizes, from $N = 2^{17}$ to $N = 2^{30}$ participants.

Figures 10(a-c) show our results: (a) shows the aggregator's computation time, and (b) and (c) show the average and maximum computation time for participants, respectively. Each graph contains three lines: two with different limits on the aggregator's computation time, and one without any limit. The overall pattern is similar to Orchard: the cost for the aggregator increases with N because it checks all the ZKPs and, at least initially, also sums up the participants' contributions; the participants' expected cost decreases with N because the chance of serving on a committee decreases, but the maximum cost is constant regardless of N. However, if we add limits, the picture changes: at some point, the aggregator has to outsource some of the computation to the participants, whose expected cost increases accordingly. (With the lowest limit, the aggregator cannot even afford to check ZKPs after $N = 2^{28}$, so the red line stops.) This option would not be available in the earlier systems, which use a single committee.

8 Related Work

Federated Learning: Although Federated Learning is superficially related to Federated Analytics, the former focuses on machine learning instead of analytical queries and thus faces very different challenges and tradeoffs. For instance, many of the attacks on Federated Learning systems (see, e.g., [17]), such as model poisoning or gradient leakage, do not have a direct equivalent in the Federated Analytics setting. For an overview of the latter, please see [16].

Local differential privacy: One way to achieve differential privacy in a federated setting is to have each participant add noise to its data locally, *before* sending it to the aggregator.

This is called *local differential privacy (LDP)* [32]. However, LDP's answers are very noisy [13], and they can be severely distorted by small coalitions of malicious participants [21, 22]. Arboretum avoids these problems by adding noise only once, using a committee. A separate concern is that, while LDP works well with the Laplace mechanism, it is not a good fit for the exponential mechanism because the quality scores need to be computed without noising.

Secrecy of the Sample: Sampling from a large set of clients in order to boost privacy is used in several differentially private algorithms, most notably differentially private SGD [3]. However, previous implementations of this sampling [5, 47] give guarantees in the local model. We are not aware of any prior solutions for large-scale federated settings.

Different trust assumptions: A key difference between Arboretum and some earlier systems is that Arboretum assumes a *single, untrusted* aggregator. For instance, the shuffle model (as, e.g., in PROCHLO [13]) requires two entities, the shuffler and the analyzer, who must not collude; the anytrust model (as, e.g., in UnLynx [34] and Prio [25]) requires a group of entities that must contain at least one honest entity; and TEE-based solutions, such as Glimmers [43], assume that certain hardware features cannot be compromised. Adding trust assumptions can reduce the cost of federated analytics, but it also creates a risk of privacy failure in case the assumptions do not hold.

Single-committee systems: Honeycrisp [53], Orchard [54], and Mycelium [52] all share Arboretum's approach of outsourcing certain computations to MPC committees. However, all three systems are restricted to a *single* committee that performs just a few simple steps (key generation, noising, and decryption). When running the exponential mechanism with more than a trivial number of possible outputs, this committee quickly becomes a bottleneck. A fourth system, Böhler and Kerschbaum [14], also uses a single committee to implement the exponential mechanism but targets smaller deployments; it was shown to scale up to one million participants.

Smaller-scale systems: Arboretum is designed for massive-scale deployments with billions of participants, such as the ones operated by Google and Apple. There is a rich literature of privacy-preserving analytics techniques for smaller deployments, based on cryptographic techniques such as distributed key generation [55], pairwise blinding [4], all-to-all MPC [19], homomorphic threshold encryption [49, 50], or secret sharing [15]. However, these solutions typically do not scale beyond a few thousand users, with the exception of [12], which can support semi-honest secure aggregation on the order of 10^9 devices using an invertible Bloom lookup table to allow scalable message distribution between clients.

