

# MPCDIFF: Testing and Repairing MPC-Hardened Deep Learning Models

这次介绍的是Qi Pang等人发表在NDSS'24上的论文MPCDIFF，论文链接如下：

<https://www.ndss-symposium.org/ndss-paper/mpcdiff-testing-and-repairing-mpc-hardened-deep-learning-models/>

开源代码如下：

<https://github.com/Qi-Pang/MPCDiff>

## 0. 研究背景

基于安全多方计算的深度学习安全预测已经得到了广泛的关注，但是现在的工作大多集中在协议优化和提升。但是和单纯的安全协议设计不同，深度学习本身的一些问题会在安全预测中也面临巨大的挑战。本文的研究表明，现有的安全预测方案在某些场景下会引入不可忽视的误差，从而使得安全预测结果和明文预测结果完全不一样，也就是说深度学习模型在安全预测下的健壮性远远弱于明文模型。本文率先研究上述问题，进而使用差分检测（differential testing）来检测误差来源，然后对模型进行自动化修复，从而提升修复后的深度学习模型在健壮性。

## 1. 问题分析与动机

在明文模型下，对抗样本是一种特殊的输入，这种输入可以通过对正常输入加入细微扰动得到，从而使得该输入对于人类来说很难区分，但是两种输入对于深度学习模型而言却是天壤之别。对抗样本的存在是模型决策边界引起的，在明文下本就存在。例如下图中的(a)。但是在安全预测中，本文发现由于安全预测协议的截断误差、近似计算等原因，决策边界带来的问题更加严重。如此一来，敌手则更容易获得对抗样本从而影响甚至控制模型的特定输出。如下图中(b)所示，很明显安全预测下的模型预测自信度远远低于明文计算结果。因此设计一种方法，自动化定位引起决策边界的问题的主要诱因，并进而修复该问题是一个亟待解决的问题。

