DUET: An Expressive Higher-Order Language and Linear Type System for Statically Enforcing Differential Privacy

JOSEPH P. NEAR, University of Vermont, USA
DAVID DARAIS, University of Vermont, USA
CHIKE ABUAH, University of Vermont, USA
TIM STEVENS, University of Vermont, USA
PRANAV GADDAMADUGU, University of California, Berkeley, USA
LUN WANG, University of California, Berkeley, USA
NEEL SOMANI, University of California, Berkeley, USA
MU ZHANG, University of Utah, USA
NIKHIL SHARMA, University of California, Berkeley, USA
ALEX SHAN, University of California, Berkeley, USA
DAWN SONG, University of California, Berkeley, USA

During the past decade, differential privacy has become the gold standard for protecting the privacy of individuals. However, verifying that a particular program provides differential privacy often remains a manual task to be completed by an expert in the field. Language-based techniques have been proposed for fully automating proofs of differential privacy via type system design, however these results have lagged behind advances in differentially-private algorithms, leaving a noticeable gap in programs which can be automatically verified while also providing state-of-the-art bounds on privacy.

We propose DUET, an expressive higher-order language, linear type system and tool for automatically verifying differential privacy of general-purpose higher-order programs. In addition to general purpose programming, DUET supports encoding machine learning algorithms such as stochastic gradient descent, as well as common auxiliary data analysis tasks such as clipping, normalization and hyperparameter tuning–each of which are particularly challenging to encode in a statically verified differential privacy framework.

We present a core design of the DUET language and linear type system, and complete key proofs about privacy for well-typed programs. We then show how to extend DUET to support realistic machine learning applications and recent variants of differential privacy which result in improved accuracy for many practical differentially private algorithms. Finally, we implement several differentially private machine learning algorithms in DUET which have never before been automatically verified by a language-based tool, and we present experimental results which demonstrate the benefits of DUET's language design in terms of accuracy of trained machine learning models.

CCS Concepts: • **Security and privacy → Logic and verification**; • **Theory of computation → Logic and verification**; **Linear logic**.

Authors' addresses: Joseph P. Near, jnear@uvm.edu, University of Vermont, USA; David Darais, David.Darais@uvm.edu, University of Vermont, USA; Chike Abuah, cabuah@uvm.edu, University of Vermont, USA; Tim Stevens, Timothy.Stevens@uvm.edu, University of Vermont, USA; Pranav Gaddamadugu, pranavsaig@berkeley.edu, University of California, Berkeley, USA; Lun Wang, wanglun@berkeley.edu, University of California, Berkeley, USA; Neel Somani, neel@berkeley.edu, University of California, Berkeley, USA; Mu Zhang, muzhang@cs.utah.edu, University of Utah, USA; Nikhil Sharma, ennsharma@berkeley.edu, University of California, Berkeley, USA; Alex Shan, alexshan@berkeley.edu, University of California, Berkeley, USA; Dawn Song, dawnsong@cs.berkeley.edu, University of California, Berkeley, USA.

2475-1421/2019/10-ART172
https://doi.org/10.1145/3360598

Additional Key Words and Phrases: Differential privacy, typechecking, machine learning

ACM Reference Format:
Joseph P. Near, David Darais, Chike Abuah, Tim Stevens, Pranav Gaddamadugu, Lun Wang, Neel Somani, Mu Zhang, Nikhil Sharma, Alex Shan, and Dawn Song. 2019. DUET: An Expressive Higher-Order Language and Linear Type System for Statically Enforcing Differential Privacy. *Proc. ACM Program. Lang.* 3, OOPSLA, Article 172 (October 2019), 30 pages. https://doi.org/10.1145/3360598

1 INTRODUCTION

Advances in big data and machine learning have achieved large-scale societal impact over the past decade. This impact is accompanied by a growing demand for data collection, aggregation and analysis at scale. This resulting explosion in the amount of data collected by organizations, however, has raised important new security and privacy concerns.

Differential privacy [Dwork 2006; Dwork et al. 2006, 2014] is a technique for addressing these issues. Differential privacy allows general statistical analysis of data while protecting *data about individuals* with a formal guarantee of privacy. Because of its desirable formal guarantees, differential privacy has received increased attention, with ongoing real-world deployments at organizations including Google [Erlingsson et al. 2014], Apple [app 2016], and the US Census [Haney et al. 2017; Machanavajjhala et al. 2008]. A number of systems for performing differentially private data analytics have been built and demonstrated to be effective [Johnson et al. 2018, 2017; McSherry 2009a; Mohan et al. 2012; Narayan and Haeberlen 2012; Proserpio et al. 2014; Roy et al. 2010].

Differential privacy plays an increasingly important role in machine learning, as recent work has shown that a trained model can leak information about data it was trained on [Fredrikson et al. 2015; Shokri et al. 2017; Wu et al. 2016]. Differential privacy provides a robust solution to this problem, and as a result, a number of differentially private algorithms have been developed for machine learning [Abadi et al. 2016; Bassily et al. 2014a; Chaudhuri et al. 2011; Friedman et al. 2016; Papernot et al. 2016; Song et al. 2013; Talwar et al. 2015; Wu et al. 2017].

Few practical approaches exist, however, for automatically proving that a *general-purpose program* satisfies differential privacy—an increasingly desirable goal, since many machine learning pipelines are expressed as programs that combine existing algorithms with custom code. Enforcing differential privacy for a new program currently requires a new, manually-written privacy proof. This process is arduous, error-prone, and must be performed by an expert in differential privacy (and re-performed, each time the program is modified).

We present DUET, a programming language, type system and tool for expressing and statically verifying privacy-preserving programs. DUET supports (1) general purpose programming features like compound datatypes and higher-order functions, (2) library functions for matrix-based computations, and (3) multiple state-of-the-art variants of differential privacy—(ϵ, δ)-differential privacy [Dwork et al. 2014], Rényi differential privacy [Mironov 2017], zero-concentrated differential privacy (zCDP) [Bun and Steinke 2016], and truncated-concentrated differential privacy (tCDP) [Bun et al. 2018]—and can be easily extended to new ones. DUET strikes a strategic balance between generality, practicality, extensibility, and precision of computed privacy bounds.

The design of DUET consists of *two* separate, mutually embedded languages, each with its own type system. The *sensitivity language* uses linear types *with* metric scaling (as in *Fuzz* [Reed and Pierce 2010]) to bound function sensitivity. The *privacy language* uses linear types *without* metric scaling (novel in DUET) to compose differentially private computations. Disallowing the use of scaling in the privacy language is essential to encode more advanced variants of differential privacy (like (ϵ, δ)) in a linear type system.

Linear typing [Barber 1996; Girard 1987] is a good fit for both privacy and sensitivity analysis because resources are tracked per-variable and combined additively. In particular, our linear typing approach to *privacy* allows for independent privacy costs for multiple function arguments, a feature shared by Fuzz and DFuzz (which only support pure ϵ-differential privacy), but not supported by prior type systems for (ϵ, δ)-differential privacy. This limitation of prior work is due to treating privacy as a computational "effect"—a property of the output via an indexed monad—as opposed to our treatment of privacy as a "co-effect"—a property of the context via linear typing.

Our main idea is to co-design *two* separate languages for privacy and sensitivity, and our main insight is that a linear type system can (1) model more powerful variants of differential privacy (like (ϵ, δ)) when strengthened to disallow scaling, and (2) interact seamlessly with a sensitivity-type system which does allow scaling. Each language embeds inside the other, and the privacy mechanisms of the underlying privacy definition (e.g. the Gaussian mechanism [Dwork et al. 2014]) form the interface between the two languages. Both languages use similar syntax and identical types. The two languages aid type checking, the proof of type soundness, and our implementation of type inference; programmers need not be intimately aware of the multi-language design.

In addition to differential-privacy primitives like the Gaussian mechanism, we provide a core language design for matrix-based data analysis tasks, such as aggregation, clipping and gradients. Key challenges we overcome in our design are how these features compose in terms of function sensitivity, and how to statically track bounds on vector norms (due to clipping, for the purposes of privacy)—and each in a way that is general enough to support a wide range of useful applications.

We demonstrate the usefulness of DUET by implementing and verifying several differentially private machine learning algorithms from the literature, including private stochastic gradient descent [Bassily et al. 2014a] and private Frank-Wolfe [Talwar et al. 2015], among many others. We also implement a variant of stochastic gradient descent suitable for deep learning. For each of these algorithms, no prior work has demonstrated an automatic verification of differential privacy, and DUET is able to automatically infer privacy bounds that equal *and in some cases improve upon* previously published manual privacy proofs.

We have implemented a typechecker and interpreter for DUET, and we use these to perform an empirical evaluation comparing the accuracy of models trained using our implementations. Although the "punchline" of the empirical results are unsurprising due to known advantages of the differential privacy definitions used (e.g., that using recent variants like zero-concentrated differential privacy results in improved accuracy), our results show the extent of the accuracy improvements for specific algorithms and further reinforce the idea that choosing the best definition consistently results in substantially better accuracy of the trained model.

Contributions. In summary, we make the following contributions:

- We present DUET, a language, linear type system and tool for expressing and automatically verifying differentially private programs. DUET supports a combination of (1) general purpose, higher order programming, (2) advanced definitions of differential privacy, (3) independent tracking of privacy costs for multiple function arguments, and (4) auxiliary differentially-private data analysis tasks such as clipping, normalization, and hyperparameter tuning.
- We formalize DUET's type system and semantics, and complete key proofs about privacy of well-typed programs.
- We demonstrate a battery of case studies consisting of medium-sized, real-world, differentially private machine learning algorithms which are successfully verified with optimal (or near-optimal) privacy bounds. In some cases, DUET infers privacy bounds which improve on the best previously published manually-verified result.

- We conduct an experimental evaluation to demonstrate Duet's feasibility in practice by training two machine learning algorithms on several non-toy real-world datasets using Duet's interpreter. These results demonstrate the effect of improved privacy bounds on the accuracy of the trained models.

2 PRELIMINARIES

2.1 Background: Differential Privacy

This section briefly summarizes the basics of differential privacy. See Dwork and Roth's reference [Dwork et al. 2014] for a detailed description. Differential privacy considers sensitive input data represented by a vector $x \in D^n$, in which x_i represents the data contributed by user i. The *distance* between two inputs $x, y \in D^n$ is $d(x,y) = |\{i | x_i \neq y_i\}|$. Two inputs x, y are *neighbors* if $d(x,y) = 1$, i.e., if they differ in only one index. A randomized mechanism $\mathcal{K} : D^n \to \mathbb{R}^d$ preserves (ϵ, δ)-differential privacy if for any neighbors $x, y \in D^n$ and any set S of possible outputs:

$$\Pr[\mathcal{K}(x) \in S] \leq e^{\epsilon} \Pr[\mathcal{K}(y) \in S] + \delta$$

The main idea is that when ϵ and δ are very small, then the resulting output distributions will be very close, and therefore nearly indistinguishible.

The ϵ parameter, also called the *privacy budget*, controls the strength of the privacy guarantee. The δ parameter allows for a non-zero probability that the guarantee fails, and is typically set to a negligible value. The case when $\delta = 0$ is called *pure* or ϵ-differential privacy; the case when $\delta > 0$ is called *approximate* or (ϵ, δ)-differential privacy. Typical values for ϵ and δ are $\epsilon \in [0.1 - 10]$ and $\delta = \frac{1}{n^2}$ where n is the number of input entries [Dwork et al. 2014].

Function sensitivity. A function's *sensitivity* is the amount its output can change when its input changes. For example, the function $f(x) = x + x$ has a sensitivity of 2, since increasing or decreasing its input by 1 has the effect of increasing or decreasing its output by 2. A real-valued function f is called *n-sensitive* if $\max_{x,y:|x-y|\leq 1} |f(x) - f(y)| = n$.

This idea can be generalized to vector-valued functions. The *global L1 sensitivity* of a query $f : D^n \to \mathbb{R}^d$ is written GS_f and defined $GS_f = \max_{x,y:d(x,y)=1} |f(x) - f(y)|_1$ where $|\_ - \_|_1$ is the $L1$ norm (i.e., sum of pointwise distances between elements). The $L2$ sensitivity is analogous, using the $L2$ norm.

Differential privacy mechanisms. Two basic differential privacy mechanisms are the *Laplace mechanism* [Dwork et al. 2006], which preserves $(\epsilon, 0)$-differential privacy, and the *Gaussian mechanism* [Dwork et al. 2014], which preserves (ϵ, δ)-differential privacy. For a function $f : D^n \to \mathbb{R}^d$ with $L1$ sensitivity of Δ_1, the Laplace mechanism adds noise drawn from $\text{Lap}(\frac{\Delta_1}{\epsilon})$ to each element of the output. For f with $L2$ sensitivity of Δ_2, the Gaussian mechanism adds noise drawn from $\mathcal{N}(0, \frac{2\Delta_2^2 \ln(1.25/\delta)}{\epsilon^2})$ to each element.

The exponential mechanism [McSherry and Talwar 2007] selects an element of a set based on the scores assigned to each element by a *scoring function*. Let $u : D^n \times \mathcal{R} \to \mathbb{R}$ be a scoring function with $L1$ sensitivity Δ. The mechanism selects and outputs an element $r \in \mathcal{R}$ with probability proportional to $\exp(\frac{\epsilon u(x,r)}{2\Delta})$, and preserves $(\epsilon, 0)$-differential privacy.

Composition. A key property of differential privacy is that differentially private computations *compose*. The sequential composition theorem says that if $\mathcal{M}_1$ and $\mathcal{M}_2$ satisfy (ϵ, δ)-differential privacy, then their combination satisfies $(2\epsilon, 2\delta)$-differential privacy.

Tighter bounds on privacy cost can be achieved using the advanced composition theorem [Dwork et al. 2014], at the expense of increasing δ. The advanced composition theorem says that for

Variant	**Sequential Composition**	k-**Loop**	**Basic Mechanism**
ϵ-DP [Dwork et al. 2014]	$\epsilon_1 + \epsilon_2 \triangleq \epsilon_1 + \epsilon_2$	$k\epsilon$	*Laplace*
(ϵ, δ)-DP [Dwork et al. 2014]	$(\epsilon_1, \delta_1) + (\epsilon_2, \delta_2) \triangleq (\epsilon_1 + \epsilon_2, \delta_1 + \delta_2)$	$(k\epsilon, k\delta)$	*Gaussian*
RDP [Mironov 2017]	$(\alpha, \epsilon_1) + (\alpha, \epsilon_2) \triangleq (\alpha, \epsilon_1 + \epsilon_2)$	$(\alpha, k\epsilon)$	*Gaussian*
zCDP [Bun and Steinke 2016]	$\rho_1 + \rho_2 \triangleq \rho_1 + \rho_2$	$k\rho$	*Gaussian*
tCDP [Bun et al. 2018]	$(\rho_1, \omega_1) + (\rho_2, \omega_2) \triangleq (\rho_1 + \rho_2, \min(\omega_1, \omega_2))$	$(k\rho, \omega)$	*Sinh-normal*

Fig. 1. Variants of Differential Privacy

$0 < \epsilon' < 1$ and $\delta' > 0$, the class of (ϵ, δ)-differentially private mechanisms satisfies $(\epsilon', k\delta + \delta')$-differential privacy under k-fold adaptive composition (e.g. a loop with k iterations) for $\epsilon' = 2\epsilon\sqrt{2k\ln(1/\delta')}$.

The moments accountant was introduced by Talwar et al. [Abadi et al. 2016] specifically for stochastic gradient descent in deep learning applications. It provides tight bounds on privacy loss in iterative applications of the Gaussian mechanism, as in SGD. The Rényi differential privacy and zero-concentrated differential privacy generalize the ideas behind the moments accountant.

Variants of differential privacy. In addition to ϵ and (ϵ, δ)-differential privacy, other variants of differential privacy with significant benefits have recently been developed. Three examples are Rényi differential privacy (RDP) [Mironov 2017], zero-concentrated differential privacy (zCDP) [Bun and Steinke 2016], and truncated concentrated differential privacy (tCDP) [Bun et al. 2018]. Each one has different privacy parameters and a different form of sequential composition, summarized in Figure 1. The basic mechanism for RDP and zCDP is the Gaussian mechanism; tCDP uses a novel *sinh-normal* mechanism [Bun et al. 2018] which decays more quickly in its tails. All three can be converted to (ϵ, δ)-differential privacy, allowing them to be compared and composed with each other. These three variants provide asymptotically tight bounds on privacy cost under composition, while at the same time eliminating the "catastrophic" privacy failure that can occur with probability δ under (ϵ, δ)-differential privacy.