Query planning: Arboretum is not the first system to use query planning to speed up privacy-preserving analytics. Conclave [57] and SMCQL [11] generate query plans that combine local clear-text processing with small MPC steps; Secrecy [42] builds query plans for outsourcing relational queries via MPC. However, these systems are all designed for a scenario with a *small* number of participants that each hold a *large* amount of data, which is the exact opposite of what we focus on here. For example, Conclave and Secrecy are limited to at most three parties, and SMCQL supports only two. Opaque [61] assumes that the private data is spread across multiple servers, but that these servers belong to a single organization and are part of a single cluster; Arboretum, in contrast, assumes a federated setting in which each device is owned by a different person.

9 Conclusion

Arboretum removes two key obstacles on the road to a broader deployment of Federated Analytics at scale. So far, analysts either had to have substantial cryptographic expertise, so they could craft a specialized design for their queries that would have an acceptable overhead, or they had to limit themselves to the small set of queries that were supported by the existing designs. Moreover, some queries were out of reach entirely because no known design was efficient enough to be usable by a typical aggregator. Arboretum helps in two ways: first, it automates a large part of the design and optimization process, and second, it adds a way for the participant devices to help with the computation. Even if each individual device can only help a little bit, the massive number of devices can nevertheless make a substantial contribution. As we have shown, this enables entirely new queries, such as categorical queries with a large number of categories, to be run at massive scales. So far, such queries would have been run in the clear out of necessity, with all the privacy risks that this entails. Arboretum provides a safer option with cryptographic protections and the strong guarantees of differential privacy.

Acknowledgments

We thank our shepherd Raluca Ada Popa and the anonymous reviewers for their thoughtful comments and suggestions. This work was supported in part by NSF grants CNS-1955670, CNS-1703936, and CNS-1750158.

References

[1] bellman. https://github.com/zkcrypto/bellman.

[2] ZoKrates. https://github.com/Zokrates/ZoKrates.

[3] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang. Deep learning with differential privacy. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2016.

[4] G. Ács and C. Castelluccia. I have a dream! (differentially private smart metering). In *Information Hiding*, 2011.

[5] N. Agarwal, A. T. Suresh, F. X. X. Yu, S. Kumar, and B. McMahan. cpsgd: Communication-efficient and differentially-private distributed

SGD. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2018.

[6] M. Albrecht, M. Chase, H. Chen, J. Ding, S. Goldwasser, S. Gorbunov, S. Halevi, J. Hoffstein, K. Laine, K. Lauter, S. Lokam, D. Micciancio, D. Moody, T. Morrison, A. Sahai, and V. Vaikuntanathan. Homomorphic encryption security standard. https://eprint.iacr.org/2019/939.

[7] Apple. Differential privacy. https://images.apple.com/privacy/docs/Differential_Privacy_Overview.pdf.

[8] E. Bagdasaryan, P. Kairouz, S. Mellem, A. Gascón, K. Bonawitz, D. Estrin, and M. Gruteser. Towards sparse federated analytics: Location heatmaps under distributed differential privacy with secure aggregation. *arXiv preprint arXiv:2111.02356*, 2021.

[9] B. Balle, G. Barthe, and M. Gaboardi. Privacy amplification by subsampling: Tight analyses via couplings and divergences. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2018.

[10] G. Barthe, B. Köpf, F. Olmedo, and S. Zanella-Béguelin. Probabilistic relational reasoning for differential privacy. In *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 2013.

[11] J. Bater, G. Elliott, C. Eggen, S. Goel, A. Kho, and J. Rogers. SMCQL: Secure querying for federated databases. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, 2017.

[12] J. H. Bell, K. A. Bonawitz, A. Gascón, T. Lepoint, and M. Raykova. Secure single-server aggregation with (poly) logarithmic overhead. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2020.

[13] A. Bittau, U. Erlingsson, P. Maniatis, I. Mironov, A. Raghunathan, D. Lie, M. Rudominer, U. Kode, J. Tinnes, and B. Seefeld. PROCHLO: Strong privacy for analytics in the crowd. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, 2017.

[14] J. Böhler and F. Kerschbaum. Secure multi-party computation of differentially private median. In *Proceedings of the USENIX Security Symposium*, 2020.

[15] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth. Practical Secure Aggregation for Federated Learning on User-Held Data. *arXiv:1611.04482 [cs, stat]*, 2016.

