Implementing Adaptations for Vision AutoRegressive Model

Kaif Shaikh [1] **Antoni Kowalczuk** [1] **Franziska Boenisch** [1] **Adam Dziedzic** [1]

Abstract

Vision AutoRegressive model (VAR) was recently introduced as an alternative to Diffusion Models (DMs) in image generation domain. In this work we focus on its adaptations, which aim to fine-tune pre-trained models to perform specific downstream tasks, like medical data generation. While for DMs there exist many techniques, adaptations for VAR remain underexplored. Similarly, differentially private (DP) adaptations—ones that aim to preserve privacy of the adaptation data—have been extensively studied for DMs, while VAR lacks such solutions. In our work, we implement and benchmark many strategies for VAR, and compare them to state-of-the-art DM adaptation strategies. We observe that VAR outperforms DMs for non-DP adaptations, however, the performance of DP suffers, which necessitates further research in private adaptations for VAR. Code is available at https://github.com/sprintml/finetuning_var_dp.

1. Introduction

Recently, Vision AutoRegressive model (VAR) (Tian et al., 2024) has been proposed as a powerful alternative to Diffusion Models (DMs) (Rombach et al., 2022) in image generation. Yet, while for DMs, there exist multiple strong methods (Xie et al., 2023; Ruiz et al., 2023; Gal et al.) to adapt pre-trained models to specific downstream tasks, like medical data generation (Kazerouni et al., 2023), similar adaptations for VAR remain underexplored. Our work makes the first step in understanding and evaluating possible adaptations, ranging from full- to parameter-efficient fine-tuning (LoRA (Hu et al., 2022), LayerNorm (Zhao et al., 2023)), by implementing and benchmarking the methods on the class-conditioned VAR.

Since the adaptation data might consist of highly-sensitive samples, it is crucial that fine-tuned models do not leak privacy. One of the methods to prevent the leakage employs Differential Privacy (DP) (Dwork, 2006) to protect the vulnerable data. We explain how to overcome limitations of the VAR code base to implement privacy-preserving adaptations, then we implement and benchmark them.

We compare the performance of our adaptations of VAR to the existing SOTA DM adaptation—DiffFit—on five downstream datasets. We find that VAR adapts faster, is computationally more efficient, and outperforms DiffFit in generation quality. Yet, DP-adaptations of VAR suffer from low generation quality, as well as slow convergence. We revisit augmentation multiplicity (De et al., 2022), a strategy that improves the performance of DP models at a cost of higher compute, and find that it is beneficial for DP fine-tuning. However, the performance of DP-adaptations remains low, which prompts for further work in that domain.

Our released code with implementation of the methods and benchmarking serves to aid the researchers and practitioners to build and evaluate novel adaptation methods, and to close the gap between DMs and VARs in that domain.

2. Background and Related Work

Image AutoRegressive models (IARs) are a class of generative models that create images by modeling $p(x) = \prod_{n=1}^{N} p(x_n|x_{<n})$, where $x = (x_1, x_2, \ldots, x_N)$ is an image represented as a sequence of N tokens, x_n is the n-th token. First proposed by Chen et al. (2020) and further developed by Tian et al. (2024); Yu et al. (2024); Han et al. (2024) they offer better generation quality than their predecessor, namely DMs (Rombach et al., 2022).

IARs are trained to minimize $\mathcal{L}_{\text{AR}} = \mathbb{E}_{x\sim\mathcal{D}}\left[-\log\left(p(x)\right)\right]$, where $\mathcal{D}$ is a dataset of tokenized images. During generation, given a prefix $x_{<n}$ the model outputs per-token logits $p(x_n|x_{<n})$, from which we iteratively sample the next token to obtain the final image $\hat{x}$. In practice, the images are tokenized using a pre-trained VQ-GAN (Esser et al., 2020), and Transformer-based (Vaswani et al., 2017) architectures like GPT-2 (Radford et al., 2019) serve as the autoregressive backbone for modeling $p(x)$.

[1]CISPA Helmholtz Center for Information Security, Saarbrücken, Germany. Correspondence to: Kaif Shaikh <kaif.shaikh@cispa.de>, Antoni Kowalczuk <antoni.kowalczuk@cispa.de>, Franziska Boenisch <boenisch@cispa.de>, Adam Dziedzic <dziedzic@cispa.de>.

Published at Data in Generative Models Workshop: The Bad, the Ugly, and the Greats (DIG-BUGS) at ICML 2025, Vancouver, Canada.

Vision AutoRegressive (VAR) model is an IAR proposed by Tian et al. (2024), which inspired SOTA image generative model—Infinity (Han et al., 2024). VAR shifts the paradigm from next-token prediction to next-scale prediction. Instead of predicting 1D token sequences in raster-scan order (top to bottom, left to right), images are processed as sequences of 2D token grids, starting from lower to higher resolution. Effectively, VAR generates images quicker than classical IARs, requiring less predictions, as the 2D token grid for each scale can be sampled simultaneously.

Full Fine-Tuning (FFT) is an adaptation technique that updates every parameter of a model, given a fine-tuning dataset. Contrary to regular training, it is initialized from a pre-trained model (instead of from random initialization), and the end goal of FFT is to tailor the model to a specific task, based on a small, domain-specific dataset.

Parameter-Efficient Fine-Tuning (PEFT) helps fine-tune a model in a more compute-efficient manner by reducing the number of trainable parameters and memory usage as compared to FFT (Xu et al., 2023). It is widely used in adapting general foundation models, pre-trained on large datasets, to specific downstream tasks, where FFT is not required to achieve satisfactory performance.

Low-Rank Adaptation (LoRA) (Hu et al., 2022) is a PEFT method widely used for Transformer-based architectures. Such models have several dense layers with weight matrices ($\boldsymbol{W}$) of full rank. However, during fine-tuning, the updates ($\Delta\boldsymbol{W} \in \mathbb{R}^{d\times k}$) to these layers exhibit a low intrinsic rank (Aghajanyan et al., 2020). LoRA builds on that phenomenon, and attempts to decompose the update into multiplication of two low-rank matrices, $\Delta\boldsymbol{W} = \boldsymbol{B}\boldsymbol{A}$, where $\boldsymbol{B} \in \mathbb{R}^{d\times r}$ and $\boldsymbol{A} \in \mathbb{R}^{r\times k}$, and rank $r \ll \min(d, k)$. During training, only $\boldsymbol{B}$ and $\boldsymbol{A}$ are updated, and other parameters of the pre-trained model are frozen. Since the trained matrices are significantly smaller than the updated weights, LoRA significantly reduces the number of fine-tuned parameters.

