Tumult Labs

Tumult Analytics: a robust, easy-to-use, scalable, and expressive framework for differential privacy

1— Introduction and design goals

Tumult Analytics is an open-source framework for releasing aggregate information from sensitive datasets with differential privacy[1] (DP). It supports many standard operations (e.g. filters, joins, maps) and aggregations (e.g. counts, averages, quantiles). It is currently used at institutions such as the IRS, the Wikimedia Foundation, and the U.S. Census Bureau.

Tumult Analytics is designed to satisfy the following desiderata:

- **Robustness.** An analyst with access to the raw data, who wants to publish a differentially private version of it, can confidently use the platform and obtain the desired privacy guarantees.
- **Ease of use.** Any data scientist or engineer can successfully apply DP to their own data, possibly after using the platform and its documentation to learn about the necessary concepts. No expert-level math knowledge or in-depth understanding of DP theory is ever required.
- **Scalability and performance.** Tumult Analytics can run DP computations on arbitrarily-sized datasets. It uses computational resources that are on the same order of magnitude as the non-private version of the queries it evaluates.
- **Expressiveness.** Tumult Analytics is sufficiently feature-rich to power real-world use cases. Additionally, the underlying privacy accounting framework is extensible enough to support the addition of new data transformation operators, aggregate functions, or even privacy notions and accounting methods, without requiring deep design changes.

The library is built in two separate components.

- Tumult Core is a privacy foundation, designed to be modular and extensible. We present its design in Section 2.
- Tumult Analytics focuses on ease-of-use, by providing a simpler interface on top of Tumult Core. We present its interface in Section 3.

We then compare Tumult Analytics with other open-source frameworks for differential privacy (Section 4).

Figure 1. An annotated Tumult Analytics program that computes the average income of all people older than 40 in an input dataframe **`income_df`**, grouped by ZIP code.

```
A  session = Session.from_dataframe(
       dataframe=income_df,
       source_id="income_data",
       privacy_budget=PureDPBudget(1.5),
   )

B  query = (
       QueryBuilder ("income_data")
      .filter("age > 40")
      .groupby(zip_codes)
      .average("income", low=0, high=10**6)
   )

C  result = session.evaluate(
       query, PureDPBudget(0.2)
   )
D  print(session.remaining_privacy_budget)
```

A Session initialization: the sensitive data is loaded and given a fixed *privacy budget*, which bounds the maximum information leakage. The interface guarantees that all the outputs of future queries stay within this budget (here, ε=1.5).

B Query definition: the user specifies which query they want to run on the data, using an interface similar to PySpark or Pandas.

C Query evaluation: the previously defined query is executed on the data, using a portion of the overall privacy budget (here, ε=0.2).

D Session inspection: the user can request how much privacy budget is left for future queries.

[1] Readers unfamiliar with differential privacy [1] can refer to [2, 3] for non-technical introductions to this field, and [4, 5] for more in-depth surveys.

