Master's Thesis

Master's Programme in Data Science

Benchmarking Differential Privacy Optimizers: A Comparative Study

Andrew Zaki

July 31, 2025

Supervisor(s): Dr. Mikko A. Heikkilä , Professor Antti Honkela

University of Helsinki

Faculty of Science

P. O. Box 68 (Pietari Kalmin katu 5)

00014 University of Helsinki

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Koulutusohjelma — Utbildningsprogram — Degree programme
Faculty of Science	Master's Programme in Data Science

Tekijä — Författare — Author

Andrew Zaki

Työn nimi — Arbetets titel — Title

Benchmarking Differential Privacy Optimizers: A Comparative Study

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidantal — Number of pages
Master's Thesis	July 31, 2025	63

Tiivistelmä — Referat — Abstract

Differential Privacy (DP) provides a mathematically principled framework for limiting the influence of individual data points during training, thereby reducing the risk of data leakage or memorization by machine learning models. As new DP optimizers continue to emerge, it becomes increasingly important to benchmark their utility under consistent and realistic conditions. This thesis introduces a unified benchmarking framework for evaluating DP optimizers under consistent experimental conditions. We benchmark several recent DP optimizers, including non-matrix methods such as DP-Dice, DP-DiSK, and DP-AdamBC, as well as matrix-mechanism-based methods that inject correlated noise across training steps instead of independent noise, adapted for single- and multi-epoch training, against the standard DP-SGD baseline. Our experiments span consistent datasets, models, privacy budgets, and training regimes, enabling a fair comparison of utility across methods. In total, we conduct 2,476 runs across all optimizers, forming one of the most comprehensive evaluations to date. To organize the analysis, we divide our methodology into three distinct setups: non-matrix optimizers (Setup 1), single-epoch matrix mechanisms (Setup 2), and multi-epoch matrix mechanisms (Setup 3). Results show that while DP-SGD remains the strongest among non-matrix optimizers, matrix-based mechanisms outperform it in relaxed privacy settings. Notably, a convergence-aware matrix mechanism variant consistently outperforms DP-SGD, even under strict privacy constraints in multi-epoch training. These findings highlight the effectiveness of matrix mechanisms in enhancing privacy-utility tradeoffs and suggest that convergence-aware designs hold promise for practical deployment in private machine learning. All benchmark data and code are publicly available to support reproducibility and further research.

ACM Computing Classification System (CCS):
Computing methodologies → Machine learning → Machine learning approaches → Neural networks
Security and privacy → Security services → Privacy-preserving protocols
Mathematics of computing → Mathematical analysis → Mathematical optimization → Continuous optimization

Avainsanat — Nyckelord — Keywords

Differential Privacy, Private Machine Learning, Correlated Noise Mechanisms, Benchmarking

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — Övriga uppgifter — Additional information

Contents

1. Introduction

As machine learning continues to be applied to sensitive domains like healthcare [AHK18], finance, and user personalization [Man16], protecting the privacy of individual data has become a critical concern. In many cases, standard training procedures risk memorizing [CLE+19] or leaking information about specific users. Differential privacy (DP) [DMNS06] provides a rigorous mathematical framework for limiting this risk. By adding independent noise to each training step, DP ensures that the model's output does not depend too much on any single data point, even in worst-case scenarios. One common method for implementing DP in deep learning is DP-SGD [ACG+16], which we use as the baseline optimizer in this thesis.

Training deep learning models with DP remains an active area of research. In particular, many alternative DP optimizers have been proposed—including DP-Dice, DP-DiSK, and DP-AdamBC—each aiming to improve utility. Additionally, matrix-mechanism-based optimizers have introduced correlated noise as an alternative to per-step independent noise injection. Yet these methods are often evaluated in isolation, under differing experimental setups, making it difficult to compare them meaningfully. This thesis addresses that problem by building a unified benchmarking framework for differentially private optimization. Using this framework, we systematically evaluate recent DP optimizers under consistent conditions—including the same datasets, models, privacy budgets, and training regimes—using DP-SGD as the baseline.

The primary goal of this work is to determine whether these newer optimizers offer better utility than DP-SGD across different tasks and privacy levels. All data[†] from our experiments, including training and test accuracies, losses, and selected hyperparameters, are available in JSON format. The benchmark code[‡] is publicly available.

[†] https://www.kaggle.com/datasets/andrewhanynady/differential-privacy-optimizer-benchmark
[‡] https://github.com/Andrew-Hany/Benchmarking-Differential-Privacy-Optimizers

2. Preliminaries and Background

This chapter introduces the key concepts and methods necessary for understanding the differentially private optimization algorithms benchmarked in this thesis. We first present the preliminaries of differential privacy, including formal definitions and mechanisms commonly used in private machine learning. We then provide background on all the DP optimizers considered in our evaluation, covering both non-matrix methods and matrix-mechanism-based approaches.

2.1 Differential Privacy

Differential privacy (DP) [Dwo06, DMNS06, Dwo11, DR14] is a widely adopted framework for providing strong, quantifiable privacy guarantees in data analysis and machine learning. By ensuring that the probability distribution of a computation's output does not change significantly when a single individual's data is modified, DP limits the influence of any one person on the overall output, thereby reducing the risk of privacy breaches. This guarantee is formalized using the concept of *adjacent datasets*, which differ by only one individual (via addition, removal, or replacement). A key strength of DP is its ability to enable useful data analysis while rigorously bounding privacy loss, though this comes with an inherent trade-off: stronger privacy typically requires injecting more randomness, which may reduce accuracy. In modern machine learning, especially in deep learning, DP is increasingly important due to models' tendency to memorize training data [CLE+19], posing risks of leaking sensitive information in settings such as medical records [AHK18], financial transactions, or personal identifiers [Man16].

DP is a mathematical framework for quantifying the privacy guarantees of algorithms operating on sensitive datasets. A randomized algorithm, or *mechanism*, is said to be differentially private if its output distribution does not change significantly when the data of a single individual in the input is modified. Two datasets D and D' are said to

be *adjacent* if they differ by the addition or removal of exactly one individual's data. This formalizes the protection of individual data contributions. Throughout this work, we adopt *sample-level adjacency*, meaning that privacy guarantees apply with respect to individual data samples.

Definition 1 (Approximate Differential Privacy [DKM+06, DMNS06]) *A randomized mechanism $\mathcal{M} : \mathcal{D} \to \mathcal{R}$ satisfies (ϵ, δ)-DP if for all adjacent datasets D, D' and all measurable subsets $S \subseteq \mathcal{R}$,*

$$\Pr[\mathcal{M}(D) \in S] \leq e^{\epsilon} \Pr[\mathcal{M}(D') \in S] + \delta. \tag{2.1}$$

*When $\delta = 0$, this is known as **pure differential privacy** (or simply, ϵ-DP).*

ADP extends **pure DP**, which strictly bounds the worst-case change in a mechanism's output distribution by a parameter ϵ. In the pure case, the change in the mechanism's output probability due to adding or removing one individual is strictly bounded by ϵ, with no exceptions. ADP relaxes this guarantee by introducing a second parameter δ, which allows the bound to be exceeded with a small additive probability. This additive slack makes the definition more flexible and is necessary for practical mechanisms such as those using Gaussian noise (see Definition 3). All final privacy guarantees in this thesis are reported using the (ϵ, δ)-DP framework.

Definition 2 (Sensitivity [Dwo06, DMNS06, Dwo11, DR14]) *Let f be a function operating on datasets. The sensitivity Δf of f is defined as:*

$$\Delta f = \max_{D,D'} \|f(D) - f(D')\|, \tag{2.2}$$

where D and D' are adjacent datasets differing by at most one element.

Sensitivity determines the scale of noise required to ensure differential privacy. Intuitively, it measures how much a function's output can change when a single data point in the input dataset is modified. Functions with higher sensitivity require more noise to achieve the same privacy guarantee [Dwo06, DMNS06, Dwo11, DR14].

The choice of norm in the definition of sensitivity depends on the privacy mechanism being used. For example, L_2-sensitivity (denoted $\Delta_2 f$, using the L_2-norm $\|\cdot\|_2$) is required for the Gaussian mechanism (see Definition 3).

To achieve differential privacy, a variety of mechanisms can be employed. While many common approaches rely on adding random noise to the output of a function, other

strategies, such as the exponential mechanism or randomized response, achieve privacy through alternative means [DR14]. In the thesis, we use the Gaussian mechanism.

Definition 3 (Gaussian Mechanism [Dwo06, DKM+06, DR14]) *Let D be a dataset and let $f : \mathcal{D} \to \mathbb{R}^k$ be a function with L_2-sensitivity $\Delta_2 f$. The Gaussian mechanism defines the release:*

$$\mathcal{M}(D) = f(D) + \mathcal{N}(0, \sigma^2 I_k), \tag{2.3}$$

where noise is sampled independently for each coordinate from a Gaussian distribution with variance σ^2. Here, k is the dimension of the output and I_k is the $k \times k$ identity matrix.

The appropriate choice of σ ensures that this mechanism satisfies (ϵ, δ)-differential privacy. The improved analysis of Balle and Wang [BW18] provides a tight calibration for the noise. Specifically, to achieve (ϵ, δ)-DP, one should choose the smallest standard deviation σ such that

$$\Phi\left(\frac{\Delta_2 f}{2\sigma} - \frac{\epsilon\sigma}{\Delta_2 f}\right) - e^{\epsilon}\Phi\left(-\frac{\Delta_2 f}{2\sigma} - \frac{\epsilon\sigma}{\Delta_2 f}\right) \leq \delta, \tag{2.4}$$

where Φ denotes the cumulative distribution function of the standard normal distribution [BW18].

In practice, when a mechanism or algorithm is applied iteratively, we need a framework to track the cumulative privacy loss. **Privacy Accountant** [ACG+16, Mir17, KJPH21] is a mathematical tool that tracks and bounds the cumulative privacy loss incurred under the composition of multiple differentially private mechanisms. In this thesis, we use the *Privacy Random Variable (PRV) accountant* [KJPH21] to compute privacy guarantees for DP-SGD [ACG+16], DP-Disk [ZBB+25], DP-Dice [ZBWH24], and DP-AdamBC [TSL24]. For matrix mechanism experiments (DP-Matrix [CMRT23, DMR+22, KMC+23]), we use the *Gaussian Differential Privacy (GDP)* framework [DRS19], as matrix mechanisms add noise through a single linear transformation, and GDP naturally captures the resulting privacy loss without requiring composition. Both PRV and GDP frameworks also support computing the required noise multiplier σ given a target (ϵ, δ), sampling rate q (i.e., the fraction of the dataset included in each minibatch), and number of steps; we use the implementation from `Opacus` [YSS+21] for this purpose.

In modern differentially private machine learning, most practical algorithms, such as DP-SGD [ACG+16] (Algorithm 1), benefit from privacy amplification by random

sampling or shuffling. In particular, Poisson subsampling [ZW19] (sampling each example independently with a fixed probability) is the default in many implementations, including popular libraries such as `Opacus` [YSS+21]. Amplification by sampling [BST14, BFTT19, ZW19] or shuffling [FMT21, EFM+19] reduces the effective privacy loss per iteration, allowing for lower noise and improved accuracy. Privacy accounting frameworks, such as GDP and PRV, do not require sampling or shuffling to be used, but they can incorporate the privacy amplification effect of Poisson sampling when present, leading to tighter cumulative privacy bounds in deep learning. Shuffling, on the other hand, typically requires dedicated analysis and is not directly supported by standard privacy accountants. However, in certain settings, especially federated or decentralized learning where sampling or shuffling is impractical or impossible, these amplification techniques cannot be used directly [KMA+21]. In these cases, alternative approaches that do not rely on sampling or shuffling, such as matrix mechanism methods [CMRT23, DMR+22, CGM+23, DMP+24, PUC+25, KL24, KMC+23], become necessary to maintain strong privacy guarantees without sacrificing utility.

2.2 Differentially Private Optimizers

In this work, we frame training as an empirical risk minimization (ERM) problem [Vap98, CMS11], which underlies all the differentially private optimizers we study. Given a dataset $D = \{x_1, \ldots, x_N\}$ and a per-example loss function $\mathcal{L}(\theta, x_i)$, the goal is to minimize the average loss over all examples:

$$\min_{\theta \in \mathbb{R}^d} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(\theta, x_i), \tag{2.5}$$

where θ denotes the model parameters, and d is the dimensionality of the parameter space (i.e., the number of trainable parameters). All optimizers studied—DP-SGD, DP-AdamBC, DP-Dice, DP-DiSK, and matrix-mechanism-based approaches—aim to solve this ERM problem subject to differential privacy constraints.

2.2.1 DP-SGD

Introduction to DP-SGD Differentially Private Stochastic Gradient Descent (DP-SGD) is a foundational technique for training deep learning models while ensuring DP [ACG+16, SCS13]. It modifies the standard SGD algorithm [Bot10, BCN18, RM51], by bounding the sensitivity of gradients and adding calibrated noise to the

gradient updates. This approach aims to limit how much the trained model's behavior can be influenced by any single data point, thereby bounding the potential for memorization or leakage of sensitive information.

Standard SGD Optimizer To minimize the empirical risk over a dataset, we use Stochastic Gradient Descent (SGD) [Bot10, BCN18, RM51], an iterative optimization algorithm widely used in machine learning, particularly for training models on large datasets. SGD estimates the gradient of the loss function using a single randomly selected data point or a small mini-batch of data points, significantly reducing the computational cost per iteration compared to using the entire dataset and allowing for faster updates. In this work, we adopt the mini-batch variant of SGD as the base optimization algorithm. The standard update rule for a parameter vector θ at iteration t is given by:

$$\theta_{t+1} = \theta_t - \eta_t \nabla \mathcal{L}(\theta_t, x_i), \tag{2.6}$$

or for a mini-batch L:

$$\theta_{t+1} = \theta_t - \eta_t \frac{1}{|L|} \sum_{x_i \in L} \nabla \mathcal{L}(\theta_t, x_i), \tag{2.7}$$

where η_t denotes the learning rate at iteration t; $\mathcal{L}(\theta)$ is the loss function; x_i is a single data point sampled from the dataset; L is a mini-batch of such data points; and $\nabla \mathcal{L}(\theta_t, x_i)$ denotes the gradient of the loss with respect to θ_t at x_i.

Key Modifications in DP-SGD The complete DP-SGD Algorithm is presented in Algorithm 1. Compared to standard SGD [Bot10, BCN18], DP-SGD [ACG+16] introduces two key modifications to ensure differential privacy:

- **Gradient Clipping:** To ensure DP, the influence of each individual data point on the gradient update must be bounded. In standard gradient-based optimization, the gradient contribution of an individual example can have unbounded sensitivity, meaning it could reveal too much information about that data point. To address this, DP-SGD clips the gradients of each example in the batch to ensure that their L2 norm does not exceed a predefined threshold α. This bounds the sensitivity of the gradients, making them suitable for adding noise.

  In our implementation, we follow a common practical variant where the clipped gradients are additionally normalized by $1/\alpha$. This rescales the result to remove dependence on the absolute clipping threshold and provides more consistent scaling for the noise addition step. This modification was introduced by De

et al. [DBH+22] and shown to improve optimization stability in practice, while remaining functionally equivalent to the original clipping rule.

Let g denote the gradient of the loss with respect to model parameters for a single data point. We define the normalized clipping operation as:

$$\texttt{clip}(g, \alpha) = \frac{1}{\alpha} \cdot \frac{g}{\max(1, \|g\|_2/\alpha)}, \tag{2.8}$$

where α is the clipping threshold. We use this definition in all algorithms that follow.

- **Noise Addition:**

  After clipping, noise is added to the aggregated gradients to ensure DP. The noise is drawn from a Gaussian distribution (see Definition 3), specifically:

  $$\mathcal{N}(0, \sigma^2 I), \tag{2.9}$$

  where σ is the noise scale (often referred to as the noise multiplier), and I is the identity matrix.

  The value of σ is chosen based on the privacy budget (ϵ, δ), the number of iterations T, and the sampling probability. These parameters are typically calibrated using privacy accounting frameworks [ACG+16, Mir17, KJPH21].

2.2.2 DP-AdamBC

Introduction to DP-AdamBC Differentially Private Adam (DP-Adam) is an extension of DP-SGD that uses Adam's [KB15] adaptive learning rates, which adjust how much each parameter is updated based on the history of past gradients, just like the standard Adam optimizer. Differentially Private Adam with Bias Correction (DP-AdamBC) [TSL24] is a recently proposed approach that aims to address a fundamental issue in DP-Adam optimization. In their work [TSL24], Tang et al. show that DP-Adam without proper bias correction behaves almost identically to DP-SGD with momentum, effectively losing its adaptive benefits. DP-AdamBC resolves this by explicitly correcting the biased moment estimates, thereby preserving Adam's original advantages while maintaining formal privacy guarantees.