[16] K. Bonawitz, P. Kairouz, B. McMahan, and D. Ramage. Federated learning and privacy: Building privacy-preserving systems for machine learning and data science on decentralized data. *ACM Queue*, 19(5):87–114, Nov. 2021.

[17] N. Bouacida and P. Mohapatra. Vulnerabilities in federated learning. *IEEE Access*, 9:63229–63249, 2021.

[18] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. Fully homomorphic encryption without bootstrapping. Cryptology ePrint Archive, Report 2011/277, 2011. https://eprint.iacr.org/2011/277.

[19] M. Burkhart, M. Strasser, D. Many, and X. A. Dimitropoulos. SEPIA: Privacy-preserving aggregation of multi-domain network events and statistics. In *Proceedings of the USENIX Security Symposium*, 2010.

[20] C. L. Canonne, G. Kamath, A. McMillan, A. Smith, and J. Ullman. The structure of optimal private tests for simple hypotheses. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, 2019.

[21] X. Cao, J. Jia, and N. Z. Gong. Data poisoning attacks to local differential privacy protocols. In *Proceedings of the USENIX Security Symposium*, 2019.

[22] A. Cheu, A. Smith, and J. Ullman. Manipulation attacks in local differential privacy. *ArXiv*, abs/1909.09630, 2019.

[23] K. Chida, D. Genkin, K. Hamada, D. Ikarashi, R. Kikuchi, Y. Lindell, and A. Nof. Fast large-scale honest-majority MPC for malicious adversaries. Cryptology ePrint Archive, Paper 2018/570, 2018. https://eprint.iacr.org/2018/570.

[24] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène. TFHE: Fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33(1):34–91, 2020.

[25] H. Corrigan-Gibbs and D. Boneh. Prio: Private, robust, and scalable computation of aggregate statistics. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2017.

[26] I. Damgård, V. Pastro, N. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Proc. CRYPTO*, 2012.

[27] B. Ding, J. Kulkarni, and S. Yekhanin. Collecting telemetry data privately. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2017.

[28] Z. Ding, Y. Wang, Y. Xiao, G. Wang, D. Zhang, and D. Kifer. Free gap estimates from the exponential mechanism, sparse vector, noisy max and related algorithms. *VLDB Journal*, Feb. 2022.

[29] D. Durfee and R. Rogers. Practical differentially private top-k selection with pay-what-you-get composition. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2019.

[30] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating Noise to Sensitivity in Private Data Analysis. In *Proceedings of the Theory of Cryptography Conference (TCC)*, 2006.

[31] C. Dwork and A. Roth. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3–4):211–407, 2014. http://dx.doi.org/10.1561/040000004.

[32] U. Erlingsson, V. Pihur, and A. Korolova. RAPPOR: Randomized aggregatable privacy-preserving ordinal response. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2014.

[33] V. Fang, L. Brown, W. Lin, W. Zheng, A. Panda, and R. A. Popa. CostCO: An automatic cost modeling framework for secure multiparty computation. Cryptology ePrint Archive, Paper 2022/332, 2022. https://eprint.iacr.org/2022/332.

[34] D. Froelicher, P. Egger, J. S. Sousa, J. L. Raisaro, Zhicong Huang, C. Mouchet, B. Ford, and J.-P. Hubaux. UnLynx: A Decentralized System for Privacy-Conscious Data Sharing. In *Proceedings of the Privacy Enhancing Technologies Symposium (PETS)*, 2017.

[35] K. Gopinath and V. H. Gupta. An extended verifiable secret redistribution protocol for archival systems. In *Proc. First International Conference on Availability, Reliability and Security*, 2006.

[36] J. Groth. On the size of pairing-based non-interactive arguments. In *Proc. EUROCRYPT*, 2016.

[37] R. Huebsch, J. M. Hellerstein, N. Lanham, B. T. Loo, S. Shenker, and I. Stoica. Querying the internet with PIER. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, 2003.

[38] C. Ilvento. Implementing the exponential mechanism with base-2 differential privacy. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2020.

