IMPLEMENTATION AND COMPARATIVE ANALYSIS OF DP-SGD & DP-FTRL

Devansh Mehta
Computer Science and Engineering
Indian Institute of Technology
Kanpur, India
devanshm24@iitk.ac.in

Roshan Kumar
Computer Science and Engineering
Indian Institute of Technology
Kanpur, India
roshank24@iitk.ac.in

***Abstract*—**
—This project investigates the training of deep neural networks under formal privacy guarantees by implementing two leading algorithms—Differentially-Private Stochastic Gradient Descent (DP-SGD) and Differentially-Private Follow-The-Regularised-Leader (DP-FTRL)—and comparing them with non-private baselines on the MNIST image classification task using identical MLP and LeNet architectures. For each method we tune clipping norms and Gaussian noise multipliers, track the cumulative privacy loss with a Rényi-DP accountant, and measure test accuracy. The study provides a clear, end-to-end reference for integrating differential privacy into real-world machine-learning pipelines and clarifies the trade-offs between utility and privacy across two techniques.

I. Introduction

Modern machine-learning models often train on sensitive data such as health records, bank transactions, and even handwritten digits that belong to real people. Standard (non-private) networks such as a multilayer perceptron (MLP) or the LeNet convolutional neural network aim only for high accuracy; on the 60 000-image MNIST dataset they can surpass 98 % test accuracy. Yet they give **no formal promise** that an observer cannot tell whether a specific record was in the training set.

Differential Privacy (DP) fixes this gap by adding carefully chosen noise so that any single example can change the model's output only by a small amount, measured by a privacy budget (ϵ, δ). In short, DP lets us balance learning quality with strict limits on how much information about one person can get leaked.

We study two widely used DP training methods. **DP-SGD** takes ordinary stochastic gradient descent, clips each sample's gradient to a fixed size, then adds Gaussian noise before every update. A privacy accountant tracks how much of the budget has been spent. The method is simple, but the added noise can lower final accuracy.

DP-FTRL (Differentially-Private Follow-The-Regularised-Leader) views training as an online problem. It stores running sums of gradients in a tree so that each data point affects only a few of those sums. That way the total noise needed stays low, especially when we train for many steps.

In our experiments, we applied both DP-SGD and DP-FTRL to the image classification task on the MNIST dataset using identical network architectures. We compared them with the non-private baselines and reported accuracy alongside the privacy budget they consume.

II. Non-Private Baseline Models

In this study a *non-private model* is a neural network trained in the usual way—optimising the loss on the full data set with no gradient clipping or noise injection. These baselines establish the predictive upper bound against which the later, differentially-private results will be judged. We implemented the following two architectures: a Multilayer Perceptron and LeNet-5.

a) Multilayer Perceptron (MLP).: A simple MLP with the following structure:

- **Input.** Each 28×28 greyscale image is flattened into a 784-dimensional vector.
- **Hidden layers.** Three dense layers with 512, 256, and 128 neurons, each followed by a ReLU activation; dropout ($p = 0.2$) is applied after every hidden layer.
- **Output layer.** A 10-unit dense layer produces the class logits; softmax is applied only at inference time.
- **Capacity.** The network contains roughly 5.7×10^5 trainable parameters, most of which reside in the first layer (784×512 weights).

b) LeNet-5 Convolutional Network.: A CNN with the following structure:

- **Convolutional feature extractor**
  - Conv 1: 6 filters of size 5×5 with ReLU
  - Max-pool 1: 2×2 window
  - Conv 2: 16 filters of size 5×5 with ReLU
  - Max-pool 2: 2×2 window
- **Fully-connected classifier**

- FC 1: 400 inputs $\rightarrow$ 120 ReLU units
    - FC 2: 120 $\rightarrow$ 84 ReLU units
    - FC 3: 84 $\rightarrow$ 10 logits (softmax at test time)
- **Capacity.** Parameter sharing keeps the total number of trainable weights to about 6.2×10^4.

c) Why LeNet Outperforms the MLP on Images.:

1) **Spatial inductive bias** – convolutional kernels detect local edges, corners, and strokes, whereas the MLP must learn such correlations from raw pixels.
2) **Translation invariance** – pooling layers make predictions robust to small spatial shifts common in handwritten digits.
3) **Hierarchical features** – successive conv-pool blocks build up from simple edge detectors to abstract digit parts, capturing global shape while retaining detail.

Empirically, LeNet-5 reaches about 99% test accuracy on MNIST, whereas the fully-connected MLP plateaus near 98% under identical training conditions. These figures form the reference point for evaluating the differentially-private versions in the remainder of this report.