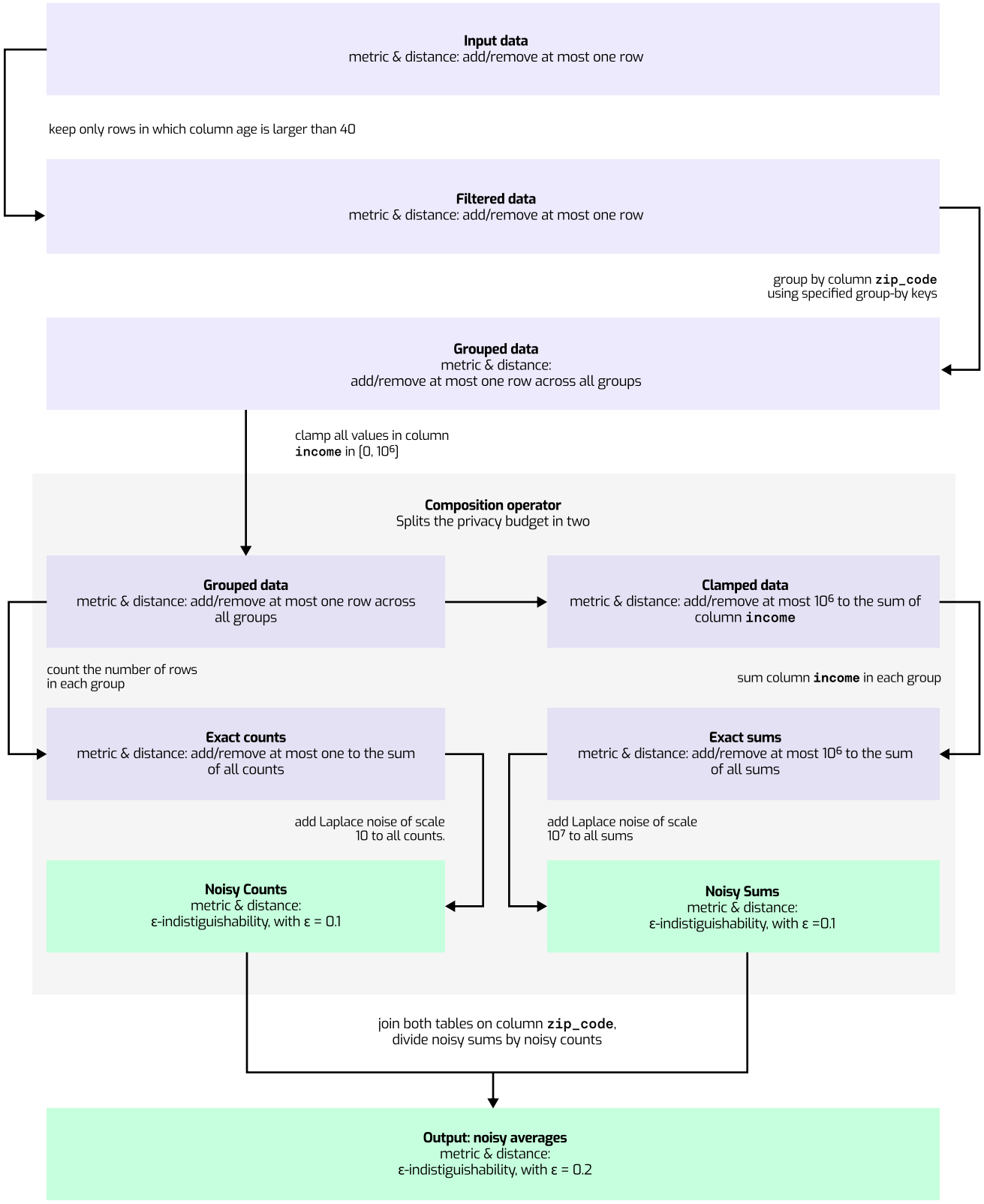

Figure 2. A simplified diagram outlining the steps taken by the data through Tumult Core transformations, measurements and operators, to implement the query in Figure 1. Each labeled arrow corresponds to a Core component. The actual series of Core components to implement this query is more complex and incorporates some additional optimizations.

2— Tumult Core

Tumult Core is the underlying framework that Tumult Analytics relies on. It is a collection of components and composition operators which allow users to implement complex differentially private mechanisms. Tumult Core is where all the privacy-critical logic lives: Tumult Analytics is only an interface to Tumult Core, and the underlying framework provides end-to-end privacy guarantees.

The fundamental component of Tumult Core is called a measurement. A measurement takes an input from some domain and produces a randomized output. The privacy properties of the measurement are captured by its input metric, a function defining distances on its input domain, its output measure, a function defining distances between probability distributions over the output domain, and its privacy function, a function that takes a distance in the input metric and returns a distance in the output measure. A measurement M with privacy function f satisfies the following property: for any pair of inputs x,y with distance at most d in the input metric, the distributions M(x) and M(y) have distance at most f(d) in the output measure.

This guarantee generalizes the usual DP guarantee in two ways. First, it gives users flexibility to define alternative notions of neighbors. In the usual definition of DP, neighbors are pairs of datasets that differ in a single record: it prevents an adversary from inferring information about a single record. However, we often want a weaker guarantee – for example, on the precise location of some person – or a stronger guarantee – for example, on all records associated with some individual. Tumult Core measurements are general enough to capture these examples, as well as other similar variants (e.g. the N section in [6]).

