Undergraduate Project Report

Privacy Preservation in Deep Learning

Submitted by:
Sarthak Paswan
220976
Department of CSE
IIT Kanpur

Under the supervision of:
Prof. Sayak Ray Chowdhury
Department of CSE
IIT Kanpur

April 2025

Abstract

This project explores the problem of privacy preservation in machine learning models, particularly in the deep learning setting. We study Differential Privacy (DP) as a framework and implement the DP-SGD algorithm to achieve private learning. Building on this, we explore the Label-DP Pro algorithm, which focuses on privatizing only the labels. We also integrate privacy accounting using Google's DP library. Our goal is to provide a comprehensive approach to training deep models with formal privacy guarantees. The results show that meaningful privacy (ε, δ) can be achieved with good model utility, and we propose future applications in LLMs.

Introduction

In the era of data-driven technologies, machine learning models are increasingly trained on **vast datasets** containing **sensitive** personal information. From healthcare records to financial transactions and user behavior, these data fuel powerful predictive models, but they also introduce serious privacy concerns. Traditional machine learning algorithms are not designed with privacy in mind and may unintentionally **memorize** and **expose** details about individual training examples.

This has led to a growing demand for formal privacy-preserving techniques that ensure that models do not **leak** sensitive information. Among these, **Differential Privacy (DP)** has emerged as the gold standard, offering strong mathematical guarantees that limit the extent to which any single data point can influence the behavior of the model.

However, applying differential privacy to complex deep learning systems presents unique challenges. Adding too much noise can severely degrade model performance, while too little noise offers inadequate protection. Striking the right balance requires careful algorithmic design and accurate measurement of privacy loss over time.

Differential Privacy

Definition 1 (Differential Privacy)**.** *Let $\varepsilon \in \mathbb{R} > 0$ and $\delta \in [0, 1)$. A randomized algorithm $\mathcal{A}$ is said to be* ***(ε, δ)-differentially private*** *if for any two adjacent datasets D and D' (differing in at most one data point), and for any subset of outputs $S \subseteq$ Range($\mathcal{A}$), the following holds:*

$$\Pr[\mathcal{A}(D) \in S] \leq e^{\varepsilon} \cdot \Pr[\mathcal{A}(D') \in S] + \delta$$

Differential Privacy is a mathematical framework designed to measure the **privacy** of individual inputs in a dataset. It guarantees that the output of an algorithm remains statistically similar, whether or not any single individual's data is included.

The privacy level is controlled using two parameters: epsilon (ϵ), which measures privacy loss, and delta (δ), which indicates the probability of a privacy breach or failure probability.

Definition 2. *A randomized training algorithm $\mathcal{A}$ is (ε, δ)-LabelDP if it is (ε, δ)-DP when two input datasets differ on the label of a single training example.*

For both notions, due to the e^{ε} factor, the *high-privacy regime* corresponds to small ε (e.g., $\varepsilon < 1$).

Differentially Private SGD Algorithm (DP-SGD)

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

Require: Initial model parameters θ_0, dataset D of size n, loss function ℓ, learning rate η_t, noise multiplier σ, clipping norm C, number of iterations T, batch size n_1

Ensure: Trained model parameters θ_T

for $t = 0$ to $T - 1$ **do**
  Sample batch $I_t \subset [n]$ of size n_1 uniformly at random
  for each $i \in I_t$ **do**
    Compute gradient: $\mathbf{g}_t(x_i, y_i) \leftarrow \nabla_\theta \ell(\theta_t, (x_i, y_i))$
    Clip gradient: $\bar{\mathbf{g}}_t(x_i, y_i) \leftarrow \frac{\mathbf{g}_t(x_i, y_i)}{\max\left(1, \frac{\|\mathbf{g}_t(x_i, y_i)\|_2}{C}\right)}$
  end for
  Aggregate gradients and add noise:
  $\tilde{\mathbf{g}}_t \leftarrow \frac{1}{n_1}\left(\sum_{i \in I_t} \bar{\mathbf{g}}_t(x_i, y_i) + \mathcal{N}(0, \sigma^2 C^2 \mathbf{I})\right)$
  Update model parameters: $\theta_{t+1} \leftarrow \theta_t - \eta_t \tilde{\mathbf{g}}_t$
end for