III. DP–SGD

Differentially-Private Stochastic Gradient Descent (DP-SGD) is the most widely used method for training deep neural networks with formal privacy guarantees. It follows the usual mini-batch SGD recipe but adds two key operations—per-example gradient clipping and calibrated Gaussian noise injection—so the influence of any single training record is provably bounded. With a moments or Rényi accountant tracking the cumulative privacy loss, the overall procedure satisfies (ε, δ)-differential privacy.

Algorithm 1 Differentially Private Stochastic Gradient Descent (DP-SGD)

```
Input: Dataset $D = \{x_1, \ldots, x_n\}$; loss $L(\theta, x)$; number of iterations $T$; learning rate $\eta$; mini-batch size $m$; privacy budget $\epsilon$; failure probability $\delta$; gradient-norm bound $G$
Initialize: Model weights $\theta_0 \in \mathbb{R}^d$ randomly,
    $\epsilon_0 = \frac{\epsilon}{\sqrt{T \log(T/\delta)}}, \quad \delta_0 = \frac{\delta}{T}$
for $t = 1$ to $T$ do
    Select a random subsample $D_t \subset D$ of $m$ points.
    for all $x_i \in D_t$ do
        $g_t(x_i) \leftarrow \nabla_{\theta_{t-1}} L(\theta_{t-1}, x_i)$
        $\overline{g}_t(x_i) \leftarrow \frac{g_t(x_i)}{\max\big(1, \|g_t(x_i)\|_2 / G\big)}$ {clip to norm $G$}
    end for all
    $\overline{g}_t \leftarrow \sum_{x_i \in D_t} \overline{g}_t(x_i)$
    $\widehat{g}_t \leftarrow \overline{g}_t + \mathcal{N}\big(0, \sigma^2 I\big), \qquad \sigma^2 = \frac{2G^2 \log(2/\delta_0)}{\epsilon_0^2}$
    $\theta_t \leftarrow \theta_{t-1} - \eta \, \frac{\widehat{g}_t}{m}$
end for
Output: Final weights $\theta_T$
```

Each training round begins by randomly sampling a mini-batch, which already adds uncertainty about any individual's presence in the update. DP-SGD then computes *per-example* gradients so that every record's influence can be bounded independently. These gradients are clipped to a fixed norm G, ensuring the update is Lipschitz with respect to a single example and thereby limiting its maximum effect on the model.

After clipping, the algorithm aggregates the gradients and injects zero-mean Gaussian noise with variance σ^2 chosen to satisfy the per-step privacy parameters (ϵ_0, δ_0). Because post-clipping sensitivity is at most G, this noise level guarantees differential privacy for that iteration. A moments or Rényi accountant composes the per-step guarantees over the T rounds, certifying that the complete training run never exceeds the global budget (ϵ, δ).

Finally, the model is updated with the noisy, averaged gradient. By bounding sensitivity and injecting calibrated noise at every update, DP-SGD delivers strong privacy guarantees while retaining much of the predictive power of standard stochastic optimisation.

Our Implementation : In our implementation, we integrated differential privacy into both the fully-connected MLP and LeNet-5 CNN architectures while maintaining the standard PyTorch training workflow. We leveraged Opacus' `PrivacyEngine` to seamlessly incorporate the privacy-preserving mechanism into our models. The implementation follows the core DP-SGD methodology with several practical considerations as detailed below.

1) **Data pipeline :** MNIST images are converted to tensors and normalised with the canonical mean and standard deviation $(0.1307, 0.3081)$. Mini-batches are drawn uniformly *without* replacement. Strict DP analysis assumes Poisson sampling, yet Opacus' accountant is calibrated for this common batching scheme, so no extra logic is required.
2) **Making the loop private :** Before training begins the model, optimiser and data loader are wrapped with the `PrivacyEngine` This single call instructs Opacus to (i) compute *per-sample* gradients, (ii) clip the ℓ_2 norm of each gradient to be bound by Norm bound (1.1), and (iii) add Gaussian noise drawn from $\mathcal{N}(0, \sigma^2 I)$ before every optimiser step. Because those operations are inserted transparently, the remainder of the training loop is identical to a standard PyTorch loop.
3) **Dual approaches to noise calibration:** We employed both common strategies for noise parameter selection. In some experiments, we fixed the noise multiplier σ upfront and allowed the privacy accountant to report the resulting privacy budget ε. This approach provided flexibility during initial model development, as the convolutional architecture required less noise to maintain comparable accuracy. For our comparative analysis between DPSGD and DPFTRL algorithms, we adopted the alternative approach—solving for the appropriate σ value that would achieve a predetermined privacy budget ε. This enabled fair comparisons between the two algorithms under equivalent privacy constraints, effectively isolating performance from privacy considerations.
4) **Live privacy accounting :** At the end of every epoch the script logs the running ε next to loss and accuracy.