Second, the privacy notion is abstracted in Tumult Core: the framework is generic enough to cover not only differential privacy, but variants of DP as well, and potentially entirely new privacy notions. Today, in addition to DP, Tumult Core supports zero-concentrated DP [7], and support for approximate DP [8] and approximate zCDP is underway. Other notions like Rényi DP [9], f-DP [10], and others (e.g. the Q section in [6]) could easily be supported.

In addition, Tumult Core also includes pre-processing components called transformations. Transformations include typical data processing operations (filters, maps, joins, ...) and pre-processing building blocks (clamping, truncation, ...) useful for DP algorithms. Transformations don't have a privacy guarantee on their own. Instead, they have a stability guarantee, which relates a distance in the transformation's input metric to a distance in its output metric.

A Tumult Core program uses composition operators to combine transformations and measurements, and implement complex algorithms. When multiple components are combined, the stability or privacy function of the resulting component is inferred inductively. This allows Tumult Core to build an end-to-end proof of the privacy guarantee of an entire program, no matter how complex. Figure 2 presents a simplified example of how the Tumult Analytics query in Figure 1 is implemented in Tumult Core. The generality of this framework allows it to integrate various complex features, including:

- **Private Joins.** Private joins are notoriously difficult [11, 12, 13, 14]: their privacy analysis requires an understanding of how the stability properties of two tables interact. Tumult Core handles this by defining a new metric that defines distances on sets of tables.
- **Complex neighboring notions.** Some datasets can contain an arbitrary number of records per user, which are keyed using a user ID. Tumult Core handles this by defining a metric for tables with privacy IDs, extending transformations to work with this metric, and adding truncation operators to convert from this metric to a metric quantifying the number of added/removed rows. Other complex neighboring relations are also implemented.
- **Generalized parallel composition.** When running DP measurements on disjoint sets of records, we can reuse privacy budget, since each record can appear in at most one of these sets [15]. Generalized parallel composition generalizes this result to non-disjoint subsets where a user's contribution across subsets is bounded. Tumult Core captures this contribution constraint in a metric between lists of datasets.
- **Adaptivity.** Tumult Core supports fully adaptive mechanisms: when performing multiple measurements on the data, each measurement can reuse the output of previous measurements, including to determine how much privacy budget it consumes.

The variety of privacy definitions and accounting techniques can easily be combined. Adding new notions of indistinguishability is as simple as defining a new measurement that supports this notion, or adding support to an existing measurement. Then, the new measurement can be combined with existing transformations to produce complex mechanisms using the new privacy guarantee. Similarly, new transformations that support e.g. privacy IDs can be combined with transformations that support more typical privacy accounting. This way, new privacy accounting techniques can be used only when necessary, and existing components can be reused where it is not. This modularity makes it easy to extend Tumult Core with new desired features.

3— Tumult Analytics

While Tumult Core is powerful, its extremely modular design can make it challenging to use directly, especially for users who are not experts in differential privacy. This is why we built Tumult Analytics as an interface layer on top of Tumult Core. Tumult Analytics is designed to be usable for non-experts, while still being capable of addressing complex practical use cases. We first discuss its design and why we believe it is approachable, using Figure 1 as an example. Second, we present some current real-world use cases of Tumult Analytics to give evidence of its power.

Computations in Tumult Analytics happen in the context of a Session, which associates a fixed privacy guarantee with a dataset (possibly containing multiple tables), and mediates the full analysis run on this data. An analyst defines the privacy budget when they create a Session, and they can query the Session for how much budget is remaining at any time. They also define the privacy definition at Session creation – both the indistinguishability definition (e.g. pure DP or zCDP) and the unit of privacy (e.g. how many records a single user contributes to in the original dataset). Session creation is the only time the analyst must provide private data. All further interactions between the analyst and the private data happen through the Session, preventing inadvertent privacy violations that may result from the analyst using the private data inappropriately. The syntax for initializing a Session is demonstrated in the first block of Figure 1.

