A First Look At Efficient And Secure On-Device LLM Inference Against KV Leakage

Huan Yang

yanghuan9812@csu.edu.cn Central South University Changsha, Hunan, China

Deyu Zhang*

zdy876@csu.edu.cn Central South University Changsha, Hunan, China

Yudong Zhao

yudong.zhao@transsion.com Shanghai Transsion CO., LTD Shanghai, China

Yuanchun Li

liyuanchun@air.tsinghua.edu.cn Tsinghua University Beijing, China

ABSTRACT

Running LLMs on end devices has garnered significant attention recently due to their advantages in privacy preservation. With the advent of lightweight LLM models and specially designed GPUs, on-device LLM inference has achieved the necessary accuracy and performance metrics.

However, we have identified that LLM inference on GPUs can leak privacy-sensitive intermediate information, specifically the KV pairs. An attacker could exploit these KV pairs to reconstruct the entire user conversation, leading to significant vulnerabilities. Existing solutions, such as Fully Homomorphic Encryption (FHE) and Trusted Execution Environments (TEE), are either too computation-intensive or resource-limited.

To address these issues, we designed KV-Shield, which operates in two phases. In the initialization phase, it permutes the weight matrices so that all KV pairs are correspondingly permuted. During the runtime phase, the attention vector is inversely permuted to ensure the correctness of the layer output. All permutation-related operations are executed within the TEE, ensuring that insecure GPUs cannot access the original KV pairs, thus preventing conversation reconstruction. Finally, we theoretically analyze the correctness of KV-Shield, along with its advantages and overhead.

ACM Reference Format:

Huan Yang, Deyu Zhang, Yudong Zhao, Yuanchun Li, and Yunxin Liu. 2018. A First Look At Efficient And Secure On-Device LLM Inference Against KV Leakage. In *MobiArch '24: Proceedings of the 19th Workshop on Mobility in the Evolving Internet Architecture, 18 November 2024, Washington, DC, USA*. ACM, New York, NY, USA, 7 pages. https://doi.org/XXXXXXXXXXXXXXX

Yunxin Liu

liuyunxin@air.tsinghua.edu.cn Tsinghua University Beijing, China

1 INTRODUCTION

Since the launch of the large language model (LLM) service by OpenAI in 2023, various LLM models based on the Transformer architecture have rapidly emerged. These models have enabled groundbreaking applications, such as expertlevel programming and advanced smartphone assistants, poised to transform the information service access paradigm, much like search engines and operating systems did in the past.

Compared to transmitting privacy-sensitive data over the Internet, on-device execution of LLMs is considered the most privacy-preserving solution [20]. Current mobile device manufacturers are competitively releasing on-device LLM deployment solutions at both the software and hardware levels. Notable examples include Apple's nearly 3-billion-parameter model and Qualcomm's Snapdragon 8 Gen 3 NPU.

While on-device LLM inference has been validated for accuracy and efficiency, it now faces the critical test of security. Unfortunately, the computing cores of mobile devices are vulnerable to various attacks, particularly information leakage [8]. For instance, the running kernel can be extracted from nearly all components of mobile GPUs, including shared, local, and texture memory. The impact of information leakage is magnified for LLMs. Leading LLM inference frameworks, like Meta's LLama [18], utilize memory caching of key-value (KV) pairs to accelerate inference. The KV cache persists throughout the entire inference process, lasting from seconds to minutes. Breaches in the KV cache can lead to the recreation of the original user conversation. A prime illustration is demonstrated in Leftoverlocal [14] on an AMD GPU, where data leaks in shared memory allowed an attacker to intercept the KV cache and replicate the entire conversation. We have replicated this attack on a Xiaomi 12 equipped with a Snapdragon 8 Gen 1 SoC.

^{*}Deyu Zhang (zdy876@csu.edu.cn) is the corresponding author.