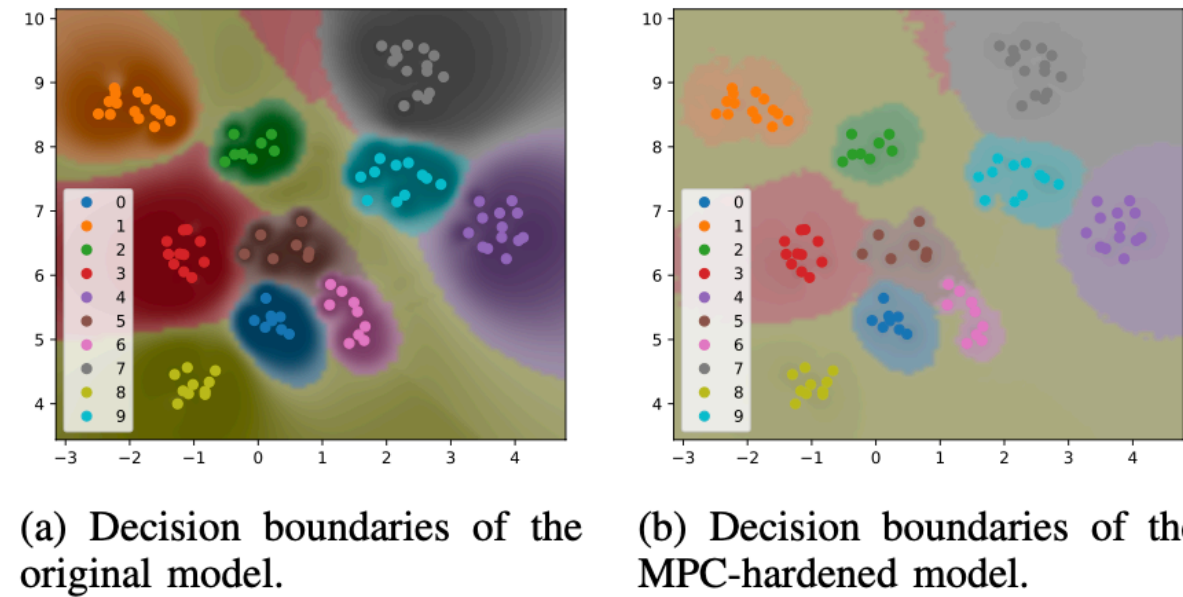


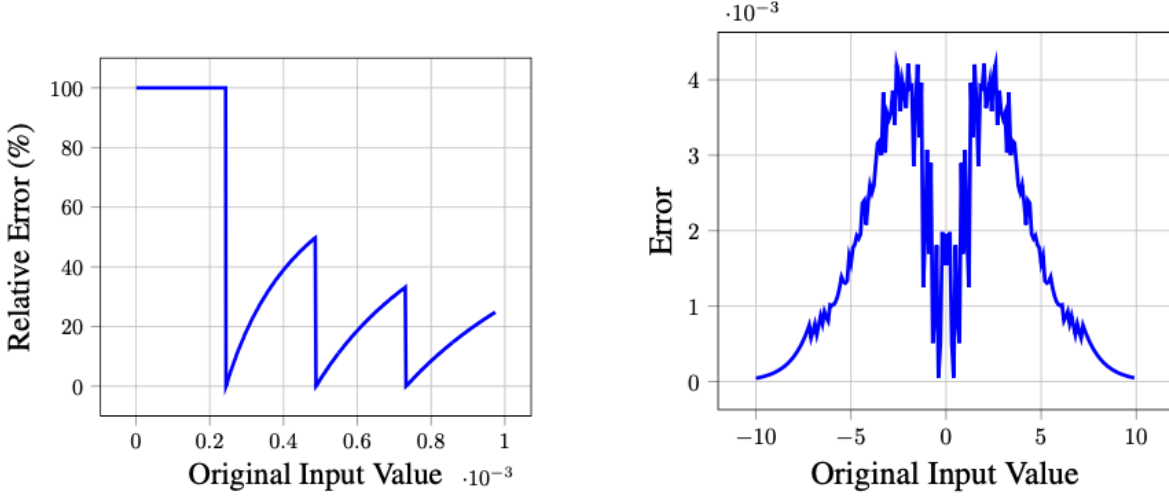
Fig. 1: Depicting decision boundaries of LeNet and its MPC-hardened version for classifying MNIST images. A darker hue denotes a higher confidence of model prediction.

## 2. MPC下深度学习误差分析

现在的深度学习安全预测方案和明文相比主要在以下两点不一样：定点数编码和复杂函数的近似计算。本文首先针对上述两个问题带来的误差进行分析：

**RC1: 定点数误差:** 明文下的模型都是浮点数类型，因此计算的精度和幅度更大，可以承载更多的计算中间结果。但是密码学协议一般都计算在整数上，因此需要将浮点数截断为定点数进而转化到整数上进行近似计算。但是编码的截断会带来误差。如图2的(a)所示，定点数编码的相对误差会随着值的变化而波动，但是不会消息。要想缓解这种误差则需要尽可能保存多浮点数小数部分的比特位 $m$ 。但是，另一方面，如果 $m$ 太大，而数据编码总的比特位数 $\ell$ 是固定的（例如 $\ell = 64$ ），太大的 $m$ 会使得定点数乘法计算的中间结果溢出，从而使得计算结果完全错误。因此选择合适的 $m$ 是解决本文问题的第一个关键点。

**RC2: 近似计算误差:** 另一方面，对复杂激活函数的计算MPC协议一般采用多项式近似计算的方式进行，例如泰勒展开等。过多的展开会导致计算、通信开销太大，但是展开的项太少会引入大量的误差。例如下图(b)表示了Sigmoid函数近似计算的误差。



(a) **RC<sub>1</sub>**: fixed-point value error.

(b) **RC<sub>2</sub>**: Sigmoid approx. error.

Fig. 2: Sample relative errors incurred from **RC<sub>1</sub>** and **RC<sub>2</sub>**.