To understand the contribution of DP-AdamBC [TSL24], it is essential to first understand the mechanics of the standard Adam optimizer [KB15], which serves as its non-private counterpart.

Algorithm 1 DP-SGD [ACG+16]

Input: Examples $\{x_1, \ldots, x_N\}$, per-example loss function $\mathcal{L}(\theta, x)$.
    Hyper Parameters: learning rate η_t, expected batch size L, sampling probability q, number of iterations T, gradient norm bound α, target privacy (ϵ, δ).
Initialize θ_1 randomly.
Computes the noise multiplier using a privacy accountant:
$\sigma = f(\epsilon, \delta, q, T)$
for $t = 1$ to T **do**
    Sample a minibatch L_t by including each x_i independently with probability q.
    Compute and clip gradient:
    for each $i \in L_t$ **do**
        $\mathbf{g}_t(x_i) \leftarrow \nabla_{\theta_t} \mathcal{L}(\theta_t, x_i)$.
        $\bar{\mathbf{g}}_t(x_i) \leftarrow \texttt{clip}(\mathbf{g}_t(x_i), \alpha)$
    end for
    Add noise:
    $\tilde{\mathbf{g}}_t \leftarrow \frac{1}{|L|} \left(\sum_{i \in L_t} \bar{\mathbf{g}}_t(x_i) + \mathcal{N}(0, \sigma^2 \mathbf{I})\right)$
    Descent:
    $\theta_{t+1} \leftarrow \theta_t - \eta_t \tilde{\mathbf{g}}_t$
end for
Output: θ_T

The Adam Optimizer [KB15] Adam is an adaptive optimization algorithm widely used in deep learning due to its fast convergence and robustness across different training regimes. Unlike SGD, which uses a fixed learning rate for all parameters, Adam adjusts learning rates individually based on first and second moment estimates of past gradients. Specifically, it maintains an exponential moving average of the gradients, denoted m_t, and an exponential moving average of the squared gradients, denoted v_t. To correct for their initial bias toward zero, Adam also computes bias-corrected estimates $\hat{m}_t$ and $\hat{v}_t$, which are used in the final parameter update. This helps it adapt to the geometry of the loss surface and improves performance, especially in sparse or noisy settings. As a result, Adam often behaves like a sign-based method, where the direction of the update tends to align with the sign of the gradient, and the magnitude of the unscaled update is typically close to 1 in confident directions; the final step size is then scaled by the learning rate. The moving averages of gradients m_t and squared gradients v_t are updated as follows:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t, \tag{2.10}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2, \tag{2.11}$$

where g_t is the gradient at iteration t, and β_1, β_2 are decay rates. The bias-corrected estimates are:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}. \tag{2.12}$$

The update step is then:

$$\theta_{t+1} = \theta_t - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \zeta}, \tag{2.13}$$

where η is the learning rate and ζ ensures numerical stability.

DP-Adam: Differentially Private Adam (DP- Adam) extends the standard Adam optimizer by incorporating DP mechanisms. It follows the same procedures as in DP-SGD (Algorithm 1), including per-sample gradient clipping, and noise addition, before replacing the final gradient descent step in DP-SGD with the Adam update rule defined in the non-private setting. The primary motivation behind DP-Adam is to ensure that the contribution of any individual data point remains private, thus offering privacy guarantees while leveraging Adam's adaptive learning rates

Limitations of DP-Adam (Bias in the Second Moment Estimate): Although DP-Adam follows the same structure as the original Adam algorithm, it introduces a key problem when noise is added for DP. The issue lies in the second moment estimate v_t, which becomes biased because of the Gaussian noise added to the gradients. This noise does not affect the first moment, but it adds a constant positive shift to the second moment [TSL24]. More precisely, the second moment in DP-Adam becomes (see Appendix A for a full derivation):

$$\mathbb{E}[v_t] = \mathbb{E}[v_t^{\text{clean}}] + (1 - \beta_2^t)\psi, \quad \text{where } \psi = \left(\frac{\sigma}{L}\right)^2. \tag{2.14}$$

This bias ψ is constant and depends only on the privacy parameters, not on the actual gradients. Because of this, even when the gradients are small and stable (low variance), the denominator in the update step gets inflated. This causes the updates to shrink more than they should, especially in directions where Adam would normally take larger steps. As a result, DP-Adam no longer behaves like a sign-based method. When this bias dominates the second moment, DP-Adam effectively turns into DP-SGD with momentum[TSL24] . The update becomes:

$$\theta_{t+1} \approx \theta_t - \eta \frac{\hat{m}_t}{\sqrt{\psi}}. \tag{2.15}$$

This means the update is just a scaled version of the first moment, without the usual adaptivity across parameters. In practice, this removes one of the main benefits of Adam, which is adjusting updates differently for each parameter based on how noisy or stable its gradients are. The problematic nature of DP-Adam's second moment estimator under noise, and its impact on true adaptivity, have been previously noted [MSH+22], with some analyses indicating that the adaptive learning rates in DP-Adam can converge to static values, questioning the long-term utility of the second moment calculations. These problems show that without correcting this bias, DP-Adam can lose its advantage and behave more like DP-SGD. This motivates the need for bias correction, which is the main idea behind DP-AdamBC [TSL24].

DP-Adam with Bias Correction To fix the issues caused by the biased second moment in DP-Adam, DP-AdamBC introduces a simple correction step. Since the bias $\psi = \left(\frac{\sigma}{L}\right)^2$ is known and constant, it can be subtracted from the second moment estimate to undo its effect [TSL24]. The corrected second moment becomes:

$$v_t^{\text{corr}} = v_t - (1 - \beta_2^t)\psi, \tag{2.16}$$

and its bias-corrected form is:

$$\hat{v}_t^{\text{corr}} = \frac{v_t^{\text{corr}}}{1 - \beta_2^t} = \frac{v_t - (1 - \beta_2^t)\psi}{1 - \beta_2^t} = \hat{v}_t - \psi, \tag{2.17}$$

and the update rule is adjusted to:

$$\theta_{t+1} = \theta_t - \eta \frac{\hat{m}_t}{\sqrt{\max(\hat{v}_t - \psi, \zeta')}}, \tag{2.18}$$

where ζ' is a stability constant used to avoid division by zero or negative values. This correction is intended to make DP-AdamBC behave more like the original Adam optimizer, particularly in cases where the gradients are low-variance but strong, by reintroducing Adam's parameter-wise adaptivity [TSL24]. In this work, we benchmark DP-AdamBC against DP-SGD to assess whether this correction leads to meaningful differences in optimizer behavior and effectiveness.

2.2.3 DP-Dice: DP-SGD with Clipped Error Feedback

One of the main limitations of DP-SGD is that it discards useful gradient signal due to clipping and the injection of Gaussian noise. Over multiple iterations, these distortions accumulate, degrading optimization quality. DP-Dice [ZBWH24] addresses this

Algorithm 2 DP-AdamBC [TSL24]

Input: Examples $\{x_1, \ldots, x_N\}$, per-example loss function $\mathcal{L}(\theta, x)$.
Hyperparameters: same as Algorithm 1, with additional β_1, β_2, small constant ζ'.
Shared Steps: Minibatch sampling, gradient computation, clipping, and noise addition as in DP-SGD Algorithm 1.
Initialize: θ_1 randomly; $m_1 = 0$; $v_1 = 0$
Computes the noise multiplier using a privacy accountant:
$\sigma = f(\epsilon, \delta, q, T)$
for $t = 1$ to T **do**
    (Compute $\tilde{\mathbf{g}}_t$ as in DP-SGD Algorithm 1)
    Optimizer Step
    $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1)\tilde{\mathbf{g}}_t$
    $v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2)\tilde{\mathbf{g}}_t^2$
    $\hat{m}_t \leftarrow m_t/(1 - \beta_1^t)$
    $\hat{v}_t \leftarrow v_t/(1 - \beta_2^t)$
    $\theta_t \leftarrow \theta_{t-1} - \eta \cdot \hat{m}_t/\sqrt{\max(\hat{v}_t - (\sigma/L)^2, \zeta')}$
end for
Output: θ_T

by incorporating an error-feedback mechanism that stores and reuses the difference between the true gradient and the clipped gradient at each step. This feedback allows the optimizer to recover information lost in previous updates while preserving formal differential privacy guarantees.

DP-Dice extends DP-SGD by maintaining a residual vector e_t that captures the cumulative error from past clipping operations. At each iteration, both the per-example gradient and the residual vector are clipped using bounds α and α_2, respectively. These are then summed to form the update direction. Noise is added to this direction before the model update. Afterward, the residual is updated using the difference between the raw gradient and the clipped gradient. This approach preserves (ϵ, δ)-DP since the noise is still injected after clipping. The full method is described in Algorithm 3, which mirrors the structure of DP-SGD (Algorithm 1) with the addition of clipped residual feedback and error correction. In principle, DP-Dice can be applied on top of any base optimizer; in our benchmark, we use it with SGD as the underlying update rule. We include DP-Dice in our benchmark to evaluate whether its feedback mechanism improves optimization utility under privacy constraints.

Algorithm 3 DP-Dice [ZBWH24]

Input: Examples $\{x_1, \ldots, x_N\}$, per-example loss function $\mathcal{L}(\theta, x)$.
Hyperparameters: same as Algorithm 1, with additional error clipping bound α_2.
Shared Steps: Minibatch sampling, gradient computation and clipping (see Algorithm 1).
Initialize: θ_1 randomly, residual vector $e_1 \leftarrow 0$
Computes the noise multiplier using a privacy accountant:
$\sigma = f(\epsilon, \delta, q, T)$
for $t = 1$ to T **do**
    Per-Example Gradient Computation and Clipping:
    (Compute $\mathbf{g}_t(x_i)$ and $\bar{\mathbf{g}}_t$ as in DP-SGD Algorithm 1)
    Raw & Clipped Gradient Averaging:
    $g_t \leftarrow \frac{1}{|L|} \sum_{i \in L} \mathbf{g}_t(x_i)$
    $\bar{g}_t \leftarrow \frac{1}{|L|} \sum_{i \in L} \bar{\mathbf{g}}_t(x_i)$
    error Feedback:
    $\bar{g}_t \leftarrow \bar{g}_t + \text{clip}(e_t, \alpha_2)$
    Noise Addition and SGD optimizer step:
    $\theta_{t+1} \leftarrow \theta_t - \eta_t(\bar{g}_t + \mathcal{N}(0, \sigma^2 \mathbf{I}))$
    Error Update:
    $e_{t+1} \leftarrow e_t + g_t - \bar{g}_t$
end for
Output: θ_T

2.2.4 DP-DiSK

Differentially private optimizer with Simplified Kalman filter (DP-DiSK) [ZBB+25] introduces a Kalman-style filtering approach to improve DP optimization. In traditional DP-SGD, each gradient is treated independently, and noise is added at every step, often overwhelming the signal and degrading convergence. To address this, DP-Disk draws inspiration from the Kalman filter [WB95, WZS16], an algorithm from signal processing. A Kalman filter is used to estimate the hidden state of a dynamic system (such as position or velocity) from noisy observations over time. It does this by recursively combining past estimates (what it already knows) with new noisy measurements (what it just observed), weighting them based on their uncertainty. This results in a smoothed estimate that is more accurate than any single noisy input.

DP-Disk applies this same principle to optimization: rather than treating each noisy

gradient as an isolated signal, it combines the current noisy update with past information to smooth out fluctuations. This allows the optimizer to retain useful signal over time, reducing the instability caused by noise while maintaining formal privacy guarantees.

DP-Disk builds on the standard DP-SGD framework by reusing the same privacy-preserving components for gradient clipping and noise addition. Specifically, it applies the same clipping function defined earlier, but instead of directly using gradients evaluated at the current model parameters, it computes a convex combination of gradients at two points: the current parameters θ_t and a shifted point $\theta_t + \gamma \mathbf{dir}_{t-1}$, where $\mathbf{dir}_{t-1}$ is the previous update direction. This weighted gradient is then clipped and averaged across the mini-batch, just as in DP-SGD (Algorithm 1). Gaussian noise is added in the same way as in DP-SGD to obtain a private gradient estimate $\tilde{\mathbf{g}}_t^{\text{noisy}}$, ensuring (ϵ, δ)-differential privacy.

The core innovation of DP-Disk lies in its use of a Kalman-style filtering step. Rather than using the noisy gradient directly for parameter updates, the algorithm recursively smooths it by combining it with the previous update direction via an exponential moving average: $\tilde{\mathbf{g}}_t = (1-\kappa)\tilde{\mathbf{g}}_{t-1} + \kappa \tilde{\mathbf{g}}_t^{\text{noisy}}$. This filtering mechanism helps retain signal across iterations and reduce the impact of injected noise. The resulting filtered gradient is then used to perform a standard SGD update. Finally, the actual update direction $\mathbf{dir}_t = \theta_{t+1} - \theta_t$ is recorded to inform the next iteration's gradient combination step. This structure allows DP-Disk to reduce variance over time, potentially improving convergence and final performance under differential privacy. We include DP-Disk in our benchmark to evaluate whether this smoothing mechanism leads to tangible improvements in private optimization under various training conditions.

Algorithm 4 DP-Disk [ZBB+25]

Input: Examples $\{x_1, \ldots, x_N\}$, per-example loss function $\mathcal{L}(\theta, x)$.
Hyperparameters: same as Algorithm 1, with additional filter coefficients γ, κ
Initialize θ_1 randomly, $\tilde{\mathbf{g}}_0 \leftarrow \tilde{\mathbf{g}}_1^{\text{noisy}}$, $\mathbf{dir}_0 \leftarrow 0$
$\sigma = f(\epsilon, \delta, q, T)$
for $t = 1$ to T **do**
    Sample a minibatch L_t by including each x_i independently with probability q
    Compute and clip gradient:
    $\bar{\mathbf{g}}_t \leftarrow \frac{1}{|L|} \sum_{x_i \in L_t} \text{clip}\left(\left(\frac{1-\kappa}{\kappa\gamma}\right) \nabla\mathcal{L}(\theta_t + \gamma\mathbf{dir}_{t-1}, x_i) + \left(1 - \frac{1-\kappa}{\kappa\gamma}\right) \nabla\mathcal{L}(\theta_t, x_i), \alpha\right)$
    Add noise:
    $\tilde{\mathbf{g}}_t^{\text{noisy}} \leftarrow \bar{\mathbf{g}}_t + \mathcal{N}(0, \sigma^2\mathbf{I})$
    Apply Kalman-style filter:
    $\tilde{\mathbf{g}}_t \leftarrow (1 - \kappa) \cdot \tilde{\mathbf{g}}_{t-1} + \kappa \cdot \tilde{\mathbf{g}}_t^{\text{noisy}}$
    Descent Using SGD update:
    $\theta_{t+1} \leftarrow \theta_t - \eta_t \tilde{\mathbf{g}}_t$
    Update direction:
    $\mathbf{dir}_t \leftarrow \theta_{t+1} - \theta_t$
end for
Output: θ_T

2.2.5 Differentially private Matrix Mechanisms

The matrix mechanism is a general framework in differential privacy for injecting noise in a structured way that reduces its impact on utility. The matrix mechanism was originally introduced in earlier work on differential privacy [HT10, LMH+15] and later extended and generalized in follow-up research [ENU20].

In optimization, matrix mechanism ideas have recently been adapted to improve differentially private training without relying on amplification techniques like subsampling [BST14, BFTT19, ZW19] or shuffling [EFM+19, FMT21], which can be impractical in settings like federated learning [KMA+21]. Recent approaches, such as *matrix-factorized private optimization techniques* [CMRT23, DMR+22, KMC+23, CGM+23, DMP+24, KL24, PUC+25], apply the matrix mechanism to structure the injected noise across iterations rather than treating each step independently.