IV. DP-FTRL

Differentially-Private Follow-The-Regularized-Leader (DP-FTRL) is an online learning method that groups past gradients in a binary-tree structure and then adds noise. At each step, the model draws a noisy summary of all past gradients from a small number of tree nodes and then updates its parameters by minimising that summary plus a regularisation term. Compared to DP-SGD, which injects noise at every gradient step and accumulates privacy loss linearly in the number of iterations, DP-FTRL's amortised noise strategy causes the total privacy cost to grow only logarithmically with the training length—resulting in stronger utility when training for many rounds or under moderate privacy budgets.

Algorithm 2 DP-Online-Learning

```
Input: Dataset $D = \{x_1, \dots, x_n\}$, loss function $L$, regularizer $\lambda$
Initialize: model weights $\theta_0 \in \mathbb{R}^d$ randomly, Binary Tree with $n$ nodes.
for each iteration $t \in \{1, \dots, n\}$ do
    1. Compute gradient
           $g_t(x_t) = \nabla_{\theta_{t-1}} L(\theta, x_t)$
    2. Add $g_t(x_t)$ to binary tree
    3. Get a noisy estimate $\tilde{s}_t$ for sum of gradients
           $\sum_{i=1}^{t} g_i(x_i)$ using binary-tree mechanism
    4. Update weights
           $\theta_t = \arg\min_{\theta} \tilde{s}_t^\top \theta + \frac{\lambda}{2}\|\theta\|^2$
Output: Final weights $\theta_n$
```

Training begins with a random weight vector θ_0 and an empty binary tree. At each time t, we receive a new data point x_t and compute its gradient

$$g_t \;=\; \nabla_{\theta_{t-1}} L(\theta_{t-1}, x_t),$$

just as in standard online learning. Rather than adding g_t directly to a running sum, we insert it into leaf t of the tree and propagate its value upward, so that each internal node stores the sum of its two children. Crucially, the first time any node becomes complete, we add a single draw of Gaussian noise to that node's value and then never modify it again.

When it is time to update the model at round t, we reconstruct a private prefix sum $\sum_{i=1}^{t} g_i$ by reading exactly the $O(\log t)$ tree nodes whose subtrees cover the first t leaves. Each node already carries its one-time noise injection, so their sum $\tilde{s}_t$ equals the true cumulative gradient plus a well-calibrated Gaussian perturbation whose total variance grows only like $\log t$.

Finally, we apply the follow-the-regularized-leader update by choosing

$$\theta_t \;=\; \arg\min_{\theta}\Big(\tilde{s}_t^\top \theta \;+\; \tfrac{\lambda}{2}\|\theta\|^2\Big),$$

which has the closed-form solution $\theta_t = -\tilde{s}_t/\lambda$. Repeating this cycle—gradient computation, one-time noise injection, noisy prefix-sum readout, and regularized update—up to $t = n$ yields a final model that closely matches the non-private regularized solution while spending only logarithmic noise per example, thus satisfying (ε, δ) - differential privacy with minimal utility loss.

Our Implementation :

1) **Weight Initialization.** All network parameters are initialised using Kaiming/Xavier schemes (via our `nn` helper) rather than PyTorch defaults. This ensures stable gradient magnitudes when noise is added later.
2) **Per-Example Gradient Computation.** For each incoming batch we calculate gradient after which Opacus clips every per-sample gradient to an ℓ_2-norm that has been specified (1.1 in our case).
3) **Adding the Clipped Gradient to the Binary Tree.** Once the clipped batch gradient is assembled, we hand it to our `TreeNoise` module, which:
   - Maintains a binary counter `self.binary` that increments each batch.
   - Flips bits in the counter to mark completed nodes in a full binary tree of height $\approx \log_2$(num_batches).
   - For each bit that flips from 0 to 1, draws a fresh Gaussian tensor (standard deviation tied to clip_norm and noise_multiplier) and stores it in the corresponding node buffer.
   - Resets any node whose bit flips back from 1 to 0, so each noise draw occurs exactly once per node per full cycle.
4) **Recovering the Noisy Prefix-Sum Gradient.** To obtain the private cumulative gradient $\tilde{s}_t$ at step t, we request the current noise tensor from `TreeNoise`, which sums the buffers of the nodes corresponding to 1-bits in the counter and then computes $\tilde{s}_t$ ensuring that the total variance grows only like $\log T$, since each leaf's gradient contributes to at most $O(\log T)$ nodes, each noised once.
5) **Standard Weight Update.** Our `FTRLOptimizer.step()` performs an in-place update
$$\theta \;=\; \theta_0 - \frac{1}{\lambda}\,\tilde{s}_t,$$
optionally mixing in momentum. This vector update is algebraically equivalent to the closed-form FTRL solution.
6) **Output of Each Step.** After calling `.step()`, the model parameters have been adjusted against the noisy cumulative gradient with the chosen regularisation and learning-rate schedule. We then record training loss and accuracy as usual.
7) **Privacy Accounting.** Differential privacy loss is tracked using Rényi Differential Privacy (RDP), which provides tighter and more flexible composition over multiple training steps.