Group privacy. Differential privacy is normally used to protect the privacy of individuals, but it turns out that protection for an individual also translates to (weaker) protection for *groups* of individuals. A mechanism which provides pure ϵ-differential privacy for individuals also provides $k\epsilon$-differential privacy for groups of size k [Dwork et al. 2014]. Group privacy also exists for (ϵ, δ)-differential privacy, RDP, zCDP, and tCDP, but the scaling of the privacy parameters is nonlinear.

2.2 Related Work

Language-based approaches for differential privacy fall into two categories: approaches based on type systems, and those based on program logics. Barthe et al. [Barthe et al. 2016d] provide a survey. The type-system based approaches are most related to our work, but program-logic-based approaches have also received considerable attention in recent years [Barthe et al. 2016b,c, 2013; Barthe and Olmedo 2013; Sato 2016; Sato et al. 2019].

Linear Type Systems. Type-system-based solutions to proving that a program adheres to differential privacy began with Reed and Pierce's *Fuzz* language [Reed and Pierce 2010], which is based on linear typing. *Fuzz*, as well as subsequent work based on linear types aided by SMT solvers [Gaboardi et al. 2013], supports type inference of privacy bounds with type-level dependency and higher-order composition of programs. However, these systems only support the original and most basic variant of differential privacy called *ϵ-differential privacy*. More recent variants, like (ϵ, δ)-differential privacy [Dwork et al. 2014] and others [Bun et al. 2018; Bun and Steinke 2016;

Mironov 2017], improve on ϵ-differential privacy by providing vastly more accurate answers for the same amount of privacy "cost" (at the expense of introducing a negligible chance of failure).

As described by Azevedo de Amorim et al. [de Amorim et al. 2018], encoding (ϵ, δ)-differential privacy in linear type systems like *Fuzz* is particularly challenging because these systems place restrictions on the interpretation of the linear function space, and (ϵ, δ)-differential privacy does not satisfy these restrictions. In particular, using *Fuzz* requires that the desired notion of privacy can be recovered from an instantiation of function sensitivity for an appropriately defined metric on probabilistic functions. No such metric can be defined for (ϵ, δ)-differential privacy, preventing a straightforward interpretation of linear functions as (ϵ, δ)-differentially private functions.

In their work, Azevedo de Amorim et al. [de Amorim et al. 2018] define a *path construction* to encode non-linear scaling via an indexed probability monad, which can be used to extend *Fuzz* with support for arbitrary relational properties (including (ϵ, δ)-differential privacy). However, this approach (1) internalizes the use of *group privacy* [Dwork et al. 2014] which in many cases provides sub-optimal bounds on privacy cost–and (2) is unable to provide privacy bounds for more than one input to a function–a useful capability of the original *Fuzz* language, and a necessary feature to obtain optimal privacy bounds for multi-argument functions.

Higher-order Relational Type Systems. Following the initial work on linear typing for differential privacy [Reed and Pierce 2010], a parallel line of work [Barthe et al. 2016a, 2015] leverages *relational refinement types* aided by SMT solvers in order to support type-level dependency of privacy parameters (à la DFuzz [Gaboardi et al. 2013]) in addition to more powerful variants of differential privacy such as (ϵ, δ)-differential privacy. These approaches support (ϵ, δ)-differential privacy, but did not support usable type inference until a recently proposed heuristic bi-directional type system [Çiçek et al. 2018]. Although a direct case study of bidirectional type inference for relational refinement types has not yet been applied to differential privacy, the possibility of such a system appears promising.

The overall technique for supporting (ϵ, δ)-differential privacy in these relational refinement type systems is similar to (and predates) Azevedo de Amorim et al.–privacy cost is tracked through an "effect" type, embodied by an indexed monad. It is this "effect"-based treatment of privacy cost that fundamentally limits these type system to not support multi-arity functions, resulting in non-optimal privacy bounds for some programs.

First-order Relational Type Systems. Yet another approach is LightDP which uses a light-weight relational type system to verify (ϵ, δ)-differential privacy bounds of first-order imperative programs [Zhang and Kifer 2017], and is suitable for verifying low-level implementations of differentially private mechanisms. A notable achievement of this work is a lightweight, automated verification of the Sparse Vector Technique [Dwork et al. 2014] (SVT). However, LightDP is not suitable for sensitivity analysis, an important component of differentially-private algorithm design. Differential privacy mechanisms often require knowledge of (or place restrictions on) function sensitivity of arguments to the mechanism. In principle, a language like *Fuzz* could be combined with LightDP to fully verify both an application which uses SVT, as well as the implementation of SVT itself.

Type Systems Enriched with Program Logics. At a high level, Fuzzi [Zhang et al. 2019] has a similar aim to Duet: supporting differential privacy for general-purpose programs and supporting recent variants of differential privacy. Duet is designed primarily as a fully-automated type system with a rich set of primitives for vector-based and higher-order programming; low-level mechanisms in Duet are opaque and trusted. On the other hand, Fuzzi is designed for general-purpose programming, low-level mechanism implementation, and their combination; however, to achieve this, Fuzzi has less support for higher-order programming and automation in typechecking.

	SA	HO	DT	MA	Rel-ext	ϵ-DP	(ϵ, δ)-DP	Rényi/zCDP/tCDP	SVT-imp
Fuzz [Reed and Pierce 2010]	✓	✓	✗	✓	✗	✓	✗	✗	✗
DFuzz [Gaboardi et al. 2013]	✓	✓	✓	✓	✗	✓	✗	✗	✗
PathC [de Amorim et al. 2018]	✓	✓	✗[1]	✗	✓	✓	✓	✓[2]	✗
HOARe^2 [Barthe et al. 2015]	✓	✓	✓	✗	✓	✓	✓	✓[2]	✗
LightDP [Zhang and Kifer 2017]	✗	✗	✓	✓	✗	✓	✓	✓[2]	✓
Fuzzi [Zhang et al. 2019]	✓	✗	✗[1]	✓	✓	✓	✓	✓	✓
DUET	✓	✓	✓	✓	✗	✓	✓	✓	✗

Fig. 2. Legend: SA = capable of sensitivity analysis; HO = support for higher order programming, program composition, and compound datatypes; DT = support for dependently typed privacy bounds; MA = support for distinct privacy bounds of multiple input arguments; Rel-ext = supports extensions to support non-differential-privacy relations; ϵ-DP = supports ϵ-differential-privacy; (ϵ, δ)-DP = supports (ϵ, δ)-differential-privacy; Rényi/zCDP/tCDP: supports Rényi, zero-concentrated and truncated concentrated differential privacy; SVT-imp: supports verified *implementation* of the sparse vector technique. 1: This limitation is not fundamental and could be supported by simple extension to underlying type theory. 2: Not described in prior work, but could be achieved through a trivial extension to existing support for (ϵ, δ)-differential privacy.

2.3 Our Approach

We show the strengths and limitations of DUET in relation to approaches from prior work in Figure 2. In particular, strengths of Duet w.r.t. prior work are: (1) DUET supports sensitivity analysis in combination with higher order programming, program composition, and compound datatypes, building on ideas from *Fuzz* (SA+HO); (2) DUET supports type-level dependency on values, which enables differentially private algorithms to be verified w.r.t. symbolic privacy parameters, building on ideas from DFuzz (DT); (3) DUET supports calculation of independent privacy costs for multiple program arguments via a novel approach (MA); and (4) DUET supports (ϵ, δ)-differential privacy—in addition to other recent powerful variants, such as Rényi, zero-concentrated and truncated concentrated differential privacy—via a novel approach ((ϵ, δ)-DP, Rényi/ZC/TC)).

In striking this balance, DUET comes with known limitations: (1) DUET is not easy to extend with new relational properties (Rel-ext); and (2) DUET is not suitable for verifying implementations of low-level mechanisms, such as the implementation of advanced composition, gradient operations, and the sparse-vector technique (SVT-imp).

3 DUET: A LANGUAGE FOR PRIVACY

This section describes the syntax, type system and formal properties of DUET. Our design of DUET is the result of two key insights.

(1) *Linear typing, when restricted to disallow scaling, can be a powerful foundation for enforcing (ϵ, δ)-differential privacy.* Privacy bounds in (ϵ, δ)-differential privacy do not scale linearly, and cannot be accurately modeled by linear type systems which permit unrestricted scaling.
(2) *Sensitivity and privacy cost are distinct properties, and warrant distinct type systems to enforce them.* Our design for DUET is a co-design of two distinct, mutually embedded languages: one for sensitivity which leverages linear typing *with* scaling a la *Fuzz*, and one for privacy which leverages linear typing *without* scaling and is novel in this work.

Before describing the syntax, semantics and types for each of DUET's two languages, we first provide some context which motivates each design decision made. We do this through several small examples and type signatures drawn from state-of-the-art type systems such as *Fuzz* [Reed and Pierce 2010], *HOARe*^{2} [Barthe et al. 2015] and Azevedo de Amorim et al's *path construction* [de Amorim et al. 2018].

3.1 Design Challenges

Higher-Order Programming. An important design goal of Duet is to support sensitivity analysis of higher-order, general purpose programs. Prior work (*Fuzz* and *HOARe*^{2}) has demonstrated exactly this, and we build on their techniques. In *Fuzz*, the types for the higher-order map function and a list of reals named xs looks like this:

$$\begin{aligned} &\text{map} : (\tau_1 \multimap_s \tau_2) \multimap_\infty \text{list } \tau_1 \multimap_s \text{list } \tau_2 \\ &xs : \text{list } \mathbb{R} \end{aligned}$$

The type of map reads: "Take as a first argument an s-sensitive function from τ_1 to τ_2 which map is allowed to use as many times as it wants. Take as second argument a list of τ_1, and return a result list of τ_2 which is s-sensitive in the list of τ_1." Two programs that use map might look like this:

$$\begin{aligned} &\text{map } (\lambda\ x \to x + 1)\ \text{xs} \qquad (1) \\ &\text{map } (\lambda\ x \to x + x)\ \text{xs} \qquad (2) \end{aligned}$$

The *Fuzz* type system reports that *(1)* is 1-sensitive in xs, and that *(2)* is 2-sensitive in xs. To arrive at this conclusion, the *Fuzz* type checker is essentially counting how many times x is used in the body of the lambda, and type soundness for *Fuzz* means that these counts correspond to the semantic property of function sensitivity.

In *HOARe*^{2} the type for map is instead:

$$\begin{aligned} \text{map} \ :\ & (\forall s'.\ \{x :: \tau_1 \mid \mathfrak{D}_{\tau_1}(x_\triangleleft, x_\triangleright) \le s'\} \to \{y :: \tau_2 \mid \mathfrak{D}_{\tau_2}(y_\triangleleft, y_\triangleright) \le s \cdot s'\}) \\ & \to \forall s'.\ \{xs :: list\ \tau_1 \mid \mathfrak{D}_{(list\ \tau_1)}(xs_\triangleleft, xs_\triangleright) \le s'\} \to \{y :: list\ \tau_2 \mid \mathfrak{D}_{(list\ \tau_2)}(ys_\triangleleft, ys_\triangleright) \le s \cdot s'\} \end{aligned}$$

This type for map means the same thing as the *Fuzz* type shown above, and *HOARe*^{2} likewise reports that *(1)* is 1-sensitive and *(2)* is 2-sensitive, each in xs, and where $\mathfrak{D}_\tau$ is some family of distance metrics indexed by types τ. To arrive at this conclusion, *HOARe*^{2} generates relational verification conditions (where, e.g., $x_\triangleleft$ is drawn from a hypothetical "first/left run" of the program, and $x_\triangleright$ is drawn from a hypothetical "second/right run" of the program) which are discharged by an external solver (e.g., SMT). In this approach, sensitivity is not concluded via an interpretation of a purely *syntactic* type system (e.g., linear typing in *Fuzz*), rather the relational *semantic* property of sensitivity (and its scaling) is embedded directly in the relational refinements of higher-order function types.

In designing Duet, we follow the design of *Fuzz* in that programs adhere to a linear type discipline, i.e., the mechanics of our type system is based on counting variables and (in some cases) scaling, and we prove a soundness theorem that says well-typed programs are guaranteed to be sensitive/private programs. Our type for map is identical to the one shown above for *Fuzz*.

Non-Linear Scaling. *Fuzz* encodes an ϵ-differentially private function as an ϵ-sensitive function which returns a monadic type $\bigcirc\ \tau$. The Laplace differential privacy mechanism is then encoded in *Fuzz* as an ϵ-sensitive function from $\mathbb{R}$ to $\bigcirc\ \mathbb{R}$:

$$\text{laplace} : \mathbb{R} \multimap_\epsilon \bigcirc\ \mathbb{R}$$

Because the metric on distributions for pure ϵ-differential privacy scales linearly, laplace can be applied to a 2-sensitive argument to achieve 2ϵ-differential privacy, e.g.:

$$\text{laplace}\ (x + x)$$

gives 2ϵ-differential privacy for x. Adding more advanced variants of differential privacy like (ϵ, δ) to *Fuzz* has proved challenging because these variants do not scale linearly. Azevedo de Amorim et al's *path construction* successfully adds (ϵ, δ)-differential privacy to *Fuzz* by tracking privacy "cost" as an index on the monadic type operator $\bigcirc_{\epsilon,\delta}$. However, in order to interpret a function application like the one shown, the group privacy property for (ϵ, δ)-differential privacy must be

used, which results in undesirable non-linear scaling of the privacy cost. The derived bound for this program using group privacy (for $k = 2$) is not $(2\epsilon, 2\delta)$ but $(2\epsilon, 2e^{\epsilon}\delta)$ [Dwork et al. 2014]. As a result, achieving a desired ϵ and δ by treating an s-sensitive function as 1-sensitive and leveraging group privacy requires adding much more noise than simply applying the Gaussian mechanism with a sensitivity of s.

In *HOARe*^{2}, the use of scaling which might warrant the use of group privacy is explicitly disallowed in the stated relational refinement type. This is in contrast to sensitivity, which likewise must explicitly allow arbitrary scaling. The type for gauss in *HOARe*^{2} (the analogous mechanism to laplace in the (ϵ, δ)-differential privacy setting) is written:

$$\text{gauss} : \{x :: \mathbb{R} \mid \mathfrak{D}_{\mathbb{R}}(x_{\triangleleft}, x_{\triangleright}) \leq 1\} \to \mathbf{M}_{\epsilon,\delta}\ \mathbb{R}$$

Notice the assumed sensitivity of x to be bounded by 1, not some arbitrary s' to be scaled in the output refinement (as was seen in the type for map in *HOARe*^{2} above). In this way, *HOARe*^{2} is able to restrict *uses* of gauss to strictly 1-sensitive arguments, a restriction that is not possible in a pure linear type system where arbitrary program composition is allowed and interpreted via scaling.

In DUET, we co-design two languages which are mutually embedded inside one another. The sensitivity language is nearly identical to *Fuzz*, supports arbitrary scaling, and is never interpreted to mean privacy. The privacy language is also linearly typed, but restricts function call parameters to be strictly 1-sensitive—a property established in the sensitivity fragment. The gauss mechanism in DUET is (essentially) given the type:

$$\text{gauss} : \mathbb{R}@\langle\epsilon, \delta\rangle \multimap^* \mathbb{R}$$

where $\multimap^*$ is the function space in DUET's privacy language, and the annotation $@\langle\epsilon, \delta\rangle$ tracks the privacy cost of that argument following a linear typing discipline.

