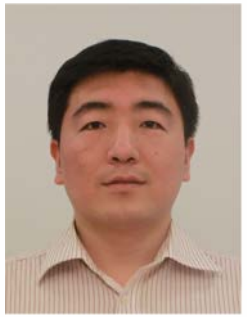
Hermes Attack: Steal DNN Models In AI Privatization Deployment Scenarios

Yueqiang Cheng*, Yuankun Zhu#, Husheng Zhou$
*Baidu Security, #University of Texas at Dallas, $Vmware

People



Dr. Yueqiang Cheng
System Security and Hardware Security
https://sites.google.com/site/strongerwillcheng/

Mr. Yuankun Zhu
GPU Security
https://www.linkedin.com/in/yuankun-zhu-b87b29a1/

Dr. Husheng Zhou
GPGPU, HPC
https://www.linkedin.com/in/hushengzhou

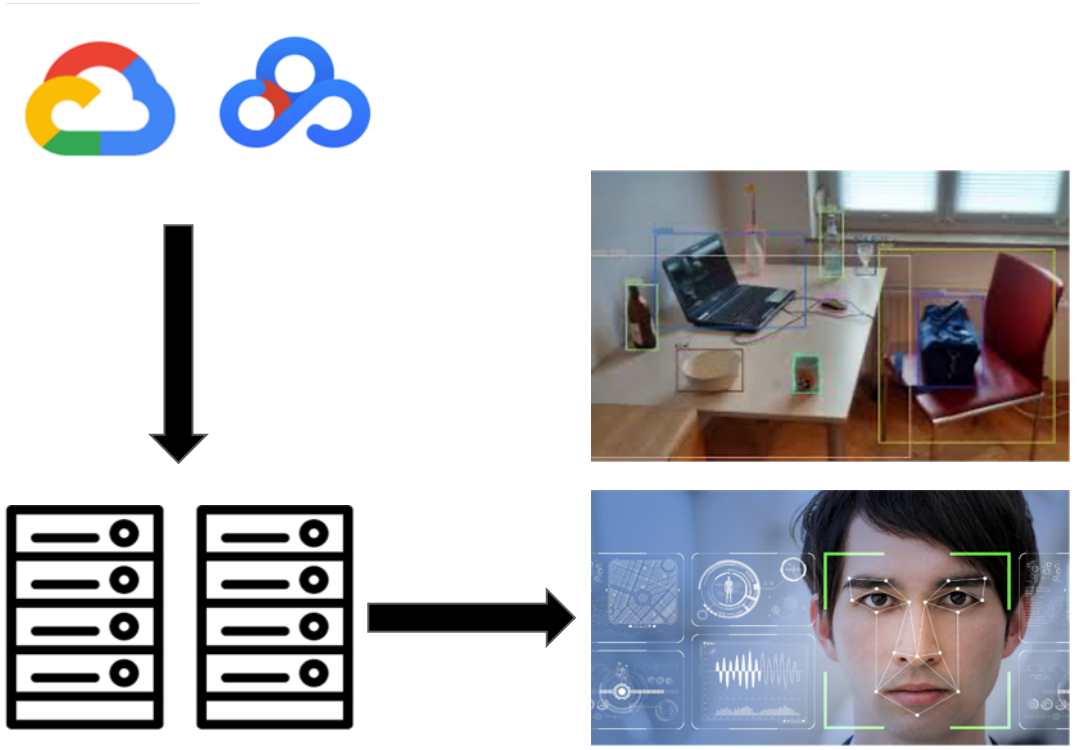
Hermes is the master of thieves and the god of stealth
https://en.wikipedia.org/wiki/Hermes