In this paper, we make the first effort to protect the KV cache saved in mobile memory during LLM inference from two perspectives:

- Making the KV cache uninformative. To achieve this, we modify the KV pairs during LLM inference so that the original conversation cannot be recreated even if the KV pairs are leaked. We evaluate the performance of two solutions: Fully Homomorphic Encryption (FHE) and permutation.
- Making the KV cache invisible. We process the KV pairs in a Trusted Execution Environment (TEE), a secure area of the main processor commonly used in mainstream ARM architectures. This ensures that the KV cache is invisible to the outside insecure world.

We demonstrate that FHE is too computation-intensive for on-device LLM inference. The size of KV pairs is too large for the memory-limited TEE, which does not support GPU acceleration, significantly limiting the runtime performance of on-device LLM inference. Based on these insights, we design KV-Shield, which employs a lightweight encryption scheme, namely permutation, ensuring that insecure GPUs can only access the permuted KV pairs at runtime. Even if the permuted KV pairs are leaked, the user conversation cannot be reconstructed. We theoretically analyze the correctness of KV-Shield and discuss its overhead.

2 BACKGROUND

This section details the risks of memory leaks on existing mobile devices and describes existing content protection schemes for large language models.

2.1 Key Value Cache for LLMs

The self-attention mechanism[19] in Transformer models is a core component used to capture dependencies between different positions in the input sequence. It flexibly focuses on different parts of the sequence to better understand the context.

Attention
$$(Q, K, V) = \text{Softmax}(\frac{QK^{\top}}{\sqrt{d\iota}})V$$
 (1)

As shown in Eq. 1, the input sequence is mapped through three weight matrices (linear transformations) to generate query (Q), key (K), and value (V) vectors. The attention scores are computed from the dot product of Q and K (where $\sqrt{d_k}$ is the vector dimension), normalized using the softmax function, and then used to obtain a weighted sum of the V vectors, capturing long-range dependencies within the sequence.

Most large language models, such as LLaMA [18] and Qwen [2], are built on the Causal Decoder architecture [22]. These models generate tokens autoregressively, determining the next token based on the past prompt and previously

generated tokens. Each time a new token's attention representation is computed, the corresponding Q-vector must be calculated with the K- and V-vectors of the past prompt and generated tokens. However, the K- and V-vectors for past prompts and generated tokens are already computed during previous generations.

Caching the K- and V-vectors in memory prevents redundant computation. Assuming the input sequence length is n and the model dimension is $d_{\rm model}$, the dimensions of the K and V matrices are $n \times d_{\rm model}$. Using cached K and V reduces the need for two matrix multiplications of size $[n-1,d] \times [d,d]$.

2.2 Threat Model

Scene Setting. We consider an adversary capable of observing GPU tasks in the normal world and exploiting vulnerabilities similar to Leftoverlocal[14] to read the high-speed shared cache contents of the device GPU, such as OpenCL's local memory or CUDA's shared memory, but unable to directly access the GPU memory to obtain the model's inputs and outputs. Our primary goal is to protect the privacy of conversations between users and the large language model (LLM), rather than focusing on protecting the model weights. Such that the adversary cannot retrieve KV cache contents from the GPU's high-speed shared cache to reconstruct user conversations. Additionally, we do not consider side-channel attacks on the Trusted Execution Environment (TEE) — we assume the TEE can safeguard the confidentiality and integrity of its internal programs and data.

How does the adversary reconstruct the user conversation? As shown in Figure 1, when the user initiates the LLM process, the attacker launches a malicious process to steal the user's KV cache. The LLM process continuously submits GPU tasks to the GPU execution queue, such as Attention Kernel and FFN Kernel, to perform model inference and generate tokens. The attacker's malicious process continuously generates monitor kernels and inserts them after the Attention Kernel. By exploiting vulnerabilities, it accesses the cached contents in the local memory of the Attention Kernel - the KV cache, and transmits this information to the attacker. The attacker can determine which open-source LLM is being used by analyzing the types of GPU tasks and the KV contents. Then, the attacker inputs the KV cache into the attention module along with a prompt provided by the attacker, thereby reconstructing the user's conversation content.