Multiple Private Parameters. Both *HOARe*^{2} and the *path construction* track (ϵ, δ)-differential privacy via an indexed monadic type, notated $\mathbf{M}_{\epsilon,\delta}$ and $\bigcirc_{\epsilon,\delta}$ respectively. E.g., a program that returns an (ϵ, δ)-differentially private real number has the type $\mathbf{M}_{\epsilon,\delta}(\mathbb{R})$ in *HOARe*^{2}. These monadic approaches to privacy inherently follow an "effect" type discipline, and as a result the monad index must track the *sum total of all privacy costs to any parameter*. For example, a small program that takes two parameters, applies a mechanism to enforce differential privacy for each parameter, and adds them together, will report a double-counting of privacy cost. E.g., in this *HOARe*^{2} program (translated to Haskell-ish "do"-notation):

$$\texttt{let}\ f = \lambda\ x\ y \to \texttt{do}\ \{\ r_1 \leftarrow \mathit{gauss}_{\epsilon,\delta}\ x\ ;\ r_2 \leftarrow \mathit{gauss}_{\epsilon,\delta}\ y\ ;\ \mathit{return}\ (r_1 + r_2)\ \}$$

The type of f in *HOARe*^{2} reports that it costs $(2\epsilon, 2\delta)$ privacy:

$$f : \{x :: \mathbb{R} \mid \mathfrak{D}_{\mathbb{R}}(x_{\triangleleft}, x_{\triangleright}) \leq 1\} \to \{y :: \mathbb{R} \mid \mathfrak{D}_{\mathbb{R}}(y_{\triangleleft}, y_{\triangleright}) \leq 1\} \to \mathbf{M}_{2\epsilon,2\delta}\ \mathbb{R}$$

This bound is too conservative in many cases: it is the best bound in the case that f is applied to the same variable for both arguments (e.g., in $f\ a\ a$), however, if f is applied to *different* variables (e.g., in $f\ a\ b$) then a privacy cost of $(2\epsilon, 2\delta)$ is still claimed, interpreted as *for either or both variables* $2\epsilon, 2\delta$ privacy is consumed. A better accounting of privacy in this second case should report (ϵ, δ)-differential privacy *independently* for both variables a and b, and such accounting is not possible in either *HOARe*^{2} or the *path construction*.

In DUET, we track privacy following a *co-effect* discipline (linear typing without scaling), as opposed to an effect discipline, in order to distinguish privacy costs independently for each variable. The type of the above program in DUET is:

$$f : (\mathbb{R}@\langle\epsilon, \delta\rangle, \mathbb{R}@\langle\epsilon, \delta\rangle) \multimap^* \mathbb{R}$$

indicating that f "costs" (ϵ, δ) for each parameter independently, and only when f is called with two identical variables as arguments are they combined as $(2\epsilon, 2\delta)$.

Due to limitations of linear logic in the absence of scaling, privacy lambdas must be multi-argument in the core design of Duet—they cannot be recovered by single-argument lambdas. As a consequence, our privacy language is not Cartesian closed.

3.2 Duet by Example

Sensitivity. Duet consists of two languages: one for tracking sensitivities (typeset in green), and one for tracking privacy cost (typeset in red). The sensitivity language is similar to that of DFuzz [Gaboardi et al. 2013]; its typing rules track the sensitivity of each variable by annotating the context. For example, the expression $x + x$ is 2-sensitive in x; the typing rules in Figure 4 allow us to conclude:

$$\{x :_2 \mathbb{R}\} \vdash x + x : \mathbb{R}$$

In this case, the context $\{x :_2 \mathbb{R}\}$ tells us that the expression is 2-sensitive in x. The same idea works for functions; for example:

$$\emptyset \vdash \lambda x : \mathbb{R} \Rightarrow x + x : \mathbb{R} \multimap_2 \mathbb{R}$$

Here, the context is empty; instead, the function's sensitivity to its argument is encoded in an annotation on its type (the 2 in $\mathbb{R} \multimap_2 \mathbb{R}$). Applying such a function to an argument *scales* the sensitivity of the argument by the sensitivity of the function. This kind of scaling is appropriate for sensitivities, and even has the correct effect for higher-order functions. For example:

$$\begin{array}{l}
\{y :_2 \mathbb{R}\} \vdash (\lambda x : \mathbb{R} \Rightarrow x + x)\ y : \mathbb{R} \\
\{y :_4 \mathbb{R}\} \vdash (\lambda x : \mathbb{R} \Rightarrow x + x)\ (y + y) : \mathbb{R} \\
\{y :_4 \mathbb{R}, z :_2 \mathbb{R}\} \vdash (\lambda x : \mathbb{R} \Rightarrow x + x)\ (y + y + z) : \mathbb{R} \\
\{y :_1 \mathbb{R}\} \vdash \lambda x : \mathbb{R} \Rightarrow y : \mathbb{R} \multimap_0 \mathbb{R} \\
\{y :_1 \mathbb{R}, z :_0 \mathbb{R}\} \vdash (\lambda x : \mathbb{R} \Rightarrow y)\ z : \mathbb{R} \\
\{y :_2 \mathbb{R}, z :_0 \mathbb{R}\} \vdash (\lambda f : \mathbb{R} \multimap_0 \mathbb{R} \Rightarrow (f\ z) + (f\ z))\ (\lambda x : \mathbb{R} \Rightarrow y) : \mathbb{R}
\end{array}$$

Privacy. Differentially private mechanisms like the Gaussian mechanism [Dwork et al. 2014] specify how to add noise to a function with a particular sensitivity in order to ensure differential privacy. In Duet, such mechanisms form the interface between the sensitivity language and the privacy language. For example:

$$\{x :_{\epsilon,\delta} \mathbb{R}\} \vdash \mathsf{gauss}[\mathbb{R}^+[2.0], \epsilon, \delta]\ \langle x \rangle\ \{x + x\} : \mathbb{R}$$

In a gauss expression, the first three elements (inside the square brackets) represent the maximum allowed sensitivity of variables in the expression's body, and the desired privacy parameters ϵ and δ. The fourth element (here, $\langle x \rangle$) is a list of variables whose privacy we are interested in tracking. Variables not in this list will be assigned infinite privacy cost.

The value of the gauss expression is the value of its fifth element (the "body"), plus enough noise to ensure the desired level of privacy. The body of a gauss expression is a sensitivity expression, and the gauss expression is well-typed only if its body typechecks in a context assigning a sensitivity to each variable of interest which does not exceed the maximum allowed sensitivity. For example, the expression $\mathsf{gauss}[\mathbb{R}^+[1.0], \epsilon, \delta]\ \langle x \rangle\ \{x + x\}$ is not well-typed, because $x + x$ is 2-sensitive in x, but the maximum allowed sensitivity is 1.

Privacy expressions like the example above are typed under a *privacy context* which records privacy cost for individual variables. The context for this example ($\{x :_{\epsilon,\delta} \mathbb{R}\}$) says that the expression provides (ϵ, δ)-differential privacy for the variable x. Tracking privacy costs using a co-effect

discipline allows precise tracking of the privacy cost for programs with multiple inputs:

$$\{x :_{\epsilon,\delta} \mathbb{R}, y :_{\epsilon,\delta} \mathbb{R}\} \vdash \mathsf{gauss}[\mathbb{R}^+[1.0], \epsilon, \delta]\ \langle x, y\rangle\ \{x + y\} : \mathbb{R}$$

The BIND rule encodes the sequential composition property of differential privacy. For example:

$$\begin{array}{l}\{x :_{2\epsilon,2\delta} \mathbb{R}\} \vdash \\ \quad v_1 \leftarrow \mathsf{gauss}[\mathbb{R}^+[1.0], \epsilon, \delta]\ \langle x\rangle\ \{x\}\ ; \\ \quad v_2 \leftarrow \mathsf{gauss}[\mathbb{R}^+[1.0], \epsilon, \delta]\ \langle x\rangle\ \{x\}\ ; \\ \quad \mathsf{return}\ v_1 + v_2 \\ : \mathbb{R}\end{array} \qquad \begin{array}{l}\{x :_{\epsilon,\delta} \mathbb{R}, y :_{\epsilon,\delta} \mathbb{R}\} \vdash \\ \quad v_1 \leftarrow \mathsf{gauss}[\mathbb{R}^+[1.0], \epsilon, \delta]\ \langle x\rangle\ \{x\}\ ; \\ \quad v_2 \leftarrow \mathsf{gauss}[\mathbb{R}^+[1.0], \epsilon, \delta]\ \langle y\rangle\ \{y\}\ ; \\ \quad \mathsf{return}\ v_1 + v_2 \\ : \mathbb{R}\end{array}$$

In the example on the left, the Gaussian mechanism is applied to x twice, so the total privacy cost for x is $(2\epsilon, 2\delta)$. In the example on the right, x and y are each used once, and their privacy costs are tracked separately. The RETURN rule provides a second interface between the sensitivity and privacy languages: a *return* expression is part of the privacy language, but its argument is a sensitivity expression. The value of a *return* expression is exactly the value of its argument, so the variables used in its argument are assigned *infinite* privacy cost. *return* expressions are therefore typically used to compute on values which are *already* differentially private (like v_1 and v_2 above), since infinite privacy cost is not a problem in that case.

Gradient descent. Machine learning problems are typically defined in terms of a *loss function* $\mathcal{L}(\theta; X, y)$ on a *model* θ, *training samples* $X = (x_1, x_2, ..., x_n)$ (in which each sample is typically represented as a *feature vector*) and corresponding *labels* $y = (y_1, y_2, ..., y_n)$ (i.e. the prediction target). The training task is to find a model $\hat{\theta}$ which minimizes the loss on the training samples (i.e. $\hat{\theta} = \mathrm{argmin}_\theta \mathcal{L}(\theta; X, y)$.

One solution to the training task is *gradient descent*, which starts with an initial guess for θ and iteratively moves in the direction of an improved θ until the current setting is close to $\hat{\theta}$. To determine which direction to move, the algorithm evaluates the *gradient* of the loss, which yields a vector representing the direction of greatest *increase* in $\mathcal{L}(\theta; X, y)$. Then, the algorithm moves in the *opposite* direction.

To ensure differential privacy for gradient-based algorithms, we need to bound the sensitivity of the gradient computation. The gradients for many kinds of convex loss functions are 1-*Lipschitz* [Wu et al. 2017]: if each sample in $X = (x_1, ..., x_n)$ has bounded $L2$ norm (i.e. $\|x_i\|_2 \leq 1$), then for all models θ and labelings y, the gradient $\nabla(\theta; X, y)$ has $L2$ sensitivity bounded by 1. For now, we will assume the existence of a function called gradient with this property (more details in Section 4).

$$\mathsf{gradient} : \mathbb{M}^{\mathsf{U}}_{L2}[1, n]\ \mathbb{R} \multimap_\infty \mathbb{M}^{\mathsf{U}}_{L\infty}[m, n]\ \mathbb{D} \multimap_{\frac{1}{m}} \mathbb{M}^{\mathsf{U}}_{L\infty}[m, 1]\ \mathbb{D} \multimap_{\frac{1}{m}} \mathbb{M}^{\mathsf{U}}_{L2}[1, n]\ \mathbb{R}$$

The function's arguments are the current θ, a $m \times n$ matrix X containing n training samples, and a $1 \times n$ matrix y containing the corresponding labels. In Duet, the type $\mathbb{M}^{\mathsf{U}}_{L\infty}[m, n]\ \mathbb{D}$ represents a $m \times n$ matrix of *discrete* real numbers; neighboring matrices of this type differ arbitrarily in a single row. The function's output is a new θ of type $\mathbb{M}^{\mathsf{U}}_{L2}[1, n]\ \mathbb{R}$, representing a matrix of real numbers with bounded $L2$ sensitivity (see Section 4 for details on matrix types). We can use the gradient function to implement a differentially private gradient descent algorithm:

$$\begin{array}{l}\mathsf{noisy\text{-}gradient\text{-}descent}(X, y, k, \epsilon, \delta) \triangleq \\ \quad \mathsf{let}\ \theta_0 = \mathsf{zeros}\ (\mathsf{cols}\ X_1)\ \mathsf{in} \\ \quad \mathsf{loop}[\delta']\ k\ \mathsf{on}\ \theta_0\ \langle X_1, y\rangle\ \{t, \theta \Rightarrow \\ \qquad g_p \leftarrow \mathsf{mgauss}[\frac{1}{m}, \epsilon, \delta]\ \langle X, y\rangle\ \{\mathsf{gradient}\ \theta\ X\ y\}\ ; \\ \qquad \mathsf{return}\ \theta - g_p \quad \}\end{array}$$

The arguments to our algorithm are the training data (X and y), the desired number of iterations k, and the privacy parameters ϵ and δ. The first line constructs an initial model θ_0 consisting of zeros for all parameters. Lines 2-4 represent the iterative part of the algorithm: k times, compute the gradient of the loss on X and y with respect to the current model, add noise to the gradient using the Gaussian mechanism, and subtract the gradient from the current model (thus moving in the opposite direction of the gradient) to improve the model.

The typing rules presented in Figure 4 allow us to derive a privacy bound for this algorithm which is equivalent to manual proof of Bassily et al. [Bassily et al. 2014b]. Based on the type of the gradient function, the $\multimap$-E rule allows us to conclude that the gradient operation is $\frac{1}{m}$-sensitive in the training data, which is reflected by the sensitivity annotations in the context:

$$\{\theta :_\infty \tau_1, X :_{\frac{1}{m}} \tau_2, y :_{\frac{1}{m}} \tau_3\} \vdash \text{gradient } \theta\ X\ y : \mathbb{M}^{\mathsf{U}}_{L2}[1,n]\ \mathbb{R}$$
$$\textit{where } \tau_1 = \mathbb{M}^{\mathsf{U}}_{L2}[1,n]\ \mathbb{R}$$
$$\tau_2 = \mathbb{M}^{\mathsf{U}}_{L\infty}[m,n]\ \mathbb{D}$$
$$\tau_3 = \mathbb{M}^{\mathsf{U}}_{L\infty}[m,1]\ \mathbb{D}$$

Next, the MGauss rule represents the use of the Gaussian mechanism, and transitions from the sensitivity language (implementing the gradient) to the privacy language (in which we use the noisy gradient). The rule allows us to conclude that since the sensitivity of the gradient computation is $\frac{1}{m}$, our use of the Gaussian mechanism satisfies (ϵ, δ)-differential privacy. This context is a *privacy* context, and its annotations represent privacy costs rather than sensitivities.

$$\{\theta :_\infty \tau_1, X :_{\langle\epsilon,\delta\rangle} \tau_2, y :_{\langle\epsilon,\delta\rangle} \tau_3\} \vdash \texttt{mgauss}[\frac{1}{m}, \epsilon, \delta]\ \texttt{<}X, y\texttt{>}\ \{\text{gradient } \theta\ X\ y\} : \mathbb{M}^{\mathsf{U}}_{L2}[1,n]\ \mathbb{R}$$

Finally, the Loop rule for advanced composition allows us to derive a bound on the total privacy cost of the iterative algorithm, based on the number of times the loop runs:

$$\{\theta :_\infty \tau_1, X :_{\langle\epsilon',k\delta+\delta'\rangle} \tau_2, y :_{\langle\epsilon',k\delta+\delta'\rangle} \tau_3\} \vdash \texttt{loop}[\delta']\ k\ \texttt{on}\ \theta_0\ \texttt{<}X_1, y\texttt{>}\ \{t, \theta \Rightarrow \ldots\} : \mathbb{M}^{\mathsf{U}}_{L2}[1,n]\ \mathbb{R}$$
$$\textit{where } \epsilon' = 2\epsilon\sqrt{2k\log(1/\delta')}$$

Variants of Differential Privacy. The typing rules presented in Figure 4 are specific to (ϵ, δ)-differential privacy, but the same framework can be easily extended to support the other variants described in Figure 1. New variants can be supported by making three simple changes: (1) Modify the *privacy cost* syntax p to describe the privacy parameters of the new variant; (2) Modify the sum operator $\_+\_$ to reflect sequential composition in the new variant; and (3) Modify the typing for basic mechanisms (e.g. gauss) to reflect corresponding mechanisms in the new variant. The extended version of this paper [Near et al. 2019] includes typing rules for the variants in Figure 1.