These methods model the full sequence of gradient updates as a linear system, represented by a *workload matrix* $A \in \mathbb{R}^{T \times T}$, where T is the total number of optimization

steps. The structure of A depends on the optimizer: for example, in standard SGD, A is a lower-triangular matrix of ones, reflecting the fact that each parameter update accumulates the sum of all past gradients, that is, the t-th row of AG represents the prefix sum $\sum_{s=1}^{t} g_s$. The optimization trajectory is represented by a matrix $\Theta \in \mathbb{R}^{T \times d}$, where each row corresponds to the model parameters at a different time step. The updates can be expressed as:

$$\Theta = \Theta_0 - \eta AG,$$

where $\Theta_0 \in \mathbb{R}^{T \times d}$ contains copies of the initial parameter vector, η is the learning rate, and $G \in \mathbb{R}^{T \times d}$ stacks the gradients over time, with each row corresponding to a gradient at a single time step. To ensure privacy, the gradients are clipped row-wise (i.e., over the model dimension) resulting in $\overline{G}$, and Gaussian noise $Z \in \mathbb{R}^{T \times d}$ is then added:

$$\Theta = \Theta_0 - \eta A(\overline{G} + Z).$$

To further improve the privacy-utility tradeoff, the workload matrix A is factorized as $A = BC$, where $C \in \mathbb{R}^{T \times T}$ transforms the problem into a lower-dimensional space focusing on the most privacy-sensitive directions, and $B \in \mathbb{R}^{T \times T}$ projects noise from this lower-dimensional space back to the full optimization space. This factorization enables the injected noise to be structured in a way that minimizes utility loss while satisfying differential privacy [CMRT23, DMR+22, KMC+23]:

$$\Theta = \Theta_0 - \eta(A\overline{G} + \text{sens}(C)BZ).$$

A particularly important case is when $B = A$ and $C = I$, which exactly recovers the behavior of standard DP-SGD, where noise is injected independently at each step. However, since each update contains fresh, uncorrelated noise, the variance of the *total accumulated noise* grows linearly with the number of steps. This leads to high variance in the cumulative update direction, especially over long training horizons. This can severely degrade optimization performance by overwhelming the true gradient signal [CMRT23, DMR+22, KMC+23].

Matrix-factorized methods aim to mitigate this by finding a decomposition $A = BC$ that shifts sensitivity into a lower-dimensional subspace, allowing for more efficient noise injection. The noise is then applied as BZ, where it is correlated across steps in a way that partially cancels out in prefix sums. This reduces the impact of noise on convergence and improves the signal-to-noise ratio during training [CMRT23, DMR+22, KMC+23].

We include matrix-mechanism-based optimizers in our benchmark to evaluate whether

these structured-noise techniques provide meaningful improvements over standard DP-SGD.

Matrix Factorization A well-chosen factorization can lead to a low-sensitivity C, which directly reduces the noise scale required for DP. Meanwhile, B can shape the correlated noise. There are three matrix factorization approaches we use in our benchmark to find a good factorization $A = BC$.

Method 1: Single-Epoch Matrix Mechanism via Fixed-Point Algorithm [DMR+22] Denisov et al. [DMR+22] propose an efficient fixed-point algorithm to solve the matrix factorization problem arising in the single-epoch matrix mechanism. The goal is to minimize the amplification of noise added for differential privacy, while maintaining sensitivity constraints. Specifically, the factorization is designed to minimize the reconstruction error between the privatized update $A\overline{G} + BZ$ and the non-private update AG.

- **Problem Setup.** The matrix A is factorized as $A = BC$, where:
  - $\text{sens}(C) \leq 1$, to ensure differential privacy,
  - $\|B\|_F^2$ is minimized, to reduce noise amplification.

  This leads to the optimization problem:

$$\min_{B,C} \|B\|_F^2 \quad \text{subject to } A = BC,\ \text{sens}(C) \leq 1. \tag{2.19}$$

- **Reformulation Using the Pseudoinverse.** Since $A = BC$, B can be written in terms of C as:

$$B = AC^{\dagger}, \tag{2.20}$$

  where $C^{\dagger} = (C^{\top}C)^{-1}C^{\top}$ is the Moore-Penrose pseudoinverse of C. The pseudoinverse is used instead of the matrix inverse because C may be singular. Substituting this into the objective gives:

$$\min_{C:\ \text{sens}(C)\leq 1} \|AC^{\dagger}\|_F^2. \tag{2.21}$$

- **Sensitivity Expression.** In the single-epoch setting, sensitivity is defined as the largest column norm of C:

$$\text{sens}(C) = \max_i \|C[:, i]\|_2. \tag{2.22}$$

  This tells us how much influence one individual example can have on the output.

- **Deriving the Trace-Based Objective.** As part of their fixed-point formulation, Denisov et al. [DMR+22] introduce the Gram matrix

$$X = C^\top C. \tag{2.23}$$

With this substitution, both the pseudoinverse $C^\dagger$ and the term $AC^\dagger$ can be expressed in terms of X:

$$C^\dagger = X^{-1}C^\top, \quad \text{and} \quad AC^\dagger = AX^{-1}C^\top. \tag{2.24}$$

Using these expressions, we can derive a trace-based form of the objective. Applying the identity $\|M\|_F^2 = \operatorname{tr}(MM^\top)$, we obtain:

$$\begin{aligned} \|AC^\dagger\|_F^2 &= \operatorname{tr}(AC^\dagger(AC^\dagger)^\top) \\ &= \operatorname{tr}(AX^{-1}C^\top CX^{-1}A^\top) \\ &= \operatorname{tr}(AX^{-1}XX^{-1}A^\top) \\ &= \operatorname{tr}(A^\top AX^{-1}). \end{aligned} \tag{2.25}$$

Since $X = C^\top C$, each diagonal entry of X corresponds to the squared ℓ_2-norm of a column of C:

$$X[i,i] = \|C[:,i]\|_2^2. \tag{2.26}$$

Since $\operatorname{sens}(C) = \max_i \|C[:,i]\|_2$, squaring both sides of the inequality $\operatorname{sens}(C) \leq 1$ yields the equivalent constraint:

$$\max_i \|C[:,i]\|_2^2 \leq 1. \tag{2.27}$$

Thus, the sensitivity constraint can be written directly in terms of X as:

$$\max_i X[i,i] \leq 1. \tag{2.28}$$

Thus, the reformulated optimization is:

$$\min_{X \succ 0} \operatorname{tr}(A^\top AX^{-1}) \quad \text{subject to } \max_i X[i,i] \leq 1. \tag{2.29}$$

- **Diagonal Mapping.** Since the trace term $\operatorname{tr}(A^\top AX^{-1})$ depends only on the diagonal entries of X^{-1}, Denisov et al. [DMR+22] define

$$v = \operatorname{diag}(X), \quad v \in \mathbb{R}^T_{>0}, \tag{2.30}$$

and construct a fixed-point iteration directly over the vector v. The full matrix $X^\star$ is later reconstructed from this diagonal after convergence.

This simplifies the objective to:

$$\operatorname{tr}(A^\top A X^{-1}) = \sum_{i=1}^{n} \frac{(A^\top A)_{ii}}{v_i}, \tag{2.31}$$

with the constraint:

$$\max_i v_i \leq 1. \tag{2.32}$$

- **Fixed-Point Iteration.** To solve the optimization, Denisov et al. [DMR+22] construct a fixed-point algorithm that searches for a vector $v \in \mathbb{R}^T_{>0}$ satisfying the condition:

$$\omega(v) = v. \tag{2.33}$$

That is, the output of the map is equal to its input, a property that defines a fixed point. The algorithm starts with an initial guess and repeatedly applies the update rule:

$$\omega(v) = \operatorname{diag}\left(\left(\operatorname{diag}(v)^{1/2} A^\top A \operatorname{diag}(v)^{1/2}\right)^{1/2}\right). \tag{2.34}$$

The iteration proceeds as follows:

$$\begin{aligned} &v^{(0)} \leftarrow \text{initial guess (e.g., random)} \\ &v^{(t+1)} = \omega(v^{(t)}) \\ \text{Stop when} \quad &\|v^{(t+1)} - v^{(t)}\| < \xi, \end{aligned} \tag{2.35}$$

where $\xi > 0$ is a fixed convergence threshold that determines when the iteration is considered sufficiently stable. The resulting vector is denoted $v^\star$, the fixed point of the iteration.

- **Reconstructing the Full Matrix $X^\star$.** Once the fixed point $v^\star$ is obtained, the optimal matrix $X^\star$ is reconstructed as:

$$X^\star = \operatorname{diag}(v^\star)^{-1/2} \left(\operatorname{diag}(v^\star)^{1/2} A^\top A \operatorname{diag}(v^\star)^{1/2}\right)^{1/2} \operatorname{diag}(v^\star)^{-1/2}. \tag{2.36}$$

If $\max_i X^\star[i,i] > 1$, rescale:

$$X^\star \leftarrow \frac{X^\star}{\max_i X^\star[i,i]}. \tag{2.37}$$

The matrix $X^\star$ is then factorized (e.g., using Cholesky decomposition or SVD) to recover C, and finally B is computed as $B = AC^\dagger$. This fixed-point method is efficient, tuning-free, and guarantees convergence under the stated constraints.

Method 2: Multi-Epoch Matrix Mechanism via Dual Optimization [CMRT23] This method extends the matrix mechanism to the **multi-epoch setting**, where each training example may participate in multiple gradient steps across time (e.g., once per epoch). As in the single-epoch case, the goal is to minimize the reconstruction error between the privatized update $A\overline{G}+BZ$ and the non-private update AG, while ensuring differential privacy through a sensitivity constraint.

Step-by-step explanation:

- **Participation Schema.** In multi-epoch training, users participate at multiple time steps. Each user has a *participation pattern* $\pi \subseteq \{1, \ldots, T\}$ indicating when they contribute. The collection of all valid patterns is called the schema Π.

  Here, the term "user" abstracts over whatever contributes to a model update at each time step. In our benchmark setting, we train using mini-batches, so when we refer to a user's participation pattern, we are describing a group of individual samples that are assigned to the same batch. These samples contribute jointly to the same update steps and thus share a single participation pattern. This shared structure is what determines the sensitivity constraints.

  In a typical (k, b)-schema:

  - k is the number of epochs (i.e., times a user participates),
  - b is the number of steps per epoch (or number of *batches* per epoch, not batch size),
  - total steps: $T = k \cdot b$.

  The schema is defined formally as:

  $$\Pi = \{\{i, i+b, i+2b, \ldots, i+(k-1)b\} \mid i = 1, \ldots, b\}\,. \tag{2.38}$$

  Example: For $k = 2$ epochs and $b = 5$ steps per epoch, the total number of time steps is $T = 10$. The participation schema is:

  $$\Pi = \{\{1, 6\}, \{2, 7\}, \{3, 8\}, \{4, 9\}, \{5, 10\}\}.$$

  Each set in Π corresponds to a user and indicates the time steps at which that user participates. In our setup, a user refers to a group of samples assigned to the same batch, contributing together to specific optimizer steps. For instance, the first user, associated with the pattern $\{1, 6\}$, contributes to the optimizer steps at time steps 1 and 6. In this example, each user participates in exactly two updates, once per epoch, with participation steps spaced 5 steps apart.

- **Sensitivity Set D.** Each participation pattern $\pi_j \in \Pi$ is represented by a binary vector $u_j \in \mathbb{R}^T$, where the t-th coordinate is 1 if the user contributes at time step t, and 0 otherwise. Formally:

$$[u_j]_t = \begin{cases} 1 & \text{if } t \in \pi_j \\ 0 & \text{otherwise.} \end{cases} \tag{2.39}$$

  In the (k, b)-participation schema, there are exactly b such patterns, one for each user, so the sensitivity set is:

$$D = \{u_1, \ldots, u_b\} \subseteq \mathbb{R}^T. \tag{2.40}$$

  This set represents the directions in which a single user's data can affect the model across time and is used to define the sensitivity constraint in the matrix mechanism optimization.

- **Generalized Sensitivity Constraint.** As in the single-epoch case, we define the Gram matrix:

$$X = C^\top C \succ 0. \tag{2.41}$$

  In the multi-epoch setting, each user may participate at multiple time steps, and thus contribute to multiple model updates. To ensure DP, we must bound the total influence a user can have across their entire participation pattern, that is, across all update steps where they are involved.

  Using the sensitivity set $D = \{u_1, \ldots, u_b\}$, each vector $u_j \in \mathbb{R}^T$ encodes the time steps where a user contributes to model parameter updates. The total squared influence of this user under the encoding matrix C is:

$$\|Cu_j\|_2^2 = u_j^\top C^\top C u_j = u_j^\top X u_j. \tag{2.42}$$

  Therefore, to ensure that the contribution of any individual user remains within the privacy budget, we enforce the constraint:

$$\max_{u \in D} u^\top X u \leq 1. \tag{2.43}$$

  This generalizes the single-epoch constraint $\max_i X[i, i] \leq 1$, which arises as a special case when each user contributes at only one time step.

- **Reformulate the Objective.** The matrix mechanism objective in the multi-epoch setting is given by:

$$\min_{X \succ 0} \operatorname{tr}(A^\top A X^{-1}) \quad \text{subject to } \max_{u \in D} u^\top X u \leq 1. \tag{2.44}$$

This is structurally identical to the single-epoch formulation: the goal is still to minimize the trace term $\operatorname{tr}(A^\top A X^{-1})$. The only change lies in the sensitivity constraint.

- **Dual Formulation and the Lagrangian.** Choquette-Choo et al. [CMRT23] derive a dual formulation of the matrix mechanism objective to enable more efficient optimization. Instead of directly enforcing the sensitivity constraint $\max_{u \in D} u^\top X u \leq 1$, they introduce a Lagrangian relaxation that transforms the constrained problem into an unconstrained one over the dual variables.

For each vector $u_j \in D$, a non-negative Lagrange multiplier $v_j \geq 0$ is introduced, yielding the Lagrangian:

$$\mathcal{L}(X, v) = \operatorname{tr}(A^\top A X^{-1}) + \sum_{u \in D} v_u (u^\top X u - 1). \tag{2.45}$$

The first term is the original objective. The second term penalizes violations of the sensitivity constraints. To simplify the Lagrangian expression, Choquette-Choo et al. [CMRT23] define the matrix:

$$U = \sum_{u \in D} v_u \, u u^\top. \tag{2.46}$$

With this substitution, the Lagrangian becomes:

$$\mathcal{L}(X, v) = \operatorname{tr}(A^\top A X^{-1}) + \operatorname{tr}(XU) - \sum_{u \in D} v_u. \tag{2.47}$$

We now minimize the Lagrangian with respect to $X \succ 0$. This is a convex optimization problem that admits a closed-form solution. Intuitively, the two trace terms in the objective represent competing effects: the first term encourages X to shrink, while the second encourages it to grow. The optimal solution is the point where these opposing forces are balanced. Although the exact derivation involves matrix calculus, the minimizer can be directly expressed in closed form, as shown in Choquette-Choo et al. [CMRT23].

$$X^{-1} A^\top A X^{-1} = U. \tag{2.48}$$

The solution to this equation is given by:

$$X = U^{-1/2} \left(U^{1/2} A^\top A U^{1/2} \right)^{1/2} U^{-1/2}. \tag{2.49}$$

Substituting this back into the Lagrangian yields:

$$\operatorname{tr}(A^\top A X^{-1}) = \operatorname{tr}(XU) = \operatorname{tr}\left(\left(U^{1/2} A^\top A U^{1/2} \right)^{1/2} \right). \tag{2.50}$$

Therefore, the dual objective becomes:

$$g(v) = 2 \cdot \operatorname{tr}\left(\left(U^{1/2} A^\top A U^{1/2} \right)^{1/2} \right) - \sum_{u \in D} v_u, \tag{2.51}$$

and the corresponding dual problem is:

$$\max_{v \geq 0} \; g(v). \tag{2.52}$$

This formulation provides two key benefits:

- It transforms the constrained primal problem into an unconstrained maximization over $v \geq 0$,
- It enables efficient numerical optimization using standard convex solvers, such as projected gradient methods.

- **Reconstructing the Full Matrix $X^\star$.**

  Once the optimal dual weights $v^\star$ are obtained, define:

$$U = \sum_j v_j^\star u_j u_j^\top. \tag{2.53}$$

  Then reconstruct the optimal Gram matrix:

$$X^\star = U^{-1/2} \left(U^{1/2} A^\top A U^{1/2} \right)^{1/2} U^{-1/2}. \tag{2.54}$$

  The matrix $X^\star$ is then factorized (e.g., using Cholesky decomposition or SVD) to recover C, and finally B is computed as $B = AC^\dagger$.

Method 3: Convergence-Aware Matrix Mechanism with Restart-Based Sequences [KMC+23] This method introduces a tighter convergence analysis that accounts for the real impact of correlated noise in stochastic optimization [KMC+23]. While previous methods focus on minimizing the noise norm $\|B\|_F$, this approach proposes a refined objective that incorporates how noise affects optimization dynamics across time steps.

The key idea is to modify the core objective

$$\min_{C:\text{sens}(C)\leq 1} \|AC^\dagger\|_F^2, \tag{2.55}$$

by inserting a specially designed matrix Λ_τ, which reweights noise directions based on their influence on convergence. This leads to the new objective:

$$\min_{C:\text{sens}(C)\leq 1} \|\Lambda_\tau AC^\dagger\|_F^2. \tag{2.56}$$

Here, $\Lambda_\tau \in \mathbb{R}^{T\times T}$ is a temporal weighting matrix derived from "restart-based virtual sequences," which are used to model the true impact of temporally correlated Gaussian noise. This matrix captures how errors accumulate across iterations and enables better alignment of noise with the algorithm's progress.