- **Why RDP?**
  - RDP composes *linearly*, making it more accurate than standard (ϵ, δ) composition, especially over many steps.
  - It allows evaluating privacy loss across different Rényi orders α, offering flexibility to select the optimal privacy guarantee.

  The process involves:
- **Step 1: Compute Maximum Squared Sensitivity.** Determine how often each batch contributes to nodes in the tree aggregation structure. The maximum squared sensitivity is:
$$S_{\max} = \max_i \sum_j c_{i,j}^2$$
where $c_{i,j}$ is the count of how many times batch i contributes to node j.
- **Step 2: Compute Effective Standard Deviation.** Adjust the noise scale based on sensitivity:
$$\sigma_{\text{effective}} = \frac{\text{noise multiplier}}{\sqrt{S_{\max}}}$$
- **Step 3: Calculate RDP.** For a given Rényi order α, the RDP of the Gaussian mechanism is:
$$\epsilon_{\text{RDP}}(\alpha) = \frac{\alpha}{2 \cdot \sigma_{\text{effective}}^2}$$
- **Step 4: Convert RDP to (ϵ, δ)-DP.** Using:
$$\epsilon = \epsilon_{\text{RDP}}(\alpha) - \frac{1}{\alpha - 1} \cdot \log\left(\delta(\alpha - 1) \cdot \left(1 - \frac{1}{\alpha}\right)^{-\alpha}\right)$$

  The α that minimizes ϵ is selected for the final guarantee.

V. Results and Observations

- **Non-Private Models**

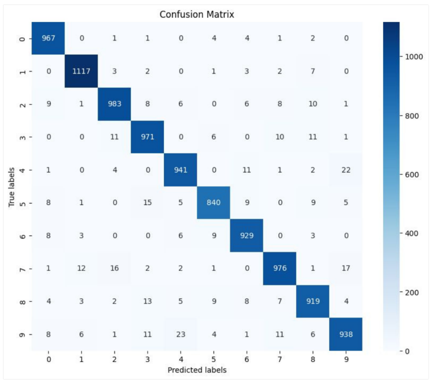

Fig. 1: MLP Confusion Matrix

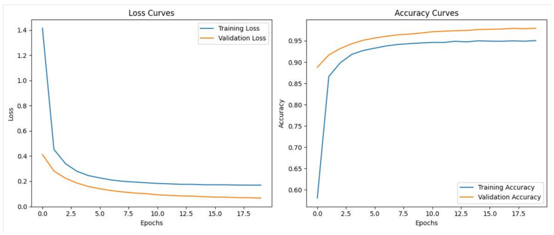

Fig. 2: MLP Loss v/s Epochs & Accuracy v/s Epochs

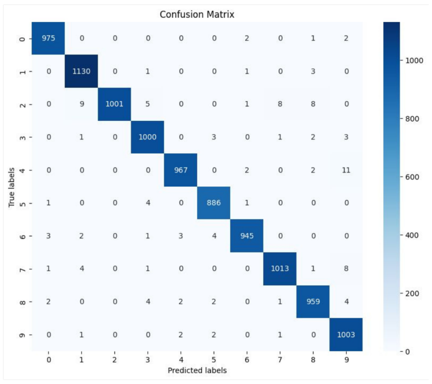

Fig. 3: LeNet Confusion Matrix

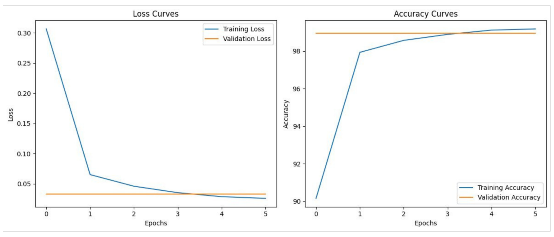

Fig. 4: LeNet Loss v/s Epochs & Accuracy v/s Epochs

- **DP-SGD**

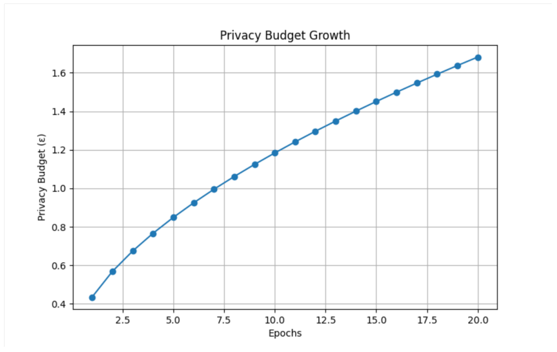

Fig. 5: Privacy Comparison ($\sigma = 1$)

