ObfuscaTune: Obfuscated Offsite Fine-tuning and Inference of Proprietary LLMs on Private Datasets

Ahmed Frikha   **Nassim Walha**
Ricardo Mendes   **Krishna Kanth Nakka**   **Xue Jiang**   **Xuebing Zhou**
Huawei Munich Research Center
ahmed.frikha1@huawei.com

Abstract

This work addresses the timely yet under-explored problem of performing inference and finetuning of a proprietary LLM owned by a model provider entity on the confidential/private data of another data owner entity, in a way that ensures the confidentiality of both the model and the data. Hereby, the finetuning is conducted offsite, i.e., on the computation infrastructure of a third-party cloud provider. We tackle this problem by proposing *ObfuscaTune*, a novel, efficient and fully utility-preserving approach that combines a simple yet effective obfuscation technique with an efficient usage of confidential computing (only 5% of the model parameters are placed on TEE). We empirically demonstrate the effectiveness of *ObfuscaTune* by validating it on GPT-2 models with different sizes on four NLP benchmark datasets. Finally, we compare to a naive version of our approach to highlight the necessity of using random matrices with low condition numbers in our approach to reduce errors induced by the obfuscation.

1 Introduction

Large Language Models (LLMs) such as GPT-4 (Achiam et al., 2023) are increasingly used due to their state-of-the-art performance in diverse tasks and productivity benefits (Noy and Zhang, 2023). While LLMs excel in zero-shot and few-shot predictions with in-context learning (Mann et al., 2020), finetuning them on domain-specific data can significantly outperform foundation models in tasks like chip design(Thakur et al., 2023; Wu et al., 2024; Liu et al., 2023).

Model providers keep their proprietary models private due to the exorbitant costs of training them[1]. To enable their users to customize or apply the proprietary models to their data, model owners provide finetuning and inference services, e.g., OpenAI finetuning API[2] and GitHub Copilot[3] respectively. Hereby, the users have to share their data with the model owners to use these services. Due to concerns of privacy leakage and competitive disadvantage, several users and commercial entities are not willing to share their private or confidential data. For e.g., Samsung banned the usage of ChatGPT after sensitive code was leaked (Ray, 2023). Hence, approaches that enable the inference and finetuning of proprietary LLMs of one stakeholder on the confidential/private data of another stakeholder in a privacy-preserving way are crucially needed.

We define the following requirement that potential methods addressing this problem must fulfill: (a) Model confidentiality: prevent leakage of the proprietary model parameters, (b) Data confidentiality: prevent data leakage, (c) Utility: the performance and results of the inference and finetuning should be comparable with and without protection, (d) Efficiency: the computational time, memory footprint and communication should remain acceptable. To the best of our knowledge, no prior work fulfill all of these requirements simultaneously. In the following, we discuss different categories of prior works.

Prior approaches based on differential privacy (DP) for inference (Igamberdiev and Habernal, 2023; Majmudar et al., 2022) and finetuning (Yu et al., 2021) focus on protecting the data. However, they do not provide any protection for the model parameters and incur significant utility losses (Req. (a) and (c) are not fulfilled). Another line of work uses cryptographic techniques, e.g., multi-party computation (MPC) and homomorphic encryption (HE) (Li et al., 2022; Liu and Liu, 2023). While the confidentiality of both the model and the data can be ensured, their substantial slowdown and communication costs are not suitable for real-time

---

[2]https://platform.openai.com/docs/guides/fine-tuning
[3]https://docs.github.com/en/copilot

applications (Req. (d) is not fulfilled). Another proposal (Xiao et al., 2023) considers sending a distilled version of the model to the client where adapter layers are finetuned on the confidential data. At inference time, the finetuned adapter are used in combination with the proprietary model on the server side. This approach does not protect inference data and leads to utility losses of up to 6% (Req. (b) and (c) are not fulfilled). The closest approach to the present work combines Trusted Execution Environments (TEE) with a lightweight encryption to address federated learning settings (Huang et al., 2024). However, such proposal protects only the finetuned LoRA parameters by using the TEE and deploys the proprietary LLM on the client-side fully or partially (Req. (a) is not fulfilled).