### 3. MPCDIFF设计

针对上述两个问题，本文设计了MPCDIFF来尽可能减少误差，并进而减少在安全预测中类似对抗样本的特殊触发样本（error-triggering inputs）的数目，提升安全预测系统的健壮性。如下图所示，本文的方案分为如下几部分：differential testing、root cause localization和repair三个阶段。

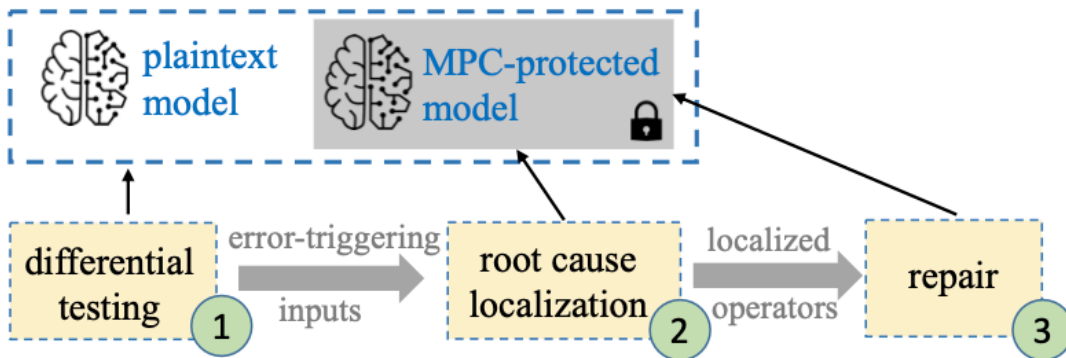


Fig. 3: The workflow of MPCDIFF.

#### 3.1 Differential Testing

在这个阶段，作者提出了上述算法1来寻找样本 $i$ 满足明文预测结果和MPC下的预测结果不同，形式化来说，则是优化下述函数：

$$\max_i : \delta = |M_p(i) - M_m(i)|$$

其中 $p$ 代表明文， $m$ 代表MPC协议。本文定义了超参门限 $T$ ，当 $\delta > T$ ，则对应的 $i$ 被认定为error-triggering输入。

当然，每一个待测的输入 $i'$ 是在选择的初始输入 $i$ 的基础上，添加一定的扰动得到的。同时，添加的扰动要满足小于训练数据的方差，否则得到的 $i'$ 则很容易分辨，失去了意义。具体的算法流程如算法1所示。

---

**Algorithm 1** Feedback-driven differential testing.
 

---

```

1: function OutputDeviation( $M_p, M_m, i', T$ )
2:    $\text{PRED}_{plaintext} \leftarrow \text{PREDICTVECTOR}(M_p(i'))$ 
3:    $\text{PRED}_{mpc} \leftarrow \text{PREDICTVECTOR}(M_m(i'))$ 
4:    $\delta \leftarrow \|\text{PRED}_{plaintext} - \text{PRED}_{mpc}\|_2$ 
5:   if  $\delta > T$  then  $\triangleright$  Output deviation larger than the threshold.
6:     return true
7:   return false

8: function DT(Corpus of Seed Inputs  $\mathcal{S}, M_p, M_m$ )
9:    $\mathcal{Q} \leftarrow \mathcal{S}, \mathcal{O} \leftarrow \emptyset$ 
10:  while #total mutations < 15,000 do
11:     $i \leftarrow \text{CHOOSENEXT}(\mathcal{Q})$ 
12:     $\text{PRED}_{plaintext} \leftarrow \text{PREDICTVECTOR}(M_p(i))$ 
13:     $\text{PRED}_{mpc} \leftarrow \text{PREDICTVECTOR}(M_m(i))$ 
14:     $T \leftarrow \|\text{PRED}_{plaintext} - \text{PRED}_{mpc}\|_2$ 
15:     $p \leftarrow \text{ASSIGNENERGY}(i)$ 
16:    for 1 ...  $p$  do
17:       $i' \leftarrow \text{MUTATE}(i)$ 
18:      if  $M_p(i') \neq M_m(i')$  then  $\triangleright$  Model prediction changes.
19:        add  $i'$  in  $\mathcal{O}$ 
20:      else if OutputDeviation( $M_p, M_m, i', T$ ) = true then
21:        add  $i'$  in  $\mathcal{Q}$ 
22:  return  $\mathcal{O}$ 

