Evaluating Differentially Private Stochastic Gradient Descent Optimizers for Training Transformer Language Models

Eric Gong
Harvard College
ericgong@college.harvard.edu

Oleg Pavliv
Harvard College
olegpavliv@college.harvard.edu

Max Peng
Harvard College
mpeng@college.harvard.edu

Abstract

The remarkable ability of LLMs to generalize to a diverse variety of tasks is made possible by training upon immense corpuses of data. These data, scraped from internet sources with little to no filtering and oversight can risk the contamination of LLM's model weights with sensitive and private information, which may now become readily available to all parties utilizing the model. As such, providing formal guarantees on limiting the extent to which any particular secret or private information found in the training data can influence model weights is of critical interest. We thus demonstrate the efficacy of differentially private stochastic gradient descent optimizers at providing privacy guarantees at training time.

1 Introduction

1.1 Background

Large Language Models (LLMs), built upon the transformer network architecture [20], have gained immense popularity in recent years, holding promise of greatly revolutionizing both academia and industry, increasing efficiency, replacing menial tasks, even supplementing the innovation and ideation traditionally limited to the most creative and ingenious human intellect. The immense applicability and generalizability of LLMs can be accredited to its vast swaths of training data. In fact, Chinchilla Scaling laws even quantify the fact that there exist predictable, measurable improvement in model generations as training data quantity increases [8]. So as to gather and utilize immense corpuses of text-based data for training models, companies turn towards scraping the internet for human-generated text. However, as the quantity of text increases, it quickly becomes intractable to fully screen and vet all text used to train the LLMs.

Among data scraped from the internet, there are undoubtedly samples problematic statements, and of potentially sensitive information. The former has been combated with fine-tuning; after a model is pre-trained on vast swathes of internet text, specific frameworks are introduced to fine-tune the LLM, tweaking model weights to remove unfavorable, incoherent, and problematic generations from the LLMs lexicon [18]. However, fine-tuning only serves to suppress such problematic generations, and cannot guarantee removal, with prompt hacking methodologies demonstrating that it is trivially feasible to coerce language models into generating outputs which are intentionally suppressed by its developers [19]. Such methodologies could be equally applied to exposing private and sensitive information inadvertently incorporated into the LLMs training data.

Unlearning—the process of removing the influence of undesirable information from an LLM without impacting unrelated information—has been explored as a potential solution to this dilemma [9]. However, unlearning primarily targets known, identified instances of undesirable information. Sensitive information that is not discovered and unlearned can remain indefinitely. This, in conjunction with the prevalence of

LLMs memorizing and regurgitating text found in training data verbatim [6] thus poses a severe security risk and privacy violation [11].

As such, it becomes critical to identifying methodologies by which it becomes possible to restrict the influence of undesirable information at training time, without external classification or detection of undesirability. Thus enters differential privacy, a field of study which not only proposes rigorous and quantitative definitions of privacy, but also provable guarantees for privacy preservation, making it a widely accepted gold standard for the purposes of statistical analysis and release of datasets [5]. Treating the training of an LLM as a form of data aggregation and release then, it becomes possible to utilize differential privacy to design and deliver guarantees which bound the extent to which any particular instance of undesirable information can influence generations elicited from the model.

In the context of LLMs, model parameters are adjusted based on training data to minimize a loss function which represents conformity to the training data. The most commonly used mechanism of minimizing the loss function is Stochastic Gradient Descent (SGD). As such, SGD can be treated as the process by which data and private information is used to influence the model parameters, which are ultimately used in inference and the generation of output, making it a primary target for differentially private frameworks, which would enable developers and researchers to bound the influence of any particular training example on the overall model weights. Thus is differentially private SGD (DP-SGD), first proposed by Abadi et. al. in 2016[1].

1.2 Motivation for Evaluation

DP-SGD is a powerful technique capable of creating provable and quantifiable privacy guarantees on the influence of training data on machine learning models, but it is not without drawbacks. In particular, the process of gradient clipping, and the addition of random noise not only increases computational overhead in an already computationally demanding workflow, but also degrades the quality of the resulting model, if given the same hyperparameters [17]. In particular, previous research has shown that to mitigate the degradation of generations caused by DP-SGD, and reach the same level of performance, it becomes necessary to train the model for more iterations, resulting in an increase in computational costs [10].

In this work, we aim to quantify the extent to which DP-SGD impacts model training and performance, by examining and comparing various implementations of DP-SGD with a non-DP SGD optimizer. So as to examine the impact of DP-SGD in a controlled environment, we opt to train a simple transformer model implementation from scratch, utilizing carefully crafted training data examples to evaluate the model across different circumstances.

2 Mathematical and Theoretical Overview

We begin with an overview of normal stochastic gradient descent, before introducing differentially private stochastic gradient descent and its particular considerations with respects to the addition of gaussian noise, particularly in determining the magnitude of noise to be added.

2.1 Stochastic Gradient Descent

Normal Stochastic Gradient Descent selects the optimal model parameters θ by taking small steps that decrease the calculated loss over the dataset. More rigorously, at each step:

1. Selects a random batch of b examples
2. Compute the average gradient across the selected examples
3. Updates θ by moving in the negative gradient direction, with the magnitude of movement controlled by the learning rate.

Algorithm 1 Overview of Standard Stochastic Gradient Descent

```
Require: Data $\{x_i\}_{i=1}^N$, loss $L(\theta, x)$, initial $\theta_0$, learning rates $\{\eta_t\}$, batch size $b$, steps $T$.
1: for $t = 1$ to $T$ do
2:     Sample mini-batch $B_t$ of size $b$.
3:     $g_t \leftarrow \frac{1}{b} \sum_{x \in B_t} \nabla_\theta L(\theta_{t-1}, x)$.
4:     $\theta_t \leftarrow \theta_{t-1} - \eta_t \, g_t$.
5: end for
```

2.2 Differentially Private Stochastic Gradient Descent

Under the differentially private variation of stochastic gradient descent, a framework of gradient clipping and gaussian noise addition is implemented to ensure that no particular sample is able to have an outsized impact on the gradient at any step of iteration [4].