Norm clipping: Proving the differential privacy guarantee of Algorithm requires bounding the influence of each individual example on $\mathbf{g}_t$. Since there is no *a priori* bound on

the size of the gradients, we *clip* each gradient in ℓ_2 norm; i.e., the gradient vector $\mathbf{g}$ is replaced by

$$\mathbf{g}/\max\left(1, \frac{\|\mathbf{g}\|_2}{C}\right),$$

for a clipping threshold C. This clipping ensures that if $\|\mathbf{g}\|_2 \leq C$, then $\mathbf{g}$ is preserved, whereas if $\|\mathbf{g}\|_2 > C$, it gets scaled down to be of norm C. We remark that gradient clipping of this form is a popular ingredient of SGD for deep networks for non-privacy reasons, though in that setting it usually suffices to clip after averaging.

Noise addition: We also add random noise to obscure the contribution of any individual example. Specifically, after aggregating the clipped gradients over a sampled batch, we add noise sampled from a Gaussian distribution. Formally, we add noise sampled from $\mathcal{N}(0, \sigma^2 C^2 \mathbf{I})$, which is a multivariate Gaussian with **mean zero** and covariance $\sigma^2 C^2 \mathbf{I}$ and C is the clipping threshold.

DP-SGD is a modification of the standard stochastic gradient descent (SGD) algorithm designed to provide formal differential privacy (DP) guarantees during the training of machine learning models. The key idea is to limit and obscure the influence of any individual training example on the learned model, thereby protecting sensitive data.

Key Properties and Guarantees:

- Bounded Sensitivity: Clipping ensures that no single example can have an outsized influence on the update, which is critical for DP guarantees.
- Randomization: Adding calibrated noise ensures that the output of the algorithm is probabilistically indistinguishable when any single data point is changed, thus providing differential privacy.
- Privacy-Utility Tradeoff: The amount of noise (controlled by σ) and the clipping norm (C) must be balanced to achieve both strong privacy and good model performance.

Label Differential Privacy

Label differential privacy (label-DP) is a relaxation of the standard differential privacy definition, tailored for machine learning scenarios where only the **labels** in the training data are considered sensitive, while the feature vectors are assumed to be public or non-sensitive. Formally, the definition of label differential privacy can be found in Definition 2 above.

In standard differential privacy (DP) algorithms such as DP-SGD, noise is typically added to the gradients at every training step to ensure that the influence of any single data point is obscured. This is necessary for providing strong privacy guarantees, but it often leads to the phenomenon of overaddition of noise, where more noise is injected than is strictly necessary for the desired privacy level, potentially degrading model utility.

Label-DP is particularly useful when the privacy risk is concentrated on the label information, allowing for stronger utility of the trained model compared to standard differential privacy, which protects both features and labels. However, care must be taken in scenarios where features and labels are not independent, as naive noise addition to labels may not provide strong privacy guarantees.

LabelDP-Pro Algorithm

Algorithm 2 LabelDP-Pro

Require: Initial model weights θ_0, training set D of size n, learning rate η_t, number of training iterations T, batch size n_1, noise multiplier σ, gradient norm bound C, and a Denoiser method

Ensure: Trained model weights θ_T

for $t = 0$ to $T - 1$ **do**
  Sample batch $I_t^G \subset [n]$ of size n_1 uniformly at random
  for each $i \in I_t^G$ **do**
    $g_t(x_i, y_i) \leftarrow \nabla_{\theta_t} \ell(\theta_t, (x_i, y_i))$
  end for
  for each $i \in I_t^G$ **do**
    $\bar{g}_t(x_i, y_i) \leftarrow \frac{g_t(x_i, y_i)}{\max\left(1, \frac{\|g_t(x_i, y_i)\|_2}{C}\right)}$
  end for
  $\tilde{g}_t \leftarrow \frac{1}{n_1}\left(\sum_{i \in I_t^G} \bar{g}_t(x_i, y_i) + \mathcal{N}(0, \sigma^2 C^2 \mathbf{I})\right)$
  $\hat{g}_t \leftarrow \text{Denoiser}(\tilde{g}_t, \theta_t, \{x_i\}_{i \in I_t^G})$ ▷ Apply denoiser
  $\theta_{t+1} \leftarrow \theta_t - \eta_t \hat{g}_t$
end for

LabelDP-Pro is an improved differentially private training algorithm designed to improve both privacy and utility during model training. It is based on the standard DP-SGD algorithm, but introduces an additional **denoising** step that leverages the model parameters and potentially the input features. The key idea is to postprocess the noisy gradients before applying them to update the model parameters.