The matrix $\Lambda_\tau = [\lambda_{tj}]_{t,j=1,\dots,T}$ is defined as:

$$\lambda_{tj} = \begin{cases} \frac{1}{\sqrt{\tau}} & j = t,\ t \neq 0 \mod \tau \\ -\frac{1}{\sqrt{\tau}} & j = \left\lfloor \frac{t}{\tau} \right\rfloor \tau,\ t \neq 0 \mod \tau,\ t > \tau \\ 1 & j = t,\ t = 0 \mod \tau \\ -1 & j = t - \tau,\ t = 0 \mod \tau,\ t > \tau \\ 0 & \text{otherwise.} \end{cases} \tag{2.57}$$

Here, τ is a restart interval that determines how often the virtual optimization sequence is reset.

While the convergence-aware formulation, based on $\Lambda_\tau A$, is introduced independently in [KMC+23], it is compatible in principle with both the fixed-point and dual optimization methods described above. We adopt this perspective in our implementation (see Section 4.3) to support convergence-aware matrix mechanisms in both single- and multi-epoch settings.

3. Related Work

Benchmarking Non-Private Optimizers While this thesis focuses on differentially private (DP) optimization, it is essential to contextualize DP optimizers within the broader field of optimization research. Schmidt et al. [SSH21] present one of the most comprehensive empirical studies of non-private deep learning optimizers in their work, *Descending through a Crowded Valley*. Their large-scale benchmark evaluates fifteen popular optimizers—including SGD, Adam, RMSProp, and their variants—across eight diverse deep learning tasks, leading to over 50,000 individual training runs.

The key takeaways from their work are threefold. First, optimizer performance is highly problem-dependent, with no single method consistently outperforming others across all benchmarks. Second, they found that using multiple optimizers with default hyperparameters often performs comparably to tuning a single optimizer extensively. Third, Adam remains a strong baseline despite the introduction of many newer methods.

This benchmark sets a high standard for empirical rigor and highlights the challenges of fair and comprehensive comparison in optimization research. Their methodology, which includes varying learning rate schedules and tuning budgets, provides a blueprint for similar evaluations in the DP setting. Indeed, our work draws inspiration from their evaluation framework, adapting it to the context of privacy-preserving learning. However, unlike Schmidt et al., our benchmark introduces additional privacy constraints, such as noise injection and gradient clipping, which fundamentally alter the optimization dynamics.

Hyperparameter Selection in DP Training Optimizing hyperparameters is crucial for achieving strong performance in differentially private (DP) deep learning. Recent work has shown that using strong pretrained models, combined with carefully tuned hyperparameters, can dramatically close the gap between private and non-private model accuracy. However, in this work, we focus on training models from scratch, so we are primarily interested in the role hyperparameter tuning plays in this more chal-

lenging setting. De et al. [DBH+22] demonstrate that, through systematic tuning of learning rate, batch size, noise multiplier, and other key parameters, it is possible to achieve state-of-the-art results on image classification benchmarks by using large batch sizes and targeted modifications to the training pipeline. Their results show that, under a fixed privacy budget, increasing the batch size (sometimes by orders of magnitude) and properly tuning the learning rate are essential for achieving high utility, and that optimal settings often deviate substantially from those used in non-private training. For example, they find that the optimal learning rate is proportional to the batch size for small batches, but stops increasing for large batches, and that the best number of training epochs grows with batch size and model scale.

However, the importance and transferability of specific hyperparameters has recently been challenged by broader replication studies. Morsbach et al. [MRS24] provide a large-scale analysis, finding that while batch size is sometimes beneficial, its effect is not consistent across tasks or architectures. Instead, their results identify the learning rate and the clipping threshold—and especially their interaction—as the primary determinants of DP-SGD performance. However, De et al. [DBH+22] propose normalizing the clipped gradient by the clipping threshold to disentangle the effect of clipping threshold from the learning rate and reduce this coupling. This is the approach [DBH+22] we adopt in our benchmark, where we fix the clipping threshold to 1. Meanwhile, the influence of batch size and number of epochs is found to be relatively minor [MRS24]. They further caution that best practices from non-private settings and hyperparameter sweeps do not always generalize to DP-SGD.

Privacy Accounting for Hyperparameter Tuning In most DP research, hyperparameter tuning is conducted using the private dataset, but only the privacy loss from training the final model is included in the reported privacy budget. This has become standard practice, as reflected in recent benchmarks [DBH+22]. Although theoretical work has proposed differentially private methods for hyperparameter selection [LT19, PS22], empirical evidence suggests that tuning hyperparameters using private data does not substantially compromise overall privacy. In particular, Pradhan et al. [PJTH25] find no statistically significant indication that hyperparameter tuning introduces additional leakage beyond what is captured in the final model. As a result, my thesis reports privacy guarantees only for the final trained model, without accounting for the cost of hyperparameter tuning.

4. Methodology

This chapter outlines the methodology used to benchmark differentially private (DP) optimization algorithms. It begins by describing the benchmarking setup, including problem definitions, hyperparameter grids, and experiment design choices tailored for consistency and reproducibility. The chapter then details the modular architecture of the benchmarking codebase and explains how various DP optimizers were implemented and integrated into the PyTorch-based Opacus framework. Together, these components support a systematic and scalable evaluation of DP optimizers across classification and generation tasks.

4.1 Benchmarking Setup

This thesis systematically benchmarks DP optimization algorithms against the baseline optimizer, DP-SGD. The primary goal is to evaluate whether alternative optimizers can provide better utility under comparable privacy guarantees. To achieve a rigorous and fair comparison, we are inspired by Schmidt et al. [SSH21] to conduct a similar benchmarking study, adapting and utilizing some of their problems. Furthermore, the selection of hyperparameter grids and experimental configurations used in this benchmark draws insights from recent comprehensive hyperparameter studies in the DP setting by De et al. [DBH+22] and Morsbach et al. [MRS24].

The tasks used in our benchmark, summarized in Table 4.1, are directly taken from Schmidt et al. [SSH21]. Their benchmark included a set of standard problems across classification and generation, which we reuse to maintain consistency with existing work. This allows us to compare DP optimizers in a setup that mirrors established benchmarks, while introducing the constraints of differential privacy.

In this thesis, we fix the clipping norm to a standard value ($\alpha = 1$) to keep experiments consistent across all runs. For the other hyperparameters, we perform grid searches

ID	Dataset	Model	Type	Params	Metric
P1	CIFAR-10	Simple CNN: 3c3d	Classification (RGB)	~250K	Accuracy
P2	Fashion-MNIST	Simple CNN: 2c2d	Classification (grayscale)	~225K	Accuracy
P3	Fashion-MNIST	ConvVAE (latent=32)	Generation (grayscale)	~1.8M	Loss

Table 4.1: Summary of benchmark datasets, tasks, and models

over learning rate, batch size, privacy budget (ϵ), and number of training epochs. This search strategy is inspired by the best practices outlined by De et al. [DBH+22] and Morsbach et al. [MRS24], while keeping in mind the limits of available compute.

Specifically, when choosing batch sizes, we were motivated by De et al.'s observation that larger batches can help improve model utility in the DP setting. They find that the optimal learning rate is proportional to the batch size for small batches, but plateaus for larger ones [DBH+22].. Therefore, our grid includes both smaller values (256, 512) and significantly larger ones (1024, 2048). For learning rate, we initially selected four values spaced logarithmically (base 10), as a standard sweep for DP-SGD. During preliminary analysis, we observed that the best-performing configurations often lay at the boundaries of this range, prompting us to include an additional learning rate for all optimizers except `DP-AdamBC`. Privacy budget (ϵ) and the number of training epochs are also varied to allow for a thorough comparison of optimizer performance under different training regimes. To ensure reproducibility, we fix random seeds across runs; in Setup 1, we additionally use multiple independent seeds to measure sensitivity to randomness, generated using `numpy`'s `SeedSequence` API [HMvdW+20]. This method ensures that each seed comes from a distinct, non-overlapping random space, following best practices for random number generation.

Throughout all experiments, we fix the δ parameter to 10^{-5}, which satisfies the standard requirement $\delta < 1/n$, where n is the number of training examples. For optimizer-specific hyperparameters—such as the adaptive moment parameters in DP-AdamBC, the filter coefficients in DP-DiSk, and the error feedback threshold in DP-Dice—we use each method's recommended default values. This ensures a fair and representative evaluation without requiring extensive per-optimizer tuning. These defaults are generally well-justified: DP-DiSk, for example, includes a thorough analysis to select effective coefficients ($\kappa = 0.7$ and $\gamma = 0.5$); DP-AdamBC uses the same values as the standard Adam optimizer ($\beta_1 = 0.9$, $\beta_2 = 0.999$); and for DP-Dice, we set the error clipping threshold to 1, matching the gradient clipping threshold.

Overall, the benchmarking strategy aims to give each optimizer a fair chance by tuning the most important hyperparameters as thoroughly as practical. It is important to

note, however, that the best hyperparameter settings can still depend on the specific task and dataset, and that all reported results should be understood in the context of the search grids and choices described here.

To gain deeper insights into the behavior of DP optimizers, we conduct extensive experiments exploring key hyperparameters. For each experiment, we save detailed data, including training loss and accuracy recorded per batch, testing loss and accuracy recorded per epoch, the noise multiplier, the resulting ϵ, estimated training time, and the hyperparameters used*. This comprehensive data collection enables a thorough analysis of the optimizer's behavior under different conditions. We design three experimental setups, that utilizes the datasets in Table 4.1 to balance comprehensiveness and scalability:

- **Setup 1: Grid-Based Evaluation of Non-Matrix DP Optimizers)**

  This setup performs an extensive grid evaluation across four optimizers: the baseline `DP-SGD` (see Section 2.2.1), `DP-AdamBC` (see Section 2.2.2), `DP-DiSk` (see Section 2.2.4), and `DP-Dice` (see Section 2.2.3). To assess sensitivity to randomness, we run five independent seeds for `DP-SGD` and `DP-AdamBC`. For `DP-DiSk` and `DP-Dice`, we reduce compute by using a single seed and omitting $\epsilon = 5$ from their configurations.

  Initially, we used four learning rates for each optimizer. However, during early analysis, we found that `DP-SGD`, `DP-DiSk`, and `DP-Dice` often achieved peak performance at the edge of the learning rate grid. To support more robust analysis, we added a fifth learning rate for those optimizers. For `DP-AdamBC`, we used a separate learning rate grid: 10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}, as its dynamics differ significantly from other methods.

  The full set of hyperparameter combinations is presented below in two separate grid expressions for clarity. While there is some overlap between them, any shared configurations are not duplicated in the experiments but are simply separated across the two formulations for readability. Here, L denotes the batch size, ϵ the target privacy level, η the learning rate, and E the number of training epochs.

$$\left\{ \begin{matrix} \text{P1} \\ \text{P2} \end{matrix} \right\}_2 \times \left\{ \begin{matrix} \texttt{DP-SGD} \\ \texttt{DP-AdamBC} \end{matrix} \right\}_2 \times \left\{ \eta \right\}_4 \times \left\{ \begin{matrix} L = 256 \\ L = 512 \\ L = 1024 \\ L = 2048 \end{matrix} \right\}_4 \times \left\{ \begin{matrix} \epsilon = 1 \\ \epsilon = 5 \\ \epsilon = 10 \end{matrix} \right\}_3 \times \left\{ \begin{matrix} E = 250 \\ E = 500 \end{matrix} \right\}_2 \times \left\{ \text{seeds} \right\}_5$$

*https://www.kaggle.com/datasets/andrewhanynady/differential-privacy-optimizer-benchmark

$$\left.\begin{matrix}\text{P1}\\\text{P2}\end{matrix}\right\}_2 \times \left.\begin{matrix}\texttt{DP-SGD}\\\texttt{DP-Dice}\\\texttt{DP-DiSk}\end{matrix}\right\}_2 \times \left.\begin{matrix}\eta = 10^{-3}\\\eta = 10^{-2}\\\eta = 10^{-1}\\\eta = 10^{0}\\\eta = 10^{1}\end{matrix}\right\}_5 \times \left.\begin{matrix}L = 256\\L = 512\\L = 1024\\L = 2048\end{matrix}\right\}_4 \times \left.\begin{matrix}\epsilon = 1\\\epsilon = 10\end{matrix}\right\}_2 \times \left.\begin{matrix}E = 250\\E = 500\end{matrix}\right\}_2$$

$$\textbf{Total} = \boxed{2272 \text{ runs}}$$

- **Setup 2: One-Epoch Matrix Comparison**

  This setup provides a controlled comparison between two DP matrix mechanism implementations (see Section 2.2.5) under a single-epoch setting. The first, which we refer to as `DP-Matrix-SE`, is based on the fixed-point matrix factorization method proposed by [DMR+22], while the second, referred to as `DP-Matrix-SE-`λ, incorporates a refined theoretical approach with linearly correlated noise from [KMC+23], which offers tighter convergence guarantees. These two methods are evaluated alongside the standard `DP-SGD` to determine whether matrix-based mechanisms can provide better utility under the same privacy conditions.

  To ensure the experiment is informative despite being limited to one epoch, we intentionally use smaller batch sizes. This increases the number of update steps, allowing us to observe optimizer dynamics more meaningfully. A similar strategy was adopted in [KMC+23], where the authors also used small batch sizes in their one-epoch matrix mechanism experiments to maximize step count. We select Problem P2, from Table 4.1, for this comparison because Fashion-MNIST is relatively easy to optimize and converges quickly. Additionally, P2 has the fewest hyperparameter variations among all benchmark problems, making it the most suitable candidate for a controlled one-epoch evaluation. Here, L denotes the batch size, ϵ the target privacy level, η the learning rate, and E the number of training epochs.

$$\text{P2}\Big\}_1 \times \left.\begin{matrix}\texttt{DP-SGD}\\\texttt{DP-Matrix-SE}\\\texttt{DP-Matrix-SE-}\lambda\end{matrix}\right\}_3 \times \left.\begin{matrix}\eta = 10^{-3}\\\eta = 10^{-2}\\\eta = 10^{-1}\\\eta = 10^{0}\\\eta = 10^{1}\end{matrix}\right\}_5 \times \left.\begin{matrix}L = 32\\L = 64\end{matrix}\right\}_2 \times \left.\begin{matrix}\epsilon = 1\\\epsilon = 10\end{matrix}\right\}_2 \times \Big\{\ \text{E} = 1\ \Big\}_1$$

$$\textbf{Total} = \boxed{60 \text{ runs}}$$

- **Setup 3: Multi-Epoch Matrix Comparison** This setup provides a controlled comparison between two differentially private matrix mechanism implementations (see Section 2.2.5) in the multi-epoch setting. The first, which we refer to as `DP-Matrix-ME`, is based on the dual optimization formulation from [CMRT23], which extends the matrix mechanism to handle repeated sample participation across training steps. The second, referred to as `DP-Matrix-ME-`λ, introduces a convergence-aware weighting matrix, as proposed in [KMC+23]. As in Setup 2, these are evaluated alongside the standard `DP-SGD` to assess their utility under comparable privacy guarantees.

  To keep the matrix mechanism computationally feasible, we limit the number of epochs and restrict the batch sizes to larger values (1024 and 2048) in this setup. This design choice is driven by the need to control the total number of training steps, denoted T, which grows with both the number of epochs and the inverse of the batch size. Since the matrix mechanism requires constructing and operating on matrices of shape $T \times T$ (for the factorization of the workload matrix $A = BC$, where $A, B, C \in \mathbb{R}^{T \times T}$, see Section 2.2.5) and $T \times d$ (where $Z \in \mathbb{R}^{T \times d}$ is a Gaussian noise matrix and d is the number of model parameters), reducing T is essential to avoid excessive memory usage and computational cost. Both the matrix factorization step and the computation of the correlated noise BZ are memory-intensive operations that must be carefully managed. The dual matrix factorization algorithm discussed in Section 2.2.5 breaks down once T exceeds a certain threshold, as the corresponding $T \times T$ matrices can no longer fit into GPU memory, which is a limit that depends on the specific GPU used. Moreover, To enable the matrix multiplication step at larger scales, we developed a custom algorithm that performs GPU-accelerated matrix multiplication in chunks using CPU-GPU streaming. Here, L denotes the batch size, ϵ the target privacy level, η the learning rate, and E the number of training epochs.

$$\left.\begin{matrix}\text{P1}\\ \text{P3}\end{matrix}\right\}_3 \times \left.\begin{matrix}\texttt{DP-SGD}\\ \texttt{DP-Matrix-ME}\\ \texttt{DP-Matrix-ME-}\lambda\end{matrix}\right\}_3 \times \left.\begin{matrix}\eta = 10^{-3}\\ \eta = 10^{-2}\\ \eta = 10^{-1}\\ \eta = 10^{0}\\ \eta = 10^{1}\\ \eta = 10^{2}\end{matrix}\right\}_6 \times \left.\begin{matrix}L = 1024\\ L = 2048\end{matrix}\right\}_2 \times \left.\begin{matrix}\epsilon = 1\\ \epsilon = 10\end{matrix}\right\}_2 \times \left.\begin{matrix}E = 30\end{matrix}\right\}_1$$

Total: 144 runs

Total Number of Experiments: $2272 + 60 + 144 =$ 2476 runs

4.2 Benchmarking Code Design

The benchmarking framework implemented in this thesis is designed with a strong emphasis on modularity, extensibility, and maintainability. It adheres to the SOLID principles of object-oriented design, introduced by Robert C. Martin [Mar02], and further emphasized in the context of software architecture and modular design [Mar17]. The SOLID acronym stands for:

- **S**ingle Responsibility Principle: A class is responsible for only one specific task, making the code easier to maintain and less prone to unintended side effects.
- **O**pen/Closed Principle: Code is structured in a way that allows new features to be added through extension rather than by modifying existing functionality.
- **L**iskov Substitution Principle: Components can be replaced with their subtypes without affecting the correctness or behavior of the system.
- **I**nterface Segregation Principle: Interfaces are designed to include only the methods relevant to the specific behavior needed, avoiding unnecessary dependencies.
- **D**ependency Inversion Principle: High-level and low-level components interact through abstractions, rather than concrete implementations, improving modularity and flexibility.