In particular clipping each gradient limits the global sensitivity, which would otherwise be infinite, and the addition of gaussian noise will suffice to mask the contributions of individual data points such that ε-DP guarantees can be achieved. Specifically:

1. Select a random batch of b examples.
2. Compute gradients individually for each example
3. Clip gradients by normalizing to at most some constant C
4. Add Gaussian noise of variance $\sigma^2 C^2$ for some computed σ providing ε, δ-DP guarantees
5. Updates θ by moving in the negative gradient direction, with the magnitude of movement controlled by the learning rate.

Algorithm 2 Overview of Differentially Private Stochastic Gradient Descent[1]

```
Require: Data $\{x_i\}$, loss $L(\theta, x)$, initial $\theta_0$, $\{\eta_t\}$, $b$, $C$, $\sigma$, $T$.
1: for $t = 1$ to $T$ do
2:     Sample $B_t$ of size $b$.
3:     for each $x \in B_t$ do
4:         $g_t(x) \leftarrow \nabla_\theta L(\theta_{t-1}, x)$.
5:         $\bar{g}_t(x) \leftarrow g_t(x) / \max\big(1, \|g_t(x)\|_2 / C\big)$.
6:     end for
7:     $\bar{g}_t \leftarrow \frac{1}{b} \sum_{x \in B_t} \bar{g}_t(x)$.
8:     $\widetilde{g}_t \leftarrow \bar{g}_t + \mathcal{N}(0, \sigma^2 C^2 I)$.
9:     $\theta_t \leftarrow \theta_{t-1} - \eta_t \widetilde{g}_t$.
10: end for
```

2.3 Gaussian Noise under Differing Composition Strategies

Having established the framework by which differential-privacy-preserving noise is to be added to the training iterations of the SGD algorithm, it then becomes a question of how much noise is necessary. In particular, we wish to quantify the value of σ that will ensure ε, δ-DP guarantees.

Thus we can turn to the DP compositions, to provide bounds on the value of σ necessary to ensure (ε, δ)-DP guarantees. In particular, the more detailed and fine-grained the approach, the more precise the bound on σ, the less noise that needs to be added at each step of the gradient descent.

For simplicity, let T be the total number of training steps taken, or number of iterations of the algorithm. Let N be the size of the training data, and $q = b/N$ be the proportion of data sampled at each iteration. Then, we wish to calculate the value of σ such that our model is (ε, δ)-DP.

2.3.1 Naive Composition Theorem

We can begin with the most simple composition theorem. Under the Naive Composition Theorem, we assume that each iteration of the optimizer is an (ε', δ')-DP release. As such, composing T such iterations or releases is $(T\varepsilon', T\delta')$-DP. As such, for this to equal an (ε, δ)-DP release as we intend, we have:

$$\varepsilon' = \frac{\varepsilon}{T}, \quad \delta' = \frac{\delta}{q\,T}.$$

Then, by Theorem 3.22 of Dwork's *Algorithmic Foundations of Differential Privacy* [4], the Gaussian mechanism (with sensitivity C) then requires

$$\sigma_{\text{naive}} = \frac{\sqrt{2\ln(1.25/\delta')}}{\varepsilon'} = \frac{q\,T\,\sqrt{2\ln\!\left(\frac{1.25\,q\,T}{\delta}\right)}}{\varepsilon}.$$

2.3.2 Advanced Composition Theorem

Accounting for the fact that the same dataset is being used across the T different iterations, we can revise our composition definition, such that we ensure that an adversary—given the releases from all T iterations—would be unable to identify whether or not a particular data point were in the dataset. This is Advanced Composition, also introduced in Dwork's *Algorithmic Foundations of Differential Privacy*, providing privacy guarantees with less noise required per iteration.

In particular, Theorem 3.20[4] tells us that if each iteration of the optimizer is (ε', δ')-DP, then the overall composition of T iterations will be $(\widetilde{\varepsilon},\ T\delta' + \delta'')$-DP, for

$$\widetilde{\varepsilon} = \varepsilon'\sqrt{2T\ln\tfrac{1}{\delta''}} + T\varepsilon'(e^{\varepsilon'} - 1).$$

By Corollary 3.21[4], this can be simplified to

$$\varepsilon < \widetilde{\varepsilon} \;=\; \frac{\varepsilon'}{2\sqrt{2T\,\ln\!\big(1/\delta'\big)}}.$$

As such, if we use this new per-step ε' in our Gaussian mechanism, it suffices to have noise

$$\sigma_{\text{advanced}} = O\!\Big(\frac{q\sqrt{T\,\ln(1/\delta)\,\ln(T/\delta)}}{\varepsilon}\Big).$$

So as to establish the efficiency of the advanced composition method over the naive composition method, we can compare the noise that would be required for the Gaussian mechanism under both cases:

$$\frac{\sigma_{\text{advanced}}}{\sigma_{\text{naive}}} = O\!\Big(\frac{q\sqrt{T\,\ln(1/\delta)\,\ln(T/\delta)}/\varepsilon}{q\,T\,\sqrt{2\ln(1.25\,q\,T/\delta)}/\varepsilon}\Big) = O\!\Big(\sqrt{\frac{\ln(1/\delta)\,\ln(T/\delta)}{2\,T\,\ln(1.25\,q\,T/\delta)}}\Big).$$

As T becomes large, $\ln(T/\delta) \approx \ln T$ and $\ln(1.25\,q\,T/\delta) \approx \ln T$, so

$$\frac{\sigma_{\text{strong}}}{\sigma_{\text{naive}}} = O\!\Big(\sqrt{\tfrac{\ln(1/\delta)}{T}}\Big) = O\!\Big(\tfrac{1}{\sqrt{T}}\Big)$$

Thus, indeed, we can confirm that advanced composition requires asymptotically less noise than does naive composition.