```

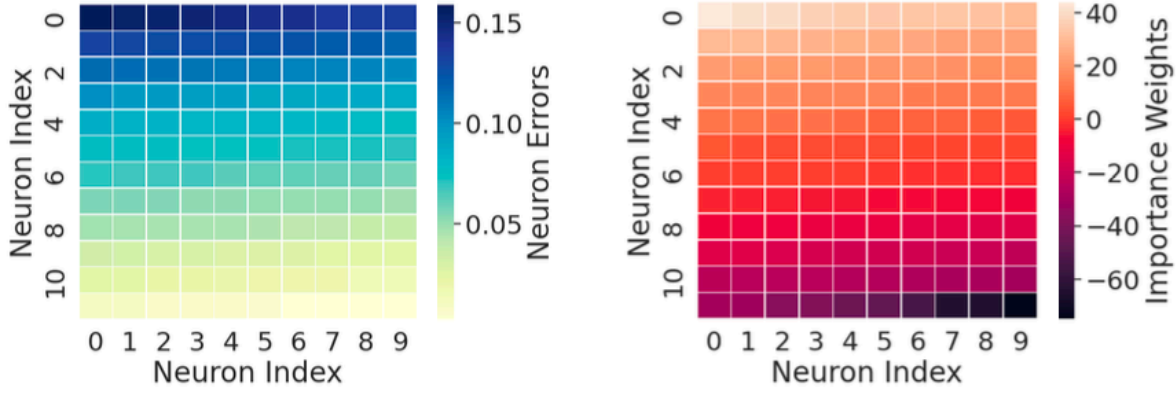
---

### 3.2 Root Cause Localization

在3.1步得到了相关的error-triggering输入之后，MPCDIFF接下来的目标则是定位产生上述输入的原因。在这一步，本文对于每一个神经元 $n_i$ 都赋予了一个权重 $w_i$ ，初始化为0。该权重刻画了神经元 $n_i$ 对于 $M_m$ 误分类的贡献。

具体来说，本文同时用 $M_p$ 和 $M_m$ 处理输入 $o \in \mathcal{O}$ ，如果两个模型之中对应神经元之间的输出差距 $\delta_i(o) > \tau_+$ ，那么 $w_i \leftarrow w_i + 1$ ；如果 $\delta_i(o) < \tau_-$ ，则 $w_i \leftarrow w_i - 1$ 。

同时，上述两个超参门限 $\tau_+$ 和 $\tau_-$ 是根据所有 $\delta_i(o)$ 设置，区分最大的三分之一和最小的三分之一的边界。热力图展示如下(a)所示。经过上述更新， $w_i$ 的热力图如下图(b)所示。



(a) Heatmap of the neurons' difference  $\delta_i(o)$ , the neurons are ranked w.r.t. their  $\delta_i(o)$ . The first four rows of the neurons will increase their importance weights  $\omega_i$ , and the last four rows will decrease their  $\omega_i$ .

(b) Heatmap of the neurons' importance weights  $\omega_i$ , the neurons are ranked w.r.t.  $\omega_i$ . The neurons with large weights are marked as important, and the neurons with small weights are marked as trivial.

Fig. 4: Heatmap of neuron difference  $\delta_i(o)$  and neuron importance weights  $\omega_i$ .