3 POTENTIAL SOLUTION ANALYSIS

We analyzed two potential solutions, namely Fully Homomorphic Encryption (FHE) and running model inference in a Trusted Execution Environment (TEE), to prevent the

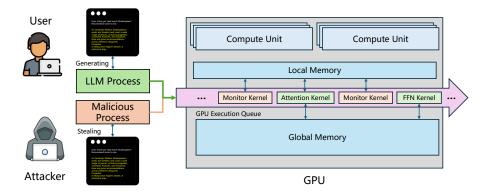


Figure 1: Attacker's workflow for stealing user contexts.

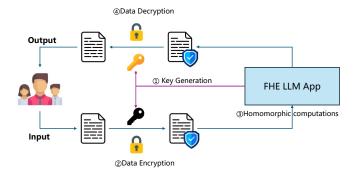


Figure 2: An Example of a Local Fully Homomorphic Encrypted Large Language Model Application

recreation of user conversations caused by KV leakage. FHE encrypts the entire LLM inference process without affecting the accuracy of the model. TEE isolates the model inference process from the GPU and insecure memory. Running model inference in TEE makes intermediate results, such as KV pairs, invisible.

3.1 The Performance of FHE

FHE is commonly employed in cloud scenario to safeguard users' data privacy. It allows for computations on encrypted user data, yielding decrypted results that are exactly the same as if the computations were performed on the unencrypted data[1]. We transition the FHE techniques to the LLM service for user privacy protection. We found that Fully Homomorphic Encryption (FHE) is too heavy for LLM inference, resulting in latency increases by nearly 6 orders of magnitude compared to plaintext inference.

As show in Figure 2, a FHE-based LLM application consists of 4 steps: 1) it generates a pair of keys to encrypt the input and decrypt the output, respectively. 2) when the user sends content to the FHE LLM application, the encryption key is

first used to encrypt the content, ensuring that the FHE-based LLM application can only receive the encrypted data. 3) the FHE-based LLM application processes the encrypted input data through homomorphic computation to derive the logically encrypted result. 4) upon the computation results being sent back to the user interface, the decryption key is utilized to decrpt the data, presenting the plain-text results on the screen.

We tested several Self-Attention implementations of LLM on Intel® $Core^{TM}$ i7-11800H processors using Pytorch[10] and two homomorphic computational libraries TenSeal[3] and ConcreteML [21]. The latency of the implementations is shown in Table 1. Tests were performed using inputs with a batch size of 1, a sequence length of 1, and a vector dimension of d_{model} . The results show that the performance of TenSeal drops by 5 orders of magnitude compared to Pytorch, while the ConcreteML library drops by 6 to 7 orders of magnitude. This shows that the computational performance of homomorphic encryption is weak, and it is difficult to meet the demand of real-time LLM inference.

The computation speed of fully homomorphic encryption is much slower than that of ordinary computation, primarily due to its reliance on complex mathematical operations,

Table 1: Inference Performance of Self-Attention

(d_{model}, num_{head})	Pytorch (s)	TenSeal (s)	ConcreteML (s)
(768,12) 1	0.00008	4.50	182.63
$(3584,16)^2$	0.00011	18.83	663.83
$(3584,28)^3$	0.00012	20.16	722.02
(4096,32) 4	0.00018	25.60	866.81

- $^1\,$ GPT2[11] and BERT[4] use 768 vector dimensions and 12 attention heads.
- ² gemma2-9b[16] uses 3584 vector dimensions and 16 attention heads.
- ³ Qwen2-7B[2] uses 3584 vector dimensions and 28 attention heads.
- ⁴ LLaMA2-7B[18] and ChatGLM3-6B[17] use 4096 vector dimensions and 32 attention heads.

data bloat, introduction of ciphertext noise, the need for multiple encryption and decryption processes, low algorithm efficiency, and unoptimized hardware.