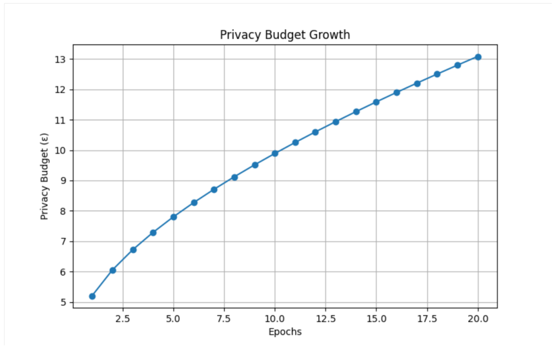

Fig. 6: Privacy Comparison ($\sigma = 0.5$)

- **Figures 5** and **6** trace identical ε curves for the MLP and LeNet because every DP–SGD hyper-parameter—batch size, clipping norm, learning-rate schedule, and the noise multiplier σ—is kept constant across both architectures.
- Over 20 epochs, ε rises slowly from ≈ 0.42 to ≈ 1.66 when $\sigma = 1$ (Fig. 5) but jumps from ≈ 5.1 to ≈ 13.1 when $\sigma = 0.5$ (Fig. 6), showing how privacy loss accelerates as the injected noise is reduced.

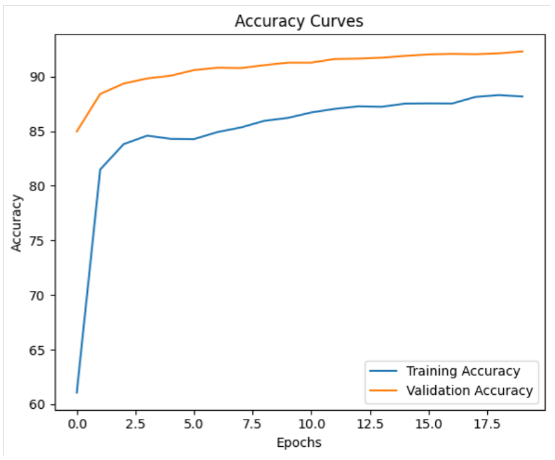

Fig. 7: Utility v/s Privacy for MLP ($\sigma = 1$)

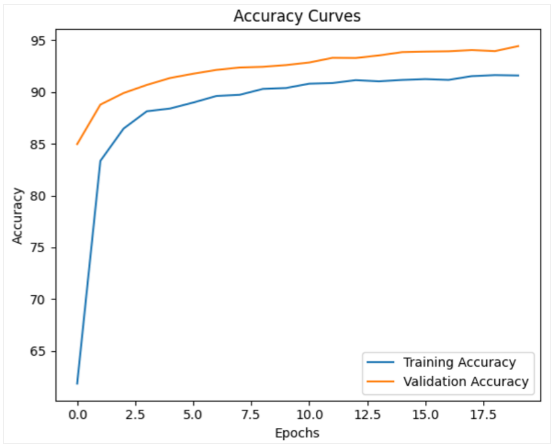

Fig. 8: Utility v/s Privacy for MLP ($\sigma = 0.5$)

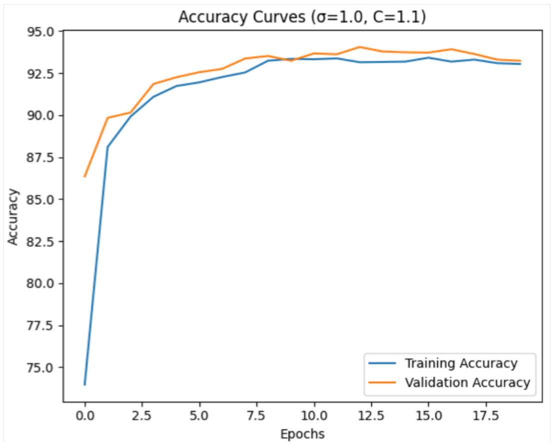

Fig. 9: Utility v/s Privacy for LeNet ($\sigma = 1$)

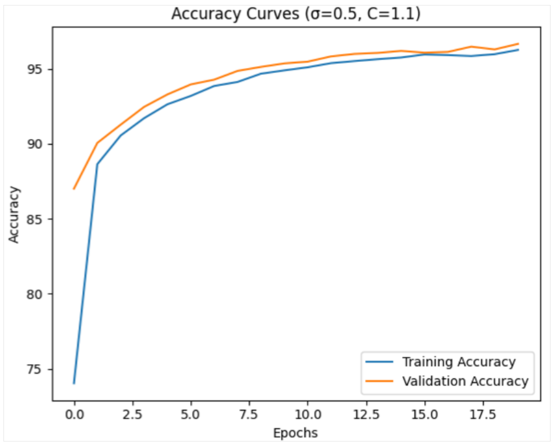

Fig. 10: Utility v/s Privacy for LeNet ($\sigma = 0.5$)