[39] M. Keller. MP-SPDZ: A versatile framework for multi-party computation. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2020.

[40] E. Kushilevitz, Y. Lindell, and T. Rabin. Information-theoretically secure protocols and security under composition. In *Proc. STOC*, 2006.

[41] L. Lamport, R. Shostak, and M. Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982.

[42] J. Liagouris, V. Kalavri, M. Faisal, and M. Varia. Secrecy: Secure collaborative analytics on secret-shared data, 2022. arXiv:2102.01048.

[43] D. Lie and P. Maniatis. Glimmers: Resolving the privacy/trust quagmire. *Proceedings of the Workshop on Hot Topics in Operating Systems (HotOS)*, 2017.

[44] E. Margolin, K. Newatia, T. Luo, E. Roth, and A. Haeberlen. Arboretum: A planner for large-scale federated analytics with differential privacy(extended version). Technical Report MS-CIS-23-03, University of Pennsylvania, 2023.

[45] F. McSherry and K. Talwar. Mechanism design via differential privacy. In *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, 2007.

[46] I. Mironov. On significance of the least significant bits for differential privacy. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2012.

[47] V. Pihur, A. Korolova, F. Liu, S. Sankuratripati, M. Yung, D. Huang, and R. Zeng. Differentially-private "draw and discard" machine learning. *arXiv preprint arXiv:1807.04369*, 2018.

[48] D. Ramage and S. Mazzocchi. Federated analytics: Collaborative data science without data collection, May 2020. Google AI Blog, https://ai.googleblog.com/2020/05/federated-analytics-collaborative-data.html.

[49] V. Rastogi and S. Nath. Differentially private aggregation of distributed time-series with transformation and encryption. In *Proceedings of the ACM SIGMOD Conference*, 2010.

[50] L. Reyzin, A. Smith, and S. Yakoubov. Turning HATE Into LOVE: Homomorphic Ad Hoc Threshold Encryption for Scalable MPC. In *Proceedings of the International Symposium on Cyber Security Cryptography and Machine Learning (CSCML)*, 2021.

[51] R. Rodrigues and P. Druschel. Peer-to-peer systems. *Communications of the ACM*, 53(10):72–82, oct 2010.

[52] E. Roth, K. Newatia, Y. Ma, K. Zhong, S. Angel, and A. Haeberlen. Mycelium: Large-scale distributed graph queries with differential privacy. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, 2021.

[53] E. Roth, D. Noble, B. Hemenway Falk, and A. Haeberlen. Honeycrisp: Large-scale differentially private aggregation without a trusted core. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, Oct. 2019.

[54] E. Roth, H. Zhang, A. Haeberlen, and B. C. Pierce. Orchard: Differentially private analytics at scale. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2020.

[55] E. Shi, T.-H. H. Chan, E. G. Rieffel, R. Chow, and D. X. Song. Privacy-preserving aggregation of time-series data. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2011.

[56] A. Smith. Differential privacy and the secrecy of the sample, Sept. 2009. https://adamdsmith.wordpress.com/2009/09/02/sample-secrecy/.

[57] 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 ACM European Conference on Computer Systems (EuroSys)*, 2019.

[58] A. Yao. Protocols for secure computations. In *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, 1982.

[59] H. Zhang, E. Roth, A. Haeberlen, B. C. Pierce, and A. Roth. Fuzzi: A three-level logic for differential privacy. In *Proceedings of the ACM SIGPLAN International Conference on Functional Programming (ICFP)*, 2019.

[60] M. Zhao, P. Aditya, A. Chen, Y. Lin, A. Haeberlen, P. Druschel, B. Maggs, B. Wishon, and M. Ponec. Peer-assisted content distribution in Akamai NetSession. In *13th ACM SIGCOMM Conference on Internet Measurement (IMC '13)*, Oct. 2013.

[61] W. Zheng, A. Dave, J. G. Beekman, R. A. Popa, J. E. Gonzalez, and I. Stoica. Opaque: An oblivious and encrypted distributed analytics platform. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2017.