Why the Denoising Step is Powerful

The Denoiser does not compromise privacy, because differential privacy is preserved under post-processing. This allows the algorithm to:

- Learn to distinguish between noise and signal in gradients.
- Potentially improve learning efficiency and generalization.
- Integrate domain knowledge or data-specific priors in the gradient correction.

LabelDP-Pro Denoisers

Table 1: A summary of denoisers for noisy gradients $\tilde{g}_t$, with optional access to the model weights θ_t and the features of the current batch $\{x_i\}_{i \in I_t^G}$. A denoiser can also access the features of the full training set. In particular, it can sample an "alternative" batch I_t^P of n_2 examples and have access to $\{x_i\}_{i \in I_t^P}$.

Denoiser	**Output**
SelfSpan	$\text{Proj}_A(\tilde{g}_t)$ for $A = \text{span}\left(\{\nabla_{\theta_t} \ell(\theta_t, (x_i, \kappa))\}_{i \in I_t^G, \kappa \in [K]}\right)$
SelfConv	$\text{Proj}_A(\tilde{g}_t)$ for $A = \text{conv}\left(\{\nabla_{\theta_t} \ell(\theta_t, (x_i, \kappa))\}_{i \in I_t^G, \kappa \in [K]}\right)$
AltConv	$\text{Proj}_A(\tilde{g}_t)$ for $A = \text{conv}\left(\{\nabla_{\theta_t} \ell(\theta_t, (x_i, \kappa))\}_{i \in I_t^P, \kappa \in [K]}\right)$

SELFSPAN Denoiser: SELFSPAN projects the noisy gradient onto the span of all per-example, per-class gradients computed from the current batch and all possible class labels. Since the true (unnoised) gradient lies in this span, the projection preserves the signal while reducing the noise component. Importantly, this span can be constructed using only the features (not the labels), exploiting the LabelDP setting.

SELFCONV Denoiser: SELFCONV further refines the denoising by projecting the noisy gradient onto the convex hull of the per-example, per-class gradients from the current batch. This approach leverages the fact that the true gradient is not just in the span, but also in the convex hull of these vectors, leading to improved noise reduction,

especially as the number of classes increases.

ALTCONV Denoiser: ALTCONV projects the noisy gradient onto the convex hull of per-example, per-class gradients computed from an independently sampled alternative batch. This batch is drawn separately from the current batch and uses only public features. ALTCONV preserves privacy amplification by subsampling, which allows for tighter privacy accounting and lower overall noise.

Memory Efficient Projection

We implement a memory-efficient algorithm to find the projected gradients in the projection-based denoisers described above.

Let $\tilde{\mathbf{g}}$ be the privatized gradient, and $\mathbf{G} = [\nabla_\theta \ell(\theta, (x_i, \kappa))]_{i \in I_t^P, \kappa \in [K]}$ be the $d \times n_2 K$ dimensional matrix of the per-example per-class gradients (in the case of SelfSpan and SelfConv, $I^P = I^G$ and $n_2 = n_1$). To project $\tilde{\mathbf{g}}$ onto the convex hull of the columns of $\mathbf{G}$, we solve $\min_{\alpha \in \Delta} \|\mathbf{G}\alpha - \tilde{g}\|^2$, where Δ is the unit simplex for $n_2 K$-dimensional vectors. We solve this optimization problem with **Projected Gradient Descent (PGD)**. Specifically, let $\alpha_0 \in \Delta$ be an initialization of the projection coefficients, and η^{PGD} be the step size, we iteratively update it as

$$\alpha_{t+1} \leftarrow \text{Proj}_\Delta \left(\alpha_t - \eta^{\text{PGD}} \mathbf{G}^\top (\mathbf{G}\alpha_t - \tilde{g}) \right).$$

The projection onto the unit simplex can be computed using the method in tensorflow:

