基于可信执行环境 TEE 的大模型保护技术

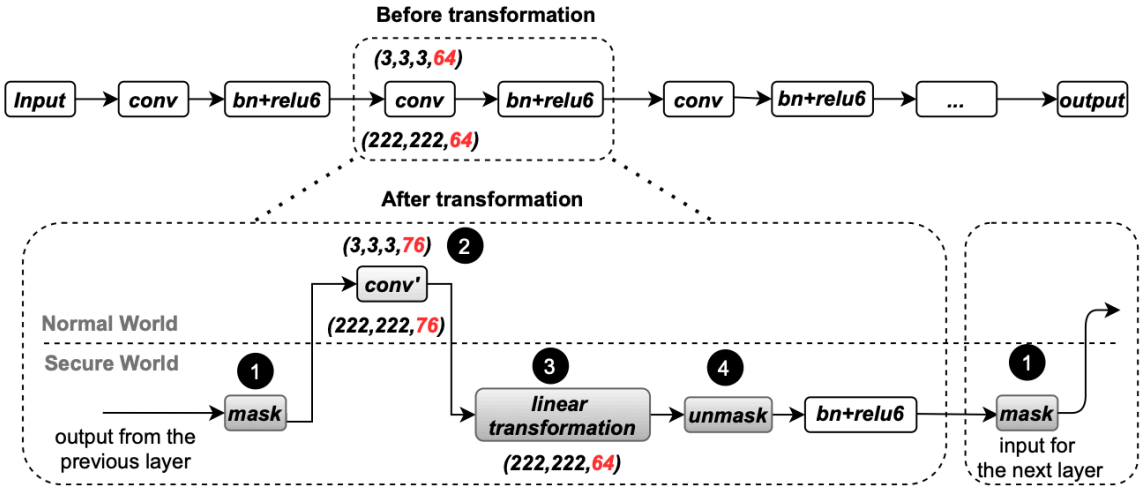
Li Kaihua

Dec 23, 2023

Problem

大模型推理依赖 GPU 计算，而目前 TEE 无法提供 GPU 加速计算能力。

ShadowNet 提出一种方法将线性算子混淆后，在不泄漏模型权重的前提下，将线性计算外包给不可信的 GPU 以获得加速。



Challenge

1. 大模型参数规模在 10B 以上才能出现涌现现象，10B 大模型参数内存占用达 20GB，TEE 难以承载如此规模的内存占用与计算量。
2. 目前在 TEE 中缺少 GPU 加速能力，而 TEE 与 GPU 之间的计算速度不匹配，造成 GPU 利用率不足与延长推理时间。
3. 大模型推理过程中，保护技术需要频繁调用 TEE，而频繁调用 TEE 引入的系统级切换开销。

Solution 1

1. 选择部分参数进行保护，减少需要在 TEE 中进行的计算量。

Q: 如何选择需要保护的参数?

A: 通过迭代地调整参数，使模型输出偏离值超过设定值。

Step 1. 设置阈值 E.

Step 2. 调整参数 W，得到 W_{err}.

Step 3. 判断 $\|Y - Y'\| > E$.

Step 4. 若满足则退出，不满足则回到 Step 1.

Solution 1

1. 选择部分参数进行保护，减少需要在 TEE 中进行的计算量。

优点：减少了需要保护的参数规模，能够有效地减少 TEE 中的内存压力和计算开销。

缺点：攻击者可以考虑通过公开的参数为基础模型，在基础模型上通过训练拟合保护的参数。

应对策略：神经网络训练的收敛难度与网络深度相关，尝试保护连续多层网络参数，使攻击者难以接受重新训练的计算代价。

Solution 2

2. 藏木于林，通过少量的冗余参数混淆，以保护正确参数。

数学基础

有矩阵 A 和矩阵 B

$$A = [a_1, a_2, ..., a_n], \quad B = [b_1, b_2, ..., b_n]$$

$$P = [p_{ij}], \text{if } p_{i'j'} = 1 \begin{cases} p_{:j}=0 \ , \ j \neq j' \\ p_{i:}=0 \ , \ i \neq i' \end{cases}$$