### 3.3 Repair

最后，定位到问题之后，MPCDIFF则需要修复问题。对于RC1，调节 $m$ 可以提升安全预测的健壮性，但是单纯的调节 $m$ 几乎不可能得到一个有效的方案。

因此，本文对于 $w_i$ 较大的神经元则使用更加复杂精确的近似算法，例如使用更多的泰勒展开项。选择的数量则是用 $\alpha$ 表示。这样增加 $\alpha$ 神经元的计算复杂度会带来更多的开销，但是实验表明开销并不是很大。另外一方面，计算方式的不同会暴露 $w_i$ 的相对差异，但本文认为这个信息的泄漏对于模型和数据隐私泄漏是可以接受的。

## 4. 实验与分析

本文在Crypten、TF-Encrypted和PySyft三个框架上验证了MPCDIFF的有效性。同时，本文在执行 $M_m$ 计算的时候都是在localhost下进行调节，因此开销在本文实验设置和数据、模型规模下是可以接受的。

下表1展示了本文选择的多种框架、模型、数据集和在不同配置下的准确率，已经相关的MPCDIFF执行结果等信息。

TABLE I: Evaluation setup and statistics.

Framework	Model	Datasets	Plaintext Accuracy	Encrypted Accuracy	#Non-linear Operations	#Multiplication Excluding Non-linear Operations	#Initial Seeds	#Initial Error-Triggering Seeds	Avg. Inference Time Per Input
CrypTen	LeNet	MNIST	98.65%	97.25%	6,734	20,844,064	2,000	32	1.57s
	MLP-Sigmoid	Credit	82.93%	80.70%	120	92,280	1,000	2	0.50s
	MLP-GELU	Bank	90.00%	89.90%	250	225,000	1,000	1	0.56s
TF-Encrypted	LeNet	MNIST	98.20%	96.90%	6,734	20,844,064	2,000	41	0.27s
	MLP-Sigmoid	Credit	82.93%	80.10%	120	92,280	1,000	12	0.04s
	MLP-GELU	Bank	90.10%	90.10%	250	225,000	1,000	2	0.05s
PySyft	LeNet	MNIST	97.95%	97.35%	6,530	20,844,064	2,000	18	2.12s
	MLP-Sigmoid	Credit	82.93%	80.70%	120	92,280	1,000	2	0.27s
	MLP-GELU	Bank	90.10%	89.40%	250	225,000	1,000	1	0.35s

进一步，图5和6则是展示了MPCDIFF发现的error-triggering样本数量和一些L2距离信息。

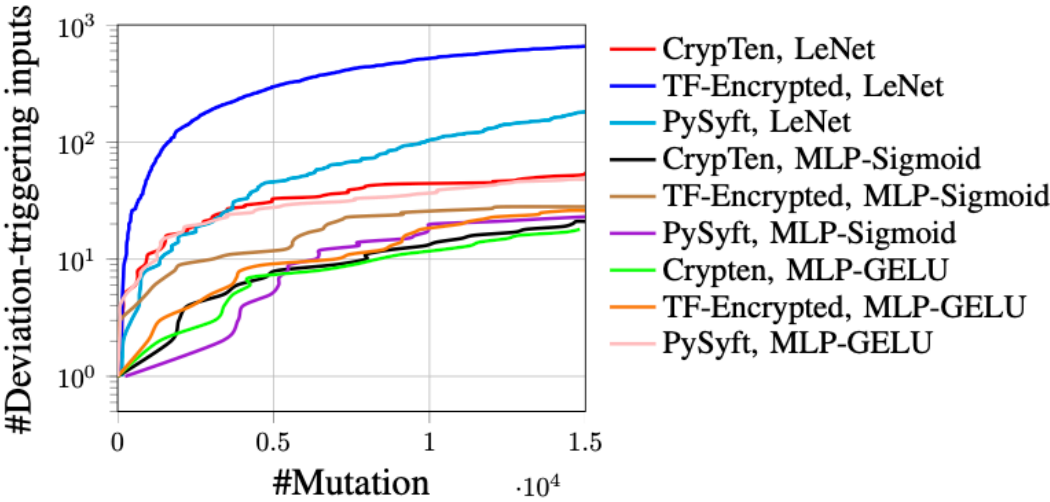


Fig. 5: #Deviation-triggering inputs found by MPCDIFF. Deviation-triggering inputs retain correct predictions in plain-text models but trigger wrong predictions in MPC-hardened models.



