Algorithm 3 Projection onto the Probability Simplex

Require: A batch of vectors $\alpha \in \mathbb{R}^{n_2 K}$
Ensure: Projected vectors onto the probability simplex

function ProjectOntoSimplex(α)
  $\alpha \leftarrow \text{cast}(\alpha, \text{float32})$
  $B \leftarrow \text{batch\_size}(\alpha)$
  $d \leftarrow \text{dimension}(\alpha)$
  $\text{sorted}_\alpha \leftarrow \text{sort}(\alpha, \text{descending})$
  $\text{cumsum} \leftarrow \text{cumulative\_sum}(\text{sorted}_\alpha, \text{axis} = -1)$
  $k \leftarrow [1, 2, \ldots, d]$
  $\theta_{\text{candidates}} \leftarrow \frac{\text{cumsum} - 1}{k}$
  $\text{mask} \leftarrow \text{sorted}_\alpha > \theta_{\text{candidates}}$
  $\rho \leftarrow$ max index where mask is true
  $\theta \leftarrow \theta_{\text{candidates}}[\rho]$
  $\text{projection} \leftarrow \max(\alpha - \theta, 0)$
  return projection
end function

The projection step is computed using methods of Wang & Carreira-Perpiñán (2013).

We next focus on efficiently computing the two subroutines: $\mathbf{u} \mapsto \mathbf{G}\mathbf{u}$ for any $\mathbf{u} \in \mathbb{R}^{n_2 K}$, and $\mathbf{v} \mapsto \mathbf{G}^\top \mathbf{v}$ for any $\mathbf{v} \in \mathbb{R}^d$. We note that given the special structure of $\mathbf{G}$, those subroutines can be computed using advanced *auto differentiation* primitives without materializing $\mathbf{G}$. We implemented the computation of $\mathbf{G}\mathbf{u}$ using the function below in **tensorflow**.

Algorithm 4 Compute_G_u

Require: *images* as the input batch, weight u and *model* be the neural network

```
function COMPUTE_G_U(images, u)
    with tf.GradientTape() as tape:
        logits ← model(images)
        loss ← tf.nn.log_softmax(logits, axis=-1)
        loss_w ← -tf.reduce_mean(u * loss)
    G_u ← tape.gradient(loss_w, model.trainable_variables)
    return G_u
end function
```

We then implemented the computaion of $\mathbf{G}^\top \mathbf{v}$ using the function below in **tensorflow**.

Algorithm 5 Compute_G_T_v

Require: *images* as the input batch, weight u and *model* be the neural network

```
function COMPUTE_G_T_V(images, v)
    with tf.autodiff.ForwardAccumulator(primals=model.trainable_variables,
tangents=v) as acc:
        logits ← model(images)
        raw_loss ← -tf.nn.log_softmax(logits, axis=-1)
    jvp ← acc.jvp(raw_loss)
    return jvp
end function
```

The entire implementation is available in the Github Repository [2].

Implementation Results

DP-SGD

Hyperparameters

The following hyperparameters were used for all experiments for the best results:

- **Clipping norm** $C = 0.1$
- **Learning rate** $(\eta) = 0.05$
- **Number of epochs** $= 10$
- **Batch_size / Dataset_size** $(q) = 0.01$
- **Delta** $(\delta) = 10^{-5}$
- **Main loop iterations** $(T) = 1000$

Noise Computation

Using THEOREM 1 from the DPSGD paper, we use the inequation

$$\sigma \geq c_2 \frac{q\sqrt{T\log(1/\delta)}}{\varepsilon}.$$

where, c_2 is a constant fixed to 1 and T is the number of iterations of the main loop. **Note** that the number of epochs here equals a $100T$ as we can assume that the entire dataset gets processed after random sampling it a 100 times.
Using the inequation we get different value of σ according to different value of ϵ. Thus, we can get the amount of noise (σ) for different privacy budget (ϵ).

Table 2: Test accuracy percentage of the DP-SGD algorithm under varying levels of Gaussian noise. The results illustrate the trade-off between model performance and the strength of differential privacy guarantees.