Hermes Attack @ Usenix Security 2021 https://arxiv.org/abs/2006.12784

Motivations

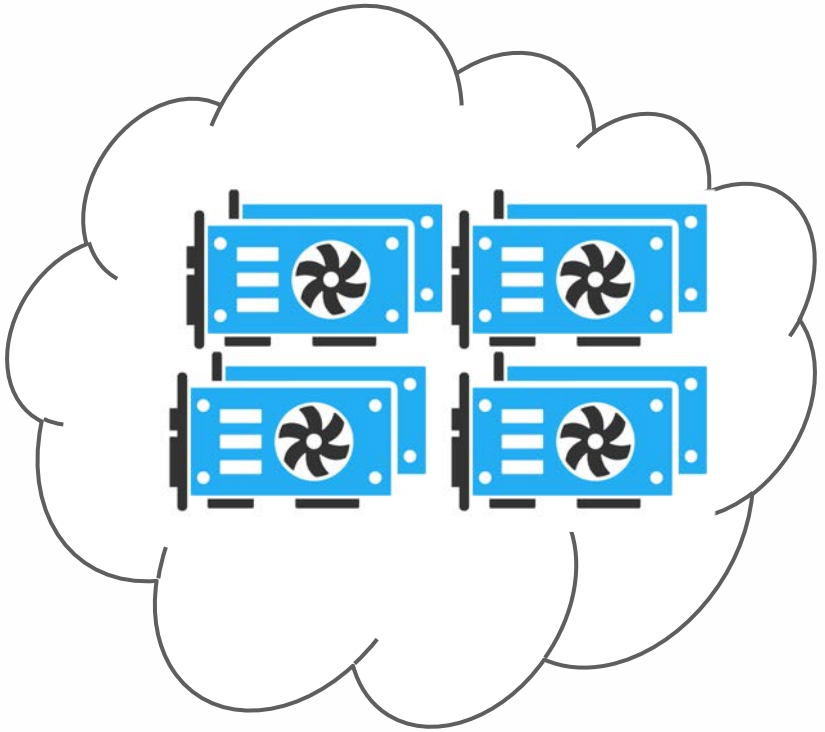
Autonomous Driving

Privatization Deployments



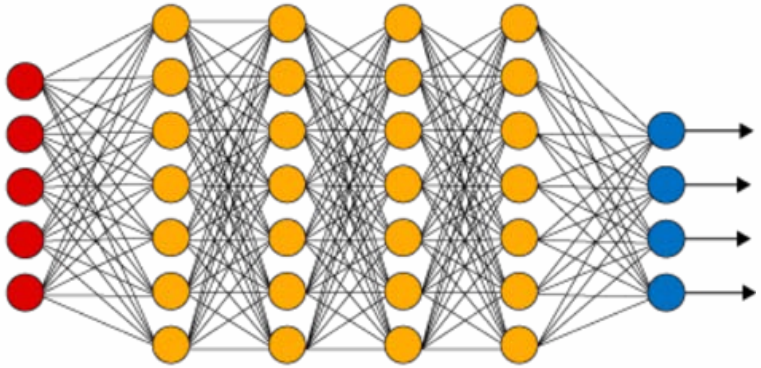
Motivations

$4.5/hr per TPU



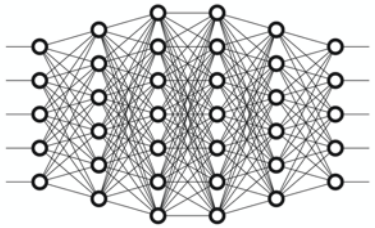
≈$400,000

DNN Model



Motivations

All existing attacks can ***only*** reconstruct ***partial*** model



Architecture

- **Kernel size = (3,3)**
- **Pool size = (3,3)**
- **Strides = (1,1)**

...