These principles guided the design of the benchmarking codebase to ensure each component has a clearly defined role and can be extended independently without modifying existing code.

4.2.1 Modular Class Design

The *Problem Classes Diagram* (Figure 4.1) outlines the modular structure for defining benchmarking problems. At the foundation, we have an abstract `DataProvider` class, which specifies two essential methods: `get_data()` and `get_default_transform()`.

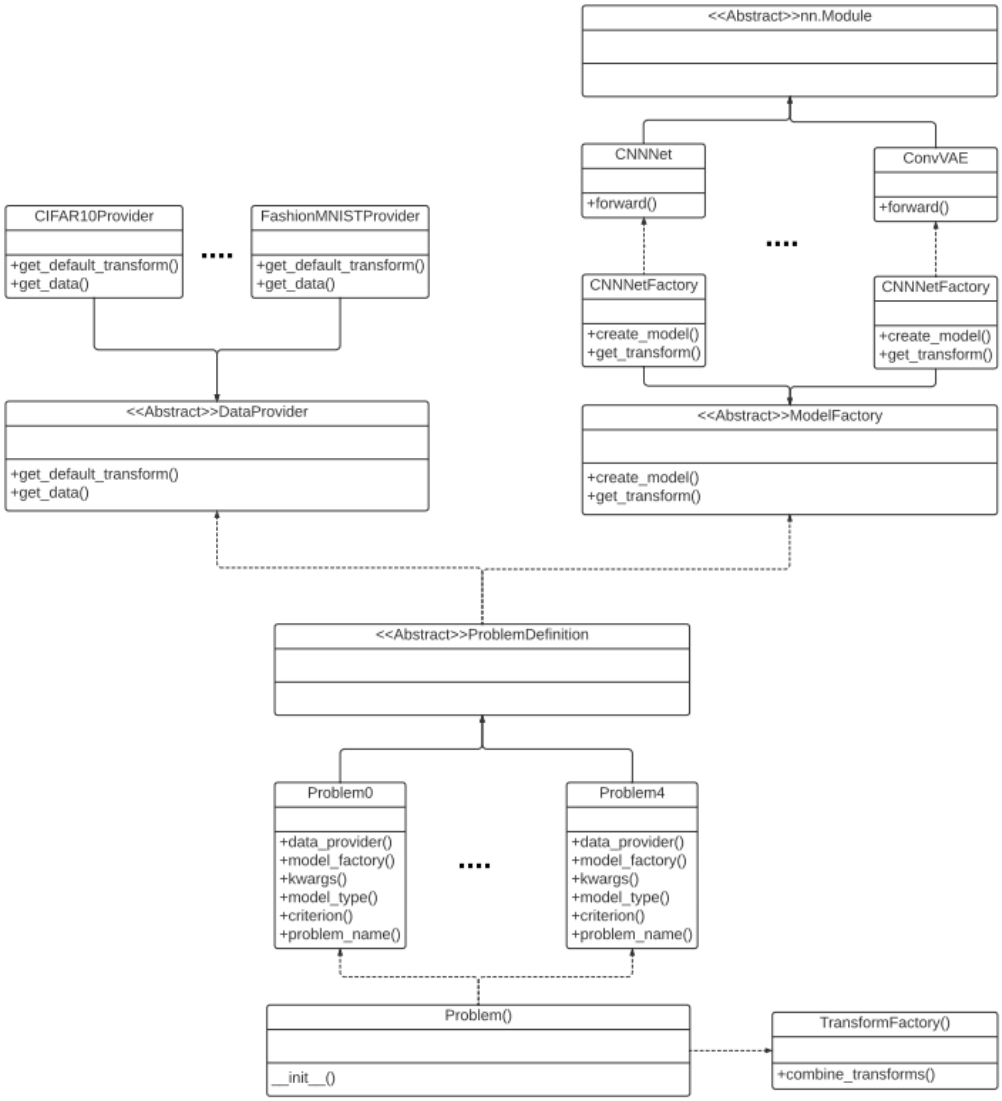


Figure 4.1: Problem Classes Diagram

These define how the dataset is loaded and what default preprocessing should be applied. Each new dataset (e.g., CIFAR10, FashionMNIST) is implemented as a subclass that overrides these methods accordingly.

Next, the `ModelFactory` is also an abstract class that provides two methods: `create_model()` and `get_transform()`. The latter returns model-specific transformation requirements, if any; otherwise, it returns `None`. Finally, the `ProblemDefinition` abstract class brings together a specific model and dataset configuration. Each benchmark problem is implemented as a subclass of this definition, specifying the model factory and data provider it depends on. It also uses a `TransformFactory` class, which has a method that combines the transformation logic from both the data and model sources to produce a consistent preprocessing pipeline.

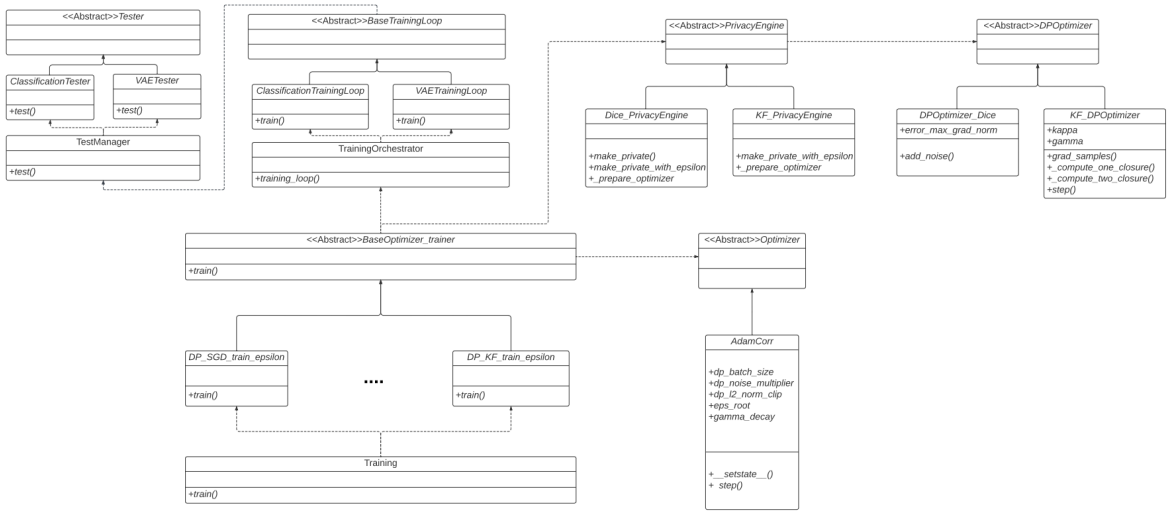


Figure 4.2: Training Classes Diagram

The *Training Classes Diagram* (Figure 4.2) represents a more complex component of the codebase. First, we introduce a new optimizer, `AdamBC`, which extends the existing `Optimizer` base class. Since it adheres to the expected interface, it can be directly combined with Opacus's standard `PrivacyEngine`, without requiring a custom privacy mechanism.

For other new DP optimizers, we implemented new optimizers by subclassing the `DPOptimizer` class from Opacus. These custom optimizers required new corresponding privacy engines, each of which inherits from Opacus's base `PrivacyEngine` class. These privacy engines are tightly coupled with their respective `DPOptimizer` implementations, ensuring proper noise injection and gradient manipulation.

Testing and training are abstracted as well: a base `Tester` class supports different evaluation strategies, with subclasses for specific task types such as `ClassificationTester` and `VAETester`. Training logic is encapsulated in the `BaseTrainingLoop` class, which integrates testing into its reporting flow. It is subclassed by problem-specific implementations like `ClassificationTrainingLoop` and `VAETrainingLoop`.

The orchestration of training is handled by the `TrainingOrchestrator` class. This component dynamically selects the appropriate training loop based on the problem definition, using a registry-based mechanism. It depends on the base training loop class and its children to execute the correct training logic.

To facilitate structured trainer design, a `BaseOptimizerTrainer` class is introduced. This class depends on the `TrainingOrchestrator`, the appropriate privacy engine, and

the specific optimizer. For each new DP-optimizer, a corresponding trainer subclass is created. This trainer coordinates the training process by invoking the orchestrator, applying the privacy engine, and using the correct optimizer (e.g., standard SGD for several matrix-based DP optimizers).

Finally, to abstract this complex composition from the user, a high-level `Training` class is provided. This class relies on the trainer registry and selects the appropriate trainer based on a user-specified DP optimizer name, offering a simple and unified interface for launching training jobs.

4.2.2 Extensibility and Integration

This architecture supports:

- Seamless integration of new datasets, models, and optimizers via subclassing.
- Modular benchmarking and evaluation workflows.
- Clear separation of concerns, aiding reproducibility and testing.

Overall, the design facilitates robust experimentation in DP and optimizer benchmarking, while maintaining a clean and extensible codebase.

4.3 Integrating DP Optimizers into PyTorch

In this section, we describe how we implemented and integrated several differentially private (DP) optimizers in PyTorch by building on top of the `Opacus` library [YSS+21]. Opacus provides core functionalities for privacy accounting, per-sample gradient computation, and noise injection. Among the optimizers in our benchmark, `DP-SGD` is already fully supported by Opacus and requires no modification. Our main contribution in this section is extending the framework to include a variety of recent DP optimizers that are not natively available in Opacus.

For `DP-AdamBC` (see Algorithm 2), we adopt the official implementation* released by the authors of [TSL24]. We wrap this implementation to interface cleanly with our Opacus-based training pipeline without making modifications to the core optimizer logic.

*https://github.com/ubc-systopia/DP-AdamBC

The `DP-DiSk` (see Algorithm 4) optimizer is implemented in the `Opacus research` module*. While we largely retain the original implementation, we found the handling of the noise multiplier, specifically the line† `noise_multiplier = noise_multiplier / norm_factor`, to be problematic. This normalization reduces the effective noise added and weakens the intended privacy guarantees. Moreover, this adjustment is not present in the theoretical formulation of the method as described in [ZBB+25]. In our implementation, we remove this normalization and use the unadjusted noise multiplier, while keeping all other components of the optimizer intact.

For `DP-Dice`, we port the publicly available JAX implementation‡ to PyTorch and integrate it with Opacus. Our implementation follows the theoretical algorithm described in [ZBWH24] (see Algorithm 3), including the error feedback update step $e_{t+1} \leftarrow e_t + g_t - \bar{g}_t$, and omits the additional clipping-based adjustment present in the reference implementation§ to stay faithful to the original formulation. In addition, there is no explicit definition of the sensitivity used in the numerical privacy accounting. We follow the expression used in the original reference code and rely on Opacus for privacy accounting and gradient management, though we note that this choice may underestimate the noise required for differential privacy guarantees.

For `DP-Matrix`, we implement the matrix-factorized optimization mechanism described in Section 2.2.5 by converting the official JAX and TensorFlow implementations to PyTorch. Specifically, we port the fixed-point solver for the single-epoch case from the Google Research Federated repository¶, and the dual optimization method for multi-epoch training from its multi-epoch counterpart‖. This unifies all matrix-based optimizers under the same PyTorch framework for consistent evaluation. Additionally, we extend both matrix factorization methods to support the convergence-aware variant described in method 3 in section 2.2.5. This is done by replacing the workload matrix A with the reweighted version $\Lambda_\tau A$, which integrates seamlessly into the existing optimization routines. Our implementation supports this extension in both the fixed-point solver for single-epoch training and the dual optimization strategy for multi-epoch settings. This enables convergence-aware matrix mechanisms to be applied without requiring structural changes to the underlying algorithms.

*https://github.com/pytorch/opacus/tree/main/research/disk_optimizer

†https://github.com/pytorch/opacus/blob/main/research/disk_optimizer/optimizers/KFoptimizer.py#L56

‡https://github.com/564612540/DiceSGD/blob/master/DiceSGD

§https://github.com/564612540/DiceSGD/blob/master/DiceSGD/optimizers_utils.py#L95

¶https://github.com/google-research/federated/blob/master/dp_matrix_factorization

‖https://github.com/google-research/federated/tree/master/multi_epoch_dp_matrix_factorization

To integrate the matrix mechanism into the Opacus pipeline, we modify the optimizer to apply structured, correlated noise generated offline. Specifically, we precompute a noise matrix Z and compute the product $BZ \in \mathbb{R}^{T \times d}$, where each row represents the cumulative correlated noise up to that step. Since the noise at step t implicitly includes contributions from all prior steps, we subtract the previously applied noise before injecting the new one. This ensures that the update at each step only reflects the incremental noise needed. This approach preserves the privacy guarantees of the matrix mechanism while remaining compatible with Opacus's per-step gradient manipulation framework. For testing purposes, we also used a lower-triangular matrix of ones for B, which produces the same behavior as DP-SGD. This provided a useful validation check, and the resulting optimizer behavior matched that of standard DP-SGD exactly.

To enable efficient matrix multiplication of the BZ product at larger scales, we implement a custom GPU-accelerated matrix multiplication routine that handles large $T \times T$ and $T \times d$ matrices by tiling them into smaller chunks. This approach streams tiles from CPU memory to the GPU, performs partial multiplications, and writes the results incrementally to disk using memory-mapped arrays. This significantly reduces GPU memory pressure and allows us to compute the full BZ product even when T is large, making matrix-based DP optimization practical on standard hardware.

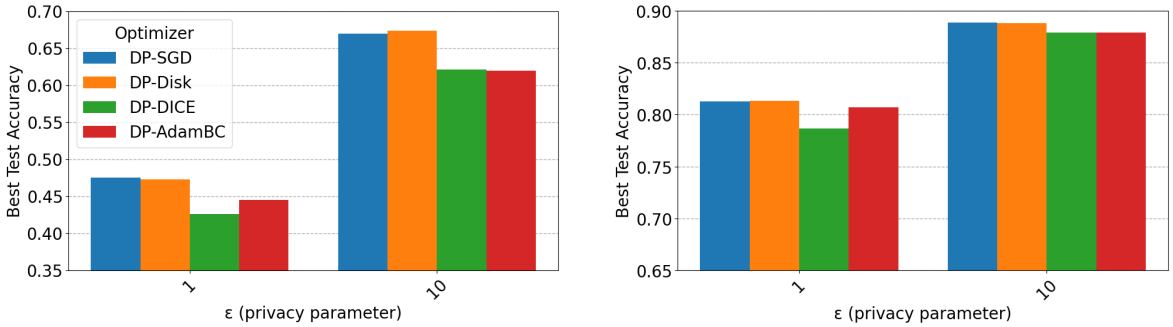
5. Results

In this chapter, we present the results of our experimental benchmarks across the three setups described in the methodology. Our goal is to evaluate and compare the behavior of differentially private (DP) optimizers across a variety of tasks and training conditions.