$$\hat{A} = [A, A']P, \quad \hat{B} = [B, B']P$$

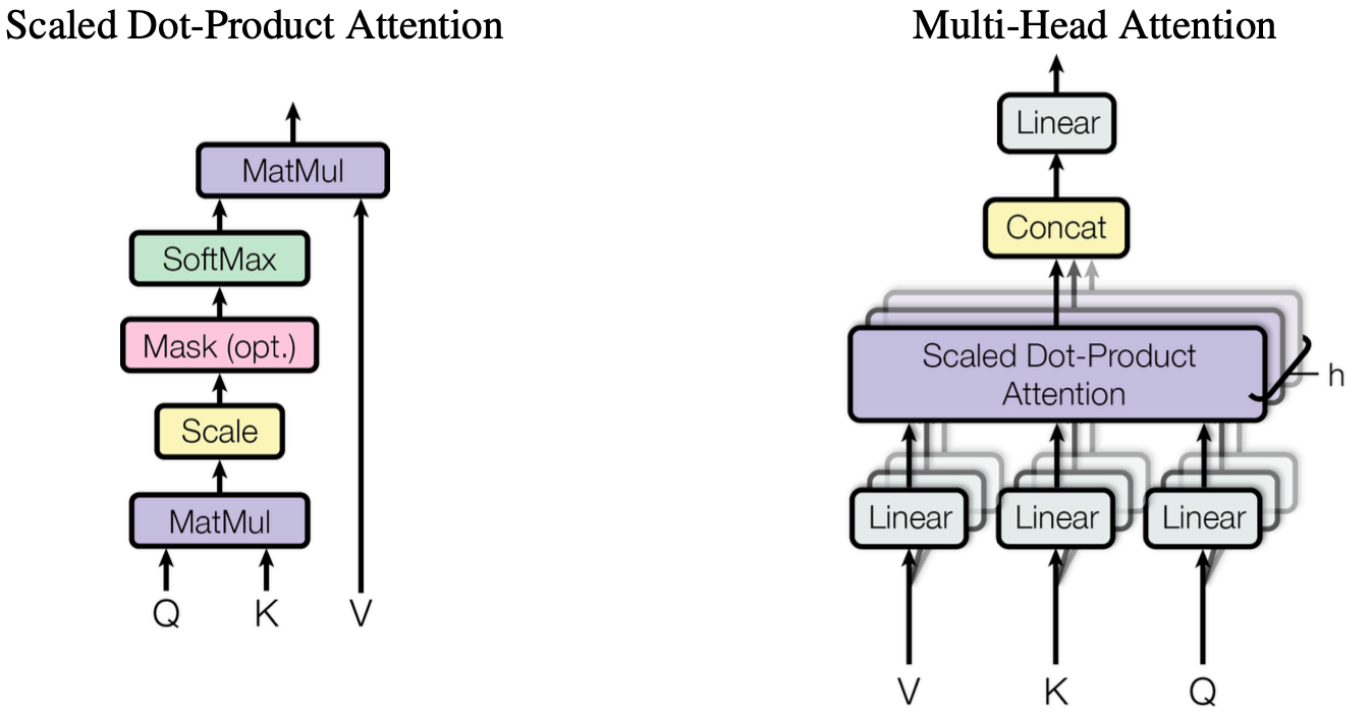
$$\hat{A}\hat{B}^T = AB^T + A'B'^T$$

说明可以通过添加随机的冗余参数 A' 和 B' 来隐藏 A 和 B。

Solution 2

2. 藏木于林，通过少量的冗余参数混淆，以保护正确参数。

Multi-Head Attention



Solution 2

2. 藏木于林，通过少量的冗余参数混淆，以保护正确参数。

Multi-Head Attention

```
def Attention(x, wq, wk, wv, wo):
  xq = x @ wq
  xk = x @ wk
  xq = xq.view(SeqLen, Head, -1)
  xk = xk.view(SeqLen, Head, -1)
  xq, xk = xq.transpose(0, 1), xk.transpose(0, 1)
  qk = xq @ xk.transpose(1, 2)
  s = F.softmax(qk, dim=-1)
  return s @ wo
```

考虑在 wq 和 wk 中引入冗余参数进行混淆。

Solution 2

2. 藏木于林，通过少量的冗余参数混淆，以保护正确参数。

pad & shuffle 正确性验证

```
# unmodified
qk = QK(x, wq, wk)
s = F.softmax(qk, -1)
# pad & shuffle
wq_, wk_, pad_q, pad_k = mask(wq, wk)
qk_ = QK(x, wq_, wk_)
pad = QK(x, pad_q, pad_k) # TEE part
s_ = F.softmax(qk_, -1)
# check
print(f"qk_ - pad == qk is {torch.allclose(qk_ - pad, qk, 10e-4)}")
-----------------------------------------------
> qk_ - pad == qk is True
```