2.3.3 DP-SGD Specific Composition methods

While Advanced Composition provides a strong bound on necessary Gaussian variance that is widely adaptable to a variety of compositions and DP mechanisms, researchers have leveraged the specific properties of SGD optimizers to define DP-SGD specific bounds on the necessary amount of Gaussian noise to ensure (ε, δ)-DP guarantees.

One such method is the Moments Accountant method introduced by Abadi et. al. [1], wherein, by taking into consideration the nature of the Gaussian mechanism as the modality by which noise is added—which Advanced Composition makes no assumptions of—it is possible to reduce the standard deviation of the Gaussian mechanism required by a logarithmic factor.

In particular, Abadi et. al. establish that for a Gaussian mechanism to preserve (ε, δ)-DP under T-fold composition, it suffices to have

$$\sigma \geq O(\frac{q\sqrt{T\ \ln(1/\delta)}}{\varepsilon}),$$

This removes a factor of $\sqrt{\ln(T/\delta)}$, making this a tighter bound on necessary Gaussian noise standard deviation than the Advanced composition theorem.

Of course, even more nuanced methods for calculating σ for the purposes of DP-SGD exist. In particular, by observing that differing degrees of utility loss is incurred when differing levels of noise are added on earlier iterations as opposed to later iterations, it is possible to vary the standard deviation for the Gaussian noise as a function of the number of iterations progressed. In doing so, it is possible to have lower Gaussian noise additions towards the later iterations, improving model performance while ensuring the same (ε, δ)-DP privacy guarantees. Opacus is an example of a DP-SGD implementation that allows for such scaling noise addition [15].

3 Methodology

So as to test the affect of DP-SGD in a controlled environment under limited access to compute, we deploy a small multi-headed transformer language model, adapting it so that it may be used with a custom DP-SGD implementation, as well as a professional DP-SGD implementation from the open-sourced Opacus project.

Code utilized throughout the evaluation process, including model implementations, figure generation scripts, and various training datasets used to evaluate the performance of the base and DP-SGD models can be found at `https://github.com/ericgong2005/Differentially_Private_SGD`

3.1 Base Transformer Model Implementation

We implement a simple transformer model to serve as a simplistic representation of commercial LLMs, complete with multiple self-attention heads, layer normalization, residual connections, random dropout, etc. As it would undoubtedly be a difficult feat to implement such a model from scratch under the given time constraints, we utilize a previous implementation of a transformer model which fits the requirements, taken from `https://github.com/ericgong2005/LLMSeuss`.

This transformer model implementation is based off of tutorials published by Andrej Karpathy, replicating portions of the transformer model implemented in the seminal "Attention is All You Need" first published 2017 [20]. In addition, specific consideration are made for various features of the implementation to ensure the construction of a model representative of larger commercial models. Such considerations include layer normalization (particularly the efficacy of pre-layer normalization[21] over the originally proposed post-layer normalization[2]), random dropout for forward passes[7], and the number of hidden layers with respect to the number of layers in the model[20].

With respects to tokenization, given the small size of the model, and the limited training data that we expect to use, it would be inefficient to utilize large-scale tokenizers designed for significantly larger lexicons, such as OpenAI's Tiktoken[1]. As such, we implement a simple word-level tokenizer which parses the training text corpus, extracting all unique words, punctuations, spaces, etc. and defines the vocabulary size based on the training text. This design choice is particularly useful in allowing for fine-grained controlling of the vocabulary size, so as to easily adapt the size of the model to fit time and compute restraints as necessary.

3.2 Custom DP-SGD Implementation

To discuss our implementation, we developed a custom implementation of Differentially Private Stochastic Gradient Descent (DP-SGD) primarily by adapting code from DP-SGD implementations, such as the one found in the `private-adam` github repository[2], which is based on the seminal DP-SGD optimizer proposed by Abadi et. al. [1]. We incorporated its `DPSGD` optimizer class into our training loop by modifying two files: inserting `optim.py`, where the optimizer logic resides, and editing `transformer_model.py`, where we replaced the default PyTorch optimizer with `DPSGD` and adjusted the pipeline accordingly.

[1]https://github.com/openai/tiktoken

[2]https://github.com/aakashks/private-adam

Our implementation follows the original algorithm by Abadi et al. (2016) [see above in Section 1]: for each batch, we compute per-example gradients, clip each to an ℓ_2 norm of $C = 1$, average the clipped gradients, and then add Gaussian noise drawn from $\mathcal{N}(0, \sigma^2 C^2 I)$ before applying the parameter update.

In our setup, we use a batch size $b = 16$ and train over 100 epochs on approximately 10,800 training tokens, yielding $T = 67{,}500$ total optimization steps. To meet (ε, δ)-DP guarantees, we calibrate the noise scale σ using the Moments Accountant bound (note the approximation of 1.12):

$$\sigma = \frac{1.12 \cdot q \cdot \sqrt{T \cdot \ln(1/\delta)}}{\varepsilon} \quad \text{where} \quad q = \frac{b}{N}$$

The computed σ is passed to the `DPSGD` constructor as `noise_scale`. Internally, the optimizer rescales this by C/b to determine the final per-coordinate standard deviation for noise injection.