3.2 Challenges in Trusted Execution Environment

Another intuitive solution to protect KV pairs is running LLM inference in the TEE, which is designed for privacy-sensitive code and data. We summarize multiple works utilizing TEE to safeguard conventional deep learning models like ResNet, VGG, and MobileNet, addressing limited memory and lack of GPU acceleration. We will discuss the inspirations from existing works and the new challenges posed by LLMs.

Incrementally feed the model layer into the TEE for model weights protection[9] [6]. DarkneTZ [9] and T-Slice [6] primarily focus on preventing model weight leakage, as effective membership inference attacks (MIAs) can reveal information about their training data. As shown in Table 2 and Table 3, TZDRAM is too small for CNNs due to the limited size of TEE trustworthy memory. DarkneTZ addresses this by slicing the CNN layer-by-layer to enable model inference within the TEE. Conversely, T-Slice [6] runs the entire model in the TEE. It dynamically splits the deep learning model into units (slices) that can be executed in TrustZone's limited trusted memory without modifying the protected deep learning model.

Offloading computation-intensive operators to GPU devices [15] [7]. TransLinkGuard [7] protects models during local inference by generating a locked model through rearranging the weights of the Transformer's fully-connected layers. During inference, TransLinkGuard rearranges the intermediate variables in the TEE to prevent model weight leakage. ShadowNet [15] observes that linear layers (including convolutional and fully connected layers) account for over 99% of the weights and computation time. It outsources these linear layers to untrusted environments (including GPUs) for acceleration without leaking model weights.

Compared to the model weights, the KV pairs are more in need of protection. For model privacy security of LLMs, KV pairs leakage leads to the recreation of user conversation. This is more direct and dangerous than the security risk of obtaining training data through model weights.

Table 2: Memory layout of the Mobiles SoC's TEE

Chips	TZDRAM¹(MiB)	Total DRAM (GiB)
RK3399	32	4
MT8173	30	2
Hikey960	16	3
Raaspberry Pi 3	15	1

¹ TZDRAM: TrustZone DRAM.

Table 3: Params, memory and FLOPs of common models

Model	Params(M)	Mem(MiB) ³	GFLOPS
Qwen2-7B[2] ¹	7070	30115.7	1680
ChatGLM3-6B[17] ¹	6240	23,794.2	1580
LLama2-7B[18] ¹	6610	25874.1	1670
$ResNet50[5]^2$	25.6	89.8	8.2
MobileNetV2[12] ²	3.5	74.9	0.6
$Vgg11[13]^2$	132.9	54.9	15.2

 $^{^{1\,}}$ indicates that it is based on the Transformer large language model.

Table 4: Size of KV pairs generated by LLM in single decode

Model	$\operatorname{shape}_{KV}$	layer _{num}	${\sf size}_{KV}^{1}$
LLaMA2-7B	$2 \times 32 \times 128$	32	seq _{len} ² × 1 MiB
ChatGLM3-6B	$2 \times 2 \times 128$	28	$seq_{len} \times 56 \text{ KiB}$
Qwen2-7B	$2 \times 4 \times 128$	28	$seq_{len} \times 112 \text{ KiB}$

¹ $\operatorname{size}_{KV} = \operatorname{seq}_{len} \times \operatorname{shape}_{KV} \times \operatorname{layer}_{num} \times \operatorname{4B}$. The size of float32 type is

As shown in Table 4 illustrates that a typical user-LLM conversation spans approximately 1000 tokens. This results in the LLM generating hundreds of MiBs or even several GiBs of KV Cache. The size of this KV Cache is over 10 times larger than the model weights of a CNN, thereby making the slicing in TEE significantly more challenging.

The lack of support for GPU acceleration in the TEE makes it challenging to efficiently perform LLM inference. Despite having adequate memory resources in the TEE, the limitation to using only the CPU for LLM inference can lead to inefficiencies. Achieving the efficiency provided by insecure GPUs while ensuring the security of KV pairs poses a significant challenge.