LayerNorm Tuning (LNTuning) (Zhao et al., 2023) is a PEFT method widely used for transforming large language models (LLMs) to Multi-Modal LLMs (MLLMs). It transitions a text understanding model to other modalities by adding new trainable parameters to targeted LayerNorm modules inside each attention block. During fine-tuning, only the newly introduced parameters are updated, and all the model's weights are kept frozen. Similarly to LoRA, LNTuning significantly reduces the number of trainable parameters.

Differential Privacy (DP) is a mathematical framework used to bound the possible privacy leakage a mechanism exhibits. In context of machine learning, the goal is to quantify and limit the extent of privacy risks associated with the data used to train a model. If presence or absence of a specific data point, *e.g.,* an image, in the training set alters the behavior of the model significantly, then an adversary can learn about this data point from the model alone, which constitutes a privacy breach. To address this issue, DP provides guarantees on the level of impact a single data point can have on the model, giving an upper bound.

Specifically, we have two datasets, $\mathcal{D}$ and $\mathcal{D}'$ differing by a single data point, and a mechanism $\mathcal{M}$, acting on $\mathcal{D}$ and $\mathcal{D}'$. Let $\mathcal{S}$ be a set of all possible outputs of $\mathcal{M}$. We say $\mathcal{M}$ is (ϵ, δ)-DP if $Pr[\mathcal{M}(D) \in \mathcal{S}] \leq e^{\epsilon} Pr[\mathcal{M}(D') \in \mathcal{S}] + \delta$. Intuitively, the maximum possible difference of outputs of $\mathcal{M}$ for any $\mathcal{D}$ and $\mathcal{D}'$ is bounded by an exponential factor dependent on ϵ, and δ provides an optional safety margin.The smaller the value of ϵ, the stronger the privacy guarantees.

The go-to algorithm to train DP models is DP-SGD (Abadi et al., 2016)—a differentially private version of the regular SGD. DP-SGD first clips per-sample gradients, averages them, and adds Gaussian noise, obtaining privatized gradients. The parameters update at i^{th} iteration is: $\theta_{i+1} = \theta_i - \eta \left(\frac{1}{L} \sum_{k=1}^{L} \text{clip}(g(x_k)) + \mathcal{N}(0, \sigma^2 C^2 I) \right)$, where $\text{clip}(g(x_k)) = g(x_k)/max(1, \frac{||g(x_k)||_2}{C})$, η is the learning rate, C is the clipping norm, σ is the noise magnitude, $g(x_k)$ is the gradient for a single input data point x_k, L is the lot size and θ are the parameters.

DP Adaptations of Image Generative Models. The noise from DP-SGD harms the convergence. Augmentation multiplicity (De et al., 2022) addresses that by averaging per-sample gradients over multiple views to increase signal to noise ratio. For DMs, DPDM (Dockhorn et al., 2023) introduced noise multiplicity, averaging per-sample gradients over multiple input noise draws, which was further extended to augmentation multiplicity by (Ghalebikesabi et al., 2023). Since limiting the number of updated parameters also boosts performance, DP-LDM (Liu et al., 2024) shifts training into a lower-dimensional latent space via a non-private encoder.

3. Fine-Tuning of VAR

In the following we adapt VAR, a novel IAR, to specific downstream tasks via fine-tuning. We begin with a description of the models and datasets used, as well as the evaluation scheme. Then, we compare the performance of VAR to the one of SOTA adaptation method for DMs, DiffFit (Xie et al., 2023), and find that VAR outperforms its DM counterpart on five different datasets. Finally, we provide insights explaining observed discrepancy, and highlight key challenges we face during the implementation stage.

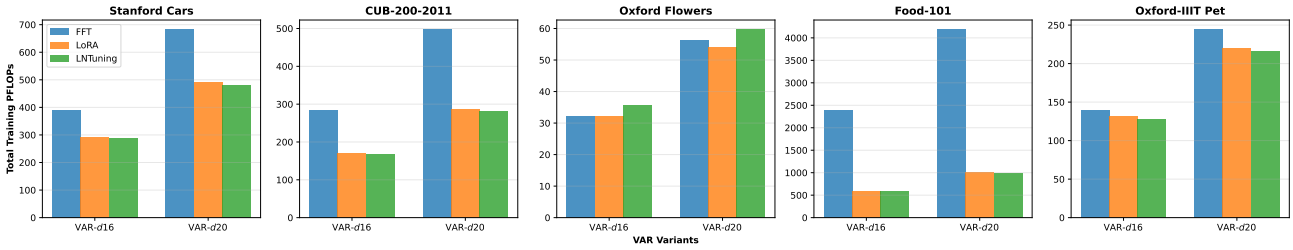

Figure 1: **Training Compute Cost (PFLOPs) Comparison Across Datasets.**

Table 1: **VAR fine-tuning outperforms DiffFit (Xie et al., 2023).** We compare FID (↓) on 5 downstream datasets between DiT-XL-2 and VAR-*d*16 and VAR-*d*20.

Model	Adaptation \ Dataset	Food-101	CUB-200-2011	Oxford Flowers	Stanford Cars	Oxford-IIIT Pet	Trainable Parameters
DiT-XL-2	FFT	10.46	5.68	21.05	9.79	-	673.8M (100%)
DiT-XL-2	LoRA	34.34	58.25	161.68	75.35	-	2.18M (0.32%)
DiT-XL-2	DiffFit (Xie et al., 2023)	6.96	5.48	20.18	9.90	-	0.83M (0.12%)
VAR *d*16	FFT	**6.11**	**5.74**	**12.08**	**7.42**	13.13	309.6M (100%)
VAR *d*16	LoRA	**6.94**	7.84	**13.18**	**8.87**	13.70	6.02M (1.91%)
VAR *d*16	LNTuning	8.01	8.15	22.82	**9.27**	14.28	100.7M (24.56%)
VAR *d*20	FFT	**5.38**	**5.58**	**11.65**	**6.31**	12.81	599.7M (100%)
VAR *d*20	LoRA	6.97	6.29	**11.16**	**9.42**	12.97	9.42M (1.54%)
VAR *d*20	LNTuning	7.00	6.07	**12.74**	**7.36**	12.86	196.7M (24.69%)

3.1. Experimental Setup

Models: We perform evaluation on VAR-*d*16 and VAR-*d*20 pre-trained on ImageNet-1k (Deng et al., 2009) to perform class-conditional image generation in 256x256 resolution, and are sourced from the repositories provided by their respective authors. We restrict ourselves to these two models, because they are of comparable sizes to DiT-XL-2 (Peebles & Xie, 2023). In Appendix C we provide results for bigger models: VAR-*d*24 and VAR-*d*30.