- **Figs. 7 vs 8** (MLP, $\sigma = 1 \rightarrow 0.5$): cutting the noise in half lifts the whole MLP accuracy curve; less perturbation gives higher utility at every epoch.
- **Figs. 9 vs 10** (LeNet, $\sigma = 1 \rightarrow 0.5$): LeNet follows the same pattern, with most of the gain in the early epochs—showing its greater sensitivity to noise at the start of training.
- **Figs. 7 vs 9** (MLP vs LeNet, $\sigma = 1$): for the same privacy budget, the convolutional LeNet outperforms the fully-connected MLP throughout, proving architecture matters for utility.
- **Figs. 8 vs 10** (MLP vs LeNet, $\sigma = 0.5$): even with a looser privacy budget, LeNet keeps the lead, suggesting architecture has more impact than further reductions in DP noise once σ is moderate.

- **DP-FTRL**

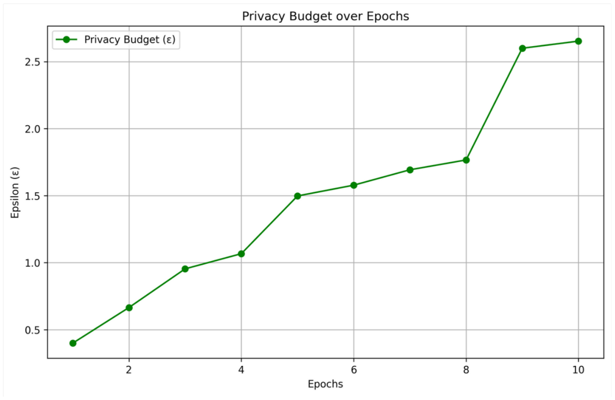

Fig. 11: Privacy Comparison (Noise Multiplier = 25)

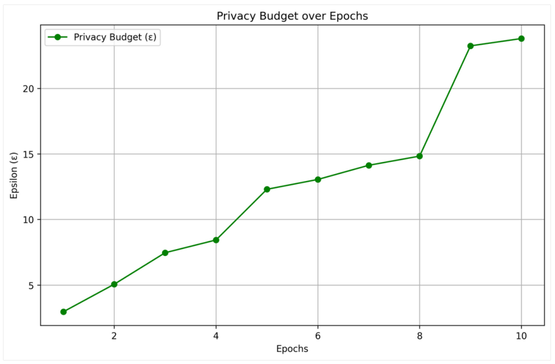

Fig. 12: Privacy Comparison (Noise Multiplier = 4)

– As seen in **Figs. 11** and **12**, DP–FTRL on LeNet keeps the privacy budget low ($\varepsilon \approx 2.7$ at epoch 10) with a large noise multiplier of 25, whereas lowering it to 4 sends ε past 24. More noise means better privacy.

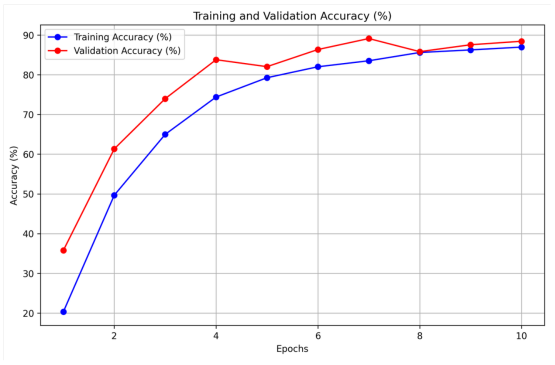

Fig. 13: Utility v/s Privacy (Noise Multiplier = 25)

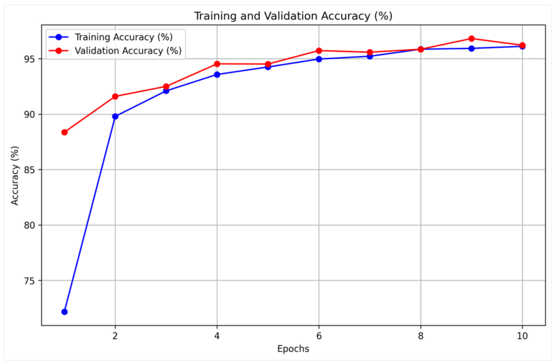

Fig. 14: Utility v/s Privacy (Noise Multiplier = 4)

– **Figs. 13** and **14** show the classic trade-off: heavy noise (multiplier 25) slows and lowers the accuracy curves, while light noise (multiplier 4) boosts accuracy sooner—better utility at the cost of weaker privacy.