In `transformer_model.py`, we replaced the default Adam optimizer with an instance of `DPSGD`. Additionally, we introduced a new import statement (`from optim import DPSGD`), added a `RUN_TAG` timestamp identifier (primarily for organizing experiment trial runs), and preserved the rest of the training loop while passing DP-specific parameters into the optimizer setup.

Unlike abstracted libraries like Opacus, our implementation requires explicit management of gradient clipping, noise injection, and privacy accounting within the training loop. Therefore, a limitation of this custom approach is that it does not include automatic privacy accounting or compatibility layers for more complex architectures. All privacy parameters (such as ε, δ, and σ) must be manually computed and passed, and the optimizer lacks enforcement of privacy bounds. In Section 4, we will compare the training behavior of this model against both the base non-private model and the Opacus-wrapped implementation.

3.3 Opacus DP-SGD Implementation

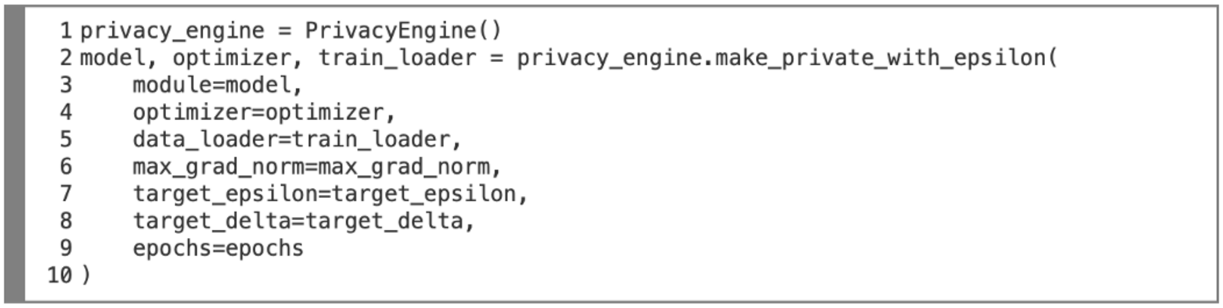
Opacus was chosen as the professional implementation of differential privacy (DP) for large language models (LLMs), as it is a leading open-source DP suite that integrates seamlessly with PyTorch tools employed in our base transformer model. Opacus was released in 2020 as "a new high-speed library for training PyTorch models with differential privacy (DP) that's more scalable than existing state-of-the-art methods"[12].

The library employs a `PrivacyEngine` abstraction to track the privacy budget and operate on model gradients by attaching to a standard PyTorch optimizer[16]. According to the Opacus documentation, DP researchers will find the implementation "easy to experiment and tinker with, allowing them to focus on what matters"[14].

However, as any third-party software engineering solution, Opacus suffers from limited flexibility and compatibility: DP with Opacus is straightforwardly implementable only on a restricted set of `nn.Module` architectures. To compute per-sample gradients, Opacus wraps model layers in a `GradSampleModule`, which uses backward hooks to capture activations and backpropagated gradients and run a custom `grad_sampler` function[13]. While effective, this hooks-based approach adds complexity and can conflict with certain transformer architectures, leading to unresolved bugs and ongoing compatibility efforts[3].

Opacus's core feature, the `PrivacyEngine`, provides two operational modes: the `make_private` method, which applies noise via the `noise_multiplier` parameter but yields inconsistent privacy budget accounting across epochs, and the `make_private_with_epsilon` method, which enforces a target ε-budget with automatic noise calibration[15]. However, wrapping bespoke training pipelines in the `PrivacyEngine` often requires adapting existing optimizers and data loaders beyond the library's advertised "out-of-the-box" simplicity[15].

To ensure fair comparison, we implemented a custom batching and transformer pipeline using PyTorch modules compatible with Opacus, preserving our base model's batching schemes and architecture. Under the hood, Opacus employs a Privacy Random Variable (PRV) accountant to compose privacy guarantees with arbitrary accuracy using privacy loss random variables[15].



```python
privacy_engine = PrivacyEngine()
model, optimizer, train_loader = privacy_engine.make_private_with_epsilon(
    module=model,
    optimizer=optimizer,
    data_loader=train_loader,
    max_grad_norm=max_grad_norm,
    target_epsilon=target_epsilon,
    target_delta=target_delta,
    epochs=epochs
)
```

Figure 1: Privacy Engine in Opacus implementation

3.4 Experimental Evaluation of DP and Non-DP Implementations

In our set of experiments, we compare the usability and performance of the professional Opacus implementation and our custom LLMSeuss-based DP-SGD. From a usability perspective, we continuously evaluate the ease of operation and the compatibility with common LLM pipeline components during our research. For performance, we benchmark both DP implementations against the baseline LLMSeuss model by training for 100 epochs across privacy budgets of $\varepsilon = 1, 10, 50$. We selected these values to assess the model utility across a spectrum of noise levels. For the aforementioned validation loss evaluation, we are using public-domain Seuss works by Theodor Seuss Geisel as the dataset. Training and validation losses will be recorded through training logs, then graphed to facilitate recognition of patterns. The qualitative assessment is discussed throughout the methodology sections when discussing the reasoning behind the training and evaluation pipeline decisions.