4 DESIGN OF KV-SHIELD

In this part, we design KV-shield to protect the original KV pairs stolen by a malicious process. We use the TEE and a simply yet effective and efficient permutation operation. We design KV-shield according to the following three principles:

- 1) Deploy the model without modification.
- 2) Keep the original KV invisible to the insecure GPUs in the REE(Rich Execution Environment).
- 3) Using GPUs in REE as much as possible to improve the inference efficiency of LLM.

² indicates that it represents a convolutional neural network.

³ indicates the peak memory usage during model inference.

 $^{^{2}}$ The size of seq_{len} is equal to the sum of the number of tokens entered by the user and the number of tokens that LLM has generated

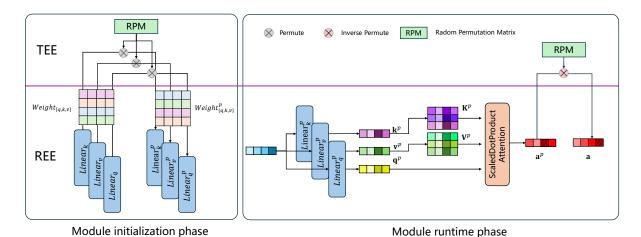


Figure 3: Workflow for End-Side Protection of KV Cache

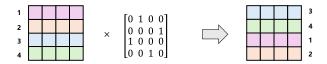


Figure 4: Workflow for Matrix Permutaion

4.1 Overview

As depicted in Figure 3, to protect the original KV pairs, we permute the weights of the linear layers in the self-attention operator. It rearranges the rows or columns of the matrices, as shown in Figure 4.By multiplying a matrix by a 01 matrix, we can realize that the rows of the matrix are disrupted. In such a way, after the GPU performs the linear layer computations, the insecure cache stores the permuted KV pairs. The corresponding permutation matrix is stored in the TEE to ensure that attackers cannot obtain the permutation information. Finally, the results of self-attention are inversely permuted through the TEE to obtain the correct results.

- 1) At the initialization of the LLM process, we randomly permute the weights of $Linear_q$, $Linear_k$, and $Linear_v$ for each layer's self-attention module, resulting in $Linear_q^p$, $Linear_k^p$, and $Linear_v^p$. The RPM (Random Permutation Matrix) is stored inside the TEE.
- 2) When executing the self-attention module at $Layer_i$, the input x undergoes three linear transformations ($Linear_q^p$, $Linear_k^p$, $Linear_v^p$) to produce the variables \mathbf{q}^p , \mathbf{k}^p , and \mathbf{v}^p . The variables \mathbf{k}^p and \mathbf{v}^p are added to the KV cache, forming \mathbf{K}^p and \mathbf{V}^p .
- 3) The variables \mathbf{q}^p , \mathbf{K}^p , and \mathbf{V}^p are then used in the Scaled Dot-Product Attention calculation to obtain the attention

results, which are sent to the TEE to recover the correct attention output.

Our design ensures that the KV matrix stored in the insecure cache is always in the form of a permuted matrix. Even if the KV pairs are leaked, without the RPM stored in the TEE, an attacker cannot effectively recover the contextual content from the transposed KV pairs.

4.2 Correctness and Security Analysis

This section theoretically analyzes the correctness of the computational process depicted in Figure 3. We use RPM to denote the random permutation matrix. Let $\mathbf{x} \in \mathbb{R}^{1 \times d}$, Weight $\in \mathbb{R}^{d \times d}$, RPM $\in \mathbb{R}^{d \times d}$, and RPM \times RPM^T = I, where I is the identity matrix.

$$Weight_{\{q,k,v\}}^{p} = Weight_{\{q,k,v\}}RPM$$
 (2)

$$\{q, k, v\}^p = xWeight_{\{q, k, v\}}^p = \{q, k, v\}RPM$$
 (3)

$$\mathbf{K}^{p} = \begin{bmatrix} \mathbf{K}^{p} \\ \mathbf{k}^{p} \end{bmatrix}, \quad \mathbf{V}^{p} = \begin{bmatrix} \mathbf{V}^{p} \\ \mathbf{v}^{p} \end{bmatrix}$$
 (4)

$$\mathbf{a}^{p} = \operatorname{Softmax}\left(\frac{\mathbf{q}^{p} \mathbf{K}^{p^{\top}}}{\sqrt{d_{k}}}\right) \mathbf{V}^{p} \tag{5}$$

$$\mathbf{a} = \mathbf{a}^p \mathbf{R} \mathbf{P} \mathbf{M}^{\mathsf{T}} \tag{6}$$