Datasets: As our downstream task, we focus on image generation in narrow domains, and fine-tune the models to perform well with small datasets. To this end, we use *Food-101* (Bossard et al., 2014) with 101k total images of 101 different categories of food, *CUB-200-2011* (Wah et al., 2011) of 11,778 images of 200 bird classes, *Oxford Flowers* (Nilsback & Zisserman, 2008) consisting of 1020 images of 102 species of flowers, *Stanford Cars* (Krause et al., 2013) with 16,185 images of 196 classes of cars, and *Oxford-IIIT Pet Dataset* (Parkhi et al., 2012) with 7,393 images of 37 different breeds of cats and dogs. More details about the datasets, *e.g.,* size of the training and validation sets, can be found in Appendix B.

Adaptations and Hyperparameters: In our study we use LoRA and LNTuning as PEFT methods, and compare them to FFT. For LoRA, we use rank $r = 16$, $\alpha = 2r$, and LoRA dropout of 0. For LoRA fine-tuning we target self-attention modules, *i.e.,* the query, key, and value matrices. Additionally, we fine-tune the projection layers of self-attention, and the Adaptive LayerNorm modules. For LNTuning, we update only the Adaptive LayerNorm modules. FFT updates all parameters of the model. We provide extended details in Appendix B.

Performance Metrics: We measure Fréchet Inception Distance (FID) (Heusel et al., 2017) to quantify generation quality. For each dataset we generate as many samples as present in each class of the respective test sets, and compute FID between images in generated set and test set. Moreover, we compute the computation cost of fine-tuning, expressed in Peta Floating Point Operations (PFLOPs). Full evaluation setup can be found in Appendix B.

3.2. Empirical Results

Compute Cost Comparison. In Figure 1 we show that across all datasets and VAR variants, FFT incurs the highest compute cost, with the exception of Oxford Flowers, where the cost is similar for all adaptations. The biggest difference is visible for Food-101, where FFT requires around $4.5\times$ more compute than parameter-efficient counterparts. Interestingly, when the size of dataset increases (Food-101 has 101k samples, Oxford Flower only 1020), the difference in cost between FFT and PEFT also increases. Among the two PEFT techniques, LNTuning exhibits a slightly smaller compute footprint than LoRA.

Performance Comparison. While we observe significant differences in compute cost between the methods, we should take other factors into consideration, *e.g.,* convergence speed, or the final generation quality. Importantly, LNTuning is the cheapest, while FFT performs best (lowest FID scores), according to the results in Table 1. LoRA strikes the middle ground between these methods: it matches FFT's FID performance, and is similarly demanding to LNTuning in terms of compute.

VAR fine-tuning achieves better performance than DiffFit, with FFT outperforming it across all models and datasets. It suggests that the adaptations of IARs might gain significance as the field progresses. We provide additional metrics in Appendix C.

VARs Converge Quickly. During our experiments we notice that VAR needs very few update steps to reach its final performance. This contrasts with the behavior observed for DMs, which tend to require extended training, which involves stochastic input noise. We investigate the convergence behavior, and in Figure 2 we show that the models achieve their final FID after few thousand steps. Interestingly, LoRA fine-tuning converges similarly fast to FFT, while LNTuning needs more update steps. We provide more insights on why VAR is faster in Appendix E.

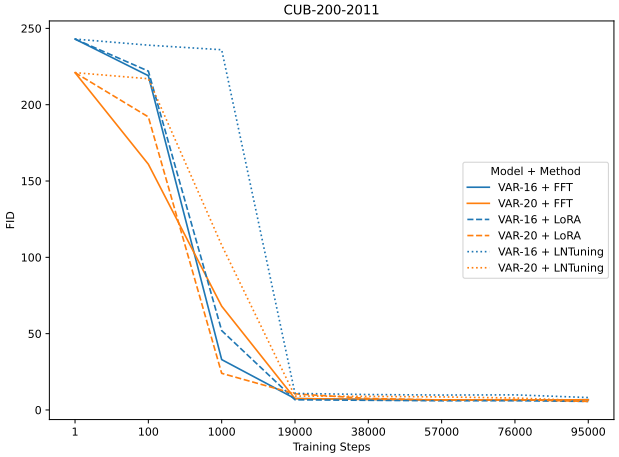

Figure 2: **VARs converge after small amount of training steps.** Dataset: CUB-200-2011.

Implementation Details The original implementation of the attention operator in VAR requires patches (Table 11) to introduce LoRA adapters. We provide more details in Appendix F.

4. Differentially-Private Adaptation for VAR

Next, we switch our focus to DP adaptations for VAR, whose goal is to preserve the privacy of potentially sensitive fine-tuning data. We show the impact of augmentation multiplicity, model's size, fine-tuning strategy, and the privacy budget ϵ on the generation quality.

4.1. Experimental Setup

Due to high computation cost of fine-tuning with DP, we only fine-tune the models on the Oxford Flowers dataset, and investigate LoRA and LNTuning adaptations. For augmentation multiplicity we use the default pre-processing image transformation of VAR, with parameter k denoting how many views per sample we craft.

4.2. Empirical Evaluation

Performance. Contrary to promising results in Section 3.2, when we fine-tune with DP, the models fail to converge. Our results in Table 3 show that we need extremely high values of ϵ to obtain acceptable generation quality. Interestingly, we observe that augmentation multiplicity yields only modest improvements, with results for LoRA in Table 2 indicating negligible gains at $k = 128$. Notably, the compute cost for $k = 128$ is around 128 times greater for $k = 1$, and increasing k further might be prohibitively expensive. LoRA appears to outperform LNTuning, according to Table 2, which might be due to the lower number of trainable parameters for LoRA (see Table 1).

Table 2: **FID (↓)** of VAR models fine-tuned using LoRA and DP with $\epsilon = 10$ on Oxford Flowers Dataset.

Model	LoRA $k = 1$	LoRA $k = 128$	LNTuning $k = 1$
VAR-$d16$	69.92	63.24	106.32
VAR-$d20$	68.92	59.29	98.2

Table 3: **FID (↓)** of VAR models fine-tuned using LoRA of varying ϵ on Oxford Flowers Dataset with augmentation multiplicity parameter $k = 32$.