Solution 2

2. 藏木于林，通过少量的冗余参数混淆，以保护正确参数。

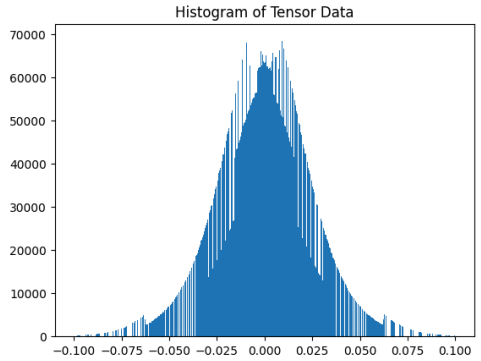
实验设置：
$\eta = 1/16$，Size(W')=Size(ηW)，
W' 按原始参数近似正态分布随机取样。
参数 (wq) 隐藏维度 $N = 4096$，head $= 32$，head_dim=128

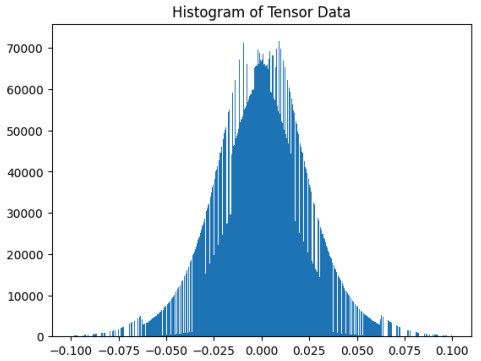
- 参数混淆处理(mask)之后，对输出结果的有多大的影响?
  计算得 $\frac{\|qk-qk\_\|_2}{\|qk\|_2} \approx 20\%\sim25\%$，$\frac{\|s-s\_\|_2}{\|s\|_2} \approx 20\%\sim30\%$，引入 6.25% 的额外参数使输出向量产生约 $20\% \sim 30\%$ 的偏离。
- 攻击者从混淆后参数中，分离出原始正确参数的代价有多大?
  $C_{128+8}^{128} \approx 10^{12}$，从每一个 head 下恢复需要尝试约 10^{12} 次组合。
  攻击者可能基于混淆后的模型，通过再训练拟合原始模型。