3.5 Canary Evaluation of DP and Non-DP Implementations

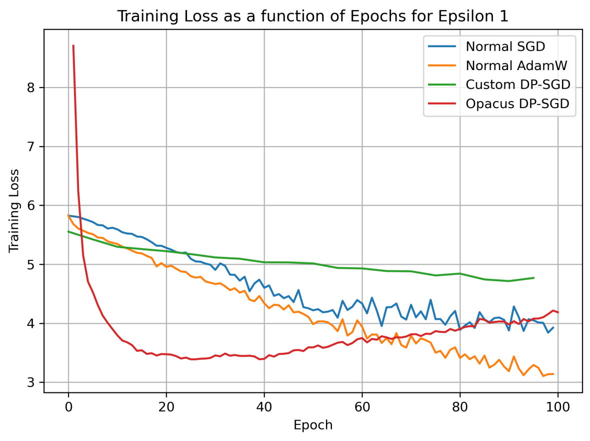
For targeted evaluation of differential privacy in the LLM domain, we utilize the *canary* technique. In our training data, we insert the phrase `The secret code is differentialPrivacy` 25 times, evenly distributed throughout the dataset. Our evaluation criterion is binary: if the model outputs the word `differentialPrivacy` within the first 50 generated tokens when prompted with `The secret code is`, we consider the canary to be *found*; otherwise, *not found*. To keep training costs reasonable, we adjusted the hyperparameters to enable more efficient canary discovery: the batch size was decreased to 1, the number of attention heads to 2, the dropout rate to 0.1, and the number of layers to 2. Training the baseline model for recollection of a canary secret proved to be significantly compute-heavy, requiring up to 10,000 epochs. While sufficient on its own, the compute and time requirements for 10,000 epochs of Opacus-DP model would be prohibitive. To decrease the time required to train, we altered the model to better hone-in on the canary and increase batch size to allow for more reasonable training time. Temperature parameter was also added to optimize the generation. Additionally, the frequency of canary sentences "The secret code is differentialPrivacy" was increased from the initial counts ranging 1-10 to 25 occurrences within the training Seuss works data. As a result, the baseline model was able to generate canary after 4,000 epochs.

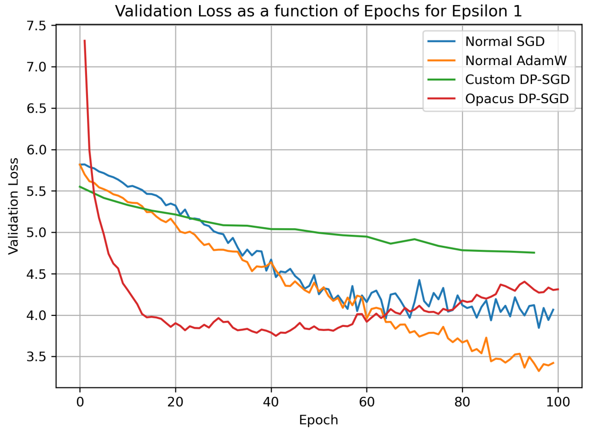
Opacus-DP compatibility limitations continued to select parameters that were outside of the norm. The model could not be trained for over 4500 epochs with any reasonable epsilon number, as we encountered the following warning: `RuntimeWarning: overflow encountered in exp z = np.log(np.where(t > np.log(1 - q), (np.exp(t) + q - 1) / q, 1))` . The root cause of the issue was the accountant choice for Opacus. While the privacy random variable (PRV) accountant, which is the default, has the tightest bounds, it can overflow in situations where the parameters are considered 'extreme.' The other accountants, Rényi DP composition or Gaussian DP approximation, were not used as compatibility issues ensued which we were not able to resolve in time due to time constraints. As a result, we optimized our baseline model to be able to achieve canary recognition with the number of epochs that Opacus with a PRV accountant would be able to train for, that is 4,000 epochs.

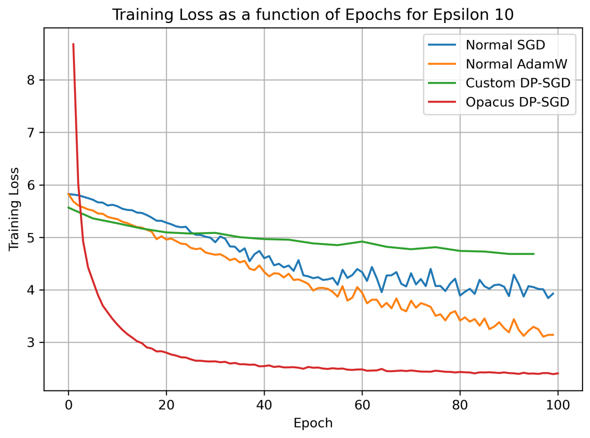
4 Results

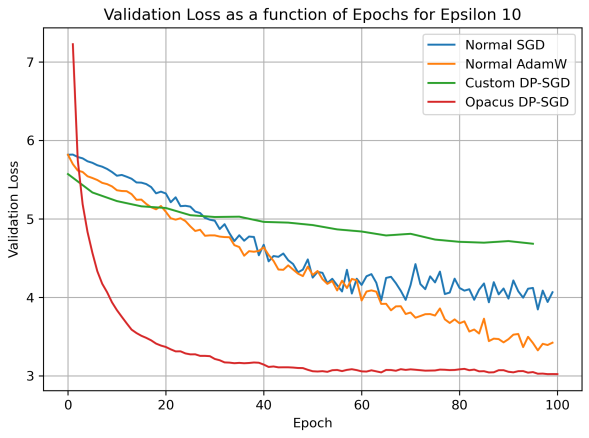
4.1 Training and Validation Loss across Differing Models

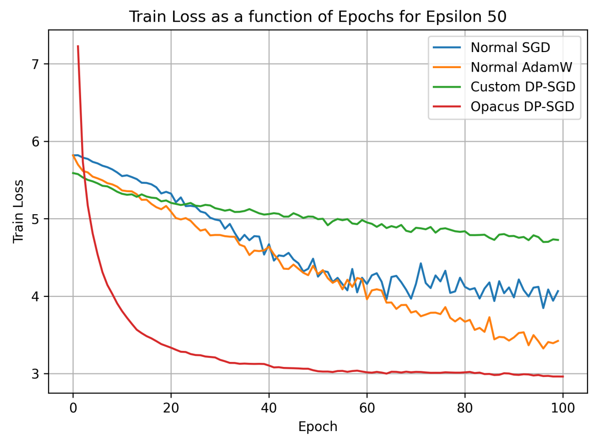
To assess the impact of differential privacy mechanisms on model performance, we analyze the training and validation loss trajectories across four optimization approaches: Normal SGD, Normal AdamW, our Custom DP-SGD implementation, and the Opacus DP-SGD implementation. As mentioned above in 2.4, we conduct three privacy budgets: $\varepsilon = 1$, 10, and 50, all using the same model architecture, dataset, and 100 training epochs. The figure below presents these results, grouped by ε value, with training and validation curves shown side-by-side for each setting.