Model	$\epsilon = 1$	$\epsilon = 10$	$\epsilon = 20$	$\epsilon = 50$	$\epsilon = 100$	$\epsilon = 500$	$\epsilon = 1000$
VAR-$d16$	196.52	60.24	52.10	46.36	41.63	35.70	35.36
VAR-$d20$	160.33	63.38	53.73	47.09	43.35	37.35	35.06

Implementation. Similarly as for LoRA adaptations, tailoring the original VAR implementation for DP fine-tuning requires patches. We identify issues with model-specific buffers and a non-standard forward function. We provide more details in Appendix F.

5. Conclusions

With the success of our adaptations of the VAR model, we expect the broader adoption of IARs tailored specific domains for image generation. Our work is the first step in the direction of IAR-specific model adaptations, and we already observe that fine-tuned VAR performs better than the SOTA adaptation strategy for DMs. Importantly, DP adaptations remain ineffective, which indicates that further research in that direction is required to enable privacy-preserving adaptations on sensitive data.

Acknowledgments

This project was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation), Project number 550224287, and by the Helmholtz Impulse and Networking Fund as part of the project "Effective Privacy-Preserving Adaptations of Foundation Models for Medical Tasks", reference number ZT-I-PF-5-227.

References

The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*.

Abadi, M., Chu, A., Goodfellow, I., McMahan, H. B., Mironov, I., Talwar, K., and Zhang, L. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS'16. ACM, October 2016. doi: 10.1145/2976749.2978318. URL http://dx.doi.org/10.1145/2976749.2978318.

Aghajanyan, A., Zettlemoyer, L., and Gupta, S. Intrinsic dimensionality explains the effectiveness of language model fine-tuning, 2020. URL https://arxiv.org/abs/2012.13255.

Bossard, L., Guillaumin, M., and Van Gool, L. Food-101–mining discriminative components with random forests. In *Computer vision–ECCV 2014: 13th European conference, zurich, Switzerland, September 6-12, 2014, proceedings, part VI 13*, pp. 446–461. Springer, 2014.

Chen, M., Radford, A., Child, R., Wu, J., Jun, H., Luan, D., and Sutskever, I. Generative pretraining from pixels. In *International conference on machine learning*, pp. 1691–1703. PMLR, 2020.

De, S., Berrada, L., Hayes, J., Smith, S. L., and Balle, B. Unlocking high-accuracy differentially private image classification through scale. *arXiv preprint arXiv:2204.13650*, 2022.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Dockhorn, T., Cao, T., Vahdat, A., and Kreis, K. Differentially private diffusion models, 2023. URL https://arxiv.org/abs/2210.09929.

Dwork, C. Differential privacy. In Bugliesi, M., Preneel, B., Sassone, V., and Wegener, I. (eds.), *Automata, Languages and Programming*, pp. 1–12, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. ISBN 978-3-540-35908-1.

Esser, P., Rombach, R., and Ommer, B. Taming transformers for high-resolution image synthesis, 2020.

Gal, R., Alaluf, Y., Atzmon, Y., Patashnik, O., Bermano, A. H., Chechik, G., and Cohen-or, D. An image is worth one word: Personalizing text-to-image generation using textual inversion. In *The Eleventh International Conference on Learning Representations*.

Ghalebikesabi, S., Berrada, L., Gowal, S., Ktena, I., Stanforth, R., Hayes, J., De, S., Smith, S. L., Wiles, O., and Balle, B. Differentially private diffusion models generate useful synthetic images. *arXiv preprint arXiv:2302.13861*, 2023.

Han, J., Liu, J., Jiang, Y., Yan, B., Zhang, Y., Yuan, Z., Peng, B., and Liu, X. Infinity: Scaling bitwise autoregressive modeling for high-resolution image synthesis, 2024. URL https://arxiv.org/abs/2412.04431.

Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., and Hochreiter, S. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Conference on Neural Information Processing Systems (NeurIPS)*, pp. 6629–6640, 2017.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., Chen, W., et al. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3, 2022.

Kazerouni, A., Aghdam, E. K., Heidari, M., Azad, R., Fayyaz, M., Hacihaliloglu, I., and Merhof, D. Diffusion models in medical imaging: A comprehensive survey. *Medical Image Analysis*, 88:102846, 2023. ISSN 1361-8415. doi: https://doi.org/10.1016/j.media.2023.102846. URL https://www.sciencedirect.com/science/article/pii/S1361841523001068.

Krause, J., Stark, M., Deng, J., and Fei-Fei, L. 3d object representations for fine-grained categorization. In *2013 IEEE International Conference on Computer Vision Workshops*, pp. 554–561, 2013. doi: 10.1109/ICCVW.2013.77.

Kynkäänniemi, T., Karras, T., Laine, S., Lehtinen, J., and Aila, T. Improved precision and recall metric for assessing generative models. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/0234c510bc6d908b28c70ff313743079-Paper.pdf.

Liu, M. F., Lyu, S., Vinaroz, M., and Park, M. Differentially private latent diffusion models, 2024. URL https://arxiv.org/abs/2305.15759.

Nilsback, M.-E. and Zisserman, A. Automated flower classification over a large number of classes. In *Indian Conference on Computer Vision, Graphics and Image Processing*, Dec 2008.

Parkhi, O. M., Vedaldi, A., Zisserman, A., and Jawahar, C. Cats and dogs. In *2012 IEEE conference on computer vision and pattern recognition*, pp. 3498–3505. IEEE, 2012.

Paszke, A. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*, 2019.

Peebles, W. and Xie, S. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 4195–4205, 2023.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. Language models are unsupervised multitask learners. 2019.

Rombach, R., Blattmann, A., Lorenz, D., Esser, P., and Ommer, B. High-resolution image synthesis with latent diffusion models. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10674–10685, 2022. doi: 10.1109/CVPR52688.2022.01042.

Ruiz, N., Li, Y., Jampani, V., Pritch, Y., Rubinstein, M., and Aberman, K. Dreambooth: Fine tuning text-to-image diffusion models for subject-driven generation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 22500–22510, 2023.

Sajjadi, M. S. M., Bachem, O., Lucic, M., Bousquet, O., and Gelly, S. Assessing generative models via precision and recall, 2018. URL https://arxiv.org/abs/1806.00035.

Tian, K., Jiang, Y., Yuan, Z., Peng, B., and Wang, L. Visual autoregressive modeling: Scalable image generation via next-scale prediction, 2024. URL https://arxiv.org/abs/2404.02905.

Tsai, Y.-L., Li, Y., Chen, Z., Chen, P.-Y., Yu, C.-M., Ren, X., and Buet-Golfouse, F. Differentially private fine-tuning of diffusion models, 2024. URL https://arxiv.org/abs/2406.01355.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Wah, C., Branson, S., Welinder, P., Perona, P., and Belongie, S. The caltech-ucsd birds-200-2011 dataset. 2011.