ϵ	σ	**Test Accuracy**
0.1	7	91.59%
0.5	1.4	91.65%
1	0.7	91.69%
5	0.14	91.73%
10	0.07	91.82%

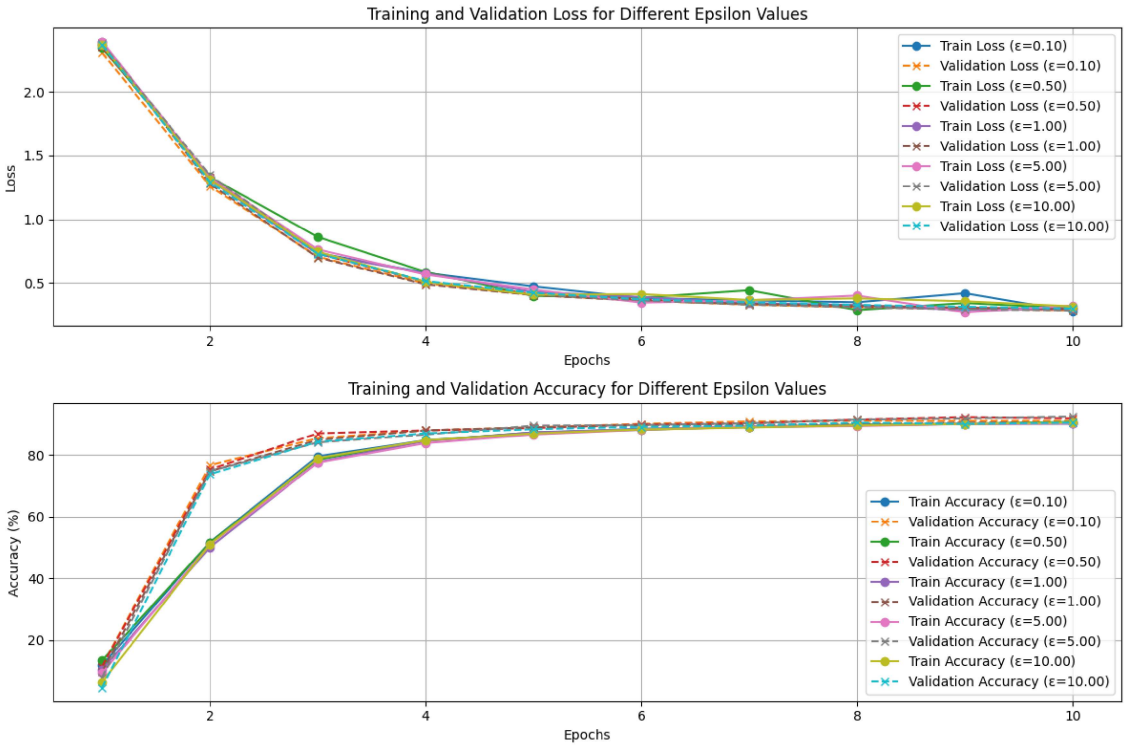

Figure 1: DPSGD Results

LabelDP-Pro ALTConv

Hyperparameters

The following hyperparameters were used for all experiments for the best results:

- **Clipping norm** $C = 0.1$
- **Learning rate** $(\eta) = 0.05$
- **Learning rate of PGD** $(\eta_{PGD}) = 0.05$
- **Number of epochs** $= 10$
- **Number of epochs of PGD** $= 20$
- **Batch_size / Dataset_size** $(q) = 0.01$
- **Delta** $(\delta) = 10^{-5}$
- **Main loop iterations** $(T) = 1000$

Table 3: Test accuracy percentage of the LabelDP-Pro ALTConv algorithm under varying levels of Gaussian noise.

ϵ	σ	**Test Accuracy**
0.1	7	76.08%
0.5	1.4	77.71%
1	0.7	77.82%
5	0.14	78.58%
10	0.07	78.69%

Noise Computation

Remains the same as the DPSGD noise computation.