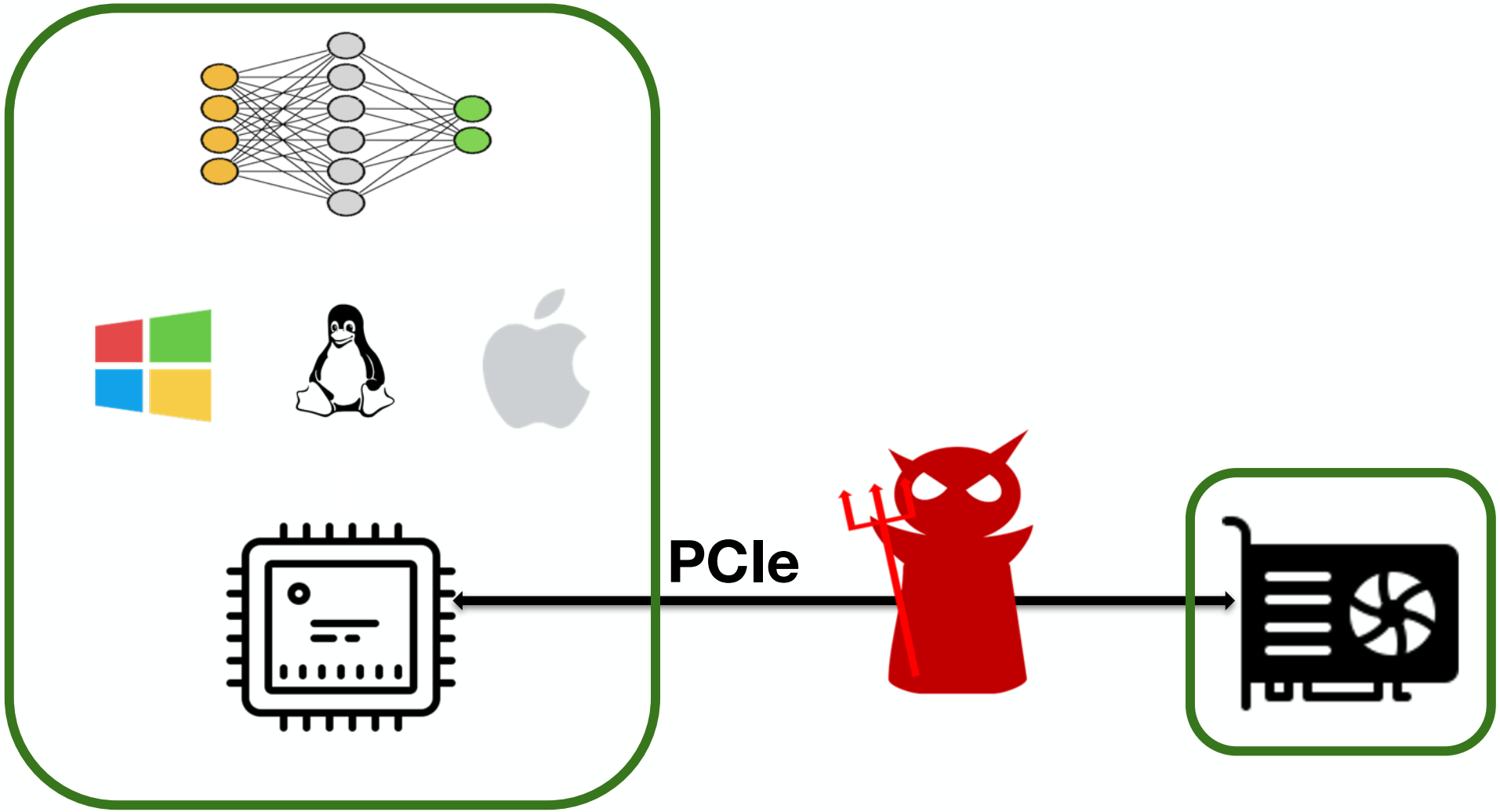
Hyper-parameters

1101010101
1010101110
1101010111
1001011001

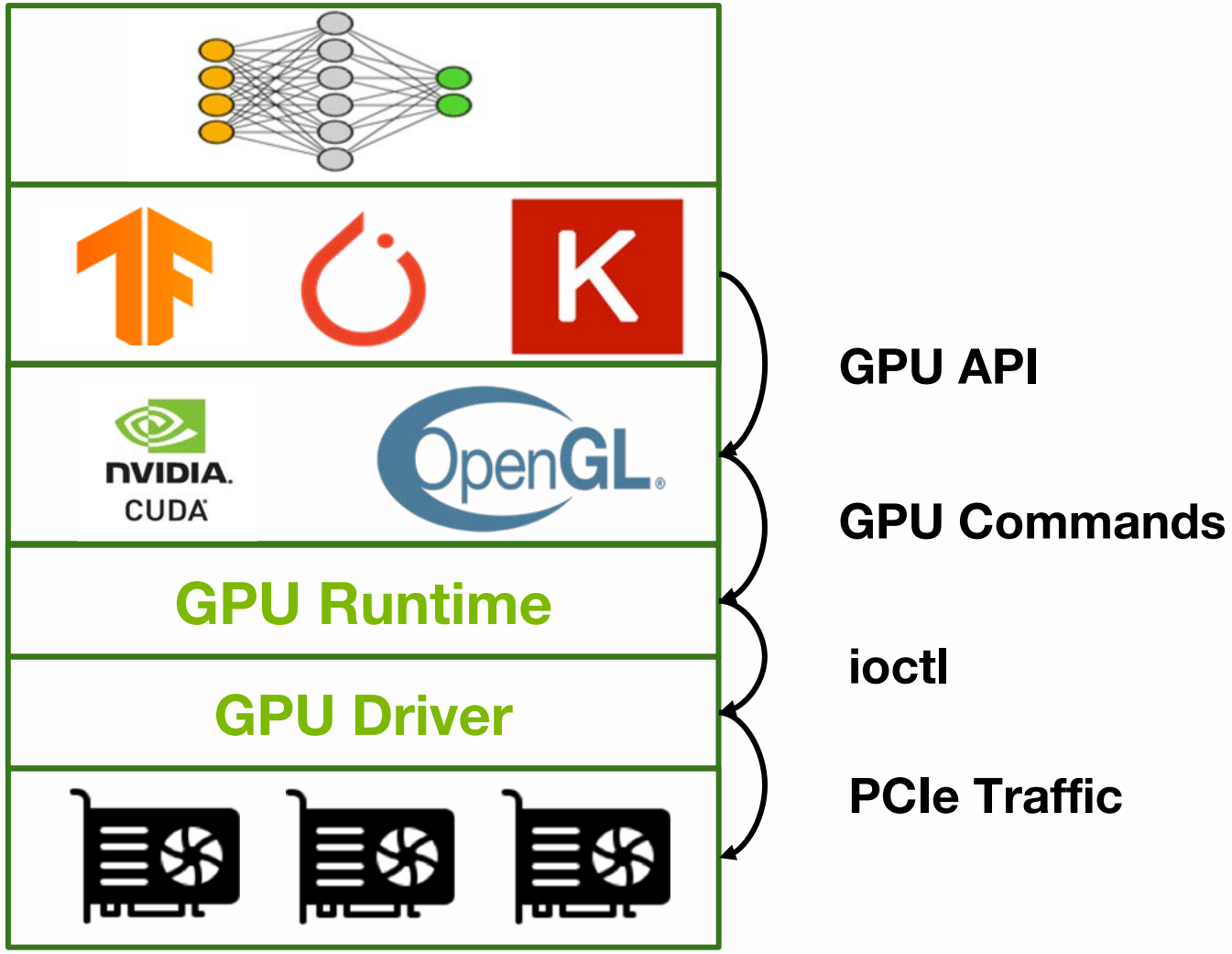
Parameters

Hermes Attack is the ***first*** work that can ***fully*** steal DNN model with ***lossless*** interface accuracy