Xie, E., Yao, L., Shi, H., Liu, Z., Zhou, D., Liu, Z., Li, J., and Li, Z. Difffit: Unlocking transferability of large diffusion models via simple parameter-efficient fine-tuning, 2023. URL https://arxiv.org/abs/2304.06648.

Xu, L., Xie, H., Qin, S.-Z. J., Tao, X., and Wang, F. L. Parameter-efficient fine-tuning methods for pretrained language models: A critical review and assessment, 2023. URL https://arxiv.org/abs/2312.12148.

Yu, Q., He, J., Deng, X., Shen, X., and Chen, L.-C. Randomized autoregressive visual generation. *arXiv preprint arXiv:2411.00776*, 2024.

Zhao, B., Tu, H., Wei, C., Mei, J., and Xie, C. Tuning layernorm in attention: Towards efficient multi-modal llm finetuning, 2023. URL https://arxiv.org/abs/2312.11420.

A. Additional Related Work

A.1. Properties of Differential Privacy

DP has the properties of group privacy, composability, and robustness to auxiliary information. Group privacy ensures graceful degradation of privacy guarantees when datasets have correlated inputs. If a dataset has k such points, then an (ϵ, δ) algorithm run on such a dataset yields ($k\epsilon$, $ke^{(k-1)\epsilon}\delta$)-DP (Dwork, 2006). Composability ensures that if in a mechanism, each of the components are differentially private then the mechanism is differentially private too. Robustness to auxiliary information implies that even with the auxiliary knowledge, DP guarantees hold against an adversary. A common approach to implement DP guarantees is to add noise, which magnitude depends on the sensitivity of the mechanism. Sensitivity of a mechanism is defined as the maximum distance between outputs of the mechanism for any two adjacent datasets. Therefore a mechanism f can be made differentially private using Gaussian noise as follows:

$$\mathcal{M}(d) = f(d) + \mathcal{N}(0, \Delta f^2 \sigma^2)$$

where Δf is the sensitivity of mechanism f and σ is the noise scale. Such a mechanism is called a *Gaussian mechanism*.

A.2. Differentially Private Diffusion Models

DPDM. Dockhorn et al. (2023) begin from a fact that the gradients at each update step are prone to high variance, because the training objective relies on the forward process, which adds noise with a varying magnitude. By default, this issue is alleviated by training for many iterations at a small batch size. However, DP-SGD computes privacy loss on a per-update basis, which means such an approach is not feasible. The non-private DM loss can be expressed as

$$l_i = \lambda(\sigma_i)||z_\theta(x_i + n_i, \sigma_i) - x_i||_2^2 \tag{1}$$

where, l_i is the loss, $x_i + n_i$ is the noisy input passed to the DM, λ is a weighting function and σ_i is the noise scale for the forward process. DPDM tackles the problem of l_i using the *noise multiplicity*, which replaces the DM objective function over one noisy sample, with the average of K noisy samples for each data point. Effectively, the new loss function can be represented as

$$\tilde{l_i} = \frac{1}{K}\sum_{k=1}^{K} \lambda(\sigma_{ik})||D_\theta(x_i + n_{ik}, \sigma_{ik}) - x_i||_2^2 \tag{2}$$

The results show a significant improvement of the performance after incorporating this mechanism to the private training. The gain from the noise multiplicity plateaus around $K = 32$.

In addition to the noise multiplicity, authors use exponential moving average to update the weights of the DM after each training step, and very large batch sizes (4096 for MNIST (lec). They also note that biasing the forward process towards higher timesteps is beneficial for the training.

Ghalebikesabi et al. (2023) show that by sampling an augmentation in addition to the timestep at each repetition in the noise multiplicity mechanism yields an improvement. They dub that approach *augmentation multiplicity*. They find that $K = 128$ yields the best results, with batch size of $16,384$. Similarly to DPDM, they sample the timesteps from a distribution different than the default $\mathcal{U}[0, T]$, $t \sim \sum_{i=1}^{K} w_i \mathcal{U}[l_i, u_i]$, where $\sum_{i=1}^{K} w_i = 1$, $0 \leq l_0, u_K \leq T$, and $u_{k-1} \leq l_k$ for $k \in \{2, \ldots, K\}$. The main difference between their work and DPDM is that they first pre-train the DM using "public" dataset without DP, and only then fine-tune on the private dataset, this time with DP.

DP-LDM. Liu et al. (2024) focus on the impact of the size of the DM on the properties of DP-SGD. Intuitively, the bigger the size of the model, the bigger the norm of the gradient, in effect the bigger the noise added by the DP-SGD mechanism. Thus, a natural idea is to limit the size of the DM. To this end, authors utilize Latent Diffusion Models (LDMs) (Rombach et al., 2022), which consist of of a VAE and a DM. VAEs are made of an encoder, which takes a high-dimensional image in the pixel space as an input, and produces a lower-dimensional latent representation, and a decoder that converts the latent back to the pixel space. Effectively, the DM is trained in the latent space, and in turn contains less parameters than a pixel-space alternative. DP-LDM first trains the VAE (without DP), and, similarly to (Ghalebikesabi et al., 2023), pre-trains the DM on "public" dataset. To further benefit from the smaller gradient norms, they fine-tune with DP only the (cross-)attention modules of the DM, which amounts to 10% of the parameters.

DP-LoRA. Tsai et al. (2024) improve over DP-LDM by utilizing a PEFT method–LoRA–to limit the gradient norms by reducing the number of trained parameters of the DM even further. LoRA adapters are applied to the QKV matrices of the attention blocks of the LDM, and the output projection layer following them.

B. Extended Experimental Details

Additional information about the datasets. All variants of VAR are pre-trained on ImageNet 256x256 (Deng et al., 2009), we finetune them using the datasets mentioned in Table 4. We use the same datasets for regular finetuning and DP-finetuning as well. All experiments in paper leverage the entire train split for finetuning. We use 100% of the test split when benchmarking the results for the experiments.

Table 4: Different datasets used across our experiments with VARs.

Dataset	Num. Classes	Training Set Size	Testing Set Size
CUB-200-2011 (Wah et al., 2011)	200	5994	5794
Food-101 (Bossard et al., 2014)	101	75750	25250
Oxford Flowers (Nilsback & Zisserman, 2008)	102	1020	6149
Oxford-IIIT Pet (Parkhi et al., 2012)	37	3680	3669
Stanford Cars (Krause et al., 2013)	196	8144	8041

Hyperparameters used in the experiments (Non-Private finetuning). We apply identical hyperparameter settings across all variants of each model: VAR-$d\{16, 20, 24, 30\}$. Larger variants typically require fewer epochs and allow larger batch sizes. The hyperparameters for VARs are given in Table 5, Table 6 and Table 7.