In Equation 2, we permute the original weight matrix **Weight** to **Weight**^p using the random permutation matrix. Through these permuted weight matrices, collectively called **Weight**^p, we compute the permuted vectors \mathbf{q}^p , \mathbf{k}^p , and \mathbf{v}^p , as shown in Equation 3.

According to our derivation, $\mathbf{a}^p \in \mathbb{R}^{1 \times d}$. In this manner, the output attention vector \mathbf{a}^p is the permuted version of the correct attention vector \mathbf{a} . Finally, within the TEE, we anti-permute \mathbf{a}^p back to \mathbf{a} .

Table 5: Overhead of permutation

d_{model}	Permute Weight (s)	Permute Result (s)
768	15.75	0.9
3584	71.44	3.7
4096	84.22	4.3

In summary, by operating on the permuted weights and inverse permuting the output attention vector, we ensure that the output of each layer remains unchanged.

Security Analysis. In KV-shield, we ensure that the plaintext of **K** and **V** are not saved in the memory of REE, thus the insecure GPUs in REE cannot process the original KV pairs directly. Even the permuted KV pairs are leaked, the attacker cannot recreate the user conversation.

5 DISCUSSION

Feasibility of KV-shield in terms of KV protection. Most LLM models have a model dimension d_{model} of around 4096, and the current TEE memory of approximately 32MB is sufficient to accommodate part of the vector storage and computation for LLMs. TEE is sufficient to accommodate a to perform the permute computation.

Overhead brought by matrix permutation in TEE. We test the efficiency of matrix permutation in the TEE on the intel 11800H in the QEMU with 16 MB TEE memory, as shown in Table 5. For ease of implementation, we implement the matrix permutation by loops. The results show that the weights and attention vectors permutation in TEE achieves the orders of seconds. To adapt to the limited TEE memory, we calculate the values of weights permutation in a block by block manner. Note that the results is just for one layer. For an entire model with over 20 layers, the latency can reach 5 minutes, which is unacceptable for users. The latency caused by vector permutation happens in the runtime phase. Although the TEE memory is sufficient for each vector permutation, the latency is still too high for real-time token generation. These results inspire us to further optimize the efficiency of KV-Shield in the future.

6 CONCLUSION

We demonstrate that a malicious process can steal KV pairs during LLM inference on the mobile GPU and reconstruct the entire user conversation, leading to significant security vulnerabilities. To address this issue, we explore potential solutions, such as FHE and TEE, to secure on-device LLM inference. We find that FHE is too resource-intensive for ondevice inference, and TEE faces limitations in both memory and computation resources. Building on these insights, we designed the KV-Shield. By permuting the weight matrix

and subsequently inversely permuting the results, KV-Shield harnesses the computational power of insecure GPU accelerators while ensuring that they cannot access the original KV pairs, thus preventing data leakage. KV-Shield operates in two phases within the TEE. During the initialization phase, it shuffles the linear weights, and in the runtime phase, it reverses the permutation of the attention vectors for each self-attention module. Given the limited size of the attention vector for each module, the TEE has sufficient resources for this operation. We analyze the theoretical accuracy of the KV shield. Moving forward, we will further optimize the performance of KV-Shield for LLM inference on the device.

ACKNOWLEDGMENTS

This work was supported in part by the National Key Research and Development Program of China under Grant 2022YFF0604504; in part by the National Science Foundation of China under Grant 62172439; in part by the Major Project of Natural Science Foundation of Hunan Province under Grant 2021JC0004; in part by the National Science Fund for Excellent Young Scholars of Hunan Province under Grant 2023JJ20076; and in part by the Central South University Innovation-Driven Research Programme under Grant 2023CXQD061.