As an example, considering the following variant of the noisy gradient descent algorithm presented earlier, but with ρ-zCDP instead of (ϵ, δ)-differential privacy. There are only two differences: the loop construct under zCDP has no δ' parameter, since standard composition yields tight bounds, and the mgauss construct has a single privacy parameter (ρ) instead of ϵ and δ.

$$\begin{array}{l}
\text{noisy-gradient-descent}(X, y, k, \rho) \triangleq \\
\quad \texttt{let}\ \theta_0 = \text{zeros}\ (\text{cols}\ X_1)\ \texttt{in} \\
\quad \texttt{loop}\ k\ \texttt{on}\ \theta_0\ \texttt{<}X_1, y\texttt{>}\ \{t, \theta \Rightarrow \\
\qquad g_p \leftarrow \texttt{mgauss}[\frac{1}{m}, \rho]\ \texttt{<}X, y\texttt{>}\ \{\text{gradient } \theta\ X\ y\}\ ; \\
\qquad \texttt{return}\ \theta - g_p \quad \}
\end{array}$$

Typechecking for this version proceeds in the same way as before, with the modified typing rules; the resulting privacy context gives both X and y a privacy cost of $k\rho$.

Mixing Variants. DUET allows mixing variants of differential privacy in a single program. For example, the total privacy cost of an algorithm is often given in (ϵ, δ) form, to enable comparing the costs of different algorithms; we can use this feature of DUET to automatically derive the cost of our zCDP-based gradient descent in terms of ϵ and δ.

$$\begin{array}{l}\text{noisy-gradient-descent}(X, y, k, \rho, \delta) \triangleq \\ \quad \texttt{let}\ \theta_0 = \text{zeros}\ (\text{cols}\ X_1)\ \texttt{in} \\ \quad \texttt{ZCDP}\ [\delta]\ \{\ \texttt{loop}\ k\ \texttt{on}\ \theta_0\ \texttt{<}X_1, y\texttt{>}\ \{t, \theta \Rightarrow \\ \qquad\qquad g_p \leftarrow \texttt{mgauss}[\frac{1}{m}, \rho]\ \texttt{<}X, y\texttt{>}\ \{\text{gradient}\ \theta\ X\ y\}\ ; \\ \qquad\qquad \texttt{return}\ \theta - g_p \quad \}\ \ \}\end{array}$$

The ZCDP {...} construct represents embedding a mechanism which satisfies ρ-zCDP in another mechanism which provides (ϵ, δ)-differential privacy. The rule for typechecking this construct encodes the property that if a mechanism satisfies ρ-zCDP, it also satisfies $(\rho + 2\sqrt{\rho \log(1/\delta)}, \delta)$-differential privacy [Bun and Steinke 2016]. Using this rule, we can derive a total privacy cost for the gradient descent algorithm in terms of ϵ and δ, but using the tight bound on composition that zCDP provides.

$$\{X :_{\langle \epsilon', \delta \rangle} \tau_2, y :_{\langle \epsilon', \delta \rangle} \tau_3, k :_\infty \tau_4, \rho :_\infty \tau_5\} \vdash \text{noisy-gradient-descent}(X, y, k, \rho, \delta) : \mathbb{M}^{\mathsf{U}}_{L2}[1, n]\ \mathbb{R}$$
$$\textit{where}\ \epsilon' = k\rho + 2\sqrt{k\rho \log(1/\delta)},\ \ \tau_3 = \mathbb{R}^+[k],\ \ \textit{and}\ \ \tau_5 = \mathbb{R}^+[\rho]$$

We might also want to nest these conversions. For example, when the dimensionality of the training data is very small, the Laplace mechanism might yield more accurate results than the Gaussian mechanism (due to the shape of the distribution). To use the Laplace mechanism in an iterative algorithm which satisfies zCDP, we can use the fact that any ϵ-differentially private mechanism also satisfies $\frac{1}{2}\epsilon^2$-zCDP; by nesting conversions, we can determine the total cost of the algorithm in terms of ϵ and δ.

$$\begin{array}{l}\text{noisy-gradient-descent}(X, y, k, \epsilon, \delta) \triangleq \\ \quad \texttt{let}\ \theta_0 = \text{zeros}\ (\text{cols}\ X_1)\ \texttt{in} \\ \quad \texttt{ZCDP}\ [\delta]\ \{\ \texttt{loop}\ k\ \texttt{on}\ \theta_0\ \texttt{<}X_1, y\texttt{>}\ \{t, \theta \Rightarrow \\ \qquad\qquad g_p \leftarrow \texttt{EPS\_DP}\ \{\ \texttt{mlaplace}[\frac{1}{m}, \epsilon]\ \texttt{<}X, y\texttt{>}\ \{\text{gradient}\ \theta\ X\ y\}\ \}\ ; \\ \qquad\qquad \texttt{return}\ \theta - g_p \quad \}\ \ \}\end{array}$$

$$\{X :_{\langle \epsilon', \delta \rangle} \tau_2, y :_{\langle \epsilon', \delta \rangle} \tau_3, k :_\infty \mathbb{R}^+, \rho :_\infty \mathbb{R}^+\} \vdash \text{noisy-gradient-descent}(X, y, k, \epsilon, \delta) : \mathbb{M}^{\mathsf{U}}_{L2}[1, n]\ \mathbb{R}$$
$$\textit{where}\ \epsilon' = \tfrac{1}{2}k\epsilon^2 + 2\sqrt{\tfrac{1}{2}k\epsilon^2 \log(1/\delta)}$$

Such nestings are sometimes useful in practice: in Section 5, we will define a variant of the Private Frank-Wolfe algorithm which uses the exponential mechanism (which satisfies ϵ-differential privacy) in a loop for which composition is performed with zCDP, and report the total privacy cost in terms of ϵ and δ.

Contextual Modal Types. A new problem arises in the design of DUET governing the interaction of sensitivity and privacy languages: in general—and for very good reasons which are detailed in the next section—let-binding intermediate results in the privacy language doesn't always preserve typeability. Not only is let-binding intermediate results desirable for code readability, it can often be *essential* in order to achieve desirable performance. Consider a loop body which performs an expensive operation that does not depend on the inner-loop parameter:

$$\begin{array}{l}\lambda\ xs\ \theta_0 \rightarrow \texttt{loop}\ k\ \texttt{times on}\ \theta_0\ \{\ \theta \rightarrow \\ \qquad\qquad \text{gauss}_{\epsilon,\delta}\ (f\ (\text{expensive}\ xs)\ \theta))\ \}\end{array}$$

$$
\begin{array}{rcll}
m, n \in \mathbb{N} & & r \in \mathbb{R} & \dot{r}, \epsilon, \delta \in \mathbb{R}^+ \qquad x, y \in \mathrm{var} \\
s \in \mathrm{sens} & ::= & \dot{r} \mid \infty & p \in \mathrm{priv} ::= \epsilon, \delta \mid \infty \\
\tau \in \mathrm{type} & ::= & \mathbb{N} \mid \mathbb{R} \mid \mathbb{N}[n] \mid \mathbb{R}^+[\dot{r}] \mid \mathsf{box}[\Gamma_s]\ \tau & \textit{numeric and box} \\
& \mid & \tau \multimap_s \tau \mid (\tau@p, ..., \tau@p) \multimap^* \tau & \textit{functions} \\
\Gamma_s \in \mathrm{tcxt}_s & \triangleq & \mathrm{var} \rightharpoonup \mathrm{sens} \times \mathrm{type} ::= \{x :_s \tau, ..., x :_s \tau\} & \textit{sens. contexts} \\
\Gamma_p \in \mathrm{tcxt}_p & \triangleq & \mathrm{var} \rightharpoonup \mathrm{priv} \times \mathrm{type} ::= \{x :_p \tau, ..., x :_p \tau\} & \textit{priv. contexts} \\
e_s \in \mathrm{exp}_s & ::= & \mathbb{N}[n] \mid \mathbb{N}[\dot{r}] \mid n \mid r \mid \mathsf{real}\ e & \textit{numeric literals} \\
& \mid & e + e \mid e - e \mid e \cdot e \mid 1/e \mid e\ \mathsf{mod}\ e & \textit{arithmetic} \\
& \mid & x \mid \mathsf{let}\ x = e\ \mathsf{in}\ e \mid e\ e & \textit{let/sens. app.} \\
& \mid & s\lambda\ x : \tau \Rightarrow e \mid p\lambda\ (x : \tau, ..., x : \tau) \Rightarrow e & \textit{sens./priv. fun.} \\
& \mid & \mathsf{box}\ e \mid \mathsf{unbox}\ e & \textit{sensitivity capture} \\
e_p \in \mathrm{exp}_p & ::= & \mathsf{return}\ e \mid x \leftarrow e\ ;\ e \mid e(e, ..., e) & \textit{ret/bind/priv. app.} \\
& \mid & \mathsf{loop}[e]\ e\ \mathsf{on}\ e\ \langle x, ..., x\rangle\ \{x, x \Rightarrow e\} & \textit{finite iteration} \\
& \mid & \mathsf{gauss}[e, e, e]\ \langle x, ..., x\rangle\ \{e\} & \textit{gaussian noise}
\end{array}
$$

Fig. 3. Core Types and Terms

A simple refactoring achieves much better performance:

$$
\begin{array}{l}
\lambda\ xs\ \theta_0 \rightarrow \mathsf{let}\ \mathrm{temp} = \mathrm{expensive}\ xs\ \mathsf{in} \\
\quad \mathsf{loop}\ k\ \mathsf{times\ on}\ \theta_0\ \{\ \theta \rightarrow \\
\quad\quad \mathrm{gauss}_{\epsilon,\delta}\ (f\ \mathrm{temp}\ \theta)\ \}
\end{array}
$$

However instead of providing (ϵ, δ)-differential privacy for xs, as was the case before the refactor, the new program provides (ϵ, δ)-differential privacy for temp—an intermediate variable we don't care about—and makes no guarantees of privacy for xs.

To accommodate this pattern we borrow ideas from *contextual modal type theory* [Nanevski et al. 2008] to allow "boxing" a sensitivity context, and "unboxing" that context at a later time. In terms of differential privacy, the argument that the above loop is differentially private relies on the fact that $\mathrm{temp} \equiv \mathrm{expensive}(xs)$ is 1-sensitive in xs (assuming expensive is 1-sensitive), a property which is lost by the typing rule for let in the privacy language. We therefore "box" this sensitivity information outside the loop, and "unbox" it inside the loop, like so:

$$
\begin{array}{l}
\lambda\ xs\ \theta_0 \rightarrow \mathsf{let}\ \mathrm{temp} = \mathsf{box}\ (\mathrm{expensive}\ xs)\ \mathsf{in} \\
\quad \mathsf{loop}\ k\ \mathsf{times\ on}\ \theta_0\ \{\ \theta \rightarrow \\
\quad\quad \mathrm{gauss}_{\epsilon,\delta}\ (f\ (\mathsf{unbox}\ \mathrm{temp})\ \theta)\ \}
\end{array}
$$

In this example, the type of temp is a $\square[xs@1]\ \mathtt{data}$ (a "box of data 1-sensitive in xs") indicating that when unboxed, temp will report 1-sensitivity w.r.t xs, not temp. f is then able to make good on its promise to gauss that the result of f is 1-sensitive in xs (assuming f is 1-sensitive in its first argument), and gauss properly reports its privacy "cost" in terms of xs, not temp.

We use exactly this pattern in many of our case studies, where *expensive* is a pre-processing operation on the input data (e.g., clipping or normalizing), and f is a machine-learning training operation, such as computing an improved model based on the current model θ and the pre-processed input data *temp*.

3.3 Duet Syntax & Typing Rules

Figure 3 shows a core subset of syntax for both languages. We only present the privacy fragment for (ϵ, δ)-differential privacy in the core formalism, although support for other variants (and combined variants) is straightforward as sketched in the previous section. See the extended version of this paper [Near et al. 2019] for the complete presentation of the full language including all advanced

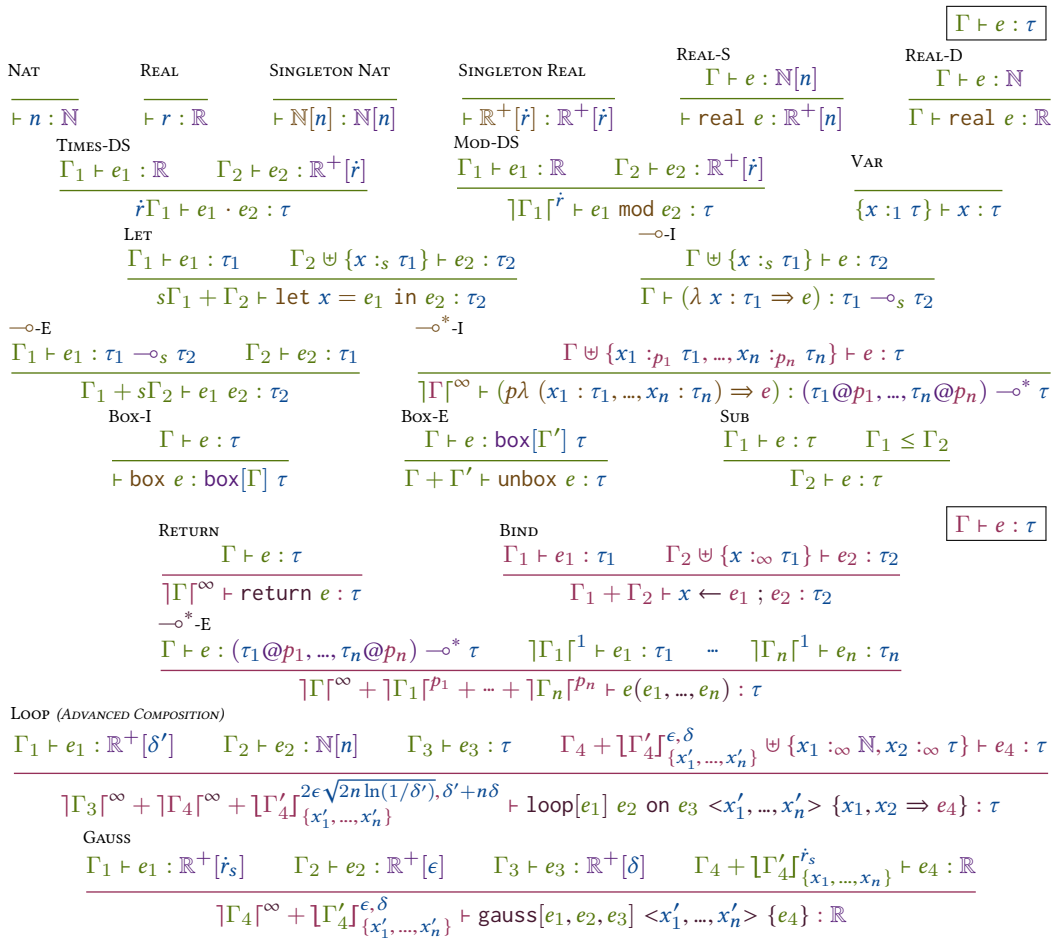

Fig. 4. Core Typing Rules

variants of differential privacy. We use color coding to distinguish between the sensitivity language, privacy language, and shared syntax between languages. The sensitivity and privacy languages share syntax for variables and types, which are typeset in blue. Expressions in the sensitivity language are typeset in green, while expressions in the privacy language are typeset in red.[1]

Types τ include base numeric types $\mathbb{N}$ and $\mathbb{R}$ and their treatment is standard. We include singleton numeric types $\mathbb{N}[n]$ and $\mathbb{R}^+[\dot{r}]$; these types classify runtime numeric values which are identical to the static index n or $\dot{r}$, e.g., $\mathbb{N}[n]$ is a type which exactly describes its runtime value as the number n. Static reals only range over non-negative values, and we write $\dot{r}$ for elements of the non-negative reals $\mathbb{R}^+$. Singleton natural numbers are used primarily to construct matrices with some statically known dimension, and to execute loops for some statically known number of iterations. Singleton real numbers and are used primarily for tracking sensitivity and privacy quantities. Novel in DUET is a "boxed" type $\mathrm{box}[\Gamma_s]\ \tau$ which delays the "payment" of a value's sensitivity, to be unboxed and

[1]Colors were chosen to minimize ambiguity for colorblind persons following a colorblind-friendly palette: http://mkweb.bcgsc.ca/colorblind/img/colorblindness.palettes.png

"paid for" in a separate context. Boxing is discussed in more detail later in this section. The sensitivity function space (a la Fuzz) is written $\tau_1 \multimap_s \tau_2$ and encodes an s-sensitive function from τ_1 to τ_2. The privacy function space (novel in Duet) is written $(\tau_1@p_1, ..., \tau_n@p_n) \multimap^* \tau$ and encodes a multi-arity function that preserves p_i-privacy for its ith argument. Privacy functions are multi-arity because functions of multiple arguments cannot be recovered from iterating functions over single arguments in the privacy language, as can be done in the sensitivity language.