Table 5: Hyperparameters for FFT.

Dataset	Learning Rate	Batch Size	Epochs
CUB-200-2011	1×10^{-4}	512	50
Food-101	1×10^{-4}	256	30
Oxford Flowers	1×10^{-4}	256	24
Oxford-IIIT Pet	1×10^{-4}	256	40
Stanford Cars	1×10^{-4}	256	50

Table 6: Hyperparameters for LoRA.

Dataset	Learning Rate	Batch Size	LoRA Rank	Epochs
CUB-200-2011	1×10^{-3}	512	16	40
Food-101	5×10^{-4}	256	16	6
Oxford Flowers	5×10^{-4}	256	16	40
Oxford-IIIT Pet	5×10^{-4}	256	16	50
Stanford Cars	5×10^{-4}	256	16	50

Table 7: Hyperparameters for LNTuning.

Dataset	Learning Rate	Batch Size	Epochs
CUB-200-2011	1×10^{-3}	512	40
Food-101	5×10^{-4}	256	6
Oxford Flowers	5×10^{-4}	256	45
Oxford-IIIT Pet	5×10^{-4}	256	50
Stanford Cars	5×10^{-4}	256	50

Hyperparameters for private finetuning. Differentially-private fine-tuning typically requires more epochs to offset the utility loss from added noise, so we increase the epoch count accordingly. All DP-finetuning experiments are conducted exclusively on the Oxford Flowers dataset. We set the total batch size BS_{total} from the sampling rate q and total examples N as:

$$BS_{\text{total}} = \lfloor q \times N \rfloor,$$

and in a multi-GPU (DDP) run with G GPUs the effective batch size per optimizer step is:

$$BS_{\text{device}} = \frac{BS_{\text{total}}}{G}.$$

Table 8: Hyperparameters for DP-finetuning (DP-$\varepsilon - 10$) on Oxford Flowers for VAR.

Method	Hyperparameter	Value
FFT	Learning Rate	1×10^{-4}
	Sample Rate (q)	0.251
	Delta (δ)	$\frac{1}{N}$
	Epochs	180
	Max Grad Norm	0.1
	Augmentation Multiplicity (k)	128
LoRA	Learning Rate	5×10^{-4}
	Sample Rate (q)	0.251
	Delta (δ)	$\frac{1}{N}$
	LoRA Rank	16
	Epochs	80
	Max Grad Norm	0.5
	Augmentation Multiplicity (k)	128
LNTuning	Learning Rate	5×10^{-4}
	Sample Rate (q)	0.251
	Delta (δ)	$\frac{1}{N}$
	Epochs	100
	Max Grad Norm	0.5
	Augmentation Multiplicity (k)	128

Profiling We use built-in profiler module inside PyTorch (Paszke, 2019) for computing the total training cost for each of our experiment. The designated profiling script utilizes the *PyTorch Profiler* to aggregate the total number of FLOps for one single training step, this involves computing the total cost for 2 events inside the training step, forward pass and the

backward pass. Finally, we calculate the cost for the whole run, based on the effective batch size and number of epochs multiplied by the cost of each step to obtain the final compute cost in PFLOps.

Compute Cost Profiling for DP-experiments Unlike the non-private baselines, each differentially-private (DP) experiment employs *Poisson subsampling*: at every update each example $i \in \{1, \dots, N\}$ is drawn independently with probability $q = \frac{\text{batch\_size}}{N}$. Because $|B_t| \sim \text{Binomial}(N, q)$, the FLOps for each update inherit a coefficient of variation $\frac{1}{\sqrt{qN}}$; hence multiplying a single-step profile by the expected step count yields only a coarse approximation of the total compute budget. Profiling only the *first* update and multiplying by an *expected* step count therefore produces a point estimate that is systematically biased and may lead to incorrect calculations. To avoid reporting a misleadingly precise figure, we refrain from quoting aggregate PFLOp totals for DP experiments and instead restrict compute-cost analysis to the deterministic, fixed-batch baselines.

C. Extended Experimental Results for Non-Private Experiments

Extended Metrics for Performance. Following (Sajjadi et al., 2018; Kynkäänniemi et al., 2019), we evaluate VAR-*d*16 and VAR-*d*20 to estimate a local k–NN manifold for both the real and the generated distributions and report **Precision** and **Recall** in Table 9. Precision (↑) is the fraction of generated samples that fall inside the real manifold and thus quantifies *sample fidelity*. Recall (↑) is the fraction of real samples that fall inside the generated manifold and therefore measures *coverage/diversity*.

Table 9: Extended Precision (**P**) and Recall (**R**) of VAR-*d*16 and VAR-*d*20 models across five downstream datasets.

Model	Adaptation	Food-101		CUB-200-2011		Oxford Flowers		Stanford Cars		Oxford-IIIT Pet		Trainable Parameters
		P	R	P	R	P	R	P	R	P	R	
VAR-*d*16	FFT	73.48%	7.65%	61.99%	58.40%	59.45%	26.57%	58.40%	40.03%	68.49%	2.67%	309.6M (100%)
VAR-*d*16	LoRA	66.89%	9.70%	63.15%	58.16%	61.68%	26.83%	47.24%	31.20%	69.77%	4.52%	6.02M (1.91%)
VAR-*d*16	LNTuning	67.67%	8.43%	48.15%	62.44%	42.57%	34.85%	43.72%	41.38%	57.23%	4.33%	100.7M (24.56%)
VAR-*d*20	FFT	73.63%	8.07%	74.74%	47.99%	64.17%	24.84%	59.01	39.68%	69.28%	2.28%	599.7M (100%)
VAR-*d*20	LoRA	68.90%	9.36%	68.24%	54.26%	59.64%	32.18%	58.97%	39.49%	68.08%	3.08%	9.42M (1.54%)
VAR-*d*20	LNTuning	69.24%	10.03%	61.23%	58.08%	59.27%	31.71%	51.11%	44.58%	65.46%	3.40%	196.7M (24.69%)

Recall values dip sharply for *Food-101* (max. $\leq 10\%$) and *Oxford-IIIT Pet* (all methods $< 5\%$). Both datasets are (i) *fine-grained*, demanding the model to cover hundreds of subtly different classes, and (ii) *out-of-domain* for the ImageNet-trained encoder used by the metric, which clusters visually similar species into overly tight neighbourhoods. Under these conditions the k-NN test rejects many legitimate but stylistically rare samples, driving recall down even for FFT.