Next, the analyst can define queries. The query language attempts to limit the amount of new concepts specific[2] to DP that the user has to learn and specify. This allows the analyst who is unfamiliar with DP to specify the statistics they want to see, and defers the process of constructing a private mechanism that approximates the answers to these queries to the Tumult Analytics engine. This query construction process uses a fluent interface and generally attempts to resemble the pandas [16] and PySpark [17] query interfaces, which users might already be familiar with. This syntax is shown on the second block of Figure 1.

Once a query has been defined, the analyst evaluates it to produce a noisy answer. This consumes privacy budget: the analyst must specify how much budget to use the query as in the third block of Figure 1. When the analyst evaluates a query, Tumult Analytics compiles it into a Tumult Core measurement that answers the query using the given privacy budget, and returns the answer to the user.

Because Tumult Core supports fully adaptive composition, the entire process is interactive: the user can process the query answers and add complex control flow (conditional statements, loops, etc.) to choose which queries to evaluate next. In the fourth block of Figure 1, the user asks the Session how much budget it has left. The answer is greater than 0, so the user can run further queries.

To help users ramp up with DP, Tumult Analytics also comes with extensive documentation [18] and multiple tutorials [19] explaining how to perform common DP analysis tasks.

Ease-of-use is only valuable if users can actually solve their real-world data analysis problems. We believe that Tumult Analytics is sufficiently powerful for such complex tasks. The features discussed in Section 2 are either available or currently being added, including private joins, multiple privacy definitions (pure differential privacy and zCDP), and support for privacy identifiers. Tumult Analytics has already been deployed in practice, either in production or as a prototype in the following cases.

- The U.S. Census Bureau is using Tumult Analytics for the Detailed DHC-A, Detailed DHC-B, and S-DHC data releases, as part of the 2020 Decennial Census. These use cases rely on several complex features outlined in Section 2: zCDP accounting, adaptivity, and tight privacy accounting with generalized parallel composition.
- The Wikimedia Foundation uses it to publish country-level statistics about the number of visitors to each Wikipedia page on each day. This use case relies on the complex privacy notion feature mentioned in Section 2.
- The U.S. Internal Revenue Service relies on Tumult Analytics to release college graduate income summaries and power the College Scorecard website [20].

The variety of privacy definitions and accounting techniques can easily be combined. Adding new notions of indistinguishability is as simple as defining a new measurement that supports this notion, or adding support to an existing measurement. Then, the new measurement can be combined with existing transformations to produce complex mechanisms using the new privacy guarantee. Similarly, new transformations that support e.g. privacy IDs can be combined with transformations that support more typical privacy accounting. This way, new privacy accounting techniques can be used only when necessary, and existing components can be reused where it is not. This modularity makes it easy to extend Tumult Core with new desired features.

2 Some, like noise type or magnitude, are either optional or entirely hidden. Others, like clamping bounds or explicit group-by keys, are required today, but work is currently underway to remove the need for users to understand and specify these.

4— Comparison to existing systems

Tumult Analytics is not the first open-source framework for differential privacy. In this section, we list other libraries, and explain how Tumult Analytics compares to them.

GoogleDP [21] is a suite of tools including "building block" libraries and higher-level frameworks (Privacy on Beam [22] written in Go, and an extension to ZetaSQL [23, 24]). Besides the obvious language difference (Tumult Analytics is a Python library), a main difference between Tumult Core and the GoogleDP building block libraries is that the latter do not provide an end-to-end guarantee: higher-level frameworks have to use them correctly and implement privacy-critical operations like privacy accounting or contribution bounding directly. Another fundamental difference is extensibility. GoogleDP tools are designed to support a specific class of queries [24], and use approximate DP for privacy accounting. Extending them to support some of Tumult Analytics' features (like zCDP accounting, parallel composition, or private joins without privacy IDs) would require deep changes throughout the framework and the building block libraries. By comparison, such changes were added to Tumult Core and Tumult Analytics in a matter of weeks when the need arose.