MPC Framework	Datasets	Error-Inducing Inputs	Avg. L2-Distance
CrypTen	MNIST	      ...	0.0018
	Credit	[0.000, 1.000, ..., 0.014, 0.039] ...	0.019
	Bank	[0.494, 0.454, ..., 0.957, 0.860] ...	0.018
TF-Encrypted	MNIST	      ...	0.0029
	Credit	[0.010, 0.000, ..., 0.276, 0.009] ...	0.0022
	Bank	[0.197, 0.636, ..., 0.000, 0.170] ...	0.032
PySyft	MNIST	      ...	0.0034
	Credit	[0.802, 0.000, ..., 0.846, 0.297] ...	0.023
	Bank	[0.049, 0.727, ..., 0.060, 0.106] ...	0.015

Fig. 6: Examples of deviation-triggering inputs found by MPCDIFF.

图7则展示了error-triggering样本和test样本的分类准确率与 $m$ 的关系(a和b)，而子图(c,d)则分别展示了修复后的模型error-triggering样本分类准确率与 $\alpha$ 和修复后模型所带来的额外开销与 $\alpha$ 的关系。



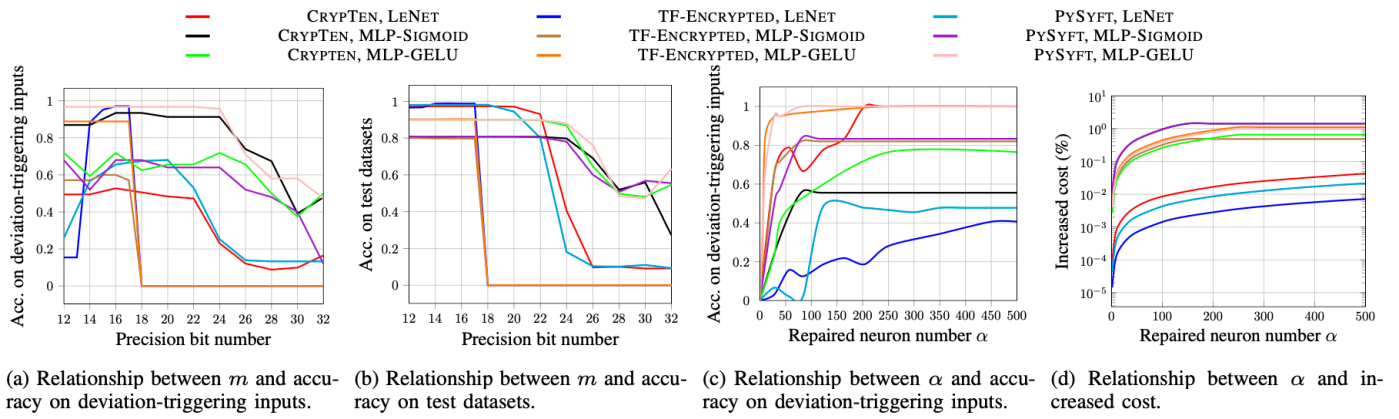


Fig. 7: Results of repairing MPC-hardened models.