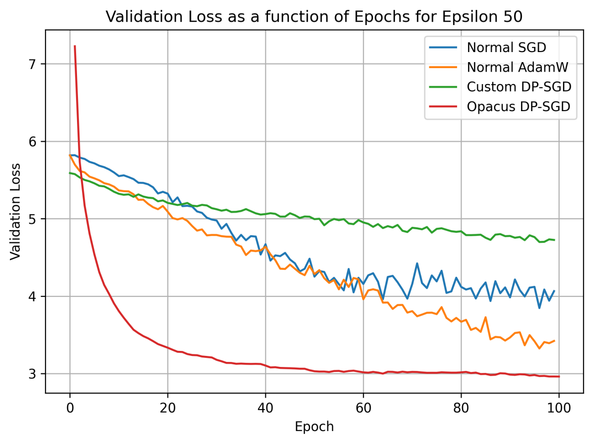












From the plots, a few key patterns emerge. First, AdamW outperforms all other models in both training and validation loss when $\varepsilon = 1$. This aligns with expectations: as an adaptive optimizer, AdamW uses dynamic learning rates and decoupled weight decay to achieve faster and more stable convergence, particularly in low-data or noisy scenarios.

Normal SGD trails AdamW but still performs reasonably well. However, Custom DP-SGD converges the slowest and to the highest loss values, a trend we see across all three privacy budgets. This is a direct result of per-example gradient clipping and the injection of Gaussian noise, both of which degrade gradient signal quality and slow optimization. Importantly, both operations are **non-adaptive**: the gradients are clipped to a fixed pre-determined ℓ_2 norm threshold regardless of context, and noise is drawn from a Gaussian distribution with a fixed standard deviation determined solely by the target privacy budget and batch size, not by the current state of training or the model's behavior of convergence.

Most interesting, however, is the behavior of the Opacus DP-SGD model, which converges extremely rapidly, even outperforming the non-private models in training loss at $\varepsilon = 10, 50$. We hypothesize that this phenomenon is the result of Opacus's internal hyperparameter tuning (adaptive learning rates) and efficient gradient accumulation mechanisms, which help mitigate the typical performance degradation associated with differential privacy. However, this comes at a cost: at $\varepsilon = 1$, we see the training loss for Opacus begins to plateau and even increase after early epochs. Here, while Opacus DP-SGD initially converges quickly, the excessive Gaussian noise eventually overwhelms the gradient signal. This leads to the model overshooting or drifting away from a local or global minimum it had previously found. In essence, Opacus's fast-converging dynamics generally help it reach a minimum quickly, but at low ε the optimizer lacks the stability to remain there due to the high noise, so the strong differential privacy guarantees impede the model's ability to converge upon a low loss value.

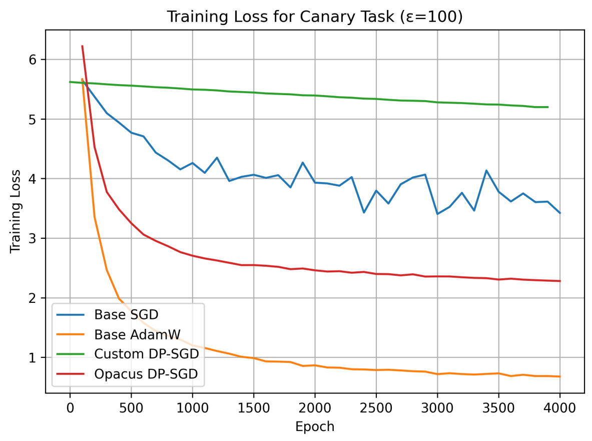
While training and validation loss curves slightly shift with increasing ε, the overall trends remain consistent. Most notably, model rankings tend to stabilize across privacy settings, with only minor deviations. We created a table of the relative performance of each optimizer below:

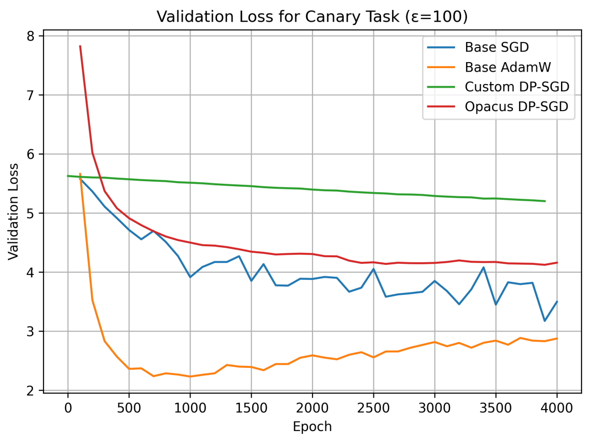
Table 1: Initial Model Ranking Across Privacy Budgets