Our contribution in the present work is threefold. First, we propose *ObfuscaTune*, a novel and efficient approach that combines TEE with a simple yet effective obfuscation technique. Our proposed approach enables finetuning and inference of LLMs in a way that preserves the confidentiality of the model and the data with no utility loss and acceptable efficiency loss, fulfilling all aforementioned requirements. Second, we empirically demonstrate the effectiveness of our method by validating it on GPT-2 models with different sizes on four NLP benchmark datasets. Hereby, only 5% of the model parameters are placed on TEE. Finally, we highlight the necessity of our obfuscation technique by comparing it to a naive obfuscation method.

2 Method

We consider a problem setting involving three stakeholders: the model provider, the data owner and the cloud provider. The objective is to perform inference and finetuning of the proprietary LLM of the model provider on the confidential/private data of the data owner, in a way that ensures the confidentiality of both the model and the data. Due to the high computation and hardware costs required, we assume that the finetuning and/or inference is performed offsite, i.e., on the computational infrastructure of the cloud provider. We assume that the cloud provider is honest-but-curious, i.e., they will perform their task correctly but will try to find extra information about the other parties assets and data.

To tackle this problem, we propose *ObfuscaTune*, an approach that addresses this problem by combining TEE and a simple yet effective obfuscation

technique, ensuring model and data confidentiality while preserving utility. Following prior works, we consider the TEE as an isolated secure zone on a potentially adversary host where the data, code and computation processes used are inaccessible from outside (Hou et al., 2021; Huang et al., 2024). Figure 1[4] presents an overview of the *ObfuscaTune* approach, which we detail next.

The model protection is ensured as follows: the model provider sends the proprietary model to the TEE on the cloud provider infrastructure. Within the TEE, the highly parameterized attention and MLP layers are protected using our obfuscation technique that we detail later and then sent outside the TEE. Since large models do not fit inside the TEE, the model layers can be sent there batchwise to be protected before leaving it. The remaining low-parameterized layers, e.g., the input, output, normalization and dropout layers, are kept on the TEE. After these steps, all model parameters are protected, either by TEE or by the obfuscation, and the majority of model parameters are outside of the TEE. We note that the TEE is controlled by authentication that ensures that only the data owner can query the model. This prevents the cloud provider from querying the model to perform model stealing (Carlini et al., 2024) or embedding inversion attacks (Li et al., 2023; Morris et al., 2023).

The data protection in *ObfuscaTune* is conducted as follows: The data owner sends an encrypted batch of data directly to the TEE where it is first decrypted and then embedded using the model input layer. The resulting embedding is protected by our obfuscation method before leaving the TEE. The text tokenization can be conducted either before or after transmitting the data on the data owner side or in the TEE, respectively.

The obfuscated feedforward pass through one transformer block is executed as follows: Outside the TEE, the obfuscated data embedding is passed through the obfuscated model layers yielding an obfuscated intermediate embedding that is sent back to the TEE. The latter is then de-obfuscated and passed through the corresponding model layers on the TEE, depending on the model architecture. Subsequently, the resulting embedding is obfuscated again and leaves the TEE to be fed to the next transformer block. Finally, the output layer is applied in the TEE and the model output is sent back to the

---

[4]Will be part of the additional page in the camera ready version upon paper acceptance.

data owner (inference case) or used to computed the loss on the TEE and perform backpropagation and parameter updates (finetuning case).

Our obfuscation method obfuscates the model parameters and data embeddings by multiplying them with random matrices that minimize numerical errors. We begin by introducing the obfuscation method and later explain how we limit the numerical errors. Let's consider a multi-head attention layer and first focus on a single attention head with key, query, value layers parameterized by W_k, W_q and W_v, respectively, and an embedding X as its input. We obfuscate the embedding X by multiplying it with a randomly generated matrix R_a, yielding X^*, and obfuscate the parameters W_k, W_q and W_v by multiplying them with the inverse of that random matrix, i.e., R_a^{-1}, yielding W_k^*, W_q^* and W_v^*. Note that multiplying the obfuscated data embeddings X with the obfuscated parameters, W_k^*, W_q^* and W_v^*, leads to the same results, Q, K and V, of the original non-obfuscated operations (Eq. 1-3). All obfuscation operations are applied inside the TEE. The remaining aforementioned operations are performed outside of the TEE.