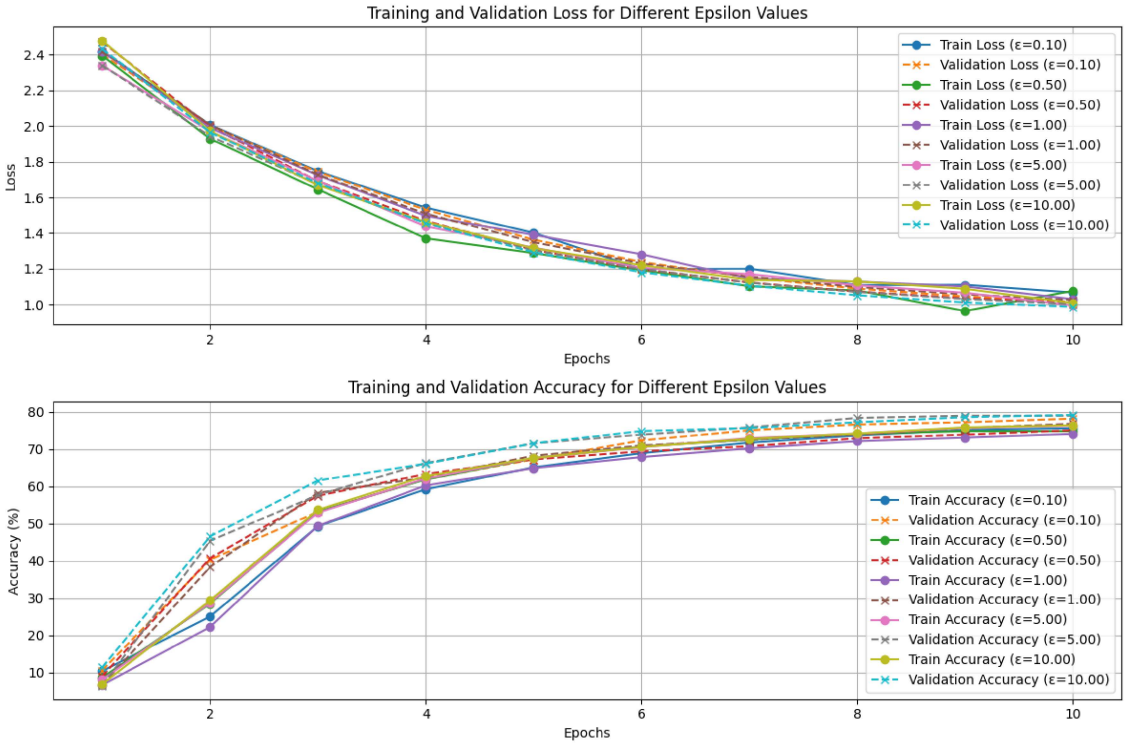

Figure 2: LabelDP-Pro ALTConv

Privacy Accounting

Privacy accounting is a fundamental component of differentially private machine learning algorithms like DP-SGD and LabelDP-Pro. It quantifies the cumulative privacy loss, typically expressed as (ϵ, δ)-incurred over multiple accesses to sensitive data during training. Each step or query in an algorithm contributes some privacy loss, and privacy accounting ensures that the total loss remains within a target privacy budget, balancing privacy guarantees with model utility.

Principles of Privacy Accounting Composition:

- *Differential privacy is composable:* the privacy loss from multiple algorithmic steps accumulates over time. Accurate accounting is essential, especially in iterative algorithms like DP-SGD, where noise is added at each step.
- *Privacy Loss Distribution (PLD):* Modern accounting techniques, such as Privacy Loss Distributions, provide tight estimates of cumulative privacy loss.
- Privacy accounting helps in selecting the parameters to achieve the right trade-off between privacy loss and accuracy.

Google's DP Accounting Library

Google DP Accounting Library Google's open-source DP Accounting Library is a widely used tool for tracking and computing privacy budgets in complex machine learning pipelines. To use the library, please find the attached link [3]. Key features used in the privacy accounting are:

- **Supports Common Mechanisms:** Used the Gaussian mechanisms for composing privacy for T number of steps.
- Used the Privacy Loss Distribution (PLD) accounting as the Gaussian noise distribution.

Privacy Accounting on LabelDP-Pro

Hyperparameters

The following hyperparameters were used for all experiments for the best results:

- **Clipping norm** $C = 0.1$
- **Learning rate** $(\eta) = 0.05$
- **Learning rate of PGD** $(\eta_{PGD}) = 0.05$

- **Number of epochs** = 10
- **Number of epochs of PGD** = 20
- **Batch_size / Dataset_size** (q) = 0.01
- **Delta** (δ) = 10^{-5}
- **Main loop iterations** (T) = 1000
- **Privacy Budget** (ϵ) = 30. Privacy budget of 30 was fixed for all the experiments.

Privacy Budget Management

During training, we set a total privacy budget of $\varepsilon = 30$ for the entire model training process. At each iteration, the cumulative privacy loss was tracked. The training algorithm was configured to terminate immediately once the cumulative privacy loss exceeded the specified budget. This approach ensures that the final model satisfies the desired differential privacy guarantee.

Table 4: Test accuracy percentage of the LabelDP-Pro ALTConv algorithm under fixed privacy budget, varying levels of Gaussian noise and T.