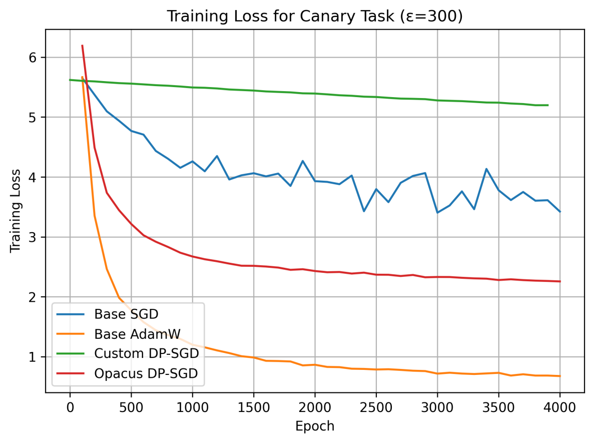
Privacy Budget (ε)	Initial Model Ranking (best to worst)
$\varepsilon = 1$	AdamW, SGD, Opacus, Custom DP-SGD
$\varepsilon = 10, 50$	Opacus, AdamW, SGD, Custom DP-SGD

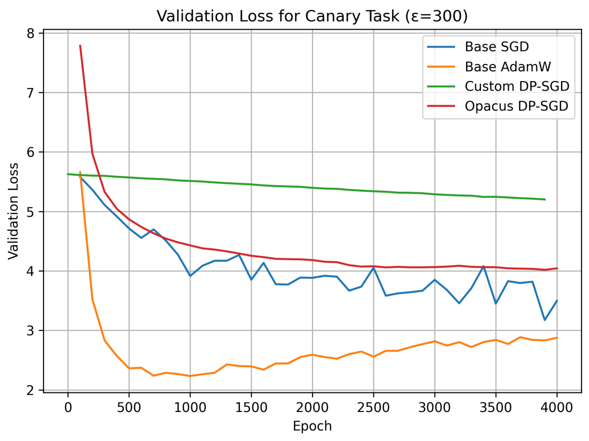
This ranking captures the general convergence behavior observed in our experiments. While Opacus shows exceptional training loss at all settings, its tendency to overfit at low ε leads to slightly poorer generalization performance, allowing AdamW and SGD to outperform it in validation loss under stricter privacy constraints.

Overall, the results demonstrate to us that DP mechanisms, especially in more naive non-professional implementations (aka hand-done by students), introduce nontrivial tradeoffs between privacy and utility. The Opacus model shows promise as a high-performance private training tool, but its aggressive optimization requires careful validation to avoid overfitting (more research needed!). Meanwhile, our Custom DP-SGD implementation prioritizes clarity at the expense of raw performance, serving as a transparent benchmark for evaluating the privacy–utility tradeoff (from an educational lens) but likely not be the best for efficacy.









4.2 Canary Evaluation

Table 2: Canary Disclosure Across Varying DP and Non-DP Models

Privacy Budget (ε)	SGD[†]	AdamW[†]	Custom-DP	Opacus
$\varepsilon = 100$	Safe	Disclosed	Safe	Safe
$\varepsilon = 300$	Safe	Disclosed	Safe	Safe

[†] Baseline models with non-DP optimizers unaffected by varying epsilon.

The table above shows the results of our canary disclosure for all four models across privacy budgets $\epsilon = 100$ and $\epsilon = 300$. After optimizing the base LLM as described in the methodology section, the AdamW optimizer baseline model correctly disclosed the secret canary after 4,000 epochs. However, the SGD optimizer baseline model failed to generate the secret canary. This can be seen due to the inability of the SGD model to converge to low loss levels, as can be seen in the graphs. For both of our DP models, we selected a privacy budget of ε 100 and 300, to allow for more noise due to a significant increase in training epochs as compared to our initial loss evaluation, which entailed 100 epochs. Custom DP model did not output the secret canary in both ε 100 and 300 budgets. Opacus DP model also did not output the secret canary in both ε 100 and 300 budgets. As such, we find that under DP optimizer implementations, even when models are trained extensively on sensitive information, here in the form of a canary, to the point of converging upon relatively low levels of loss, specific secret information remains concealed even with seemingly large privacy budgets. This result reinforces the strength of DP guarantees.

Most revealing, however, is the direct comparison between the AdamW baseline and the Custom DP-SGD implementation. Structurally, our Custom DP-SGD is derived from the AdamW framework (retaining its decoupled weight decay and overall update logic) but simply replaces the adaptive moment updates with per-example clipping and Gaussian noise. Essentially, the only difference between the two optimizers is the addition of DP mechanisms. Therefore, it's striking to see that this change alone is sufficient to prevent the model from memorizing and disclosing the canary, even after 4,000 epochs of training. By making the direct comparison, this suggests that DP-SGD's privacy protections stem not only from how it behaves during training, but also by structurally disabling the optimizer behaviors (like more granular signal retention) that enable memorization and thus leakage from our AdamW baseline in the first place.

In addition, we note that at 4,000 training epochs, the asymptotic loss convergence between the different models is substantially different from that which we observed when training for low epoch counts. In particular, when training for 4,000 epochs, we find that the model that converges upon the lowest loss is in fact, the AdamW base model, followed by the opacus model, followed by the SGD base model, followed by the custom DP SGD model. As such, we find that the addition of noise—as seen in Opacus—is more useful in overcoming local minima within a non-convex landscape towards the early phases of training. However, as the model begins to converge closer to the true minima, the noise in fact prevents the model from sensitively discovering this minima, resulting in an asymptotically higher loss than the base models.

We also note that while the validation loss starts to increase after 500 epochs for the AdamW base model—indicating model overfitting—we find that no such phenomenon occurs in the two DP models. In particular, this indicates that the use of DP SGD optimizers mitigate model overfitting, and increases the ability of the model to generalize. This observation is intuitive, and follows naturally from the definition of DP and the guarantees that it provides: in restricting the extent to which the training data can affect the model weights, as per DP guarantees, the DP SGD optimizers necessarily prevent overfitting on training data.