5.1 Optimizer Comparisons

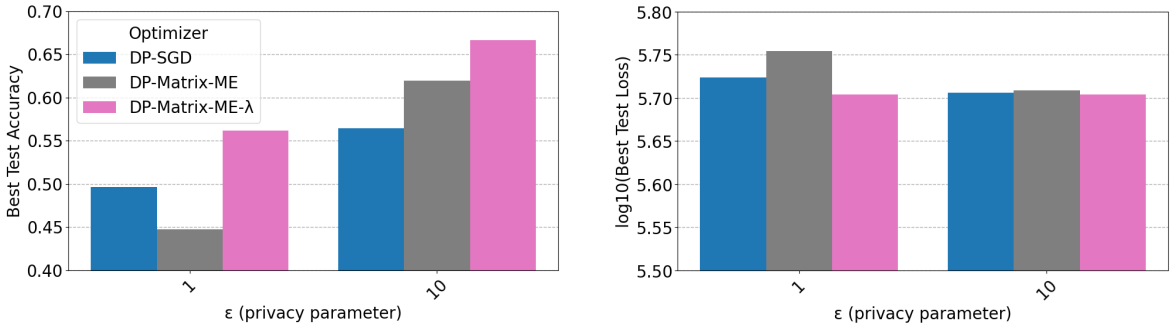
Across Setup 1, in Figures 5.1 and 5.4, we find that DP-SGD consistently delivers the best overall performance across privacy levels and hyperparameter settings, with DP-DiSK often matching its accuracy. DP-AdamBC performs reasonably well, though not as strongly as DP-SGD or DP-DiSK, while DP-DICE shows the lowest accuracy.

Across both the single-epoch (see Setup 2) and multi-epoch (see Setup 3) experiments, in Figure 5.2, we find that matrix-mechanism-based optimizers can offer utility improvements over `DP-SGD`, particularly when privacy constraints are relaxed. In the single-epoch setting, both the fixed-point variant (`DP-Matrix-SE`) and the convergence-aware version (`DP-Matrix-SE-`λ) outperform `DP-SGD` at $\epsilon = 10$, with the fixed-point variant achieving the highest accuracy overall, while `DP-SGD` remains stronger at $\epsilon = 1$. In contrast, the multi-epoch results show that the convergence-aware variant (`DP-Matrix-ME-`λ) consistently outperforms both `DP-SGD` and `DP-Matrix-ME`, including in high-noise regimes like $\epsilon = 1$. Notably, `DP-Matrix-ME` does not compete with `DP-SGD` at $\epsilon = 1$, and while it shows better utility in CIFAR-10 classification under relaxed privacy, it exhibits slightly higher loss than `DP-SGD` on the FashionMNIST generation task even in low-noise settings.

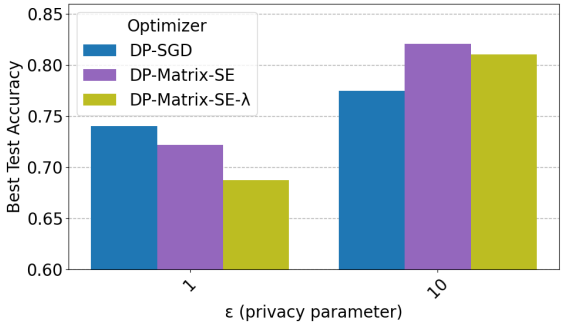


(a) CIFAR-10 **(b)** Fashion-MNIST

Figure 5.1: Setup 1: Non-matrix optimizers comparison: for each optimizer, we plot the highest accuracy achieved across all tested hyperparameter settings. DP-SGD consistently achieves the highest accuracy, while DP-DiSK remains nearly identical across both privacy regimes.



(a) CIFAR-10 (30 epochs) **(b)** Fashion-VAE (30 epoch)



(c) Fashion-MNIST (single epoch)

Figure 5.2: Setup 2 & 3: Matrix mechanism optimizer comparison. For each optimizer, we plot the best observed utility across all tested hyperparameter settings. Subplots (a) and (c) report accuracy, while (b) shows log loss. In the multi-epoch runs (a, b), DP-Matrix-ME-λ consistently outperforms DP-Matrix-ME and DP-SGD across all ϵ values. In the single-epoch setting (c), DP-SGD leads under strict privacy (low ϵ), while both matrix-based methods surpass it as privacy is relaxed.

5.2 Non-Matrix Hyperparameter Analysis

Setup 1 is designed to benchmark DP optimizers across a comprehensive hyperparameter grid on two representative image classification tasks: CIFAR-10 with a 3c3d CNN (P1) and Fashion-MNIST with a 2c2d CNN (P2). It includes evaluations of DP-SGD, DP-AdamBC, DP-DiSK, and DP-Dice.

Seeds Sentivity Analysis To evaluate the consistency of training outcomes under different random seeds, we use Setup 1 to conduct a seed sensitivity analysis across two problems (CIFAR-10 and Fashion-MNIST) and two optimizers (DP-SGD and DP-AdamBC). For each learning rate, we identify the best-performing hyperparameter configuration (excluding seed) and report the mean and standard deviation of test accuracy across all seeds. This isolates the variability due solely to random initialization. As shown in Figure 5.3, the standard deviation across seeds is consistently small for all learning rates, indicating low sensitivity to random initialization and suggesting that reliable hyperparameter configurations can be identified without requiring repeated runs across seeds. Based on these results, we observe that random initialization has minimal impact on test performance, so we do not run additional seeds in the rest of this study. However, while these findings hold for DP-SGD and DP-AdamBC, we caution that this behavior may not generalize to all optimizers or tasks.

CIFAR-10 For this section, we used a fixed random seed and evaluated performance at two privacy levels ($\epsilon = 1$ and 10), allowing a fair comparison across optimizers, including DP-DiSK and DP-Dice, which were run under limited compute. The key trends discussed here also hold for the Fashion-MNIST task (see Appendix B).

Figure 5.4 shows test accuracy across hyperparameter settings for each optimizer at $\epsilon = 1$ and $\epsilon = 10$. `DP-SGD` performs consistently well, reaching top accuracy across a wide range of settings. `DP-DiSK` closely matches `DP-SGD` at both privacy levels, with nearly identical best accuracies and stable performance. `DP-AdamBC` also performs well, achieving high accuracy across several learning rates, though with slightly lower peak values. `DP-Dice` shows the weakest results, with lower accuracy across all settings. For DP-SGD, DP-DiSK, and DP-Dice, the best accuracy at $\epsilon = 1$ is achieved at a learning rate of either 0.01 or 0.1, depending on the specific setting. In contrast, for $\epsilon = 10$, all three consistently perform best at a learning rate of 1.0. DP-AdamBC follows a different trend, with the best accuracy occurring at 10^{-5} when $\epsilon = 1$, and shifting to 10^{-4} at $\epsilon = 10$. These results confirm a general shift toward higher optimal learning

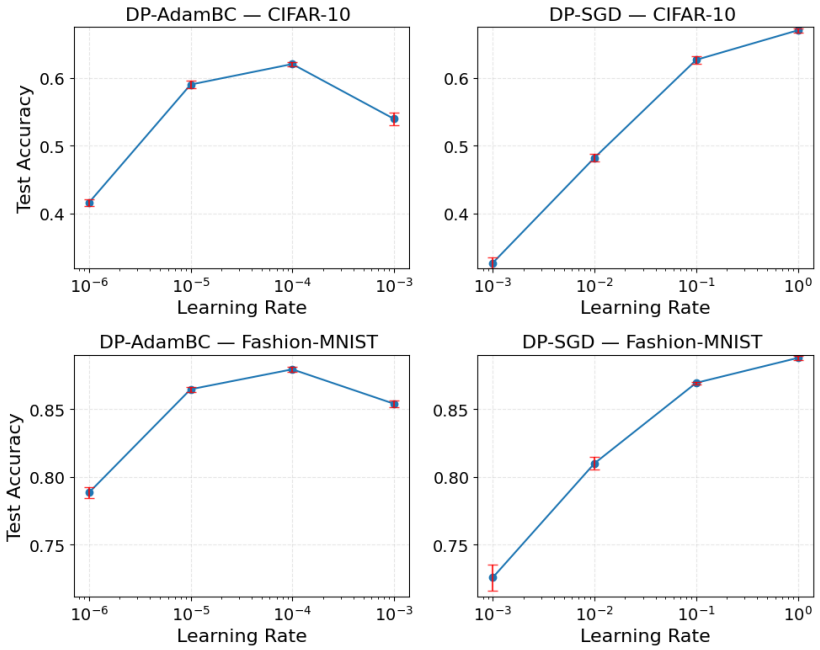


Figure 5.3: Seed sensitivity of test accuracy across learning rates for each optimizer (columns) and problem (rows), using the best hyperparameters per learning rate. Results show minimal variance, indicating low sensitivity to random initialization.

rates under weaker privacy across most optimizers.

Figure 5.4 shows that increasing the number of training epochs from 250 to 500 does not consistently improve performance. In many cases, the best accuracy is already reached at 250 epochs, and using 500 epochs yields similar results. Among the best-performing configurations, 250 epochs is always either equal to or slightly better than 500, except for DP-AdamBC at $\epsilon = 1$, where 500 epochs yields a modest improvement of about 1 percentage point. While in a few non-optimal hyperparameter settings, 500 epochs leads to larger gains, the improvements remain modest, typically no more than 3–4 percentage points. In figure 5.5, We also observe that the optimal number of epochs often changes depending on the batch size, and this relationship varies across privacy levels and optimizers. However, there is no consistent or monotonic trend between batch size and the preferred number of epochs, further suggesting that these two hyperparameters should be tuned jointly rather than independently.

The relationship between batch size and testing accuracy depends strongly on the learning rate, as shown in Figures 5.4 and 5.5. When using smaller learning rates, smaller batch sizes tend to give better results across all optimizers. One intuitive explanation is that small batch sizes introduce higher gradient noise (i.e., higher variance), so smaller learning rates help take more cautious steps and avoid instability. However, as the learning rate increases, this pattern reverses: larger batch sizes start to perform better.

Figure 5.4: Setup 1: Heatmaps showing the best test accuracy for each optimizer on the CIFAR-10 problem, across learning rates (x-axis) and either batch size or number of epochs (y-axis), at $\epsilon = 1$ (top row) and $\epsilon = 10$ (bottom row). Each cell reports the best accuracy found over all other hyperparameters. Red boxes mark the highest accuracy per optimizer. DP-SGD consistently achieves the best results, followed by DP-DiSK, then DP-AdamBC, with DP-Dice performing worst.

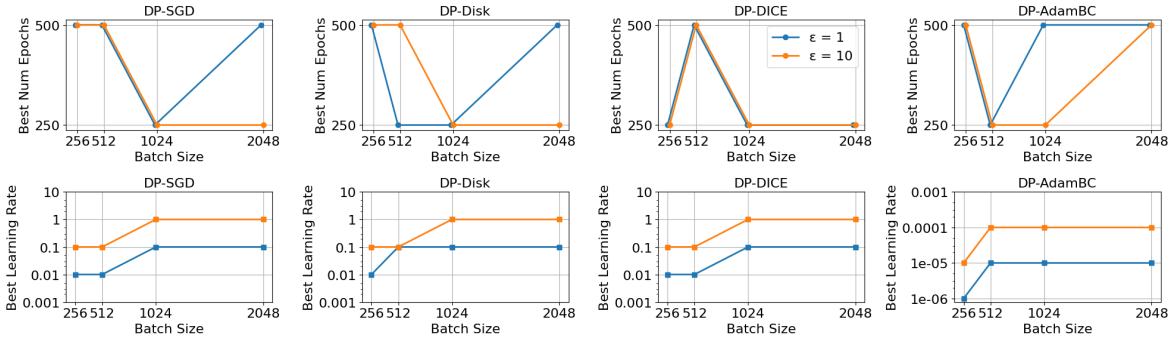


Figure 5.5: Best number of epochs (top row) and learning rate (bottom row) selected for each optimizer at different batch sizes and privacy levels. No consistent pattern is observed between batch size and the optimal number of epochs, but the optimal number of epochs often changes as the batch size varies. In contrast, a clearer trend emerges for learning rate: smaller batch sizes tend to perform best at lower learning rates, while larger batch sizes benefit from higher learning rates.

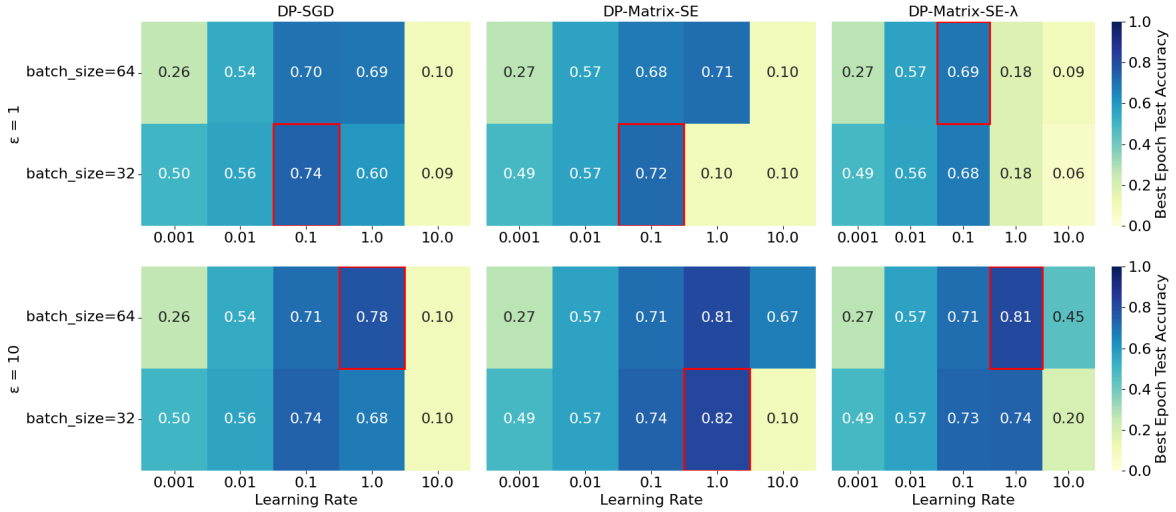


Figure 5.6: Setup 2: Heatmaps showing the best test accuracy for each optimizer on the Fashion-MNIST problem, across learning rates (x-axis) and batch sizes (y-axis), at $\epsilon = 1$ (top row) and $\epsilon = 10$ (bottom row). Each cell reports the best accuracy found over all other hyperparameters. Red boxes mark the highest accuracy per optimizer. While DP-SGD achieves the highest accuracy at $\epsilon = 1$, matrix mechanism-based optimizers outperform DP-SGD when the privacy budget is relaxed ($\epsilon = 10$).

On the CIFAR-10 task, we find that DP-SGD consistently delivers the best overall performance across privacy levels and hyperparameter settings, followed closely by DP-DiSK. DP-AdamBC performs moderately well but lags behind the top two, while DP-DICE shows the least stability and lowest accuracy. Across all optimizers, we observe that the best learning rate increases with the privacy budget, batch size interacts strongly with learning rate, and 250 epochs are often sufficient to reach peak accuracy. It is also important to note that the optimal batch size can differ depending on the number of epochs used, so these hyperparameters should be tuned jointly. Importantly, these same trends are also observed in the Fashion-MNIST task (see Appendix B), confirming the generality of our findings across datasets.

5.3 Single-Epoch Matrix Hyperparameter Analysis

Setup 2 evaluates matrix-mechanism-based optimizers against DP-SGD in a controlled single-epoch setting on the Fashion-MNIST classification task. It compares two matrix methods, one using a fixed-point factorization (`DP-Matrix-SE`), and the other incorporating a convergence-aware formulation (`DP-Matrix-SE-`λ), to assess whether theoretical benefits from structured noise injection translate into practical accuracy gains. Since the experiment is limited to a single epoch, we use small batch sizes to

increase the number of update steps, following the same design choice used in prior matrix mechanism work [KMC+23].

Figure 5.6 shows the best test accuracy across learning rates for each optimizer under both $\epsilon = 1$ and $\epsilon = 10$, with batch sizes 32 and 64. At $\epsilon = 10$, both matrix-mechanism variants outperform DP-SGD, with `DP-Matrix-SE` achieving the highest overall accuracy, followed by the `DP-Matrix-SE-`λ variant. At $\epsilon = 1$, DP-SGD slightly leads overall, although the gap is small in several configurations.

Across all optimizers, we observe consistent trends in the optimal learning rate: at $\epsilon = 1$, the best results occur at a learning rate of 0.1, while at $\epsilon = 10$, the optimum shifts to 1.0. At $\epsilon = 1$, `DP-Matrix-SE` slightly outperforms DP-SGD in some settings, but DP-SGD still achieves the highest test accuracy overall, about 2 percentage points higher than the matrix-based method. In contrast, `DP-Matrix-SE-`λ performs noticeably worse at this privacy level. At $\epsilon = 10$, both matrix optimizers improve significantly, with `DP-Matrix-SE` outperforming DP-SGD by up to 4 percentage points, and `DP-Matrix-SE-`λ also showing competitive performance. These results suggest that matrix-based methods can offer clear utility benefits in low-noise settings, while DP-SGD remains stronger under stricter privacy.