而下图则展示了修复后模型和原始模型的决策自信分数，可以看到修复后的模型和原始明文模型已经基本一致了。

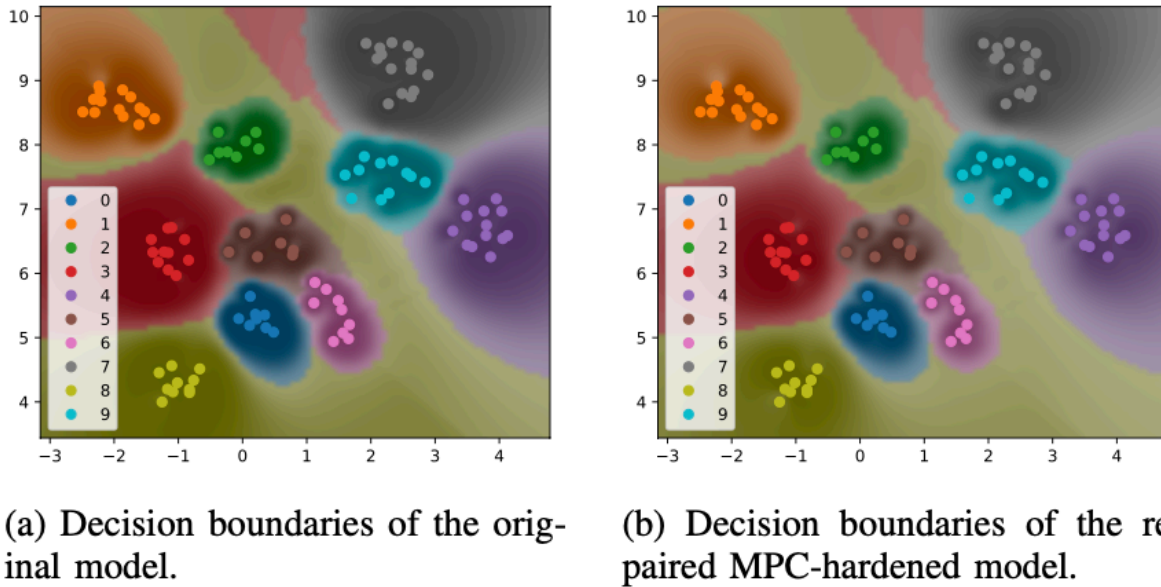


Fig. 8: Depicting decision boundaries of LeNet and its repaired MPC-hardened version for classifying MNIST. A darker hue denotes a higher confidence of model prediction.

最后，作者再次用MPCDIFF测量修复后模型的error-triggering样本的数量，可以发现这时能发现的样本数量已经大大降低了。

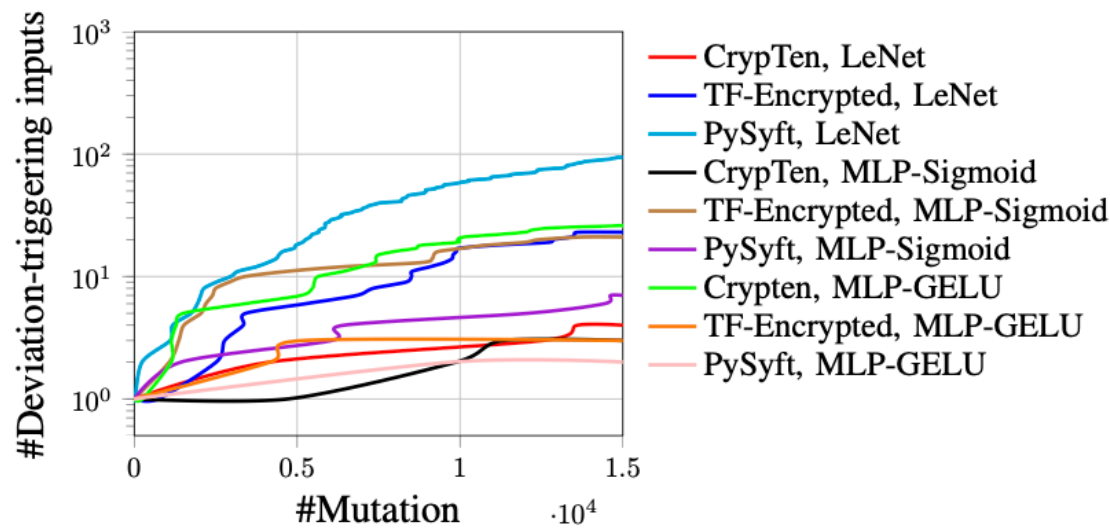


Fig. 9: #Deviation-triggering inputs found by MPCDIFF on repaired models.

需要说明的是，即便是MPCDIFF不能发现修复后模型任何的error-triggering样本了，那也不能说明修复后的模型已经完全没有了MPC协议带来的error-triggering问题，只能说明健壮性有所改善和提升。