REFERENCES

- Abbas Acar, Hidayet Aksu, A Selcuk Uluagac, and Mauro Conti. 2018.
 A survey on homomorphic encryption schemes: Theory and implementation. ACM Computing Surveys (Csur) 51, 4 (2018), 1–35.
- [2] Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. 2023. Qwen technical report. arXiv preprint arXiv:2309.16609 (2023).
- [3] Ayoub Benaissa, Bilal Retiat, Bogdan Cebere, and Alaa Eddine Belfedhal. 2021. TenSEAL: A Library for Encrypted Tensor Operations Using Homomorphic Encryption. arXiv:2104.03152 [cs.CR]
- [4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805 (2018).
- [5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition. 770–778.
- [6] Md Shihabul Islam, Mahmoud Zamani, Chung Hwan Kim, Latifur Khan, and Kevin W Hamlen. 2023. Confidential execution of deep learning inference at the untrusted edge with arm trustzone. In Proceedings of the Thirteenth ACM Conference on Data and Application Security and Privacy. 153–164.
- [7] Qinfeng Li, Zhiqiang Shen, Zhenghan Qin, Yangfan Xie, Xuhong Zhang, Tianyu Du, and Jianwei Yin. 2024. TransLinkGuard: Safeguarding Transformer Models Against Model Stealing in Edge Deployment. arXiv preprint arXiv:2404.11121 (2024).
- [8] Sparsh Mittal, SB Abhinaya, Manish Reddy, and Irfan Ali. 2018. A survey of techniques for improving security of gpus. *Journal of Hardware and Systems Security* 2 (2018), 266–285.

- [9] Fan Mo, Ali Shahin Shamsabadi, Kleomenis Katevas, Soteris Demetriou, Ilias Leontiadis, Andrea Cavallaro, and Hamed Haddadi. 2020. Darknetz: towards model privacy at the edge using trusted execution environments. In Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services. 161–174.
- [10] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch. (2017).
- [11] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language Models are Unsupervised Multitask Learners. (2019).
- [12] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In Proceedings of the IEEE conference on computer vision and pattern recognition. 4510–4520.
- [13] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014).
- [14] Tyler Sorensen and Heidy Khlaaf. 2024. LeftoverLocals: Listening to LLM Responses Through Leaked GPU Local Memory. arXiv preprint arXiv:2401.16603 (2024).
- [15] Zhichuang Sun, Ruimin Sun, Changming Liu, Amrita Roy Chowdhury, Long Lu, and Somesh Jha. 2023. Shadownet: A secure and efficient on-device model inference system for convolutional neural networks. In 2023 IEEE Symposium on Security and Privacy (SP). IEEE, 1596–1612.

- [16] Gemma Team. 2024. Gemma. (2024). https://doi.org/10.34740/ KAGGLE/M/3301
- [17] GLM Team, Aohan Zeng, Bin Xu, Bowen Wang, Chenhui Zhang, Da Yin, Diego Rojas, Guanyu Feng, Hanlin Zhao, Hanyu Lai, et al. 2024. ChatGLM: A Family of Large Language Models from GLM-130B to GLM-4 All Tools. arXiv e-prints (2024), arXiv-2406.
- [18] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. LLaMA: Open and Efficient Foundation Language Models. arXiv:2302.13971 [cs.CL]
- [19] 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 (2017).
- [20] Yuanchun Li Wangsong Yin, Mengwei Xu and Xuanzhe Liu. 2024. LLM as a System Service on Mobile Devices. arXiv prepring arXiv:2403.11805 (2024).
- [21] Zama. 2022. Concrete ML: a Privacy-Preserving Machine Learning Library using Fully Homomorphic Encryption for Data Scientists. https://github.com/zama-ai/concrete-ml.
- [22] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. 2023. A survey of large language models. arXiv preprint arXiv:2303.18223 (2023).