Solution 2

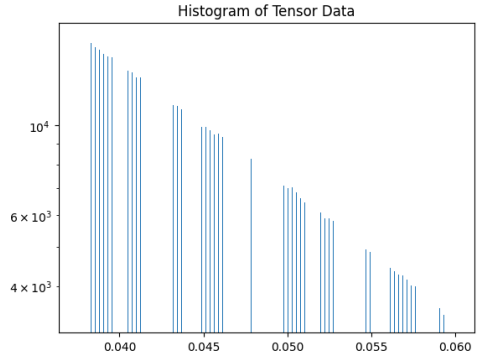
2. 藏木于林，通过少量的冗余参数混淆，以保护正确参数。

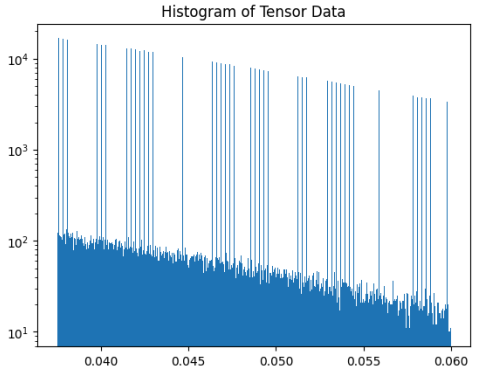
从整体上难以区分混淆前后参数(wq)分布（左图为未混淆）





观察局部分布(左图为未混淆，纵坐标采用对数化）





Solution 2

2. 藏木于林，通过少量的冗余参数混淆，以保护正确参数。

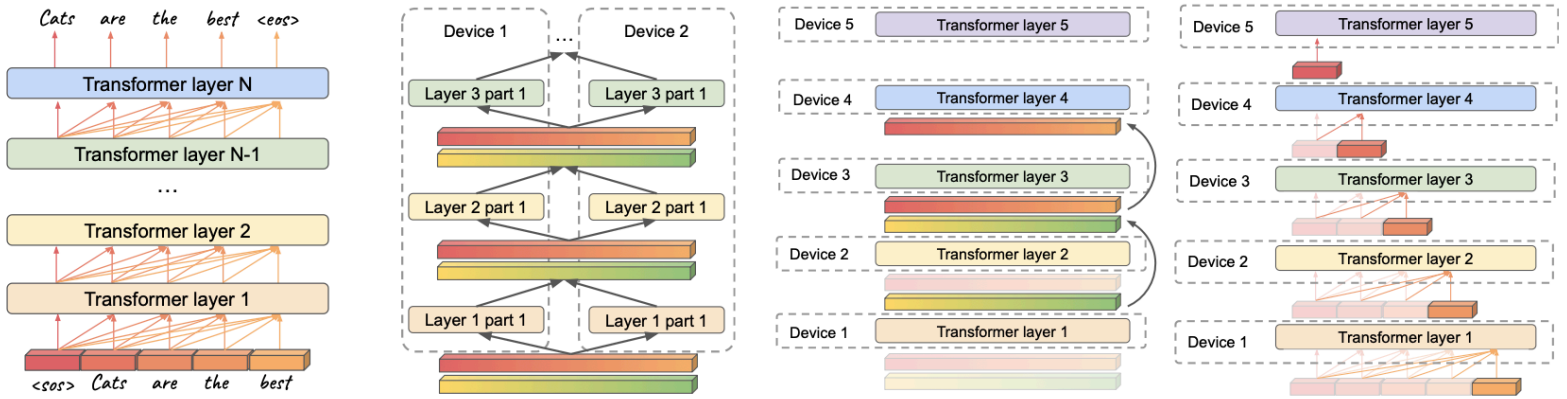
“聪明的”攻击者可能通过数据分布过滤混淆的参数，以减少尝试次数。此外，可能通过配合再训练，以拟合原始模型，造成模型权重参数泄漏。

应对策略：

- 基于原始参数 W 的数据分布，精心设计 W' 的数据分布，使得混淆后的 $\widehat{W}$ 数据分布难以辨别。
- 控制 η 倍率，增大再训练拟合原始模型的难度，权衡安全性与计算开销。
- 结合 Solution 1，以尽可能小的计算开销代价，将更多的计算过程 (Softmax…) 迁移至 TEE 环境，使攻击者恢复难度非线性增加。

Solution 3

3. Token-based Pipeline 与 GPU 虚拟化分割



(a) Transformer-based LM (b) Operation partitioning (Megatron-LM) (c) Microbatch-based pipeline parallelism (GPipe) (d) Token-based pipeline parallelism (TeraPipe)

Figure 1. Different approaches of model parallel training of Transformer-based LMs. (a) shows a standard multi-layer Transformer LM. In each layer, each position only takes only its previous positions as input. (b) shows operation partitioning (Shoeybi et al., 2019). An allreduce operation is required to synchronize the results of each layer. (c) shows microbatch-based pipeline parallelism (Huang et al., 2019), which allows different microbatches (red and green bars) to be executed on different layers of the DNN in parallel. (d) show TeraPipe (our work), which pipelines along the token dimension.

TeraPipe 一文中提出一种 Token-based 的流水线并行技术，在已有的并行维度上，添加了 Token 级别的并行维度。

Solution 3

3. Token-based Pipeline 与 GPU 虚拟化分割

简单地，有计算序列

$$T_i = \{A_1, F_1, A_2, F_2, ..., A_n, F_n\}$$

n 为 Transformer 层数，A 为 Attention 层，F 为 FeedForward 层，假设 A_i 依赖 TEE 进行计算，F_i 完全不依赖于 TEE 进行计算。

基于 Token-based Pipeline 构建流水线 $P = \{T_1, T_2, ..., T_n\}$，有

$$P_{i,j} = T_{i_j}$$

$$T_{i_j} = \begin{cases} A_{\lfloor \frac{j}{2} \rfloor + 1}, & 2 \nmid j \\ F_{\frac{j}{2}}, & 2 \mid j \end{cases}$$

Solution 3

3. Token-based Pipeline 与 GPU 虚拟化分割

若以 3 Token 构成流水线，当 $2 \nmid j$ 时，

$$
\begin{aligned}
P_{1,j-1} &= F_{\frac{j-1}{2}} \\
P_{2,j} &= A_{\lfloor \frac{j}{2} \rfloor + 1} \\
P_{3,j+1} &= F_{\frac{j+1}{2}}
\end{aligned}
$$

此时，可以借助 Nvidia MPS 虚拟化技术实现 P_1, P_3 的并行计算。

容易发现此场景下，下一阶段的 Pipeline 为 A, F, A，即无法充分利用 GPU 进行加速计算，因此在设计上需要进一步考虑安全计算与 Pipeline 之间的协调，以尽可能增大 GPU 吞吐量。