The output H of the attention head is computed (Eq. 4) and concatenated with the other heads outputs, yielding H_{allheads} (Vaswani et al., 2017). H_{allheads} is then multiplied by the projection layer parameters W_o^* that are obfuscated by another randomly generated random matrix R_b, yielding the obfuscated output O^* of the multi-head attention layer (Eq. 5). Finally, this obfuscated output is sent to the TEE where is it de-obfuscated via multiplication with the inverse of the random matrix, i.e., R_b^{-1}. The bias term of this last projection layer has to be added after de-obfuscation and is therefore kept unobfuscated on the TEE. The obfuscation of the MLP layers of the proprietary LLM is conducted in an analogous manner to the obfuscation of the multi-head attention layers. Fig. 2 shows an overview of all operations conducted in GPT-2 (Radford et al., 2019) with annotations of which operations are performed inside or outside the TEE and on obfuscated or de-obfuscated variables.

Note that using the same or different random matrices to obfuscate different transformer blocks does not impact our method. Note that the layers that are kept on TEE involve non-linearities, e.g., layer-norm, and therefore cannot be applied to obfuscated variables since the subsequent de-obfuscation would not yield the same result. These layers have a low number of parameters compared

to the attention and MLP layers placed outside of TEE, e.g., only ca. 5% of the parameters of GPT2-XL are kept on TEE while 95% are obfuscated and placed outside of TEE, in our experiments.

$$Q = (X^T R_a)(R_a^{-1} W_q) = X^{*T} W_q^* \quad (1)$$
$$K = (X^T R_a)(R_a^{-1} W_k) = X^{*T} W_k^* \quad (2)$$
$$V = (X^T R_a)(R_a^{-1} W_v) = X^{*T} W_v^* \quad (3)$$
$$H = \text{Dropout}(\text{Softmax}(QK^T))V \quad (4)$$
$$O^* = H_{\text{allheads}}^T W_o^* \quad (5)$$
$$O = O^* R_b^{-1} \quad (6)$$

Note that all data embeddings and parameters that are accessible to the adversary, i.e., the ones that are processed outside of the TEE, are obfuscated, except for the intermediate embeddings Q, K and V. Note that these embeddings cannot be inverted with state-of-the-art embedding inversion attacks (Li et al., 2023; Morris et al., 2023) as these require a high number of model queries. This is not possible in this case, since querying the TEE requires authentication. A potential adversary would be interested in recovering a total of 5 unknown variables, i.e., the data embeddings X and the model parameters W_k, W_q, W_v and W_o, while having access to only 4 equations involving them (Eq. 1-3 and Eq. 5). Hence, it is not possible to compute them analytically. For an additional layer of protection, model obfuscation with new randomly generated matrices can be conducted regularly, e.g., every day or every hour, although we believe this is not required. The model obfuscation can be performed very efficiently (ca. 10 seconds on a middle range GPU for a GPT2-XL model).

The minimal error property of our obfuscation method is designed to limit numerical errors resulting from the inverse computations of the random matrices as well as errors resulting from matrix multiplication between the random matrix and the data embeddings or model parameters. We use only orthogonal random matrices, as they have the minimum condition number of 1 (see Appendix B). We do this by setting our random matrices R_a and R_b to be the Q matrix computed by applying a QR-decomposition to a randomly generated matrix, as Q is always orthogonal. In this case, the inverse computation is fully error-free since the inverse of an orthogonal matrix is its transposed version which is an error-free operation.

3 Experimental evaluation

The conducted experiments aim to address the following key questions: **(a)** What is the impact of applying *ObfuscaTune* on utility, i.e., how do models finetuned with *ObfuscaTune* compare to the normally finetuned models? **(b)** How does our obfuscation method using orthogonal random matrices compare to naively using any random matrices?