PipelineDP [25] is a Python framework built atop the GoogleDP building block libraries. It follows roughly the same design as the two GoogleDP frameworks, so the comparison from the previous paragraph applies here as well. One major difference is that PipelineDP's design is backend-agnostic: it can run on multiple data processing frameworks, like Beam, Spark, or locally. This is an advantage over Tumult Analytics, which fully relies on Spark.

OpenDP[3] [26] is inspired by a programming framework proposed by [29]. This framework was also the inspiration for Tumult Core, so there are similarities between both projects. Besides feature-richness (the features mentioned in Section 2 do not exist yet in OpenDP), a major difference between the two projects is scalability: all components in OpenDP are written in Rust, so transformations like group-by operations and aggregations assume that all the data fits in memory on a single machine. This makes it unsuitable for large-scale data processing use cases.

SmartNoise SQL [30] is a high-level framework to run differentially private SQL queries. It uses some primitives from OpenDP, like noise addition, but (maybe due to the scalability limitations mentioned above) uses the SQL backend directly for most data processing operations. As a result, most of the privacy-critical logic is implemented directly in SmartNoise SQL instead of using the underlying framework as was originally envisioned [29]. This means that even though OpenDP can generate a proof of privacy for a complex program from its simple components, SmartNoise SQL cannot. This architectural choice also means that SmartNoise SQL cannot directly benefit from OpenDP's extensibility.

Diffprivlib [31] is a library of DP mechanisms written in Python. It relies on NumPy [32, 33] for all the underlying computations, which has the same scalability limits as OpenDP, and also creates security vulnerabilities [34]. diffprivlib is also not built on an extensible framework like OpenDP or Tumult Core; it relies exclusively on approximate DP, and can only protect individual records in a single table.

Other systems have been proposed in the literature, and proofs of concept for these systems have been published on open-source platforms: this is the case for Chorus [35, 36] or PINQ [37, 15]. Since these libraries are prototypes that are not actively maintained, we do not extensively compare Tumult Analytics to them.

[3] The software library [26], not to be confused with the larger project with the same name [27]. Another software library under the same project is SmartNoise Core [28], which is an older version of OpenDP and is now deprecated.

References

[1] C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating noise to sensitivity in private data analysis," in *Theory of cryptography conference*. Springer, 2006, pp. 265–284.

[2] A. Wood, M. Altman, A. Bembenek, M. Bun, M. Gaboardi, J. Honaker, K. Nissim, D. R. O'Brien, T. Steinke, and S. Vadhan, "Differential privacy: A primer for a non-technical audience," *Vand. J. Ent. & Tech. L.*, vol. 21, p. 209, 2018.

[3] D. Desfontaines, "A friendly, non-technical introduction to differential privacy," https://desfontain.es/privacy/friendly-intro-to-differential-privacy.html, 09 2021, Ted is writing things (personal blog).

[4] C. Dwork, A. Roth et al., "The algorithmic foundations of differential privacy," *Foundations and Trends® in Theoretical Computer Science*, vol. 9, no. 3–4, pp. 211–407, 2014.

[5] S. Vadhan, "*The complexity of differential privacy*," in Tutorials on the Foundations of Cryptography. Springer, 2017, pp. 347–450.

[6] D. Desfontaines and B. Pejó, "SoK: differential privacies," *Proceedings on privacy enhancing technologies*, vol. 2020, no. 2, pp. 288–313, 2020.

[7] M. Bun and T. Steinke, "Concentrated differential privacy: Simplifications, extensions, and lower bounds," in *Theory of Cryptography Conference*. Springer, 2016, pp. 635–658.

[8] C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov, and M. Naor, "Our data, ourselves: Privacy via distributed noise generation," in *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 2006, pp. 486–503.

[9] I. Mironov, "Rényi differential privacy," in 2017 *IEEE 30th computer security foundations symposium (CSF)*. IEEE, 2017, pp. 263–275.

[10] J. Dong, A. Roth, and W. J. Su, "Gaussian differential privacy," *arXiv preprint arXiv:1905.02383*, 2019.