FLOPs Compute Cost Analysis. We show extended FLOPs compute-cost analysis for VAR-*d*{16, 20, 24, 30} in Figure 3. These results expand upon our primary evaluation, which originally included only the VAR-*d*16 and VAR-*d*20 models. By adding the larger VAR-*d*24 and VAR-*d*30 variants, we provide a full picture of how scaling the autoregressive backbone affects total PFLOPs for each adaptation method.

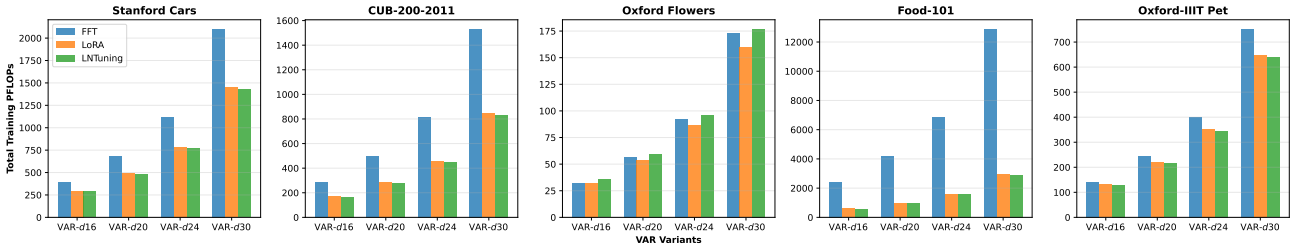

Figure 3: Extended Training Compute Cost (PFLOPs) Comparison Across Datasets.

The plots clearly show that LoRA and LNTuning deliver greater compute savings as model size grows. While for FFT the PFLOPs increase steeply from VAR-*d*16 through VAR-*d*30, both PEFT methods remain comparatively flat—especially on

large datasets like Food-101 and Stanford Cars. All compute costs were calculated using the identical hyperparameters specified in Tables 5 to 7, reinforcing that PEFT is the most scalable strategy under a fixed compute budget.

On the Oxford Flowers dataset, both LoRA and LNTuning were scheduled for more epochs than FFT (*e.g.,* 30 vs. 24), yet their total PFLOPs remain lower. This occurs because the per-step overhead of PEFT methods is substantially smaller than that of FFT, so even with an extended training schedule, the aggregate compute cost stays below FFT's. Consequently, PEFT retains its efficiency advantage on Flowers despite requiring more epochs.

D. Ablation Study

LoRA outperforms LNTuning in DP experiments.

We check what happens with the performance of VAR when we select different parameters to fine-tune with DP-SGD. To this end, we run a controlled ablation on VAR-$d16$ using the Oxford Flowers dataset. We use the same hyperparameters as in Appendix B. We keep LoRA rank fixed at 16 and vary which weight blocks receive the low-rank adapters: (i) LoRA-A updates only the attention projections W_{qkv} and the output projections; (ii) LoRA-M updates only the MLP projections fc_1 and fc_2; (iii) LoRA-AM + LN combines (i) + (ii) and additionally includes all LayerNorm weights; Finally, (iv) we include LNTuning as the baseline that trains an adapter for LayerNorm only. Effectively, we ablate over the capacity (number of parameters to fine-tune) and layers to fine-tune. Table 10 reports FID and number of trainable parameters for each variant.

Table 10: Effect of selectively finetuning specific components of VAR-$d16$ with DP $\varepsilon = 10$ on Oxford Flowers Dataset with augmentation multiplicity $k = 128$ to compare FID (↓) scores.

Adaptation Variant	FID (↓)	Trainable Parameters	Notes
LoRA-A	78.35	1.57M (0.50%)	Finetuning only Attention Layers
LoRA-M	81.78	2.62M (0.84%)	Finetuning only MLP Layers.
LoRA-AM + LN	63.24	6.02M (1.91%)	Main Reported LoRA Method
LNTuning	116.64	100.7M (24.56%)	Main Reported LNTuning Method

Our results clearly show that a proper selection of layers to fine-tune is more crucial than the number of fine-tuned parameters. This happens due to the specifics of privacy training: information from gradients costs privacy, how the signal is allocated in the model affects its final performance. Low-rank adapters placed in the attention and MLP projections capture task-specific directions that resists DP perturbations, whereas LNTuning lacks the expressive power to counteract the injected noise. Consequently, our main LoRA-AM + LN configuration delivers the best privacy–utility trade-off, outperforming LNTuning in FID while updating fewer than 2% of the weights.

E. Why VAR converges faster?

Results in Figure 2 indicate that VAR converges surprisingly fast to its downstream task when fine-tuned. We argue that this behavior stems from VAR's training objective. Specifically, VAR is trained with the standard cross-entropy loss between predicted and ground-truth tokens. With this objective, the gradient signal remains strong even early in training. Instead, DMs minimize a denoising score-matching loss, which tries to reconstruct clean latents from various noise levels. The signal in this objective is diluted across T noise scales; gradients for very noisy time-steps are dominated by injected Gaussian noise, slowing effective learning until the model acquires a good global estimate of the score function.

F. Implementation Details of Adaptations and DP for VAR

IARs have model architectures identical to the popular transformer models like GPT-2 (Radford et al., 2019). The IARs' state-of-the-art next-token prediction mechanism excels at generating high-resolution images significantly faster than the average diffusion model. During our research, we implemented our own finetuning pipeline for Non-private and Private finetuning of IARs. The code base allows finetuning pre-trained IARs on private datasets with DP guarantee and compare that to our Non-Private baseline. Our implementation presents private IARs retaining high quality images on several datasets.

F.1. Overcoming Technical Barriers in Differentially Private IARs

Implementing DP guarantees is not straightforward with many IARs as their functioning is different than how DMs generally work. Each implementation, Private or Non-Private requires some modification to allow different finetuning adaptations function properly. Considering one of the class conditional IAR like VAR (Tian et al., 2024) registers buffers which are not supported by Opacus due to their nature of being a non-trainable parameter as well as the *SelfAttention* mechanism causes empty gradient flow when implementing LoRA finetuning. To overcome these constraints we patch the internal code and calling the modified classes from the updated module while retaining the same performance and efficiency as before.

Table 11: Various patching mechanisms implemented to successfully perform private and non-private finetuning with VARs.