5 Conclusion

5.1 Future Works

This project reveals many interesting findings which warrants further evaluation. Of particular interest are the trends related to the Opacus model loss at low epsilon values. In particular, we find that the training loss in fact increases after reaching an initial minima for $\varepsilon = 1$. Further investigation on the nature of this training loss degradation would be of great interest. In particular, by extending the number of iterations, it may be possible to discover whether or not the increase in training loss is systematic, or—more likely–simply a result of random fluctuations around a converged minima, in which case the variance in the fluctuations could be characterized.

5.2 Final Words

DP SGD offers attractive guarantees on the extent to which any particular instance of training data can affect the weights of a transformer model. One may believe that these privacy guarantees come at the cost of performance, and this would not be incorrect; when attempting to converge upon low loss levels, non-DP variants (such as AdamW optimizers) asymptotically outperform the training loss and validation loss of even professional DP SGD models.

However, the addition of noise and the guarantees provided by DP SGD in fact also exhibit beneficial outcomes in terms of model performance. In particular, on training runs for which the number of epochs is low in comparison to the quantity of training data, DP SGD models can actually result in faster convergence, with the addition of gaussian noise preventing the model from falling into local minima. In addition, we find that when the number of epochs is larger, the privacy guarantees of DP-SGD in fact also guarantee that the model does not overfit on training data, which would disrupt performance during validation. As such, we find DP SGD to be a useful tool, that, although not without its drawbacks, may have potential use in specific data analysis and training situations.

Acknowledgments

We would like to express our sincerest gratitude to professors Salil Vadhan, James Honaker, and Priyanka Nanayakkara for the guidance and the insight they have imparted upon us through their graduate course on differential privacy. The pedagogical approach taken by these professors—combining theory with real-world examples and practical implementations—ultimately inspired the initial direction of our investigations. We also wish to extend our deepest thanks to Mr. Zachary Ratliff, whose feedback and suggestions were integral for shaping the approaches and experimentation conducted within this study.

With respects to the investigations and execution of this study, EG implemented the initial SGD and AdamW transformer model, OP adapted the initial transformer model with the Opacus DP-SGD optimizer, and MP adapted the transformer model with the custom DP-SGD optimizer. EG and MP carried out evaluations of the models across low ε privacy budgets, and OP proposed and implemented the canary-based evaluations of the various models. EG investigated the algorithmic and mathematical properties of DP SGD, drafting the provided explanations alongside background and introductory information. MP and OP drafted the methodological approaches for the low privacy budget and canary evaluations respectively. All authors contributed to the analysis and interpretation of results.

References

[1] Martin Abadi, Andy Chu, Ian Goodfellow, 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*, pages 308–318. ACM, 2016.

[2] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

[3] PyTorch Opacus Contributors. Opacus github repository. GitHub, 2024. `https://github.com/pytorch/opacus`.

[4] Cynthia Dwork and Aaron Roth. *The Algorithmic Foundations of Differential Privacy*, volume 9 of *Foundations and Trends in Theoretical Computer Science*. Now Publishers, 2014.

[5] Ferdinando Fioretto, Pascal Van Hentenryck, and Juba Ziani. Differential privacy overview and fundamental techniques, 2024.

[6] Valentin Hartmann, Anshuman Suri, Vincent Bindschaedler, David Evans, Shruti Tople, and Robert West. Sok: Memorization in general-purpose large language models, 2023.

[7] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors, 2012.

[8] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. Training compute-optimal large language models, 2022.

[9] Sijia Liu, Yuanshun Yao, Jinghan Jia, Stephen Casper, Nathalie Baracaldo, Peter Hase, Yuguang Yao, Chris Yuhao Liu, Xiaojun Xu, Hang Li, Kush R. Varshney, Mohit Bansal, Sanmi Koyejo, and Yang Liu. Rethinking machine unlearning for large language models, 2024.

[10] Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. Learning differentially private recurrent language models. In *International Conference on Learning Representations*, 2018.

[11] Tarun Ram Menta, Susmit Agrawal, and Chirag Agarwal. Analyzing memorization in large language models through the lens of model attribution, 2025.

[12] Meta AI. Introducing `Opacus`: A high-speed library for training `PyTorch` models with differential privacy. Blog post, 2020. `https://ai.meta.com/blog/introducing-opacus-a-high-speed-library-for-training-pytorch-models-with-differential-privacy/`.

[13] Meta Platforms, Inc. Gradsamplemodule. Opacus API Documentation, 2025. `https://opacus.ai/api/grad_sample_module.html`.

[14] Meta Platforms, Inc. Introduction. Opacus Documentation, 2025. `https://opacus.ai/docs/introduction`.

[15] Meta Platforms, Inc. Privacy engine. Opacus API Documentation, 2025. `https://opacus.ai/api/privacy_engine.html`.

[16] Meta Platforms, Inc. `Opacus`: Train pytorch models with differential privacy. Website, 2025. `https://opacus.ai`.

[17] Nicolas Papernot, Martin Abadi, Ulfar Erlingsson, Ian Goodfellow, and Kunal Talwar. Semi-supervised knowledge transfer for deep learning from private training data. In *International Conference on Learning Representations*, 2017.

[18] Venkatesh Balavadhani Parthasarathy, Ahtsham Zafar, Aafaq Khan, and Arsalan Shahid. The ultimate guide to fine-tuning llms from basics to breakthroughs: An exhaustive review of technologies, research, best practices, applied research challenges and opportunities, 2024.

[19] Baha Rababah, Shang, Wu, Matthew Kwiatkowski, Carson Leung, and Cuneyt Gurcan Akcora. Sok: Prompt hacking of large language models, 2024.

[20] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

[21] Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tie-Yan Liu. On layer normalization in the transformer architecture, 2020.