We apply our method to GPT2 (Radford et al., 2019) models with different sizes, ranging from 117 million to 1.5 billion parameters. We implement *ObfuscaTune* on top of the nanoGPT implementation (Karpathy, 2023). All our experiments perform LoRA-finetuning (Hu et al., 2022). Hereby, the LoRA parameters are randomly initialized and placed outside of the TEE. We apply LoRA to all linear and attention layers. Further hyperparameters are specified in the appendix.

In each *ObfuscaTune* experiment, we use 2 GPU devices, one that is placed outside of TEE and another that simulates the TEE. We believe this is reasonable since high-end GPUs have TEE support (Apsey et al., 2023). We evaluate the finetuning with *ObfuscaTune* and with a naive version that uses any random matrices on 4 question-answering benchmark datasets, including WebQuestions (WebQs) (Berant et al., 2013), OpenBookQA (OBQA) (Mihaylov et al., 2018), PIQA (Bisk et al., 2020) and SciQ (Welbl et al., 2017). We evaluate all models using `lm-eval-harness`[5].

Setting	WebQs	OBQA	PIQA	SciQ
GPT2-Small				
Unprotected	16.0	23.0	64.1	91.1
Protected (random)	0.0	15.4	53.1	19.7
Protected (**ours**)	16.8	23.6	64.8	91.7
GPT2-Medium				
Unprotected	24.1	29.2	69.1	92.2
Protected (random)	0.0	14.4	52.0	20.0
Protected (**ours**)	24.5	28.6	68.9	92.4
GPT2-Large				
Unprotected	30.0	35.0	72.1	93.3
Protected (random)	0.0	14.4	52.0	19.7
Protected (**ours**)	29.7	32.2	72.3	93.0
GPT2-XL				
Unprotected	32.4	34.2	74.1	93.5
Protected (random)	0.0	14.8	52.5	20.5
Protected (**ours**)	32.6	33.2	73.9	93.6

Table 1: Test accuracy results (%) yielded by normally finetuned models (unprotected) and models which are protected by *ObfuscaTune* as well as a naive version of our method that uses an arbitrary random matrix with a non-optimized condition number (random).

CN	1	8	32	128	160	random
Accuracy	16.8	15.5	15.2	14.7	0.3	0.0

Table 2: Test accuracy results (%) yielded by GPT2-small models finetuned on WebQs with *ObfuscaTune* using matrices with different condition numbers (CN).

Table 1 presents our main experimental results. We find that models finetuned with our method achieve a performance comparable to models finetuned without model and data protection. This observation is consistent across all model sizes and benchmark datasets. Besides, models that are finetuned with a naive method that uses arbitrary random matrices incur substantial utility loss due to the high accumulation of errors. Furthermore, we evaluate the impact of using random matrices with different condition numbers and empirically confirm that higher condition numbers deteriorate performance (Tab. 2, details in Appendix B).

We also measure the percentage of model parameters present on TEE after model obfuscation to be **5.2%** for GPT2-XL, which highlights a substantial efficiency increase compared to naively shielding the whole model inside the TEE. Finally, we measure the runtime of the finetuning and find that using *ObfuscaTune* leads to a slowdown of 1.5x to 4.3x, for GPT2-small and GPT2-XL respectively. This is substantially lower than slowdowns yielded by cryptographic techniques, e.g., ca. 10^2 using MPC (Knott et al., 2021) and 10^5 using HE (Lou and Jiang, 2021) with significantly smaller models.

4 Conclusion

This work tackled the timely but underexplored problem of performing offsite inference and finetuning of a proprietary LLM owned by a model provider entity on the confidential/private data of another data owner entity, in a way that ensures the confidentiality of both the model and the data. Our proposed approach, ObfuscaTune, achieves this by combining a simple yet effective obfuscation technique with an efficient usage of confidential computing (only 5% of the model parameters are placed on TEE). Our extensive empirical evaluation on four NLP benchmark datasets and different models highlights the effectiveness of our method and emphasizes the importance of using random matrices with low condition numbers for preserving high utility. In future work, we will investigate the effectiveness of our approach to RAG-systems.