[11] I. Kotsogiannis, Y. Tao, X. He, M. Fanaeepour, A. Machanavajjhala, M. Hay, and G. Miklau, "PrivateSQL: a differentially private SQL query engine," *Proceedings of the VLDB Endowment*, vol. 12, no. 11, pp. 1371–1384, 2019.

[12] W. Dong, J. Fang, K. Yi, Y. Tao, and A. Machanavajjhala, "R2T: Instance-optimal truncation for differentially private query evaluation with foreign keys," in *Proc. ACM SIGMOD International Conference on Management of Data*, 2022.

[13] N. Johnson, J. P. Near, and D. Song, "Towards practical differential privacy for sql queries, "*Proceedings of the VLDB Endowment*, vol. 11, no. 5, pp. 526–539, 2018.

[14] Y. Tao, X. He, A. Machanavajjhala, and S. Roy, "Computing local sensitivities of counting queries with joins," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 2020, pp. 479–494.

[15] F. D. McSherry, "Privacy Integrated Queries: an extensible platform for privacy-preserving data analysis," in Proceedings of the 2009 ACM SIGMOD International Conference on Management of data, 2009, pp. 19–30.

[16] The pandas development team, "pandas-dev/pandas: Pandas," Feb. 2020. [Online]. Available: https://doi.org/10.5281/zenodo.3509134

[17] T. P. development team, "PySpark." [Online]. Available: https://spark.apache.org/docs/latest/api/python/

[18] "Tumult Analytics documentation." [Online]. Available: https://docs.tmlt.dev/analytics/latest/

[19] "Tumult Analytics tutorials." [Online]. Available: https://docs.tmlt.dev/analytics/latest/tutorials

[20] "College Scorecard," https://collegescorecard.ed.gov/, accessed: 2022-11-23.

[21] "Google's differential privacy libraries." [Online]. Available: https://github.com/google/differential-privacy

[22] "Privacy on Beam." [Online]. Available: https://github.com/google/differential-privacy/tree/main/privacy-on-beam

[23] "ZetaSQL differential privacy extension." [Online]. Available: https://github.com/google/differential-privacy/tree/main/examples/zetasql

[24] R. J. Wilson, C. Y. Zhang, W. Lam, D. Desfontaines, D. Simmons-Marengo, and B. Gipson, "Differentially private SQL with bounded user contribution," *Proceedings on Privacy Enhancing Technologies*, vol. 2, pp. 230–250, 2020.

[25] "PipelineDP," accessed: 2022-11-23. [Online]. Available: https://pipelinedp.io/

[26] "OpenDP," accessed: 2022-11-23. [Online]. Available: https://github.com/opendp/opendp

[27] "OpenDP," accessed: 2022-11-23. [Online]. Available: https://opendp.org/

[28] "SmartNoise Core," accessed: 2022-11-23. [Online]. Available: https://github.com/opendp/smartnoise-core

[29] M. Gaboardi, M. Hay, and S. Vadhan, "A programming framework for OpenDP," *Manuscript, May*, 2020.

[30] "SmartNoise SQL," accessed: 2022-11-23. [Online]. Available: https://github.com/opendp/smartnoise-sdk/tree/main/sql

[31] "diffprivlib," accessed: 2022-11-23. [Online]. Available: https://github.com/IBM/differential-privacy-library

[32] "NumPy," accessed: 2022-11-23. [Online]. Available: https://numpy.org/

[33] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2

[34] S. Haney, D. Desfontaines, L. Hartman, R. Shrestha, and M. Hay, "Precision-based attacks and interval refining: how to break, then fix, differential privacy on finite computers," *arXiv preprint arXiv:2207.13793*, 2022.

[35] "Chorus," accessed: 2022-11-23. [Online]. Available: https://github.com/uvm-plaid/chorus

[36] N. Johnson, J. P. Near, J. M. Hellerstein, and D. Song, "Chorus: a programming framework for building scalable differential privacy mechanisms," in *2020 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2020, pp. 535–551.

[37] "Privacy Integrated Queries, v0.1.1," accessed: 2022-11-23. [Online]. Available: https://github.com/LLGemini/PINQ