In our implementation and extended presentation of Duet in the extended version of this paper, we generalize the static representations of natural numbers and reals to symbolic expression η, which may be arbitrary symbolic polynomial formulas including variables. E.g., suppose ϵ is a type-level variable ranging over real numbers and $x{:}\mathbb{N}[\epsilon]$, then $2x{:}\mathbb{N}[2\epsilon]$. Our type checker knows this is the same type as $\mathbb{N}[\epsilon{+}\epsilon]$ using a custom solver we implemented but do not describe in this paper. Because the typelevel representation of a natural number can be a variable, its value is therefore not statically *determined*, rather it is statically *tracked* via typelevel symbolic formulas.

Type contexts in the sensitivity language Γ_s track the *sensitivity* s of each free variable whereas in the privacy language Γ_p they track *privacy cost* p. Sensitivities are non-negative reals $\dot{r}$ extended with a distinguished infinity element ∞, and privacy costs are specific to the current privacy *mode*. In the case of (ϵ, δ)-differential privacy, p has the form ϵ, δ or ∞ where ϵ and δ range over $\mathbb{R}^+$.

We reuse notation conventions from Fuzz for manipulating contexts, e.g., $\Gamma_1{+}\Gamma_2$ is partial and defined only when both contexts agree on the type of each variable; adding contexts adds sensitivities pointwise, i.e., $\{x{:}_{s_1+s_2}\tau\} \in \Gamma_1{+}\Gamma_2$ when $\{x{:}_{s_1}\tau\} \in \Gamma_1$ and $\{x{:}_{s_2}\tau\} \in \Gamma_2$; and scaling contexts scales sensitivities pointwise, i.e., $\{x{:}_{ss'}\tau\} \in s\Gamma$ when $\{x{:}_{s'}\tau\} \in \Gamma$.

We introduce a new operation not shown in prior work called *truncation* and written $\rceil s_1 \lceil^{s_2}$ for truncating a sensitivity and $\rceil\Gamma\lceil^s$ for truncating a sensitivity context, which is pointwise truncation of sensitivities. Sensitivity truncation $\rceil\_\lceil^s$ maps 0 to 0 and any other value to s:

$$\rceil\_\lceil - \in \text{sens} \times \text{sens} \to \text{sens} \qquad \rceil s_1 \lceil^{s_2} \triangleq \begin{cases} 0 & \text{if} \quad s_1 = 0 \\ s_2 & \text{if} \quad s_1 \neq 0 \end{cases}$$

Truncation is defined analogously for privacies $\rceil p_1 \lceil^{p_2}$, for converting between sensitivities and privacies $\rceil s \lceil^p$ and $\rceil p \lceil^s$, and also for liftings of these operations pointwise over contexts $\rceil\Gamma\lceil^p$, $\rceil\Gamma\lceil^p$ and $\rceil\Gamma\lceil^s$. Sensitivity truncation is used for typing the modulus operator, and truncating between sensitivities and privacies is always to ∞/∞ and appears frequently in typing rules that embed sensitivity terms in privacy terms and vice versa.

The syntax and language features for both sensitivity and privacy languages are discussed next alongside their typing rules. Figure 4 shows a core subset of typing rules for both languages. In the typing rules, the languages embed within each other—sensitivity typing contexts are transformed into privacy contexts and vice versa. Type rules are written in logical style with an explicit subsumption rule, although a purely algorithmic presentation is possible (not shown) following ideas from Azevedo de Amorim et al [De Amorim et al. 2014] which serves as the basis for our implementation.

3.4 Sensitivity Language

Duet's sensitivity language is similar to that of DFuzz [Gaboardi et al. 2013], except that we extend it with significant new tools for machine learning in Section 4. We do not present standard linear logic connectives such as sums, additive products and multiplicative products (a la Fuzz), or symbolic type-level expressions (a la DFuzz), although each are implemented in our tool and described formally in the extended version of this paper [Near et al. 2019]. We do not formalize or implement general recursive types in order to ensure that all Duet programs terminate. Including general recursive types would be straightforward in Duet (following the design of Fuzz), however

such a decision comes with known limitations. As described in Fuzz [Reed and Pierce 2010], requiring that all functions terminate is necessary in order to give both sound and useful types to primitives like set-filter. The design space for the combination of sensitivity types and nontermination is subtle, and discussed extensively in prior work [Azevedo de Amorim et al. 2017; Reed and Pierce 2010].

Typing for literal values is immediate (NAT, REAL). Singleton values are constructed using the same syntax as their types, and where the type level representation is identical to the literal (SINGLETON NAT, SINGLETON REAL). Naturals can be converted to real numbers through the explicit conversion operation `real` (REAL-S, REAL-D). For the purposes of sensitivity analysis, statically known numbers are considered constant, and as a consequence any term that uses one is considered 0-sensitive in the statically known term. The result of this is that the sensitivity environment Γ associated with the subterm at singleton type is dropped from the output environment, e.g., in REAL-S. This dropping is justified by our metric space interpretation $[\![\mathbb{N}[n]]\!]$ for statically known numbers as singleton sets $\{n\}$, and because for all $x, y \in [\![\mathbb{N}[n]]\!]$, $x = y$ and therefore $|x - y| = 0$.

Type rules for arithmetic operations are given in multiple variations, depending on whether or not each argument is tracked statically or dynamically. We show only the rule for multiplication when the left argument is dynamic and the right argument is static (TIMES-DS). The resulting sensitivity environment reports the sensitivities of e_1 scaled by $\dot{r}$—the statically known value of e_2—and the sensitivities for e_2 are not reported because its value is fixed and cannot vary, as discussed above. When both arguments are dynamic, the resulting sensitivity environment is $\infty(\Gamma_1 + \Gamma_2)$, i.e., all potentially sensitive variables for each expression are bumped to infinity. The modulus operation is similar to multiplication in that we have cases for each variation of static or dynamic arguments, however the context is truncated rather than scaled in the case of one singleton-typed parameter; we show only this static-dynamic variant in the figure (MOD-DS).

Typing for variables (VAR) and functions ($\multimap$-I, $\multimap$-E) is the same as in Fuzz: variables are reported in the sensitivity environment with sensitivity 1; and closures are created by annotating the arrow with the sensitivity s of the argument in the body, and by reporting the rest of the sensitivities Γ from the function body as the sensitivity of whole closure as a whole; and function application scales the argument by the function's sensitivity s.

The first new (w.r.t. DFuzz) term in our sensitivity language is the privacy lambda. Privacy lambdas are multi-arity (as opposed to single-arity sensitivity lambdas) because the privacy language does not support currying to recover multi-argument functions. Privacy lambdas are created in the *sensitivity* language with $p\lambda\ (x : \tau, ..., x : \tau) \Rightarrow e$ and applied in the *privacy* language with $e(e, ..., e)$. The typing rule for privacy lambdas ($\multimap^*$-I) types the body of the lambda in a privacy type context extended with its formal parameters, and the privacy cost of each parameter is annotated on its function argument type. Unlike sensitivity lambdas, the privacy cost of variables in the closure environment are not preserved in the resulting typing judgment. The reason for this is twofold: (1) the final "cost" for variables in the closure environment depends on how many times the closure is called, and in the absence of this knowledge, we must conservatively assume that it could be called an infinite number of times, and (2) the interpretation of an ∞-sensitive function coincides with that of an ∞-private function, so we can soundly convert between ∞-privacy-cost and ∞-sensitivity contexts freely using truncation.

The final two new terms in our sensitivity language are introduction and elimination forms for "boxes" (BOX-I and BOX-E). Boxes have no operational behavior and are purely a type-level mechanism for tracking sensitivity. The rules for box introduction capture the sensitivity context of the expression, and the rule for box elimination pays for that cost at a later time. Boxes are reminiscent of *contextual modal type theory* [Nanevski et al. 2008]—they allow temporary capture of a linear context via boxing—thereby deferring its payment—and re-introduction of the context at later time

via unboxing. In a linear type system that supports scaling, this boxing would not be necessary, but it becomes necessary in our system to achieve the desired operational behavior when interacting with the privacy language, which does not support scaling. E.g., in many of our examples we perform some pre-processing on the database parameter (such as clipping) and then use this parameter in the body of a loop. Without boxing, the only way to achieve the desired semantics is to re-clip the input (a deterministic operation) every time around the loop—boxing allows you to clip on the outside of the loop and remember that privacy costs should be "billed" to the initial input.

3.5 Privacy Language

Duet's privacy language is designed specifically to enable the composition of individual differentially private computations. It has a linear type system, but unlike the sensitivity language, annotations instead track privacy cost, and the privacy language *does not allow scaling* of these annotations, that is, the notation $p\Gamma$ is not used and cannot be defined. Syntax `return` e and $x \leftarrow e; e$ (pronounced "bind") are standard from Fuzz, as are their typing rules (Return, Bind), except for our explicit conversion from a sensitivity context Γ to a privacy context Γ by truncation to infinity in the conclusion of Return. Bind encodes exactly the post-processing property of differential privacy—it allows e_2 to use the value computed by e_1 any number of times after paying for it once.

Privacy application $e(e, ..., e)$ applies a privacy function ($p\lambda$, created in the sensitivity language) to a sequence of *1-sensitivity* arguments—the sensitivity is enforced by the typing rule. The type rule ($\multimap^*$-E) checks that the first term produces a privacy function and applies its privacy costs to function arguments which are restricted by the type system to be 1-sensitive. We use truncation in well-typed hypothesis for $e_1 \ldots e_n$ to encode the restriction that the argument must be 1-sensitive. This restriction is crucial for type soundness—arbitrary terms cannot be given tight privacy bounds statically due to the lack of a tight scaling operation in the model for (ϵ, δ)-differential privacy. The same is true for other advanced variants of differential privacy.

The `loop` expression is for loop iteration fixed to a statically known number of iterations. The syntax includes a list of variables ($<x, ..., x>$) to indicate which variables should be considered when calculating final privacy costs, as explained shortly. The typing rule (Loop) encodes advanced composition for (ϵ, δ)-differential privacy. e_1 is the δ' parameter to the advanced composition bound and e_2 is the number of loop iterations—each of these values must be statically known, which we encode with singleton types (a la DFuzz). Statically known values are fixed and their sensitivities do not appear in the resulting context. e_3 is the initial value passed to the loop, and for which no claim is made of privacy, indicated by truncation to infinity. e_4 is a loop body with free variables x_1 and x_2 which will be iterated e_2 times with the first variable bound to the iteration index, and the second variable bound to the loop state, where e_3 is used as the starting value. The loop body e_4 is checked in a privacy context $\Gamma_4 + \lfloor \Gamma'_4 \rfloor^{\epsilon,\delta}_{\{x'_1,...,x'_n\}}$, shorthand for $\rceil\lfloor \Gamma'_4 \rfloor_{\{x'_1,...,x'_n\}}\lceil^{\epsilon,\delta}$ where $\lfloor \Gamma'_4 \rfloor_{\{x'_1,...,x'_n\}}$ is a context restricted to only the variables $x'_1, ..., x'_n$. The ϵ, δ is an upper bound on the privacy cost of the variables x'_i in the loop body, and the resulting privacy bound is restricted to only those variables. This allows variables for which the programmer is not interested in tracking privacy to appear in Γ_4 in the premise, and the rule's conclusion makes no claims about privacy for these variables. We make use of this feature in all of our examples programs.

The `gauss` expression is a *mechanism* of (ϵ, δ)-differential privacy; other mechanisms are used for other privacy variants. Like the loop expression, mechanism expressions take a list of variables to indicate which variables should be considered in the final privacy cost. The typing rule (Gauss) is similar in spirit to Loop: it takes parameters to the mechanism which must be statically known (encoded as singleton types), a list of variables to consider for the purposes of the resulting privacy

bound, and a term $\{e\}$ for which there is a bound $\dot{r}$ on the sensitivity of free variables $x_1, ..., x_n$. The resulting privacy guarantee is that the term in brackets $\{e\}$ is ϵ, δ differentially private. Whereas `loop` and advanced composition consider a *privacy term* loop body with an upper bound on *privacy leakage*, `gauss` considers a *sensitivity term* body with an upper bound on its *sensitivity*.

3.6 Metatheory

We denote sensitivity language terms $e \in \text{exp}$ into total, functional, linear maps between metric spaces—the same model as the terminating fragment of Fuzz. Every term in our language terminates by design, which dramatically simplifies our models and proofs. This restriction poses no issues in implementing most differentially private machine learning algorithms, because such algorithms typically terminate in a statically determined number of loop iterations in order to achieve a particular privacy cost.

Types in DUET denote metric spaces, as in Fuzz. We notate metric spaces D, their underlying carrier set $\|D\|$, and their distance metric $|x - y|_D$, or $|x - y|$ where D can be inferred from context. Sensitivity typing judgments $\Gamma \vdash e : \tau$ denote linear maps from a scaled cartesian product interpretation of Γ:

$$[\![\overbrace{\{x_1 :_{s_1} \tau_1, ..., x_n :_{s_n} \tau_n\}}^{\Gamma} \vdash \tau]\!] \triangleq !_{s_1}[\![\tau_1]\!] \otimes \cdots \otimes !_{s_n}[\![\tau_n]\!] \multimap [\![\tau]\!]$$

Although we do not make metric space scaling explicit in our syntax (for the purposes of effective type inference, a la DFuzz [De Amorim et al. 2014]), scaling becomes apparent explicitly in our model. Privacy judgments $\Gamma \vdash e : \tau$ denote *probabilistic, privacy preserving maps* from an *unscaled* product interpretation of Γ:

$$[\![\overbrace{\{x_1 :_{p_1} \tau_1, ..., x_n :_{p_n} \tau_n\}}^{\Gamma} \vdash \tau]\!] \triangleq ([\![\tau_1]\!]@p_1, ..., [\![\tau_n]\!]@p_n) \multimap^* \|[\![\tau]\!]\|$$

The multi-arity (ϵ, δ)-differential-privacy-preserving map is defined:

$$\begin{aligned}(D_1@(\epsilon_1, \delta_1), ..., D_n@(\epsilon_n, \delta_n)) \multimap^* X \triangleq \\ \{\, f \in \|D_1\| \times \cdots \times \|D_n\| \to \mathcal{D}(X) \\ \mid |x_i - y|_{D_i} \leq 1 \Rightarrow \Pr[f(x_1, ..., x_i, ..., x_n) = d] \leq e^{\epsilon_i} \Pr[f(x_1, ..., y, ..., x_n) = d] + \delta_i \,\}\end{aligned}$$

where $\mathcal{D}(X)$ is a distribution over elements in X.

We give a full semantic account of typing in the extended version of this paper [Near et al. 2019], as well as prove key type soundness lemmas, many of which appeal to well-known differential privacy proofs from the literature.

The final soundness theorem, proven by induction over typing derivations, is that the denotations for well-typed open terms e_s and e_p in well-typed environments γ_s and γ_p are contained in the denotation of their typing contexts $\Gamma_s \vdash \tau$ and $\Gamma_p \vdash \tau$.

THEOREM 3.1.

(1) If $\Gamma_p \vdash e_p : \tau$ *and* $\Gamma_p \vdash \gamma_p$ *then* $[\![e_p]\!]^{\gamma_p} \in [\![\Gamma_p \vdash \tau]\!]$
(2) If $\Gamma_s \vdash e_s : \tau$ *and* $\Gamma_s \vdash \gamma_s$ *then* $[\![e_s]\!]^{\gamma_s} \in [\![\Gamma_s \vdash \tau]\!]$

A corollary is that any well-typed privacy lambda function satisfies (ϵ, δ)-differential privacy for each of its arguments w.r.t. that argument's privacy annotation used in typing.

4 LANGUAGE TOOLS FOR MACHINE LEARNING

Machine learning algorithms typically operate over a training set of *samples*, and implementations of these algorithms often represent datasets using matrices. To express these algorithms, DUET includes a core matrix API which encodes sensitivity and privacy properties of matrix operations.