5 Limitations

One potential limitation of our work is that despite testing on different models and datasets, we focused on the same model architecture, i.e., GPT2. However, most of the other LLMs are composed on the same building blocks, which makes the application of our method to them straightforward. Another limitation might be that while the slowdown incured by *ObfuscaTune* is substantially lower than other technologies, e.g., MPC and HE, it might still be unsuitable for some applications where efficiency has a higher importance than privacy

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Emily Apsey, Phil Rogers, Michael O'Connor, and Rob Nertney. 2023. Confidential computing on nvidia h100 gpus for secure and trustworthy ai, august 2023.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on freebase from question-answer pairs. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1533–1544.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. 2020. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 7432–7439.

Nicholas Carlini, Daniel Paleka, Krishnamurthy Dj Dvijotham, Thomas Steinke, Jonathan Hayase, A Feder Cooper, Katherine Lee, Matthew Jagielski, Milad Nasr, Arthur Conmy, et al. 2024. Stealing part of a production language model. *arXiv preprint arXiv:2403.06634*.

Gene H Golub and Charles F Van Loan. 2013. *Matrix computations*. JHU press.

Jiahui Hou, Huiqi Liu, Yunxin Liu, Yu Wang, Peng-Jun Wan, and Xiang-Yang Li. 2021. Model protection: Real-time privacy-preserving inference service for model privacy at the edge. *IEEE Transactions on Dependable and Secure Computing*, 19(6):4270–4284.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Wei Huang, Yinggui Wang, Anda Cheng, Aihui Zhou, Chaofan Yu, and Lei Wang. 2024. A fast, performant, secure distributed training framework for large language model. *arXiv preprint arXiv:2401.09796*.

Timour Igamberdiev and Ivan Habernal. 2023. Dp-bart for privatized text rewriting under local differential privacy. *arXiv preprint arXiv:2302.07636*.

Andrej Karpathy. 2023. nanogpt.

Will Knight. 2023. Openai's ceo says the age of giant ai models is already over, april 2023.

Brian Knott, Shobha Venkataraman, Awni Hannun, Shubho Sengupta, Mark Ibrahim, and Laurens van der Maaten. 2021. Crypten: Secure multi-party computation meets machine learning. *Advances in Neural Information Processing Systems*, 34:4961–4973.

Dacheng Li, Rulin Shao, Hongyi Wang, Han Guo, Eric P Xing, and Hao Zhang. 2022. Mpcformer: fast, performant and private transformer inference with mpc. *arXiv preprint arXiv:2211.01452*.

Haoran Li, Mingshi Xu, and Yangqiu Song. 2023. Sentence embedding leaks more information than you expect: Generative embedding inversion attack to recover the whole sentence. *arXiv preprint arXiv:2305.03010*.

Mingjie Liu, Teodor-Dumitru Ene, Robert Kirby, Chris Cheng, Nathaniel Pinckney, Rongjian Liang, Jonah Alben, Himyanshu Anand, Sanmitra Banerjee, Ismet Bayraktaroglu, et al. 2023. Chipnemo: Domain-adapted llms for chip design. *arXiv preprint arXiv:2311.00176*.

Xuanqi Liu and Zhuotao Liu. 2023. Llms can understand encrypted prompt: Towards privacy-computing friendly transformers. *arXiv preprint arXiv:2305.18396*.

Qian Lou and Lei Jiang. 2021. Hemet: a homomorphic-encryption-friendly privacy-preserving mobile neural network architecture. In *International conference on machine learning*, pages 7102–7110. PMLR.

Jimit Majmudar, Christophe Dupuy, Charith Peris, Sami Smaili, Rahul Gupta, and Richard Zemel. 2022. Differentially private decoding in large language models. *arXiv preprint arXiv:2205.13621*.