Adaptation	**Patch Buffers**	**Override Forward Function**	**Patch *SelfAttention***
Full Fine-tuning	×	×	×
LoRA Fine-tuning	×	×	✓
LayerNorm Fine-tuning	×	×	×
DP-Full Fine-tuning	✓	✓	×
DP-LoRA Fine-tuning	✓	✓	✓
DP-LayerNorm Fine-tuning	✓	✓	×

In this paper, we illustrate our approach to managing buffers, as they are typically found in most of the IAR implementations discussed.

F.2. Patching Buffers to handle DP-finetuning

IARs like VAR (Tian et al., 2024) model register three important buffers that are crucial for its functioning. These buffers primarily handle embeddings, attention mask and bias making them highly important in order for VARs to function. The first buffer *lvl_1L*, is a level index tensor where each element contains the pyramid level index for the corresponding position in the sequence based on the patch numbers. This helps the model distinguish between tokens from different pyramid levels in the hierarchical structure. Another buffer registered in the VAR model is *attn_bias_for_masking* which is an Auto-regressive attention mask to ensure that tokens can only attend from the same or earlier pyramid levels. Lastly, the *zero_k_bias* is a zero-filled tensor with shape *embed_dim* that serves as a placeholder bias for the key projection component. While Query and Value have learnable biases and are initialized as parameters, the *zero_k_bias* is always passed as a fixed zero value tensor.

Unlike parameters, buffers are not updated at all and are not meant to have gradients computed either in batches or per-sample. Since the DP-SGD algorithm solely relies on computing gradients for each individual sample, Opacus does not handle the non-trainable buffer and causes the trainer to break immediately. Generally, buffers are removed in standard DP-SGD practices to avoid these errors although removing buffers from the VAR model results in unstable and garbled image generation since tokens are not positioned consistently since *lvl_1L* buffer does not exist and the attention bias does not exist to ensure correct level of scaled resolution.

We overcome buffer removal by not registering them at all in the first place and then passing them as functions with property decorators so Opacus doesn't create any sort of conflict with those.

This way we do not have to register any buffers in the first place for them to conflict with the finetuning process while every time the model calls these newly patched functions they are computed on the fly and act like model attributes. This approach is lightweight and flexible with different types of buffers which are essential while making sure the arbitrary computation does not slow down the finetuning process.

```
attn_bias_for_masking = torch.where(
    d >= dT, 0., -torch.inf
).reshape(1, 1, self.L, self.L)

self.register_buffer(
    'attn_bias_for_masking',
    attn_bias_for_masking.contiguous()
)
```

(a): Original buffer implementation

```
@property
def attn_bias_for_masking(self):
    d = torch.cat([
        torch.full((pn * pn,), i,
                   device=self.pos_1LC.device)
        for i, pn in enumerate(self.patch_nums)
    ]).view(1, self.L, 1)
    dT = d.transpose(1, 2)
    attn_bias = torch.where(d >= dT,
                            0.0, -torch.inf
                    ).reshape(1, 1, self.L, self.L)
    return attn_bias.contiguous()
```

(b): Patched buffer implementation

Figure 4: Buffer patching enables skipping *register_buffer* calls to avoid parameter incompatibility with Opacus.

F.3. Addressing Gradient Flow Issues in *SelfAttention* Layers for LoRA & DP-LoRA in VARs

The *SelfAttention* component of the the VAR model contains two base layers we target when passing LoRA configuration to the model. One of the layers *mat_qkv* was computed by combining several operations in a single line, performing a linear transformation, adding the bias and reshaping to match the tensor shape. When implementing LoRA finetuning on this layer caused empty gradient propagation meaning the gradients were not flowing through this layer at all due to the condensed approach of *qkv* computation. The problem occurs because Opacus needs to track and clip gradients at all times and non-initialized gradients causing to break the gradient flow. Simply meaning when using parameter-efficient fine-tuning methods like LoRA that rely on precise gradient computation for their low-rank updates the gradient computation becomes opaque to the DP mechanism.

We resolved this issue by decomposing the self-attention operation into its constituent parts, which benefits both standard LoRA and differentially private DP-LoRA implementations. Our patched approach follows three distinct steps: first applying the linear transformation through the LoRA-augmented layer *self.mat_qkv(x),* then explicitly adding biases as a separate operation, and finally reshaping the output tensor to avoid any tensor shape mismatches.

For standard LoRA, this separation ensures that the low-rank adaptation matrices properly participate in the computation graph, allowing gradients to flow correctly through both the base parameters and the LoRA adaptation matrices during backpropagation. For DP-LoRA specifically, this explicit separation is even more crucial, as it enables Opacus to correctly track, compute, and clip gradients at each step of the computation.

The modification maintains the mathematical equivalence of the operation while significantly improving the training dynamics for both approaches.

F.4. Override Forward Function

VAR models define a custom *forward* method with two separate inputs *label_B* and *x_BLCv_wo_first_l*, plus on-the-fly randomness for conditional dropout. Opacus's DP-SGD hooks expect a single, standard *forward(self, input)* signature and deterministic operations. When it encounters extra arguments or stochastic operations inside the forward pass, the privacy accounting and per-sample gradient machinery break, leading to trainer errors.

```
qkv = F.linear(
    input=x,
    weight=self.mat_qkv.weight,
    bias=torch.cat(
        (self.q_bias, self.zero_k_bias, self.v_bias)
    )
).view(B, L, 3, self.num_heads, self.head_dim)
```

(a): Original *qkv* computation method

```
qkv_raw = self.mat_qkv(x)
qkv = qkv_raw + torch.cat(
    (self.q_bias, self.zero_k_bias, self.v_bias)
).view(1, 1, -1)
qkv = qkv.view(B, L, 3, self.num_heads, self.head_dim)
```

(b): Patched *qkv* implementation

Figure 5: Patching *qkv* computation to add support for LoRA fine-tuning in VAR models (Tian et al., 2024).

To restore compatibility, we override the model's *forward* by accepting a single concatenated tensor *concat_tensor*. Before computing the forward pass, we externally concatenate the expanded labels and the inputs along the feature dimension. Inside the new *forward*, we unpack *label_B* and *x_BLCv_wo_first_l* by slicing *concat_tensor*, thereby presenting Opacus with the canonical single argument signature it requires.

Once unpacked, we replicate the original embedding, positional-encoding, and attention-bias computations exactly as before. The unpacked *label_B* drives the class embeddings and start-of-sequence token, while *x_BLCv_wo_first_l* populates the remainder of the input sequence. We then proceed through the usual per-layer AdaLN self-attention and final logits projection unmodified, ensuring functional parity with the upstream VAR implementation.