We add a matrix type $\mathbb{M}^c_\ell[m,n]\ \tau$, encode vectors as single-row matrices, and add typing rules for gradient computations that encode desirable properties. We also introduce a type for matrix indices $\texttt{idx}[n]$ for type-safe indexing. These new types are shown in Figure 6, along with sensitivity operations on matrices—encoded as library functions because their types can be encoded using existing connectives—and new matrix-level differential privacy mechanisms—encoded as primitive syntactic forms because their types *cannot* be expressed using existing type-level connectives.

In the matrix type $\mathbb{M}^c_\ell[m,n]\ \tau$, the m and n parameters refer to the number of rows and columns in the matrix, respectively. The ℓ parameter determines the distance metric used for the matrix metric for the purposes of sensitivity analysis; the c parameter is used to specify a norm bound on each row of the matrix, which will be useful when applying gradient functions.

4.1 Distance Metrics for Matrices

Differentially private machine learning algorithms typically move from one distance metric on matrices and vectors to another as the algorithm progresses. For example, two input training datasets are neighbors if they differ on exactly one sample (i.e. one row of the matrix), but they may differ arbitrarily in that row. After computing a gradient, the algorithm may consider the $L2$ sensitivity of the resulting vector—i.e. two gradients g_1 and g_2 are neighbors if $\|g_1 - g_2\|_2 \leq 1$. These are very different notions of distance—but the first is required by the definition of differential privacy, and the second is required as a condition on the input to the Gaussian mechanism.

The ℓ annotation on matrix types in Duet enables specifying the desired notion of distance between *rows*. The annotation is one of $L\infty$, $L1$, or $L2$; an annotation of $L\infty$, for example, means that the distance between two rows is equal to the $L\infty$ norm of the difference between the rows. The distance between two matrices is always equal to the sum of the distances between rows. The distance metric for the element datatype τ determines the distance between two corresponding elements, and the row metric ℓ specifies how to combine elementwise distances to determine the distance between two rows.

Figure 5 presents the complete set of distance metrics for matrices, as well as real numbers and the new domain data for elements of the $\mathbb{D}$ type, which is operationally a copy of $\mathbb{R}$ but with a discrete distance metric. Many combinations are possible, including the following common ones:

Ex. 1: $|X - X'|_{\mathbb{M}^U_{L\infty}[m,\,n]\ \mathbb{D}} = \sum_i \max_j |X_{i,j} - X'_{i,j}|_{\mathbb{D}}$

Distance is the *number of rows on which X and X' differ*; commonly used to describe neighboring input datasets.

Ex. 2: $|X - X'|_{\mathbb{M}^U_{L1}[m,\,n]\ \mathbb{R}} = \sum_i \sum_j |X_{i,j} - X'_{i,j}|_{\mathbb{R}}$

Distance is the *sum of elementwise differences.*

Ex. 3: $|X - X'|_{\mathbb{M}^U_{L2}[m,\,n]\ \mathbb{R}} = \sum_i \sqrt{\sum_j |X_{i,j} - X'_{i,j}|^2_{\mathbb{R}}}$

Distance is *sum of the $L2$ norm of the differences between corresponding rows.*

Ex. 4: $|X - X'|_{\mathbb{M}^U_{L2}[1,\,n]\ \mathbb{R}} = \sqrt{\sum_j |X_{1,j} - X'_{1,j}|^2_{\mathbb{R}}}$

Represents a vector; distance is $L2$ sensitivity for vectors, as required by the Gaussian mechanism.

These distance metrics are used in the types of library functions which operate over matrices.

4.2 Matrix Operations

Figure 6 summarizes the matrix operations available in Duet's API. We focus on the non-standard operations which are designed specifically for sensitivity or privacy applications. For example, fr-sens allows converting between notions of distance between rows; when converting from $L2$ to

Domain	*Carrier:* $X \in \text{set}$	*Metric:* $\lvert \_ - \_ \rvert \in X \to X \to \mathbb{R} \uplus \{\infty\}$
real	$\mathbb{R}$	$\lvert r_1 - r_2 \rvert \triangleq \lvert r_1 - r_2 \rvert_{\mathbb{R}}$
data	$\mathbb{R}$	$\lvert r_1 - r_2 \rvert \triangleq \begin{cases} 0 & \text{when } r_1 = r_2 \\ 1 & \text{when } r_1 \neq r_2 \end{cases}$
$\text{matrix}[n_1, n_2]_{L\infty}(D)$	$\mathbb{M}[n_1, n_2](\lVert D \rVert)$	$\lvert m_1 - m_2 \rvert \triangleq \sum_i \max_j \lvert m_1[i,j] - m_2[i,j] \rvert_D$
$\text{matrix}[n_1, n_2]_{L1}(D)$	$\mathbb{M}[n_1, n_2](\lVert D \rVert)$	$\lvert m_1 - m_2 \rvert \triangleq \sum_{i,j} \lvert m_1[i,j] - m_2[i,j] \rvert_D$
$\text{matrix}[n_1, n_2]_{L2}(D)$	$\mathbb{M}[n_1, n_2](\lVert D \rVert)$	$\lvert m_1 - m_2 \rvert \triangleq \sum_i \sqrt{\sum_j \lvert m_1[i,j] - m_2[i,j] \rvert_D^2}$

Fig. 5. Distance Metrics for Matrices

$$\ell \in \text{norm} ::= L1 \mid L2 \mid L\infty \qquad c \in \text{clip} ::= \ell \mid U \qquad \tau \in \text{type} ::= \ldots \mid \mathbb{D} \mid \texttt{idx}[n] \mid \mathbb{M}^c_\ell[n, n]\ \tau$$

$$\begin{aligned}
\text{rows} &: \mathbb{M}^c_\ell[m,n]\ \tau \multimap_0 \mathbb{N}[m] & \text{convert} &: \mathbb{M}^\ell_{\ell'}[m,n]\ \mathbb{D} \multimap_1 \mathbb{M}^{\mathsf{U}}_\ell[m,n]\ \mathbb{R} \\
\text{cols} &: \mathbb{M}^c_\ell[m,n]\ \tau \multimap_0 \mathbb{N}[n] & \text{clip}^\ell &: \mathbb{M}^c_{\ell'}[m,n]\ \mathbb{D} \multimap_1 \mathbb{M}^\ell_{\ell'}[m,n]\ \mathbb{D} \\
\text{discf} &: (\tau \multimap_\infty \mathbb{R}) \multimap_1 \tau \multimap_1 \mathbb{D} & \text{fr-sens}^{L\infty} &: \mathbb{M}^c_{L\infty}[m,n]\ \tau \multimap_{\sqrt{n}} \mathbb{M}^c_{L2}[m,n]\ \tau \\
\text{undisc} &: \mathbb{D} \multimap_\infty \mathbb{R} & \text{fr-sens}^{L2} &: \mathbb{M}^c_{L2}[m,n]\ \tau \multimap_{\sqrt{n}} \mathbb{M}^c_{L1}[m,n]\ \tau \\
\text{transpose} &: \mathbb{M}^c_{L1}[m,n]\ \tau \multimap_1 \mathbb{M}^{\mathsf{U}}_{L1}[n,m]\ \tau & \text{to-sens}^\ell &: \mathbb{M}^c_{L1}[m,n]\ \tau \multimap_1 \mathbb{M}^c_\ell[m,n]\ \tau
\end{aligned}$$

$$\begin{aligned}
\text{mcreate} &: \mathbb{N}[m] \multimap_0 \mathbb{N}[n] \multimap_0 (\texttt{idx}[m] \multimap_\infty \texttt{idx}[n] \multimap_\infty \tau) \multimap_{mn} \mathbb{M}^{\mathsf{U}}_{L1}[m,n]\ \tau \\
_\#[_,_] &: \mathbb{M}^c_\ell[m,n]\ \tau \multimap_1 \texttt{idx}[m] \multimap_\infty \texttt{idx}[n] \multimap_\infty \tau \\
_\#[_,_ \mapsto _] &: \mathbb{M}^c_\ell[m,n]\ \tau \multimap_1 \texttt{idx}[m] \multimap_\infty \texttt{idx}[n] \multimap_\infty \tau \multimap_1 \mathbb{M}^{\mathsf{U}}_\ell[m,n]\ \tau \\
\text{fld} &: (\tau_1 \multimap_{s_1} \tau_2 \multimap_{s_2} \tau_3) \multimap_{mn} \tau_2 \multimap_{s_2^{mn}} \mathbb{M}^c_{L1}[m,n]\ \tau_1 \multimap_{s_1} \tau_2 \\
\text{map} &: (\tau_1 \multimap_s \tau_2) \multimap_{mn} \mathbb{M}^c_\ell[m,n]\ \tau_1 \multimap_s \mathbb{M}^{\mathsf{U}}_\ell[m,n]\ \tau_2 \\
\text{fld-row} &: (\tau_1 \multimap_{s_1} \tau_2 \multimap_{s_2} \tau_2) \multimap_m \tau_2 \multimap_{s_2{}^m} \mathbb{M}^c_\ell[m,n]\ \tau_1 \multimap_{s_1} \mathbb{M}^{\mathsf{U}}_\ell[m,1]\ \tau_2 \\
\text{map-row} &: (\mathbb{M}^{c_1}_{\ell_1}[1,n_1]\ \tau_1 \multimap_s \mathbb{M}^{c_2}_{\ell_2}[1,n_2]\ \tau_2) \multimap_m \mathbb{M}^{c_1}_{\ell_1}[m,n_1]\ \tau_1 \multimap_s \mathbb{M}^{c_2}_{\ell_2}[m,n_2]\ \tau_2 \\
\text{L}\nabla^g_\ell[_;_,_] &: \mathbb{M}^\ell_{\ell'}[1,n]\ \mathbb{R} \multimap_\infty \mathbb{M}^\ell_{\ell''}[1,n]\ \mathbb{D} \multimap_1 \mathbb{D} \multimap_1 \mathbb{M}^{\mathsf{U}}_\ell[1,n]\ \mathbb{R} \\
\text{U}\nabla[_;_,_] &: \mathbb{M}^\ell_{\ell'}[1,n]\ \mathbb{R} \multimap_\infty \mathbb{M}^\ell_{L\infty}[1,n]\ \mathbb{D} \multimap_1 \mathbb{D} \multimap_1 \mathbb{M}^{\mathsf{U}}_{L\infty}[1,n]\ \mathbb{D}
\end{aligned}$$

$$\begin{aligned}
\text{above-threshold} &: (\mathbb{M}^c_\ell[1,n]\ (\tau \multimap_1 \mathbb{R})@\infty,\ \mathbb{R}^+[\epsilon]@0,\ \tau@\langle\epsilon,0\rangle,\ \mathbb{R}@\infty) \multimap^* \texttt{idx}[n] \\
\text{pfld-rows} &: (\mathbb{M}^{c_1}_{L\infty}[m,n_1]\ \mathbb{D}@\langle\epsilon,\delta\rangle,\ \mathbb{M}^{c_2}_{L\infty}[m,n_2]\ \mathbb{D}@\langle\epsilon,\delta\rangle, \\
&\quad ((\mathbb{M}^{c_1}_{L\infty}[1,n_1]\ \mathbb{D}@\langle\epsilon,\delta\rangle,\ \mathbb{M}^{c_2}_{L\infty}[1,n_2]\ \mathbb{D}@\langle\epsilon,\delta\rangle,\ \mathbb{D}@\infty) \multimap^* \tau)@\infty, \\
&\quad \tau@\infty \\
&\quad) \multimap^* \tau \\
\text{sample} &: (\mathbb{N}[m_2]@\langle 0,0\rangle, \\
&\quad \mathbb{M}^c_{L\infty}[m_1,n_1]\ \mathbb{D}@\langle 2m_2\epsilon_1/m_1, m_2\delta_1/m_1\rangle,\ \mathbb{M}^c_{L\infty}[m_1,n_2]\ \mathbb{D}@\langle 2m_2\epsilon_2/m_1, m_2\delta_2/m_1\rangle, \\
&\quad ((\mathbb{M}^c_{L\infty}[m_2,n_1]\ \mathbb{D}@\langle\epsilon_1,\delta_1\rangle,\ \mathbb{M}^c_{L\infty}[m_2,n_2]\ \mathbb{D}@\langle\epsilon_2,\delta_2\rangle) \multimap^* \tau)@\infty \\
&\quad) \multimap^* \tau
\end{aligned}$$

$\boxed{\Gamma \vdash e : \tau}$

MGAUSS

$$\frac{\Gamma_1 \vdash e_1 : \mathbb{R}^+[\dot{r}] \qquad \Gamma_2 \vdash e_2 : \mathbb{R}^+[\epsilon] \qquad \Gamma_3 \vdash e_3 : \mathbb{R}^+[\delta] \qquad \Gamma_4 + \lfloor\Gamma_5\rfloor^{\dot{r}}_{\{x_1,\ldots,x_n\}} \vdash e_4 : \mathbb{M}^c_{L2}[m,n]\ \mathbb{R}}{\lceil\Gamma_1 + \Gamma_2 + \Gamma_3\rceil^{0,0} + \lceil\Gamma_4\rceil^\infty + \lceil\Gamma_5\rceil^{\epsilon,\delta} \vdash \texttt{mgauss}[e_1,e_2,e_3]\ \langle x_1,\ldots,x_n\rangle\ \{e_4\} : \mathbb{M}^U_{L\infty}[m,n]\ \mathbb{R}}$$

EXPONENTIAL

$$\frac{\Gamma_1 \vdash e_1 : \mathbb{R}^+[\dot{r}] \qquad \Gamma_2 \vdash e_2 : \mathbb{R}^+[\epsilon] \qquad \Gamma_3 \vdash e_3 : \mathbb{M}^c_\ell[1,m](\tau) \qquad \Gamma_4 + \lfloor\Gamma_5\rfloor^{\dot{r}}_{\{x_1,\ldots,x_n\}} \uplus \{x :_\infty \tau\} \vdash e_4 : \mathbb{R}}{\lceil\Gamma_1 + \Gamma_2\rceil^{0,0} + \lceil\Gamma_3 + \Gamma_4\rceil^\infty + \lceil\Gamma_5\rceil^{\epsilon,0} \vdash \texttt{exponential}[e_1,e_2]\ \langle x_1,\ldots,x_n\rangle\ e_3\ \{x \Rightarrow e_4\} : \tau}$$

Fig. 6. Matrix Typing Rules

$L1$, the distance between two rows may increase by $\sqrt{n}$ (by Cauchy-Schwarz), so the corresponding version of fr-sens has a sensitivity annotation of $\sqrt{n}$.

undisc allows converting from discrete to standard reals, and is infinitely sensitive. discf allows converting an infinitely sensitive function which returns a real to a 1-sensitive function returning a discrete real; we can recover a 1-sensitive function from reals to discrete reals ($\mathrm{disc} : \mathbb{R} \multimap_1 \mathbb{D}$) by applying `discf` to the identity function.

above-threshold encodes the *Sparse Vector Technique* [Dwork et al. 2014], discussed in the extended version of this paper [Near et al. 2019]. pfld-rows encodes parallel composition of privacy mechanisms, and is discussed in Section 5.5. sample performs random subsampling with privacy amplification, and is discussed in Section 5.4.

Gradients are computed using $\mathrm{L}\nabla^{g}_{\ell}[\_;\_,\_]$ and $\mathrm{U}\nabla[\_;\_,\_]$. The first represents an ℓ-Lipschitz gradient (typical in convex optimization problems like logistic regression) like the gradient function introduced in Section 3.2; it is a 1-sensitive function which produces a matrix of real numbers. The second represents a gradient *without* a known Lipschitz constant (typical in non-convex optimization problems, including training neural networks); it produces a matrix of discrete reals. We demonstrate applications of both in Section 5.

In order to produce a matrix with sensitivity bound $\mathsf{L2}$, $L\nabla^{g}_{\mathsf{L2}}$ requires input of type $\mathbb{M}^{L2}_{\ell}[m,n]\ \mathbb{D}$ for any ℓ. We obtain such a matrix by *clipping*, a common operation in differentially private machine learning. Clipping scales each row of a matrix to ensure its c norm (for $c \in \{L\infty, L1, L2\}$) is less than 1:

$$\mathrm{clip}^{c}\ x_i \triangleq \begin{cases} \frac{x_i}{\|x_i\|_c} & \text{if} \quad \|x_i\|_c > 1 \\ x_i & \text{if} \quad \|x_i\|_c \leq 1 \end{cases}$$

The clipping process is encoded in Duet as clip (Figure 6), which introduces a new bound on the c norm of its output.