Solution 4

4. 使用压缩和量化技术，减少 TEE 的数据移动

Layer	Input (MB)	28MB+1vCPU Latency (s)	28MB+1vCPU Evictions (#)	56MB+2vCPU Latency (s)	56MB+2vCPU Evictions (#)	112MB+4vCPU Latency (s)	112MB+4vCPU Evictions (#)	168MB+8vCPU Latency (s)	168MB+8vCPU Evictions (#)
Outside TEE									
19	392	0.030	-	0.029	-	0.030	-	0.029	-
21	64	0.005	-	0.005	-	0.005	-	0.005	-
23	16	0.001	-	0.001	-	0.001	-	0.001	-
Quant./Comp. in TEE									
19	392	0.542	55,707	0.498	55,258	0.497	16,167	0.464	16,168
21	64	0.090	9,194	0.094	11,157	0.076	2,660	0.075	2,660
23	16	0.023	2,090	0.020	2,500	0.020	715	0.022	715
Unmodified in TEE									
19	392	0.987	101,113	1.158	102,328	1.213	101,270	1.124	103,536
21	64	0.162	16,481	0.191	16,516	0.203	16,458	0.185	16,523
23	16	0.039	4,090	0.046	4,165	0.050	4,030	0.045	4,041

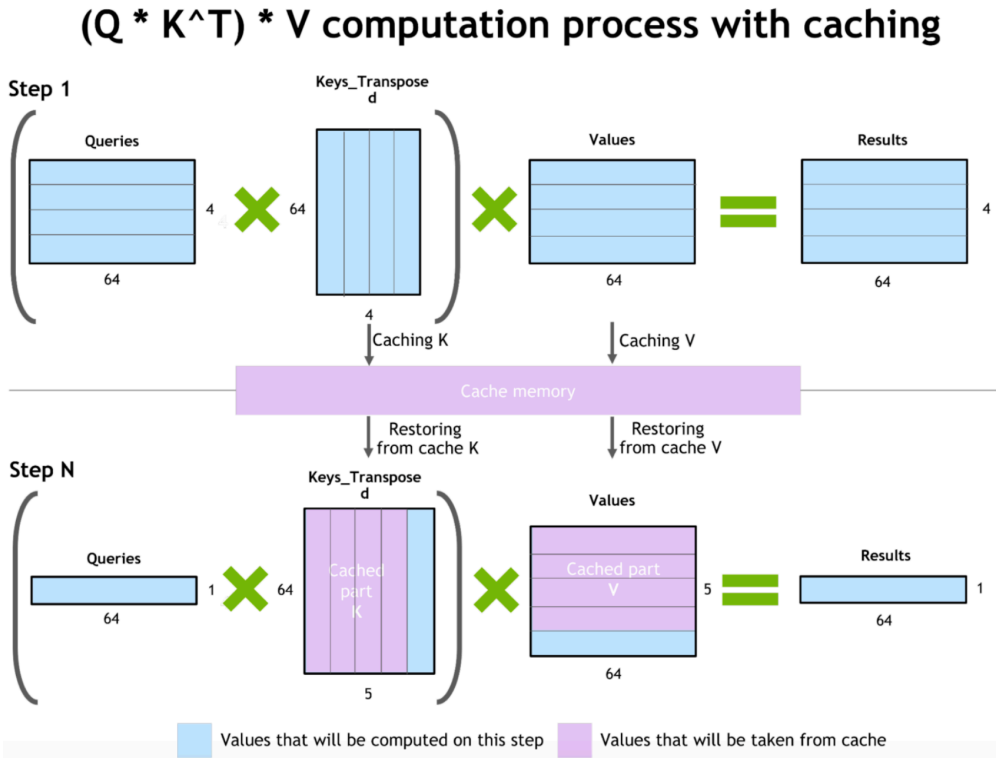
TABLE IV: Latency for Fully-Connected Layers using Quantization and Compression. The compression and quantization ratios are respectively 32:5 and 32:16. Numbers averaged over 30 runs.

有研究指出来自 TEE 计算达部分开销来源于移动数据至 TEE 中所产生的换页，加解密及内存的完整性检查等操作。

- 考虑通过量化和压缩技术，减少需要移动的数据量。
- 考虑 Switchless 零切换技术减少 TEE 移动数据的开销。

Solution 5

5. 充分考虑流水线场景下的缓存切换，以减少 TEE 调用开销。



考虑 Transformer kv cache 与 Solution 3 中的 Token-based Pipeline 配合，使 TEE 缓存移动与计算过程并行。