Ben Mann, N Ryder, M Subbiah, J Kaplan, P Dhariwal, A Neelakantan, P Shyam, G Sastry, A Askell, S Agarwal, et al. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. 2018. Can a suit of armor conduct electricity? a new dataset for open book question answering. *arXiv preprint arXiv:1809.02789*.

John X Morris, Volodymyr Kuleshov, Vitaly Shmatikov, and Alexander M Rush. 2023. Text embeddings reveal (almost) as much as text. *arXiv preprint arXiv:2310.06816*.

Shakked Noy and Whitney Zhang. 2023. Experimental evidence on the productivity effects of generative artificial intelligence. *Science*, 381(6654):187–192.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

Siladitya Ray. 2023. Samsung bans chatgpt among employees after sensitive code leak, may 2023.

Shailja Thakur, Baleegh Ahmad, Zhenxing Fan, Hammond Pearce, Benjamin Tan, Ramesh Karri, Brendan Dolan-Gavitt, and Siddharth Garg. 2023. Benchmarking large language models for automated verilog rtl code generation. In *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6. IEEE.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Johannes Welbl, Nelson F Liu, and Matt Gardner. 2017. Crowdsourcing multiple choice science questions. *arXiv preprint arXiv:1707.06209*.

Haoyuan Wu, Zhuolun He, Xinyun Zhang, Xufeng Yao, Su Zheng, Haisheng Zheng, and Bei Yu. 2024. Chateda: A large language model powered autonomous agent for eda. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*.

Guangxuan Xiao, Ji Lin, and Song Han. 2023. Offsite-tuning: Transfer learning without full model. *arXiv preprint arXiv:2302.04870*.

Da Yu, Saurabh Naik, Arturs Backurs, Sivakanth Gopi, Huseyin A Inan, Gautam Kamath, Janardhan Kulkarni, Yin Tat Lee, Andre Manoel, Lukas Wutschitz, et al. 2021. Differentially private fine-tuning of language models. *arXiv preprint arXiv:2110.06500*.

A   Hyperparameters

We train all models for 10 epochs. We perform validation at the end of every epoch and use early stopping with a patience of 3. We use a learning rate of $3e-5$ and a batch size of 1. We keep the other hyperparameters unchanged from (Karpathy, 2023). For LoRA, we use the hyperparameters: $r = 16$, $\alpha = 32$ and apply dropout with 0.05. We did not perform hyperparameter tuning, which highlights the robustness of our method. We did all experiments on middle-range GPUs. Each experiment took between less than 1 and 8 GPU hours, depending on he model size and dataset.

B   Effect of the condition number

The condition number κ of a matrix A is defined as $\kappa(A) = \frac{M}{m}$, where $M = \max \frac{\|Ax\|}{\|x\|}$ measures how much the mapping induced by that matrix can stretch vectors and $m = \min \frac{\|Ax\|}{\|x\|}$ measures how much it can shrink vectors. It determines how much a relative error in the input reflects on the output for solving linear systems, matrix inversion or matrix-vector multiplication (Golub and Van Loan, 2013). Such numerical errors get accumulated and increase with the number of sequential matrix multiplication operations, i.e., the deeper the model the higher the accumulated error. We minimize the numerical errors by minimizing the condition number of the random matrix.

In this work, we consider the condition number w.r.t the ℓ_2 norm. Since orthogonal matrices induce isometries, i.e $\|Ax\|_2 = \|x\|_2$ for all x, we get $\kappa(A) = 1$ for every orthogonal matrix A. Note that singular matrices have the highest (worst) possible condition number, which is ∞, since for a singular matrix A, $m = \min \frac{\|Ax\|}{\|x\|} = 0$. On the other side, from the definition we see that the lowest possible κ is 1.