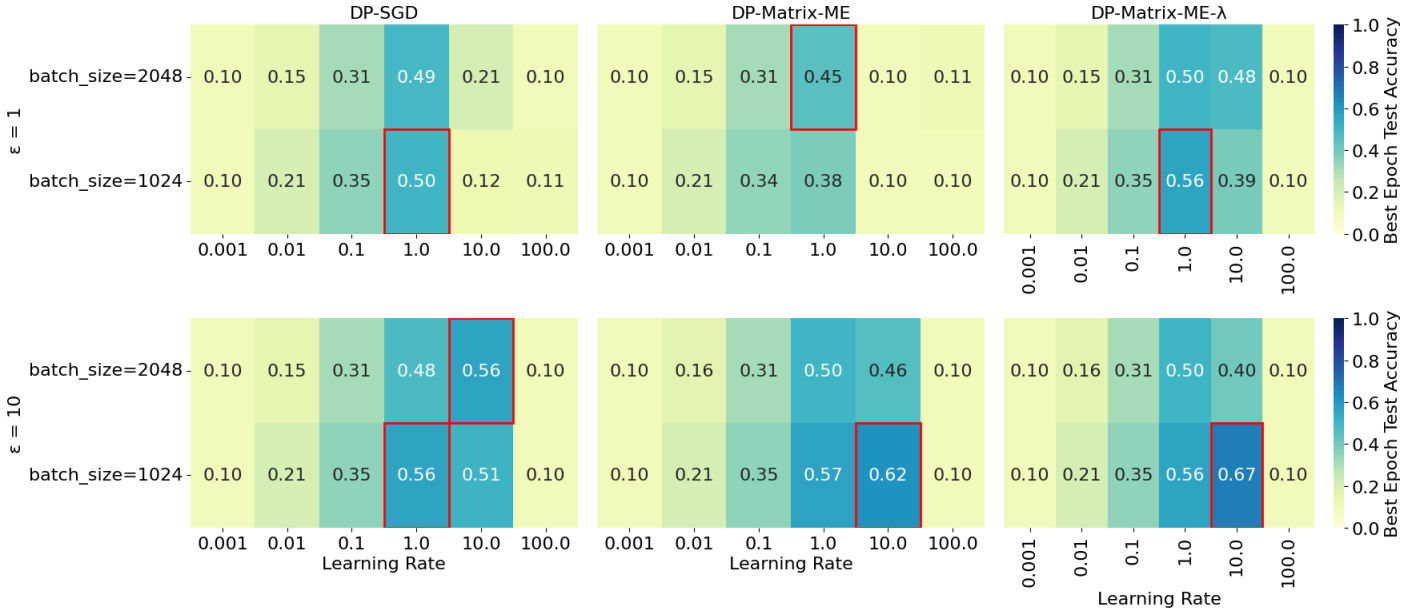


Figure 5.7: Setup 3: Heatmaps showing the best test accuracy for each optimizer on the CIFAR-10 problem, across learning rates (x-axis) and batch sizes (y-axis), at $\epsilon = 1$ (top row) and $\epsilon = 10$ (bottom row). Each cell reports the best accuracy found over all other hyperparameters. Red boxes mark the highest accuracy per optimizer. DP-Matrix-ME-λ consistently outperforms DP-SGD across both privacy levels, demonstrating the benefits of convergence-aware matrix mechanisms in multi-epoch training.

5.4 Multi-Epoch Matrix Hyperparameter Analysis

This setup evaluates matrix-mechanism-based optimizers in the multi-epoch training regime, where each sample may participate in multiple steps. Unlike the single-epoch formulation used in Setup 2, we now apply the dual optimization framework described in the theory section (see Section 2.2.5), which accommodates repeated participation through structured sensitivity accounting. Specifically, we compare two matrix methods: `DP-Matrix-ME`, which uses the core dual-based factorization, and `DP-Matrix-ME-`λ, a convergence-aware variant that introduces a temporal weighting matrix Λ_τ to prioritize noise directions based on their influence on convergence. These methods are benchmarked against DP-SGD to assess whether the theoretical advantages of structured, correlated noise translate into practical utility gains under realistic multi-epoch privacy constraints.

To keep the comparison tractable, this setup is evaluated on two benchmark tasks – CIFAR-10 classification (P1), and Fashion-MNIST generation (P3) – under a controlled hyperparameter sweep. We vary learning rates, batch sizes (1024, 2048), and privacy levels ($\epsilon = 1, 10$), and fix the number of training epochs to 30. This controlled setup allows us to assess whether matrix-based methods scale effectively in multi-epoch settings and whether their theoretical benefits hold under varied privacy constraints.

CIFAR-10 Figure 5.7 shows the test accuracy of DP-SGD and two matrix-mechanism variants on CIFAR-10 in the multi-epoch setting. Across both privacy levels, `DP-Matrix-ME-`λ consistently outperforms the other optimizers. At $\epsilon = 1$, it achieves the highest accuracy (0.56), followed by DP-SGD (0.50) and `DP-Matrix-ME` (0.45). At $\epsilon = 10$, both matrix methods outperform DP-SGD, with `DP-Matrix-ME-`λ reaching 0.67 and `DP-Matrix-ME` achieving 0.62, while DP-SGD peaks at 0.56. Optimal learning rates shift with privacy level: for all optimizers, the best performance is obtained at LR = 1.0 under $\epsilon = 1$, and at LR = 10.0 under $\epsilon = 10$. These results indicate that structured matrix noise not only scales well to multi-epoch training but can consistently improve accuracy, especially when the Λ_τ-based weighting is used, and unlike in the single-epoch setting, it offers benefits even under high privacy (i.e., $\epsilon = 1$).

Fashion-MNIST Generation In this task, we evaluate matrix-mechanism-based optimizers on a generative modeling problem using a variational autoencoder (VAE) [KW14] trained on Fashion-MNIST. Unlike classification tasks, this is an unsupervised setting, and our focus is on optimizer utility, measured by the total test loss

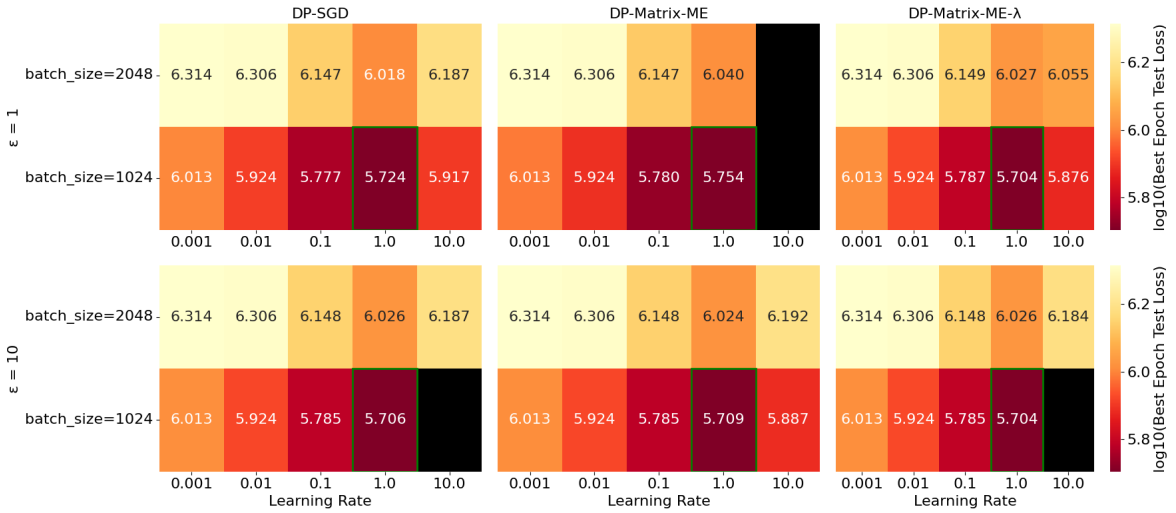


Figure 5.8: Setup 3: Heatmaps showing the best test loss for each optimizer on the Fashion-MNIST generation problem, across learning rates (x-axis) and batch sizes (y-axis), at $\epsilon = 1$ (top row) and $\epsilon = 10$ (bottom row). Each cell reports the best loss found over all other hyperparameters. Red boxes mark the lowest test loss per optimizer. Black boxes indicate configurations that resulted in infinite loss. DP-Matrix-ME-λ consistently outperforms DP-SGD across both privacy levels, demonstrating the benefits of convergence-aware matrix mechanisms in multi-epoch training.

of the variational autoencoder (VAE), which includes both reconstruction error and a regularization term based on Kullback-Leibler (KL) divergence [KW14]. This setup allows us to assess whether structured noise injection leads to better loss minimization under DP in generative models.

Figure 5.8 reports test losses in $\log_{10}$ scale. The heatmaps show the best test loss (i.e., the minimum loss achieved at any epoch) for each optimizer across different learning rates, privacy levels, and batch sizes. At $\epsilon = 1$, `DP-Matrix-ME-`λ achieves the lowest test loss, outperforming both `DP-SGD` and `DP-Matrix-ME`, which rank second and third respectively. A similar pattern holds at $\epsilon = 10$, though the performance gap between the optimizers becomes smaller. These results align with the theoretical motivation behind the convergence-aware variant (`DP-Matrix-ME-`λ), where their tighter, convergence-aware approach—using a temporal weighting matrix—can lead to improved utility under DP.

6. Discussion

Our primary goal was to evaluate how different DP optimizers perform in practice by comparing their utility to our strong baseline DP-SGD. In particular, we aimed to identify whether and when alternative optimizers offer practical improvements over DP-SGD under various training conditions and privacy budgets.

Optimizers compared to DP-SGD We begin our discussion by reaffirming that DP-SGD is the most effective non-matrix optimizer across all evaluated privacy levels and training conditions. Given its consistent and robust performance, we see no reason to consider alternative methods such as DP-AdamBC, DP-DICE, or DP-DiSK, which offer no tangible benefits in practice.

In the single-epoch setting, matrix mechanism-based optimizers align well with theoretical expectations from matrix mechanism literature. Prior work [PUC+25] suggest that in the high-noise regime (i.e., small ϵ), DP-SGD benefits significantly from privacy amplification via sampling and typically performs better. In contrast, correlated noise mechanisms such as matrix methods do not benefit from amplification in the same way and often underperform in this regime. Our results confirm this: DP-SGD achieves higher accuracy than matrix-based methods at $\epsilon = 1$. However, as the privacy constraint relaxes and noise decreases (e.g., $\epsilon = 10$), the theoretical advantage of correlated noise becomes apparent. In this setting, matrix mechanisms outperform DP-SGD, suggesting that structured noise injection can offer better utility in low-noise regimes where amplification is less effective or infeasible.

In the multi-epoch setting, the non-convergence-aware matrix mechanism (DP-Matrix-ME) follows prior theoretical expectations from matrix mechanism literature [PUC+25]. In contrast, the convergence-aware variant (DP-Matrix-ME-λ) outperforms DP-SGD in both low-noise and high-noise regimes. While this matches recent theoretical predictions under relaxed privacy, it goes beyond what has been demonstrated in prior work [PUC+25] for the high-noise regime, where correlated noise

is typically less effective. This suggests that incorporating convergence awareness into the noise structure may substantially improve the utility of matrix mechanisms even when privacy constraints are tight, highlighting their potential as practical alternatives to `DP-SGD` in multi-epoch training, provided that GPU memory is sufficient to support the matrix factorization required for the given number of training steps and model size (see Section 4.3 and Setup 3).

Hyperparameter Discussion Our findings support trends observed in prior work. We observe that increasing the number of training epochs has limited effect on final model accuracy, consistent with Morsbach et al. [MRS24], who report that epoch count plays a minor role in DP-SGD performance. Additionally, we confirm that batch size and learning rate interact strongly: smaller batch sizes perform better at lower learning rates, while larger batch sizes require higher learning rates, aligning with observations from De et al. [DBH+22]. These results reinforce the importance of tuning batch size and learning rate jointly, and suggest that simply increasing the number of training epochs may offer limited benefit in DP settings.

Limitations and Future Work This study focuses on two task types, classification and generation, suggesting that future work could explore additional problems such as natural language processing. Moreover, matrix mechanisms impose significant GPU memory demands, particularly in multi-epoch settings, which limited our ability to scale to longer training runs. In addition, while we evaluated three representative matrix-based methods, several recent variants were not included and should be benchmarked in future work. Finally, limited computing resources constrained our ability to explore broader domains such as federated learning, additional datasets, varied architectures, and pretrained models, all of which may exhibit different optimization dynamics and represent promising directions for future investigation.

7. Conclusion

This thesis introduced a unified benchmarking framework for evaluating differentially private (DP) optimizers under consistent experimental conditions. We applied this framework to a set of recent DP optimizers, including non-matrix methods such as DP-Dice, DP-DiSK, and DP-AdamBC, as well as matrix-mechanism-based optimizers adapted for both single and multi-epoch training. All optimizers were compared against DP-SGD.

Our goal was to assess whether alternative optimizers could improve utility under differential privacy constraints. We found that while `DP-SGD` remains the most effective among non-matrix methods, convergence-aware matrix mechanism variant (`DP-Matrix-ME-`λ) consistently outperformed it in multi-epoch training. Other matrix-based approaches, across both single-epoch and multi-epoch settings, followed expected trends, performing better than `DP-SGD` under relaxed privacy but worse under strict privacy. These results suggest that matrix mechanisms are a strong choice under relaxed privacy constraints, while the convergence-aware variant shows promise even in strict privacy settings, outperforming `DP-SGD` in multi-epoch training.

Future work could benchmark more recent matrix mechanism variants not covered here and investigate how pretrained models behave under different DP optimizers. The framework could also be extended to new domains such as natural language processing or federated learning. Additionally, scaling matrix mechanisms remains challenging due to GPU memory limitations during matrix construction; addressing these constraints is essential to make such methods practical for larger models and broader deployment.

Bibliography

[ACG+16] Martín Abadi, Andy Chu, Ian J. Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 308–318. ACM, 2016.

[AHK18] Karim Abouelmehdi, Abderrahim Beni Hssane, and Hayat Khaloufi. Big healthcare data: preserving security and privacy. *J. Big Data*, 5:1, 2018.

[BCN18] Léon Bottou, Frank E. Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *SIAM Rev.*, 60(2):223–311, 2018.

[BFTT19] Raef Bassily, Vitaly Feldman, Kunal Talwar, and Abhradeep Guha Thakurta. Private stochastic convex optimization with optimal rates. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 11279–11288, 2019.

[Bot10] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In Yves Lechevallier and Gilbert Saporta, editors, *19th International Conference on Computational Statistics, COMPSTAT 2010, Paris, France, August 22-27, 2010 - Keynote, Invited and Contributed Papers*, pages 177–186. Physica-Verlag, 2010.

[BST14] Raef Bassily, Adam D. Smith, and Abhradeep Thakurta. Private empirical risk minimization: Efficient algorithms and tight error bounds. In *55th IEEE Annual Symposium on Foundations of Computer Science,*

FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014, pages 464–473. IEEE Computer Society, 2014.

[BW18] Borja Balle and Yu-Xiang Wang. Improving the Gaussian mechanism for differential privacy: Analytical calibration and optimal denoising. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 403–412. PMLR, 2018.

[CGM+23] Christopher A. Choquette-Choo, Arun Ganesh, Ryan McKenna, H. Brendan McMahan, John Rush, Abhradeep Guha Thakurta, and Zheng Xu. (amplified) banded matrix factorization: A unified approach to private training. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023.

[CLE+19] Nicholas Carlini, Chang Liu, Úlfar Erlingsson, Jernej Kos, and Dawn Song. The secret sharer: Evaluating and testing unintended memorization in neural networks. In Nadia Heninger and Patrick Traynor, editors, *28th USENIX Security Symposium, USENIX Security 2019, Santa Clara, CA, USA, August 14-16, 2019*, pages 267–284. USENIX Association, 2019.

[CMRT23] Christopher A. Choquette-Choo, Hugh Brendan McMahan, J. Keith Rush, and Abhradeep Guha Thakurta. Multi-epoch matrix factorization mechanisms for private machine learning. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 5924–5963. PMLR, 2023.

[CMS11] Kamalika Chaudhuri, Claire Monteleoni, and Anand D. Sarwate. Differentially private empirical risk minimization. *J. Mach. Learn. Res.*, 12:1069–1109, 2011.

[DBH+22] Soham De, Leonard Berrada, Jamie Hayes, Samuel L. Smith, and Borja Balle. Unlocking high-accuracy differentially private image classification through scale. *CoRR*, abs/2204.13650, 2022.

[DKM+06] Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. Our data, ourselves: Privacy via distributed noise generation. In Serge Vaudenay, editor, *Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings*, volume 4004 of *Lecture Notes in Computer Science*, pages 486–503. Springer, 2006.