TABLE I: DP–SGD in **MLP** Architecture Results

Batch Size	Epochs	σ	ε	Accuracy (%)	Training Time (s)	Inference Time (s)
256	20	1.00	1.68	92.22	226.42	0.0195
256	20	0.75	3.33	93.17	223.11	0.0237
256	20	0.50	13.09	94.13	234.67	0.0223

Learning rate $= 0.05$, $\delta = 10^{-5}$, gradient-norm clip $= 1.1$.

- As seen in **Table I**, with the batch size and epochs fixed at 256 and 20, shrinking the noise multiplier from $\sigma = 1$ to 0.5 pushes ε up from 1.68 to 13.09 while nudging accuracy from 92 % to 94 %, illustrating the familiar privacy–utility trade-off.

TABLE II: DP–SGD in **LeNet** Architecture Results

Batch Size	Epochs	σ	ε	Accuracy (%)	Training Time (s)	Inference Time (s)
256	20	1.00	1.68	93.63	228.99	0.0289
256	20	0.75	3.33	95.38	227.91	0.0314
256	20	0.50	13.09	96.61	253.08	0.0229

Learning rate $= 0.05$, $\delta = 10^{-5}$, gradient-norm clip $= 1.1$.

- As seen in **Table II**, the same training budget shows the same trend—ε jumps from 1.68 to 13.09 as σ drops, and accuracy climbs from 93.6 % to 96.6 %—demonstrating that less noise yields higher utility but weaker privacy.

TABLE III: DP–FTRL in **LeNet** Architecture Results

Batch Size	Epochs	Noise Multiplier	ε	Accuracy (%)	Training Time (s)	Inference Time (s)
500	10	25	2.65	90.18	107.60	0.0293
500	10	20	3.41	91.27	111.02	0.0280
500	10	12.5	5.84	93.63	107.92	0.0291
500	10	8	9.92	95.21	109.17	0.0307
500	10	4	23.80	96.17	109.17	0.0298

Learning rate $= 1$, $\delta = 10^{-5}$, gradient-norm clip $= 1.1$.

- As seen in **Table III**, keeping the batch size, epochs, and every other hyper-parameter fixed, lowering the noise multiplier (e.g. $25 \rightarrow 4$) injects less Gaussian noise per update; the privacy accountant therefore reports a larger ε (weaker privacy) while the cleaner gradients let the model learn slightly better, so accuracy inches upward.
- Training ($\approx$ 108–111 s) and inference ($\approx$ 0.029 s) times remain virtually unchanged because run-time is dominated by forward and backward passes through the network; sampling and adding Gaussian noise adds only negligible overhead.

TABLE IV: DP–SGD *vs.* DP–FTRL on LeNet (equal batch size = 500)

Mechanism	ε	δ	Accuracy (%)	σ	Training Time (s)	Batch Size	Epochs	Learning Rate
DP-SGD	3.41	10^{-5}	95.70	0.872	231.89	500	20	0.1
DP-FTRL	3.41	10^{-5}	91.27	0.044	111.02	500	10	1.0
DP-SGD	5.71	10^{-5}	96.02	0.710	240.03	500	20	0.1
DP-FTRL	5.71	10^{-5}	93.57	0.028	105.61	500	10	1.0
DP-SGD	13.14	10^{-5}	96.60	0.545	246.68	500	20	0.1
DP-FTRL	13.14	10^{-5}	95.55	0.014	109.30	500	10	1.0

- As seen in **Table IV**, with the batch size, ϵ and δ fixed, DP-SGD needs a much larger noise multiplier ($\sigma \approx 0.9, 0.7, 0.5$) than DP-FTRL ($\sigma$ well below 0.1) because it performs twice as many updates (20 epochs versus 10). More steps mean more chances to leak information, so extra noise is added to keep the same privacy budget. Those extra updates also let DP-SGD refine the weights longer, which is why its accuracy is 2–4 percentage points higher in every row.

- As ϵ relaxes from about 3 to 13, the noise injected by both methods becomes small compared with the gradient signal; the models see almost "clean" data, so their final accuracies converge (96.6% vs 95.5%). When privacy budget is relaxed, the DPFTRL algorithm provides comparable accuracy to the DPSGD (It could even outperform the DPSGD for moderate budgets).
- DP-FTRL uses a higher learning rate (1.0) and fewer epochs (10), which leads to significantly lower training time but slightly reduced accuracy. In contrast, DP-SGD uses a smaller learning rate (0.1) and more epochs (20), resulting in better accuracy but longer training times.