Let $\sigma_{max}(A)$ and $\sigma_{min}(A)$ respectively be the largest and the smallest singular values of the matrix A. For the ℓ_2-induced operator norm norm the following holds :

$$\|A\| = \max \frac{\|Ax\|}{\|x\|} = \sigma_{max}(A).$$

On the other hand, for A square and non-singular

$$\min_x \frac{\|Ax\|}{\|x\|} = \min_y \frac{\|y\|}{\|A^{-1}y\|}$$
$$= \frac{1}{\max_y \frac{\|A^{-1}y\|}{\|y\|}}$$
$$= \frac{1}{\|A^{-1}\|}$$
$$= \frac{1}{\sigma_{max}(A^{-1})} = \sigma_{min}(A).$$

Finally we get for every square and non-singular matrix A:

$$\kappa(A) = \frac{\sigma_{max}(A)}{\sigma_{min}(A)}$$

The last equation makes it possible to generate random matrices R of a given predefined condition number $\kappa(R)$. First we generate random matrices A and B using the standard normal distribution. We then apply QR-decomposition on A and B to
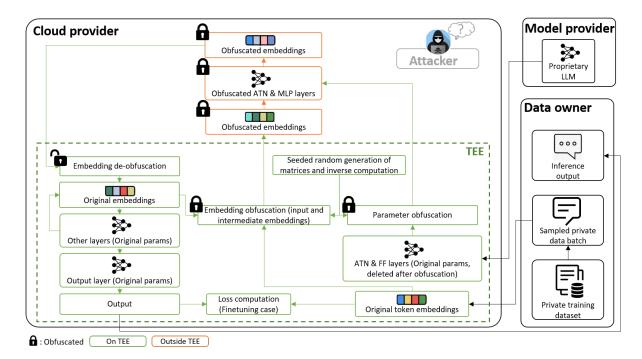
Figure 1: **Overview of the proposed *ObfuscaTune***, composed by the three stakeholders: model provider, which seeks to keep the model confidential, data owner, which uses the model (finetuning or inference) while preserving privacy of their data, and cloud provider which provides the computation infrastructure, while potentially trying to eavesdrop on the data or steal the model. *ObfuscaTune* provides the necessary protection by keeping very few components of the model within a TEE, and obfuscating the remaining ones, effectively and efficiently preventing data or model stealing. This Figure will be part of the additional page in the camera ready version upon paper acceptance.
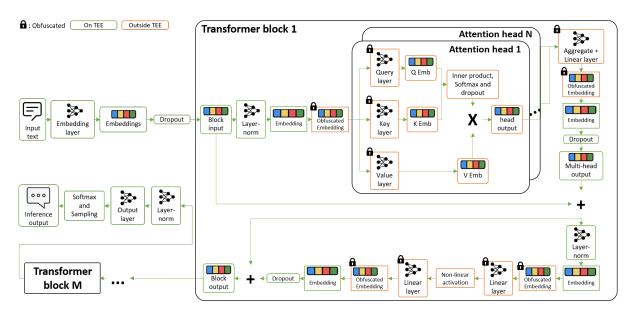

Figure 2: **Detailed architecture of the GPT-2 with M layers using *ObfuscaTune***. Diagram blocks in green are within the TEE, while the orange are outside the TEE. This diagram illustrates how the data is successfully sent from and to the TEE, while being obfuscated while outside the TEE. Note that both the input text and output text are always within the TEE to prevent inversion attacks. Note that the non-activation applied after the first MLP (bottom) is applied on the de-obfuscated embedding. The same applies for the softmax non-linear function.

generate two orthogonal matrices Q_A and Q_B. We then choose a random positive value for the largest singular value of the final matrix R and we set $\sigma_{min}(R) = \frac{\sigma_{max}(R)}{\kappa(R)}$. The remaining singular values can be sampled randomly from the uniform distribution between $\sigma_{min}(R)$ and $\sigma_{max}(R)$. Then we construct the diagonal matrix S with the singular values on the diagonal. Note that S^{-1} is the diagonal matrix with the inverses of the singular values on the diagonal. Then we define R to be having the following singular value decomposition:

$$R = Q_A S Q_B. \tag{7}$$

And can calculate $R^{-1} = Q_B^T S^{-1} Q_A^T$ with minimal rounding errors. We use this approach to generate random matrices of a given condition number and monitor the effect of the condition number on the test accuracy of the final model. The results are showcased in table 2 show indeed that it is curcial to have a low condition number, otherwise the training degenerates.