4.3 Vector-Valued Privacy Mechanisms

Both the Laplace and Gaussian mechanisms are capable of operating directly over vectors; the Laplace mechanism adds noise calibrated to the $L1$ sensitivity of the vector, while the Gaussian mechanism uses its $L2$ sensitivity. With the addition of matrices to Duet, we can introduce typing rules for these vector-valued mechanisms, using single-row matrices to represent vectors. We present the typing rule for MGauss in Figure 6; the rule for MLaplace is similar. We also introduce a typing rule for the exponential mechanism, which picks one element out of an input vector based on a sensitive scoring function (Figure 6, rule Exponential).

5 CASE STUDIES

In this section, we demonstrate the use of Duet to express and verify a number of different algorithms for differentially private machine learning.

There are four basic approaches to differentially private convex optimization: input perturbation [Chaudhuri et al. 2011], objective perturbation [Chaudhuri et al. 2011], gradient perturbation [Bassily et al. 2014b; Song et al. 2013], and output perturbation [Chaudhuri et al. 2011; Wu et al. 2017]. Of these, the latter three are known to provide competitive accuracy, and the latter two (gradient perturbation and output perturbation) are the most widely used; our first two case studies verify these two techniques. Our third case study verifies the noisy Frank-Wolfe algorithm [Talwar et al. 2015], a variant of gradient perturbation especially suited to high-dimensional datasets.

Our next three case studies demonstrate the use of Duet to verify commonly-used variations on the above algorithms, including various kinds of minibatching and a gradient clipping approach used in deep learning. Our final three case studies explore techniques for preprocessing input

datasets so that the preconditions of the above algorithms are satisfied. In Section 5.6, we discuss the use of DUET to combine all of these components—many of which leverage *different* variants of differential privacy—to build a complete machine learning system. Our case studies are summarized in the following table.

Technique	Ref.	§	Privacy Concept
Optimization Algorithms			
Noisy Gradient Descent	[Bassily et al. 2014b]	5.1	Composition
Gradient Descent w/ Output Perturbation	[Wu et al. 2017]	5.2	Parallel comp. (sens.)
Noisy Frank-Wolfe	[Talwar et al. 2015]	5.3	Exponential mechanism
Variations on Gradient Descent			
Minibatching	[Bassily et al. 2014b]	5.4	Amplification by subsampling
Parallel-composition minibatching	—	5.5	Parallel composition
Gradient clipping	[Abadi et al. 2016]	†	Sensitivity bounds
Preprocessing & Deployment			
Hyperparameter tuning	[Chaudhuri and Vinterbo 2013]	†	Exponential mechanism
Adaptive clipping	—	†	Sparse Vector Technique
Z-Score normalization	[skl 2019]	†	Composition
Combining All of the Above		5.6	Composition

5.1 Noisy Gradient Descent

We begin with a fully-worked version of the differentially-private gradient descent algorithm from Section 3.2. This algorithm was first proposed by Song et al. [Song et al. 2013] and later refined by Bassily et al. [Bassily et al. 2014b]. Gradient descent is a simple but effective training algorithm in machine learning, and has been applied in a wide range of contexts, from simple linear models to deep neural networks. The program below implements noisy gradient descent in DUET (without minibatching, though we will extend it with minibatching in Section 5.4). It performs k iterations of gradient descent, starting from an initial guess θ_0 consisting of all zeros. At each iteration, the algorithm computes a noisy gradient using noisy-grad, scales the gradient by the *learning rate* η, and subtracts the result from the current model θ to arrive at the updated model.

```
noisy-grad(θ, X, y, ϵ, δ) ≜
  let s = ℝ[1.0]/real (rows X) in
  let z = zeros (cols X) in
  let gs = mmap-row (sλ X_i y_i ⇒
             L∇[LR, L2][θ; X_i, y_i]) X y in
  let g = fld-row (sλ x1 x2 ⇒ x1 + x2) z gs in
  let g_s = map (sλ x ⇒ s · x) g in
  mgauss[s, ϵ, δ] <X, y> {g_s}
```

```
zeros(n) ≜ mcreate_{L∞} 1 n (sλ i j ⇒ 0.0)

noisy-gradient-descent(X, y, k, η, ϵ, δ) ≜
  let X_1 = box (mclip^{L2} X) in
  let θ_0 = zeros (cols X_1) in
  loop[δ'] k on θ_0 <X_1, y> {t, θ ⇒
      g_p ← noisy-grad θ (unbox X_1) y ϵ δ ;
      return θ − η · g_p   }
```

Under (ϵ, δ)-differential privacy, DUET derives a total privacy cost of $(2\epsilon\sqrt{2k\log(1/\delta')}, k\delta+\delta')$-differential privacy for this implementation, which matches the total cost manually proven by Bassily et al. [Bassily et al. 2014b]. DUET can also derive a total cost for other privacy variants: the same program satisfies $k\rho$-zCDP, or $(\alpha, k\epsilon)$-RDP.

5.2 Output Perturbation Gradient Descent

An alternative to gradient perturbation is *output perturbation*—adding noise to the final trained model, rather than during the training process. Wu et al. [Wu et al. 2017] present a competitive

† Due to space constraints, these case studies appear in the extended version of this paper [Near et al. 2019].

algorithm based on this idea, which works by bounding the *total sensitivity* (rather than privacy) of the iterative gradient descent process. Their algorithm leverages *parallel composition* for sensitivity: it divides the dataset into small chunks called *minibatches*, and each iteration of the algorithm processes one minibatch. A single pass over all minibatches (and thus, the whole dataset) is often called an *epoch*. If the dataset has size m and each minibatch is of size b, then each epoch comprises m/b iterations of the training algorithm. This approach to minibatching is often used (without privacy) in deep learning. The sensitivity of a complete epoch in this technique is just $1/b$.

We encode parallel composition for sensitivity in Duet using the `mfold-row` function, defined in Section 4, whose type matches that of `foldl` for lists in the *Fuzz* type system [Reed and Pierce 2010]. `mfold-row` considers each row to be a "minibatch" of size 1, but is easily extended to consider multiple rows at a time (as in our encoding below). Duet derives a sensitivity bound of k/b for the training process, and a total privacy cost of (ϵ, δ)-differential privacy, matching the manual analysis of Wu et al. [Wu et al. 2017].

```
gd-output-perturbation(xs, ys, k, η, ϵ, δ) ≜
  let m₀ = zeros (cols X) in
  let c = box (mclip^L2 xs) in
  let s = real k/real b in
  mgauss[s, ϵ, δ] <xs, ys> {
    loop k on m₀ { a, θ ⇒
      mfold-row b, θ, unbox c, ys { θ, xb, yb ⇒
        let g = ∇^LR_L2[θ ; xb, yb] in
          θ − η · g } } }
```

5.3 Noisy Frank-Wolfe

We next consider a variation on gradient perturbation called the private Frank-Wolfe algorithm [Talwar et al. 2015]. This algorithm has dimension-independent utility, making it useful for high-dimensional datasets. In each iteration, the algorithm takes a step of fixed size in a *single* dimension, using the exponential mechanism to choose the best direction based on the gradient. The sensitivity of each update is therefore dependent on the L_∞ norm of each sample, rather than the L_2 norm.

```
frank-wolfe X y k ϵ δ ≜
  let X₁ = clip-matrix_L∞ X in
  let d = cols X in
  let θ₀ = zeros d in
  let idxs = mcreate_L∞[1,2·d]{i,j ⇒
               ⟨j mod d, sign(j − d)⟩} in
  ZCDP [δ] { loop k on θ₀ {t, θ ⇒
    let μ = 1.0/((real t) + 2.0) in
    let g = L∇^LR_L∞[θ; X₁, y] in
    ⟨i, s⟩ ← EPS_DP {
      exponential[1/rows X₁, ϵ] idxs {⟨i, s⟩ ⇒
        s · g#[0, i]} ;}
    let g_p = (zeros d)#[0, i ↦ s · 100] in
    return ((1.0 − μ) · θ) + (μ · g_p) } }
```

Our implementation uses the exponential mechanism to select the direction in which the gradient has its maximum value, then updates θ in only the selected dimension. To get the right sensitivity, we compute the gradient with $L\nabla^{LR}_{L\infty}$, which requires an $L\infty$ norm bound on its input and ensures bounded $L\infty$ sensitivity.

We mix several variants of differential privacy in this implementation. Each use of the exponential mechanism provides ϵ-differential privacy; each iteration of the loop satisfies $\frac{1}{2}\epsilon^2$-zCDP, and the whole algorithm satisfies $(\frac{1}{2}\epsilon^2 + 2\sqrt{\frac{1}{2}\epsilon^2 \log(1/\delta)}, \delta)$-differential privacy. The use of zCDP for composition is an improvement over the manual analysis of Talwar et al. [Talwar et al. 2015], which used advanced composition.

5.4 Minibatching

An alternative form of minibatching to the one discussed in Section 5.2 is to randomly sample a subset of of the data in each iteration. Bassily et al. [Bassily et al. 2014b] present an algorithm for differentially private stochastic gradient descent based on this idea: their approach samples a single random example from the training to compute the gradient in each iteration, and leverages the idea of *privacy amplification* to improve privacy cost. The privacy amplification lemma states

that if mechanism $\mathcal{M}(D)$ provides (ϵ, δ)-differential privacy for the datset D of size n, then running $\mathcal{M}$ on uniformly random γn entries of D (for $\gamma \leq 1$) provides $(2\gamma\epsilon, \gamma\delta)$-differential privacy [Bassily et al. 2014b; Wang et al. 2018] (this bound is loose, but used here for readability).

We encode the privacy amplification lemma in DUET using the `sample` construct defined in Section 4. Similar privacy amplification lemmas exist for RDP [Wang et al. 2018] and tCDP [Bun et al. 2018], but not for zCDP. We can use sampling with privacy amplification to implement minibatching SGD in DUET. Under (ϵ, δ)-differential privacy with privacy amplification, DUET derives a total privacy cost of $(4(b/m)\epsilon\sqrt{2k\log(1/\delta')}, (b/m)k\delta + \delta')$-differential privacy for this algorithm, which is equivalent to the manual proof of Bassily et al. [Bassily et al. 2014b].

```
minibatch-gradient-descent X y k b η ϵ δ ≜
  let X₁ = clip-matrix X in
  loop [δ] k on zeros (cols X₁) <X₁, y> {t, θ ⇒
    sample b on X₁, y {X₁', y' ⇒
      g_p ← noisy-grad θ X₁' y' ϵ δ ; return θ − η · g_p }}
```

5.5 Parallel-Composition Minibatching

As a final form of minibatching, we consider extending the parallel composition approach used by Wu et al. [Wu et al. 2017] for *sensitivity* to parallel composition of *privacy mechanisms* for minibatching in the gradient perturbation approach from Section 5.1. Since the minibatches are disjoint in this approach, we can leverage the parallel composition property for privacy mechanisms (McSherry [McSherry 2009b], Theorem 4; Dwork & Lei [Dwork and Lei 2009], Corollary 20), which states that running an (ϵ, δ)-differentially private mechanism k times on k disjoint subsets of a database yields (ϵ, δ)-differential privacy. We encode this concept in DUET using the `pfld-rows` construct defined in Section 4. The arguments to `pfld-rows` include the dataset and a function representing an (ϵ, δ)-differentially private mechanism, and `pfld-rows` ensures (ϵ, δ)-differential privacy for the dataset. This version considers minibatches of size 1, and is easily extended to consider other sizes. We can use `pfld-rows` to implement epoch-based minibatching with gradient perturbation, even for privacy variants like zCDP which do not admit sampling:

```
epoch b ρ η ≜
  pλ xs ys θ ⇒
    let s = ℝ⁺[1.0]/real b in
    g ← mgauss[s, ρ] <xs, ys> {∇^LR[θ ; xs, ys] } ;
    return θ − η · g
```

```
epoch-minibatch-GD X y ρ η k b ≜
  let m₀ = zeros (cols xs) in
  loop k on m₀ <X, y> {a, θ ⇒
    pfld-rows(b, θ, mclip^L2 X, y, epoch b ρ η)
  }
```

This algorithm is similar in concept to the output perturbation approach of Wu et al. [Wu et al. 2017], but leverages parallel composition of privacy mechanisms for gradient perturbation instead, and has not been previously published. The algorithm runs k epochs with a batch size of b, for a total of kb iterations. Duet derives a privacy cost of $k\rho$-zCDP for the algorithm.

5.6 Composing Privacy Variants to Build Complete Learning Systems

Putting together the pieces we have described to build real machine learning systems that preserve differential privacy often requires mixing privacy variants in order to obtain optimal results. We can use DUET's ability to mix variants of differential privacy to combine components in a way that optimizes the use of the privacy budget. We demonstrate this

```
adaptiveClippingGradientDescent xs ys k ϵ δ ηs bs ≜
  means ← colMeans(xs, ϵ, δ, bs);
  scales ← EPS_DP { colScaleParams(xs, ϵ, bs, means) };
  let xs_n = box (normalize xs means scales) in
  η ← pick_η(unbox xs_n, ys, k, ϵ, δ, ηs);
  ZCDP[δ]{noisyGradientDescentZCDP(b·(unbox xs_n), ys, k, η, ϵ, δ)}
```

ability with an example that performs several data-dependent analyses as pre-processing steps before training a model. Our example uses Duet's ability to mix variants to compose z-score normalization (using both pure ϵ and (ϵ, δ)-differential privacy), hyperparameter tuning (with (ϵ, δ)-differential privacy), and gradient descent (with zCDP), returning a total (ϵ, δ) privacy cost.

6 IMPLEMENTATION & EVALUATION

This section describes our implementation of Duet, and our empirical evaluation of Duet's ability to produce accurate differentially private models. Our results demonstrate that the state-of-the-art privacy bounds derivable by Duet can result in huge gains in accuracy for a given level of privacy.

6.1 Implementation & Typechecking Performance

We have implemented a prototype of Duet in Haskell that includes type inference of privacy bounds, and an interpreter that runs on all examples described in this paper. We do not implement Hindley-Milner-style constraint-based type inference of quantified types; our type inference is syntax-directed and limited to construction of privacy bounds as symbolic formulas over input variables. Our implementation of type inference roughly follows the bottom-up approach of *DFuzz*'s implementation [De Amorim et al. 2014]. Type checking requires solving constraints over symbolic expressions containing log and square root operations. Prior work (*DFuzz* and *HOARe*^{2}) uses an SMT solver during typechecking to check validity of these constraints, but SMT solvers typically do not support operators like log and square root, and struggle in the presence of non-linear formulas. Because of these limitations, we implement a custom solver for inequalities over symbolic real expressions instead of relying on support from off-the-shelf solvers. Our custom solver is based on a simple decidable (but incomplete) theory which supports log and square root operations, and a more general subset of non-linear (polynomial) formulas than typical SMT theories.

Technique	LOC	Time (ms)
Noisy G.D.	23	0.51ms
G.D. + Output Pert.	25	0.39ms
Noisy Frank-Wolfe	31	0.59ms
Minibatching	26	0.51ms
Parallel minibatching	42	0.65ms
Gradient clipping	21	0.40ms
Hyperparameter tuning	125	3.87ms
Adaptive clipping	68	1.01ms
Z-Score normalization	104	1.51ms

Fig. 7. Summary of Typechecking Performance on Case Study Programs

The Duet typechecker demonstrates very practical performance. Figure 7 summarizes the number of lines of code and typechecking time for each of our case study programs; even medium-size programs with many functions typecheck in just a few milliseconds. Our implementation is open source and freely available on GitHub at: https://github.com/uvm-plaid/duet.

6.2 Evaluation of Private Gradient Descent and Private Frank-Wolfe

We also study the accuracy of the models produced by the Duet implementations of private gradient descent and private Frank-Wolfe in Section 5. We evaluate both algorithms on 4 datasets. Details about the datasets can be found in Figure 8.