σ	T	**Epochs**	**Test Accuracy**
1	100	1	42.80%
5	600	6	73.49%
10	2200	22	81.01%

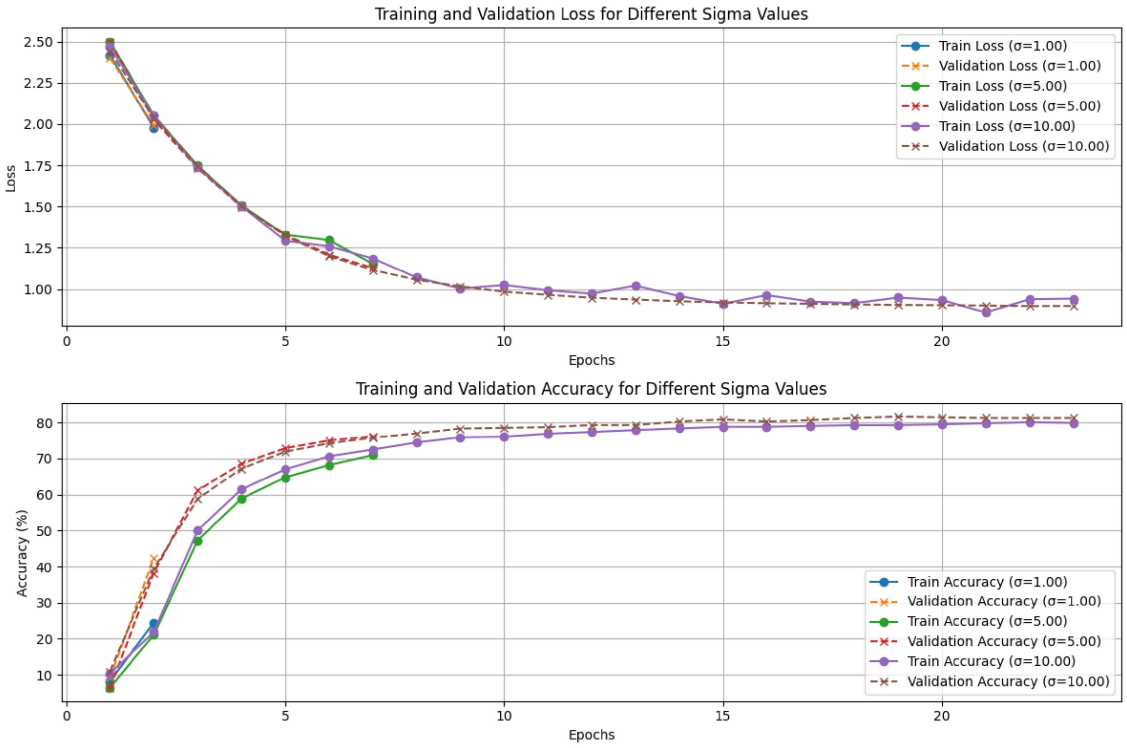

Figure 3: Privacy Accounting on LabelDP-Pro

Conclusion and Future Work

In this project, we explored privacy-preserving machine learning with a focus on the LabelDP-Pro algorithm, which is designed to provide strong privacy guarantees for sensitive labels while leveraging public features for improved utility. Through the implementation and experimentation with LabelDP-Pro, we demonstrated that it is possible to achieve meaningful differential privacy guarantees (in terms of (ε, δ)) without incurring excessive utility loss.

A promising direction for future work is the application of the LabelDP-Pro framework to large language models (LLMs). Given the scale and sensitivity of data involved in training LLMs, incorporating label differential privacy could provide substantial privacy benefits while maintaining high model performance.

Bibliography

[1] Weiran Wang and Miguel A. Carreira-Perpiñán. *Projection onto the probability simplex: An efficient algorithm with a simple proof, and an application 2013*, https://arxiv.org/pdf/1309.1541

[2] *Privacy Preservation in Deep Learning - Github Repository*, https://github.com/AlgoSarthak/Privacy-Preservation-in-Deep-Learning-UGP-

[3] *Google's DP Library documentation,* https://github.com/google/differential-privacy

[4] Abadi et al., "Deep Learning with Differential Privacy," 2016. https://arxiv.org/pdf/1607.00133

[5] LabelDP-Pro: LEARNING WITH LABEL DIFFERENTIAL PRIVACY VIA PROJECTIONS, https://openreview.net/pdf?id=JnYaF3vv3G