[DMNS06] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam D. Smith. Calibrating noise to sensitivity in private data analysis. In Shai Halevi and Tal Rabin, editors, *Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006, Proceedings*, volume 3876 of *Lecture Notes in Computer Science*, pages 265–284. Springer, 2006.

[DMP+24] Krishnamurthy Dj Dvijotham, H. Brendan McMahan, Krishna Pillutla, Thomas Steinke, and Abhradeep Thakurta. Efficient and near-optimal noise generation for streaming differential privacy. In *65th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2024, Chicago, IL, USA, October 27-30, 2024*, pages 2306–2317. IEEE, 2024.

[DMR+22] Sergey Denisov, H. Brendan McMahan, John Rush, Adam D. Smith, and Abhradeep Guha Thakurta. Improved differential privacy for SGD via optimal private linear operators on adaptive streams. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022.

[DR14] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9(3-4):211–407, 2014.

[DRS19] Jinshuo Dong, Aaron Roth, and Weijie J. Su. Gaussian differential privacy. *CoRR*, abs/1905.02383, 2019.

[Dwo06] Cynthia Dwork. Differential privacy. In *Automata, Languages and Programming, 33rd International Colloquium, ICALP 2006, Venice, Italy,*

July 10-14, 2006, Proceedings, Part II, volume 4052 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2006.

[Dwo11] Cynthia Dwork. A firm foundation for private data analysis. *Commun. ACM*, 54(1):86–95, 2011.

[EFM+19] Úlfar Erlingsson, Vitaly Feldman, Ilya Mironov, Ananth Raghunathan, Kunal Talwar, and Abhradeep Thakurta. Amplification by shuffling: From local to central differential privacy via anonymity. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 2468–2479. SIAM, 2019.

[ENU20] Alexander Edmonds, Aleksandar Nikolov, and Jonathan R. Ullman. The power of factorization mechanisms in local and central differential privacy. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 425–438. ACM, 2020.

[FMT21] Vitaly Feldman, Audra McMillan, and Kunal Talwar. Hiding among the clones: A simple and nearly optimal analysis of privacy amplification by shuffling. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 954–964. IEEE, 2021.

[HMvdW+20] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[HT10] Moritz Hardt and Kunal Talwar. On the geometry of differential privacy. In Leonard J. Schulman, editor, *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 705–714. ACM, 2010.

[KB15] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[KJPH21] Antti Koskela, Joonas Jälkö, Lukas Prediger, and Antti Honkela. Tight differential privacy for discrete-valued mechanisms and for the subsampled Gaussian mechanism using FFT. In Arindam Banerjee and Kenji Fukumizu, editors, *The 24th International Conference on Artificial Intelligence and Statistics, AISTATS 2021, April 13-15, 2021, Virtual Event*, volume 130 of *Proceedings of Machine Learning Research*, pages 3358–3366. PMLR, 2021.

[KL24] Nikita P. Kalinin and Christoph H. Lampert. Banded square root matrix factorization for differentially private model training. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang, editors, *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024.

[KMA+21] Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista A. Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G. L. D'Oliveira, Hubert Eichner, Salim El Rouayheb, David Evans, Josh Gardner, Zachary Garrett, Adrià Gascón, Badih Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaïd Harchaoui, Chaoyang He, Lie He, Zhouyuan Huo, Ben Hutchinson, Justin Hsu, Martin Jaggi, Tara Javidi, Gauri Joshi, Mikhail Khodak, Jakub Konečný, Aleksandra Korolova, Farinaz Koushanfar, Sanmi Koyejo, Tancrède Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer Özgür, Rasmus Pagh, Hang Qi, Daniel Ramage, Ramesh Raskar, Mariana Raykova, Dawn Song, Weikang Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian Tramèr, Praneeth Vepakomma, Jianyu Wang, Li Xiong, Zheng Xu, Qiang Yang, Felix X. Yu, Han Yu, and Sen Zhao. Advances and open problems in federated learning. *Found. Trends Mach. Learn.*, 14(1-2):1–210, 2021.

[KMC+23] Anastasia Koloskova, Ryan McKenna, Zachary Charles, John Keith Rush, and H. Brendan McMahan. Gradient descent with linearly cor-

related noise: Theory and applications to differential privacy. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023.

[KW14] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In Yoshua Bengio and Yann LeCun, editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

[LMH+15] Chao Li, Gerome Miklau, Michael Hay, Andrew McGregor, and Vibhor Rastogi. The matrix mechanism: optimizing linear counting queries under differential privacy. *VLDB J.*, 24(6):757–781, 2015.

[LT19] Jingcheng Liu and Kunal Talwar. Private selection from private candidates. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 298–309. ACM, 2019.

[Man16] Alessandro Mantelero. Personal data for decisional purposes in the age of analytics: From an individual to a collective dimension of data protection. *Comput. Law Secur. Rev.*, 32(2):238–255, 2016.

[Mar02] Robert C. Martin. *Agile Software Development: Principles, Patterns, and Practices*. Prentice Hall, 2002.

[Mar17] Robert C. Martin. *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Prentice Hall, 2017.

[Mir17] Ilya Mironov. Rényi differential privacy. In *30th IEEE Computer Security Foundations Symposium, CSF 2017, Santa Barbara, CA, USA, August 21-25, 2017*, pages 263–275. IEEE Computer Society, 2017.

[MRS24] Felix Morsbach, Jan Reubold, and Thorsten Strufe. R+R: understanding hyperparameter effects in DP-SGD. In *Annual Computer Security Applications Conference, ACSAC 2024, Honolulu, HI, USA, December 9-13, 2024*, pages 1217–1230. IEEE, 2024.

[MSH+22] Shubhankar Mohapatra, Sajin Sasy, Xi He, Gautam Kamath, and Om Thakkar. The role of adaptive optimizers for honest private hy-

perparameter selection. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022*, pages 7806–7813. AAAI Press, 2022.

[PJTH25] Gauri Pradhan, Joonas Jälkö, Marlon Tobaben, and Antti Honkela. Hyperparameters in score-based membership inference attacks. In *IEEE Conference on Secure and Trustworthy Machine Learning, SaTML 2025, Copenhagen, Denmark, April 9-11, 2025*, pages 362–384. IEEE, 2025.

[PS22] Nicolas Papernot and Thomas Steinke. Hyperparameter tuning with renyi differential privacy. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022.

[PUC+25] Krishna Pillutla, Jalaj Upadhyay, Christopher A. Choquette-Choo, Krishnamurthy Dvijotham, Arun Ganesh, Monika Henzinger, Jonathan Katz, Ryan McKenna, H. Brendan McMahan, Keith Rush, Thomas Steinke, and Abhradeep Thakurta. Correlated noise mechanisms for differentially private learning. *CoRR*, abs/2506.08201, 2025.

[RM51] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3):400–407, 1951.

[SCS13] Shuang Song, Kamalika Chaudhuri, and Anand D. Sarwate. Stochastic gradient descent with differentially private updates. In *IEEE Global Conference on Signal and Information Processing, GlobalSIP 2013, Austin, TX, USA, December 3-5, 2013*, pages 245–248. IEEE, 2013.

[SSH21] Robin M. Schmidt, Frank Schneider, and Philipp Hennig. Descending through a crowded valley - benchmarking deep learning optimizers. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 9367–9376. PMLR, 2021.

[TSL24] Qiaoyue Tang, Frederick Shpilevskiy, and Mathias Lécuyer. DP-AdamBC: Your DP-Adam is actually DP-SGD (unless you apply bias correction). In Michael J. Wooldridge, Jennifer G. Dy, and Sriraam

Natarajan, editors, *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20-27, 2024, Vancouver, Canada*, pages 15276–15283. AAAI Press, 2024.

[Vap98] Vladimir Vapnik. *Statistical learning theory.* Wiley, 1998.

[WB95] Greg Welch and Gary Bishop. An introduction to the kalman filter. Technical report, University of North Carolina at Chapel Hill, 1995.

[WZS16] Yilin Wu, Qian Zhang, and Zhiping Shen. Kalman filtering with multiplicative and additive noises. In *2016 12th World Congress on Intelligent Control and Automation (WCICA)*, pages 483–487. IEEE, 2016.

[YSS+21] Ashkan Yousefpour, Igor Shilov, Alexandre Sablayrolles, Davide Testuggine, Karthik Prasad, Mani Malek, John Nguyen, Sayan Ghosh, Akash Bharadwaj, Jessica Zhao, Graham Cormode, and Ilya Mironov. Opacus: User-friendly differential privacy library in PyTorch. *arXiv preprint arXiv:2109.12298*, 2021.

[ZBB+25] Xinwei Zhang, Zhiqi Bu, Borja Balle, Mingyi Hong, Meisam Razaviyayn, and Vahab Mirrokni. Disk: Differentially private optimizer with simplified kalman filter for noise reduction. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025.

[ZBWH24] Xinwei Zhang, Zhiqi Bu, Steven Wu, and Mingyi Hong. Differentially private SGD without clipping bias: An error-feedback approach. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024.

[ZW19] Yuqing Zhu and Yu-Xiang Wang. Poission subsampled rényi differential privacy. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 7634–7642. PMLR, 2019.

Appendix A. DP-AdamBC Bias Derivation

In this appendix, we derive the second moment bias introduced in DP-Adam due to the addition of Gaussian noise for differential privacy. This bias affects the optimizer's adaptivity, and motivates the correction proposed in DP-AdamBC [TSL24].

1. DP Gradients with Clipping and Noise

In DP-Adam, each per-example gradient is clipped and normalized using a threshold α as follows:

$$\texttt{clip}(g, \alpha) = \frac{1}{\alpha} \cdot \frac{g}{\max(1, \|g\|_2/\alpha)}, \tag{A.1}$$

The privatized gradient at iteration t is then computed as:

$$\tilde{g}_t = \bar{g}_t + \frac{1}{L} z_t, \quad z_t \sim \mathcal{N}(0, \sigma^2 I), \tag{A.2}$$

where:

- $\bar{g}_t$ is the average of the clipped gradients over a mini-batch of size L,
- σ is the noise multiplier determined by the privacy accountant,
- L is the expected batch size.

Because the noise is added to the average clipped gradient, the effective variance of each coordinate in $\tilde{g}_t$ becomes:

$$\text{Var}[\tilde{g}_t] = \frac{\sigma^2}{L^2}. \tag{A.3}$$

2. Second Moment Estimate in DP-Adam

The second moment estimate in Adam is updated using an exponential moving average:

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)\tilde{g}_t^2. \tag{A.4}$$

Taking the expectation and expanding recursively:

$$\mathbb{E}[v_t] = (1-\beta_2)\sum_{\tau=1}^{t}\beta_2^{t-\tau}\mathbb{E}[\tilde{g}_\tau^2]. \tag{A.5}$$

We decompose the squared privatized gradient into its clean and noise components:

$$\mathbb{E}[\tilde{g}_\tau^2] = \mathbb{E}[\bar{g}_\tau^2] + \frac{\sigma^2}{L^2}. \tag{A.6}$$

Substituting this into the recurrence:

$$\mathbb{E}[v_t] = (1-\beta_2)\sum_{\tau=1}^{t}\beta_2^{t-\tau}\mathbb{E}[\bar{g}_\tau^2] + (1-\beta_2)\sum_{\tau=1}^{t}\beta_2^{t-\tau}\cdot\frac{\sigma^2}{L^2}. \tag{A.7}$$

The second term is a geometric series. To simplify the expression, we perform a change of variable by letting $k = t - \tau$. This transforms the index of summation and yields a standard geometric series:

$$(1-\beta_2)\sum_{\tau=1}^{t}\beta_2^{t-\tau} = (1-\beta_2)\sum_{k=0}^{t-1}\beta_2^{k} = (1-\beta_2)\cdot\frac{1-\beta_2^t}{1-\beta_2} = 1-\beta_2^t. \tag{A.8}$$

Thus, the expected second moment becomes:

$$\mathbb{E}[v_t] = \mathbb{E}[v_t^{\text{clean}}] + (1-\beta_2^t)\psi, \quad \text{where } \psi = \left(\frac{\sigma}{L}\right)^2. \tag{A.9}$$

3. Update Step

In regions where the true gradients are small or stable, the noise term dominates. Then, the second moment reduces to:

$$v_t \approx (1-\beta_2^t)\psi. \tag{A.10}$$

Taking the bias-corrected version:

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t} \approx \psi. \tag{A.11}$$

Then, the DP-Adam update becomes:

$$\theta_{t+1} = \theta_t - \eta\frac{\hat{m}_t}{\sqrt{\hat{v}_t}+\zeta'} \approx \theta_t - \eta\frac{\hat{m}_t}{\sqrt{\psi}}, \tag{A.12}$$

where ζ' is a small constant added for numerical stability.

Appendix B. Setup 1: Fashion-MNIST Results

The following figures show the results of our experiments on the Fashion-MNIST dataset under the same setup as CIFAR-10. The patterns of optimizer behavior, learning rate sensitivity, batch size interaction, and performance ranking are all consistent with those observed in CIFAR-10. For detailed analysis, refer to Section 5.2.

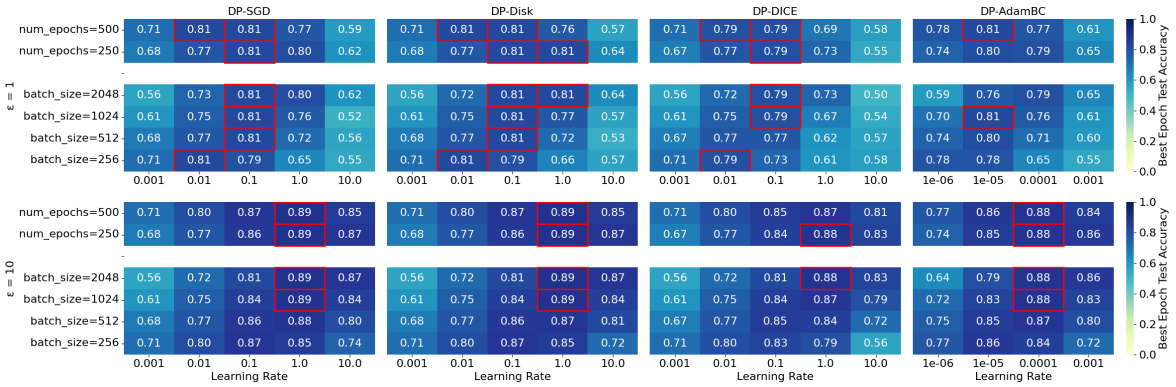


Figure B.1: Setup 1: Heatmaps showing the best test accuracy for each optimizer on the Fashion-MNIST problem, across learning rates (x-axis) and either batch size or number of epochs (y-axis), at $\epsilon = 1$ (top row) and $\epsilon = 10$ (bottom row). Each cell reports the best accuracy found over all other hyperparameters. Red boxes mark the highest accuracy per optimizer. DP-SGD consistently achieves the best results, followed by DP-DiSK, then DP-AdamBC, with DP-Dice performing worst.

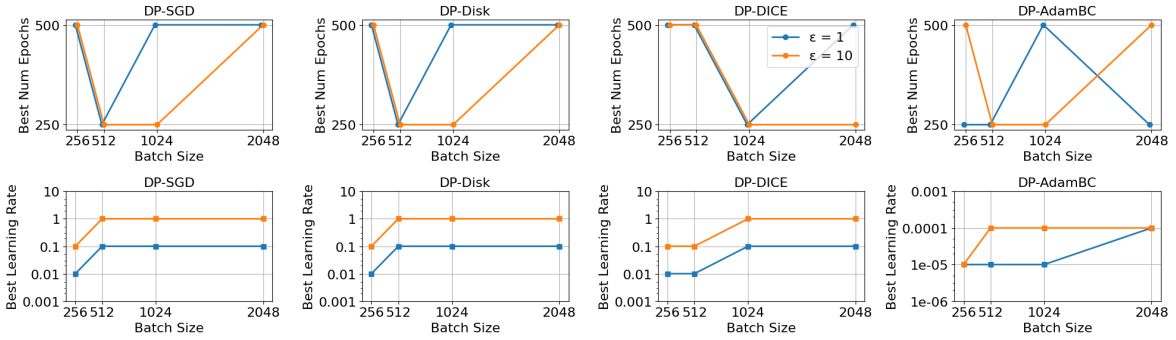


Figure B.2: Best number of epochs (top row) and learning rate (bottom row) selected for each optimizer at different batch sizes and privacy levels. No consistent pattern is observed between batch size and the optimal number of epochs, but the optimal number of epochs often changes as the batch size varies. In contrast, a clearer trend emerges for learning rate: smaller batch sizes tend to perform best at lower learning rates, while larger batch sizes benefit from higher learning rates.