We ran both algorithms on each dataset with per-iteration $\epsilon_i \in \{0.0001, 0.001, 0.01, 0.1\}$ and then used Duet to derive the corresponding total privacy cost. We fixed $\delta = \frac{1}{n^2}$, where n is the size of the dataset. For private gradient descent, we set $\eta = 1.0$, and for private Frank-Wolfe we set the size of each corner $c = 100$.

We randomly shuffled each dataset, then chose 80% of the dataset as training data and reserved 20% for testing. We ran each training algorithm 5 times on the training data, and take the average testing error over all 5, to account for the randomness in the training process.

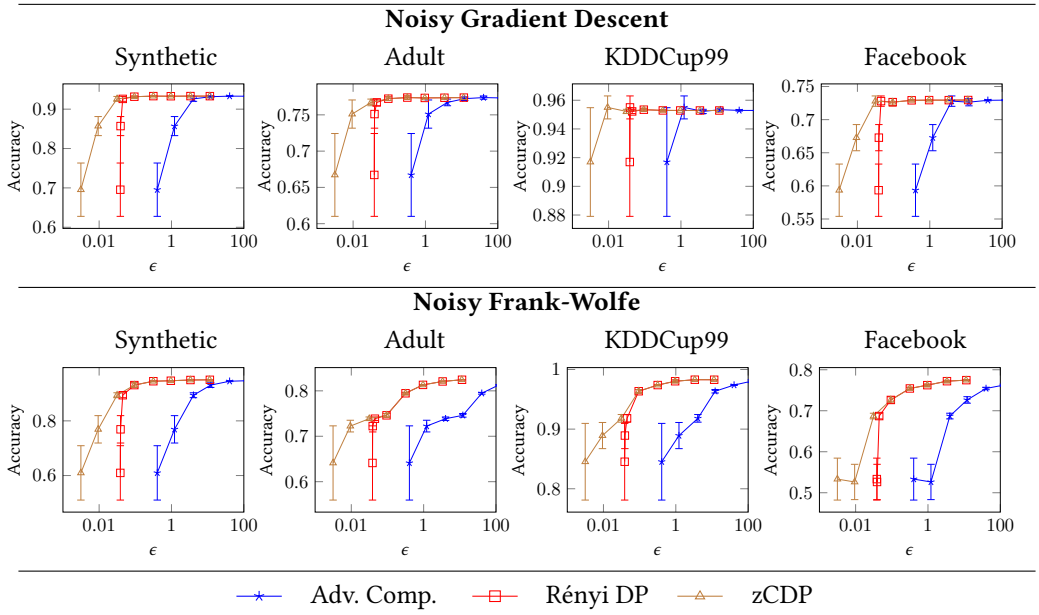

Fig. 9. Accuracy Results for Noisy Gradient Descent (Top) and Noisy Frank-Wolfe (Bottom).

We present the results in Figure 9. Both algorithms are capable of generating accurate models at reasonable values of ϵ. Note that all three models in the results provide *exactly the same privacy guarantee* for a given value of ϵ, yet their accuracies vary significantly—demonstrating the advantages of recently developed variants of differential privacy.

Dataset	Samples	Dim.
Synthetic	10,000	20
Adult	45,220	104
KDDCup99	70,000	114
Facebook	40,949	54

Fig. 8. Dataset Used in Accuracy Evaluation

7 CONCLUSION

We have presented DUET, a language and type system for expressing and statically verifying privacy-preserving programs. Unlike previous work, DUET is agnostic to the underlying privacy definition, and requires only that it support sequential composition and post-processing. We have extended DUET to support several recent variants of differential privacy, and our case studies demonstrate that DUET derives state-of-the-art privacy bounds for a number of useful machine learning algorithms. We have implemented a prototype of DUET, and our experimental results demonstrate the benefits of flexibility in privacy definition.

ACKNOWLEDGMENTS

The authors would like to thank Arthur Azevedo de Amorim, Justin Hsu, and Om Thakkar for their helpful comments. This work was supported by the Center for Long-Term Cybersecurity, Alibaba AIR, DARPA & SPAWAR via N66001-15-C-4066, IARPA via 2019-1902070008, and NSF award 1901278. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes not withstanding any copyright annotation therein. The views, opinions, and/or findings expressed are those of the authors and should not be interpreted as representing the official views or policies of any US Government agency.

REFERENCES

2016. Apple previews iOS 10, the biggest iOS release ever. http://www.apple.com/newsroom/2016/06/apple-previews-ios-10-biggest-ios-release-ever.html.

2019. scikit-learn: Standardization, or mean removal and variance scaling. https://scikit-learn.org/stable/modules/preprocessing.html#preprocessing-scaler

Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. 2016. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 308–318.

Arthur Azevedo de Amorim, Marco Gaboardi, Justin Hsu, Shin-ya Katsumata, and Ikram Cherigui. 2017. A semantic account of metric preservation. In *POPL*, Vol. 52. ACM, 545–556.

Andrew Barber. 1996. *Dual Intuitionistic Linear Logic.* Technical Report ECS-LFCS-96-347. University of Edinburgh.

Gilles Barthe, Gian Pietro Farina, Marco Gaboardi, Emilio Jesus Gallego Arias, Andy Gordon, Justin Hsu, and Pierre-Yves Strub. 2016a. Differentially Private Bayesian Programming. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS '16)*. ACM, New York, NY, USA, 68–79. https://doi.org/10.1145/2976749.2978371

Gilles Barthe, Noémie Fong, Marco Gaboardi, Benjamin Grégoire, Justin Hsu, and Pierre-Yves Strub. 2016b. Advanced probabilistic couplings for differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 55–67.

Gilles Barthe, Marco Gaboardi, Emilio Jesús Gallego Arias, Justin Hsu, Aaron Roth, and Pierre-Yves Strub. 2015. Higher-Order Approximate Relational Refinement Types for Mechanism Design and Differential Privacy. In *POPL*. ACM, 55–68.

Gilles Barthe, Marco Gaboardi, Benjamin Grégoire, Justin Hsu, and Pierre-Yves Strub. 2016c. Proving differential privacy via probabilistic couplings. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*. ACM, 749–758.

Gilles Barthe, Marco Gaboardi, Justin Hsu, and Benjamin Pierce. 2016d. Programming language techniques for differential privacy. *ACM SIGLOG News* 3, 1 (2016), 34–53.

Gilles Barthe, Boris Köpf, Federico Olmedo, and Santiago Zanella-Béguelin. 2013. Probabilistic relational reasoning for differential privacy. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 35, 3 (2013), 9.

Gilles Barthe and Federico Olmedo. 2013. Beyond differential privacy: Composition theorems and relational logic for f-divergences between probabilistic programs. In *International Colloquium on Automata, Languages, and Programming*. Springer, 49–60.

Raef Bassily, Adam Smith, and Abhradeep Thakurta. 2014a. Private empirical risk minimization: Efficient algorithms and tight error bounds. In *Foundations of Computer Science (FOCS), 2014 IEEE 55th Annual Symposium on*. IEEE, 464–473.

Raef Bassily, Adam Smith, and Abhradeep Thakurta. 2014b. Private empirical risk minimization: Efficient algorithms and tight error bounds. In *Foundations of Computer Science (FOCS), 2014 IEEE 55th Annual Symposium on*. IEEE, 464–473.

Mark Bun, Cynthia Dwork, Guy N Rothblum, and Thomas Steinke. 2018. Composable and versatile privacy via truncated CDP. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*. ACM, 74–86.

Mark Bun and Thomas Steinke. 2016. Concentrated differential privacy: Simplifications, extensions, and lower bounds. In *Theory of Cryptography Conference*. Springer, 635–658.

Kamalika Chaudhuri, Claire Monteleoni, and Anand D Sarwate. 2011. Differentially private empirical risk minimization. *Journal of Machine Learning Research* 12, Mar (2011), 1069–1109.

Kamalika Chaudhuri and Staal A Vinterbo. 2013. A stability-based validation procedure for differentially private machine learning. In *Advances in Neural Information Processing Systems*. 2652–2660.

Ezgi Çiçek, Weihao Qu, Gilles Barthe, Marco Gaboardi, and Deepak Garg. 2018. Bidirectional Type Checking for Relational Properties. *CoRR* abs/1812.05067 (2018). arXiv:1812.05067 http://arxiv.org/abs/1812.05067

Arthur Azevedo De Amorim, Marco Gaboardi, Emilio Jesús Gallego Arias, and Justin Hsu. 2014. Really Natural Linear Indexed Type Checking. In *Proceedings of the 26nd 2014 International Symposium on Implementation and Application of Functional Languages*. ACM, 5.

Arthur Azevedo de Amorim, Marco Gaboardi, Justin Hsu, and Shin-ya Katsumata. 2018. Metric Semantics for Probabilistic Relational Reasoning. *CoRR* abs/1807.05091 (2018). arXiv:1807.05091 http://arxiv.org/abs/1807.05091

Cynthia Dwork. 2006. Differential Privacy. In *Automata, Languages and Programming*, Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener (Eds.). Lecture Notes in Computer Science, Vol. 4052. Springer Berlin Heidelberg, 1–12. https://doi.org/10.1007/11787006_1

Cynthia Dwork and Jing Lei. 2009. Differential privacy and robust statistics. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*. ACM, 371–380.

Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. 2006. Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography Conference*. Springer, 265–284.

Cynthia Dwork, Aaron Roth, et al. 2014. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science* 9, 3–4 (2014), 211–407.

Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. 2014. Rappor: Randomized aggregatable privacy-preserving ordinal response. In *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*. ACM, 1054–1067.

Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. 2015. Model Inversion Attacks That Exploit Confidence Information and Basic Countermeasures. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security (CCS '15)*. ACM, New York, NY, USA, 1322–1333. https://doi.org/10.1145/2810103.2813677

Arik Friedman, Shlomo Berkovsky, and Mohamed Ali Kaafar. 2016. A differential privacy framework for matrix factorization recommender systems. *User Modeling and User-Adapted Interaction* 26, 5 (2016), 425–458.

Marco Gaboardi, Andreas Haeberlen, Justin Hsu, Arjun Narayan, and Benjamin C Pierce. 2013. Linear dependent types for differential privacy. In *POPL*, Vol. 48. ACM, 357–370.

Jean-Yves Girard. 1987. Linear Logic. *Theor. Comput. Sci.* 50, 1 (Jan. 1987), 1–102. https://doi.org/10.1016/0304-3975(87)90045-4

Samuel Haney, Ashwin Machanavajjhala, John M Abowd, Matthew Graham, Mark Kutzbach, and Lars Vilhuber. 2017. Utility cost of formal privacy for releasing national employer-employee statistics. In *Proceedings of the 2017 ACM International Conference on Management of Data*. ACM, 1339–1354.

Noah Johnson, Joseph P Near, and Dawn Song. 2018. Towards practical differential privacy for SQL queries. *Proceedings of the VLDB Endowment* 11, 5 (2018), 526–539.

Noah M. Johnson, Joseph P. Near, and Dawn Xiaodong Song. 2017. Towards Practical Differential Privacy for SQL Queries. *CoRR* abs/1706.09479 (2017). http://arxiv.org/abs/1706.09479

Ashwin Machanavajjhala, Daniel Kifer, John Abowd, Johannes Gehrke, and Lars Vilhuber. 2008. Privacy: Theory meets practice on the map. In *Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*. IEEE Computer Society, 277–286.

Frank McSherry and Kunal Talwar. 2007. Mechanism design via differential privacy. In *Foundations of Computer Science, 2007. FOCS'07. 48th Annual IEEE Symposium on*. IEEE, 94–103.

Frank D McSherry. 2009a. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*. ACM, 19–30.

Frank D McSherry. 2009b. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*. ACM, 19–30.

Ilya Mironov. 2017. Renyi differential privacy. In *Computer Security Foundations Symposium (CSF), 2017 IEEE 30th*. IEEE, 263–275.

Prashanth Mohan, Abhradeep Thakurta, Elaine Shi, Dawn Song, and David Culler. 2012. GUPT: privacy preserving data analysis made easy. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. ACM, 349–360.

Aleksandar Nanevski, Frank Pfenning, and Brigitte Pientka. 2008. Contextual Modal Type Theory. *ACM Trans. Comput. Logic* 9, 3, Article 23 (June 2008), 49 pages. https://doi.org/10.1145/1352582.1352591

Arjun Narayan and Andreas Haeberlen. 2012. DJoin: differentially private join queries over distributed databases. In *Presented as part of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*. 149–162.

Joseph P. Near, David Darais, Chike Abuah, Tim Stevens, Pranav Gaddamadugu, Lun Wang, Neel Somani, Mu Zhang, Nikhil Sharma, Alex Shan, and Dawn Song. 2019. Duet: An Expressive Higher-order Language and Linear Type System for Statically Enforcing Differential Privacy. *CoRR* abs/1909.02481 (2019). https://arxiv.org/abs/1909.02481

Nicolas Papernot, Martín Abadi, Ulfar Erlingsson, Ian Goodfellow, and Kunal Talwar. 2016. Semi-supervised knowledge transfer for deep learning from private training data. *arXiv preprint arXiv:1610.05755* (2016).

Davide Proserpio, Sharon Goldberg, and Frank McSherry. 2014. Calibrating data to sensitivity in private data analysis: A platform for differentially-private analysis of weighted datasets. *PVLDB* 7, 8 (2014), 637–648.

Jason Reed and Benjamin C Pierce. 2010. Distance makes the types grow stronger: a calculus for differential privacy. *ICFP* 45, 9 (2010), 157–168.

Indrajit Roy, Srinath TV Setty, Ann Kilzer, Vitaly Shmatikov, and Emmett Witchel. 2010. Airavat: Security and Privacy for MapReduce.. In *NSDI*, Vol. 10. 297–312.

Tetsuya Sato. 2016. Approximate relational Hoare logic for continuous random samplings. *Electronic Notes in Theoretical Computer Science* 325 (2016), 277–298.

Tetsuya Sato, Gilles Barthe, Marco Gaboardi, Justin Hsu, and Shin-ya Katsumata. 2019. Approximate span liftings: Compositional semantics for relaxations of differential privacy. In *2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. IEEE, 1–14.

R. Shokri, M. Stronati, C. Song, and V. Shmatikov. 2017. Membership Inference Attacks Against Machine Learning Models. In *2017 IEEE Symposium on Security and Privacy (SP)*. 3–18. https://doi.org/10.1109/SP.2017.41

Shuang Song, Kamalika Chaudhuri, and Anand D Sarwate. 2013. Stochastic gradient descent with differentially private updates. In *Global Conference on Signal and Information Processing (GlobalSIP), 2013 IEEE*. IEEE, 245–248.

Kunal Talwar, Abhradeep Guha Thakurta, and Li Zhang. 2015. Nearly optimal private lasso. In *Advances in Neural Information Processing Systems*. 3025–3033.

Yu-Xiang Wang, Borja Balle, and Shiva Kasiviswanathan. 2018. Subsampled Rényi Differential Privacy and Analytical Moments Accountant. *CoRR* abs/1808.00087 (2018). arXiv:1808.00087 http://arxiv.org/abs/1808.00087

X. Wu, M. Fredrikson, S. Jha, and J. F. Naughton. 2016. A Methodology for Formalizing Model-Inversion Attacks. In *2016 IEEE 29th Computer Security Foundations Symposium (CSF)*. 355–370. https://doi.org/10.1109/CSF.2016.32

Xi Wu, Fengan Li, Arun Kumar, Kamalika Chaudhuri, Somesh Jha, and Jeffrey Naughton. 2017. Bolt-on Differential Privacy for Scalable Stochastic Gradient Descent-based Analytics. In *Proceedings of the 2017 ACM International Conference on Management of Data (SIGMOD '17)*. ACM, New York, NY, USA, 1307–1322. https://doi.org/10.1145/3035918.3064047

Danfeng Zhang and Daniel Kifer. 2017. LightDP: Towards automating differential privacy proofs. In *POPL*, Vol. 52. ACM, 888–901.

Hengchu Zhang, Edo Roth, Andreas Haeberlen, Benjamin C. Pierce, and Aaron Roth. 2019. Fuzzi: A Three-Level Logic for Differential Privacy. Accepted for publication in PACMPL / ICFP 2019.