VI. Challenges and Pain Points

Implementing and evaluating DP-SGD and DP-FTRL on MNIST revealed several practical challenges and pain points:

1. Computational and Memory Overhead

- **Per-Example Gradients:** Computing and storing per-sample gradients (required by DP-SGD/Opacus) multiplies memory use by the mini-batch size and can exhaust GPU RAM on larger networks.
- **Tree Aggregation Buffers:** DP-FTRL's binary-tree noise protocol maintains $O(\log T)$ noise tensors per parameter, adding storage and slowdown when the number of steps T grows.

2. Hyperparameter Tuning Complexity

- **Clip Norm and Noise Scale:** The privacy–utility trade-off is highly sensitive to the clipping bound C and noise multiplier σ. Finding a stable pair often requires sweeping dozens of combinations.
- **Learning Rate Interaction:** Noisy updates can destabilize standard learning-rate schedules. We observed that fixed or plateau-based schedules work better than aggressive decay, but tuning remains data- and architecture-specific.
- **Batch Size vs. Privacy Budget:** Larger batches reduce variance in gradient estimates but consume privacy faster (via sensitivity), forcing a careful balance between iteration count and privacy loss.

3. Privacy Accounting and Debugging

- **RDP Conversion Nuances:** Converting Rényi DP to (ε, δ) requires an α-grid search; choosing too coarse a grid can under- or over-estimate the true budget.
- **Reproducibility:** Small changes in random seeds or sampling order can produce diverging privacy reports, making end-to-end debugging of ε tricky.

VII. Discussion and Future Work

The empirical study confirms the classic privacy–utility tension that lies at the heart of differential privacy. Whenever the Gaussian noise added to each update was reduced—either by lowering the noise multiplier in DP-FTRL or the σ value in DP-SGD—the privacy budget ϵ rose sharply while test accuracy crept upward. This pattern is clearest in DP-SGD, where cutting σ from 1 to 0.5 pushed ϵ from about 1.7 to 13 but lifted MLP accuracy only two percentage points and LeNet by three. The two optimisers, however, spend their budgets very differently. DP-SGD injects noise at every step, so to satisfy the same (ϵ, δ) it must use a much larger σ than DP-FTRL, which noises a logarithmic-sized tree of accumulated gradients. The price DP-SGD pays for its stronger per-step privacy control is longer wall-clock time—a doubling of epochs translated into roughly twice the training time in our runs. Architecture also matters: under every privacy setting the convolution-based LeNet converted the same ϵ into higher accuracy than the fully-connected MLP, indicating that spatial weight sharing lets the model tolerate injected noise more gracefully.

Several avenues could sharpen these findings. First, repeating the experiments on more challenging datasets such as CIFAR-10 or Fashion-MNIST would test whether the observed gaps persist once the baseline accuracy ceiling is lower. Second, recent work on adaptive clipping and noise schedules suggests that re-estimating the clip bound or σ online can save privacy budget without hurting accuracy; integrating such techniques may shift the sweet spot further toward stronger privacy. Third, trying deeper backbones—ResNets or even vision transformers—would show whether added depth amplifies or dampens the effect of DP noise. Finally, combining DP-FTRL with federated or split-learning protocols and using tighter Rényi or f-DP accountants could provide stronger end-to-end privacy with little extra cost.

VIII. Conclusion

This project delivered a full PyTorch pipeline for training image-classification models under formal differential-privacy guarantees and compared two leading methods—DP-SGD and DP-FTRL—on identical MLP and LeNet architectures. The results make three points clear. (i) Lowering the noise level invariably boosts accuracy but inflates the privacy budget, so the trade-off cannot be avoided, only managed. (ii) At the same (ϵ, δ), DP-SGD achieves slightly better utility than DP-FTRL, yet it needs more updates and a larger σ, doubling training time in our settings. (iii) Architectural choice is pivotal: LeNet consistently outperformed the MLP for every privacy budget examined. Together these findings give practitioners two practical dials—noise magnitude and optimiser choice—for navigating the privacy–accuracy landscape, while the accompanying code and tables offer a ready reference for bringing differential privacy into real-world image-classification workflows.

References

[1] Martín Abadi, Andy Chu, Ian Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. *Deep learning with differential privacy*. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS '16), pages 308–318, Vienna, Austria, 2016. ACM.

[2] Peter Kairouz, Brendan McMahan, Shuang Song, Om Thakkar, Abhradeep Thakurta, and Zheng Xu. *Practical and private (deep) learning without sampling or shuffling*. In Proceedings of the 38th International Conference on Machine Learning (ICML 2021), PMLR 139, pages 5213–5224, 2021.

Project GitHub Link: https://github.com/ghosh-sarbajit/CS798L_DPSGD_Roshan_Devansh