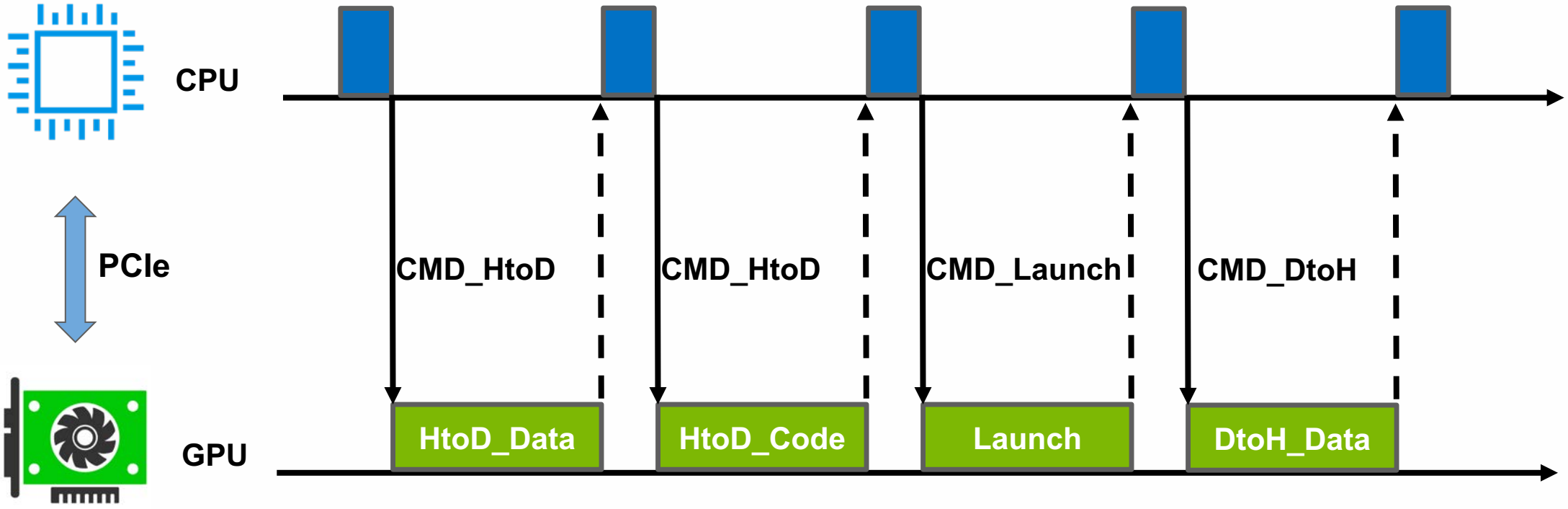
Adversarial Scenario



DNN System Stack



How GPU Works

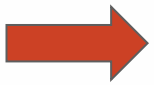


O1: **Command driven** interaction working mode

O2: All data and code are passed through **PCIe bus**

Challenges and Solutions

C1: Closed-source Code and Undocumented Data Structures

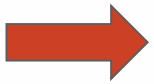


S1: Reverse Engineering

C2: Numerous Noises and Chaotic Orders in PCIe packets

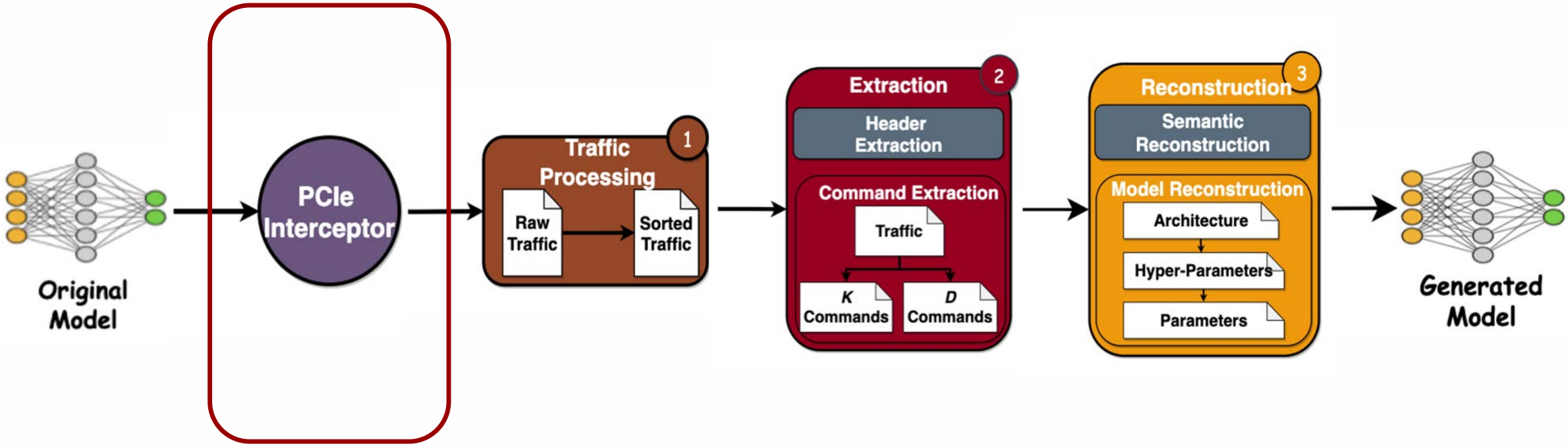
S2: Sanitization and Order Correction

C3: Semantic Loss in PCIe Traffic

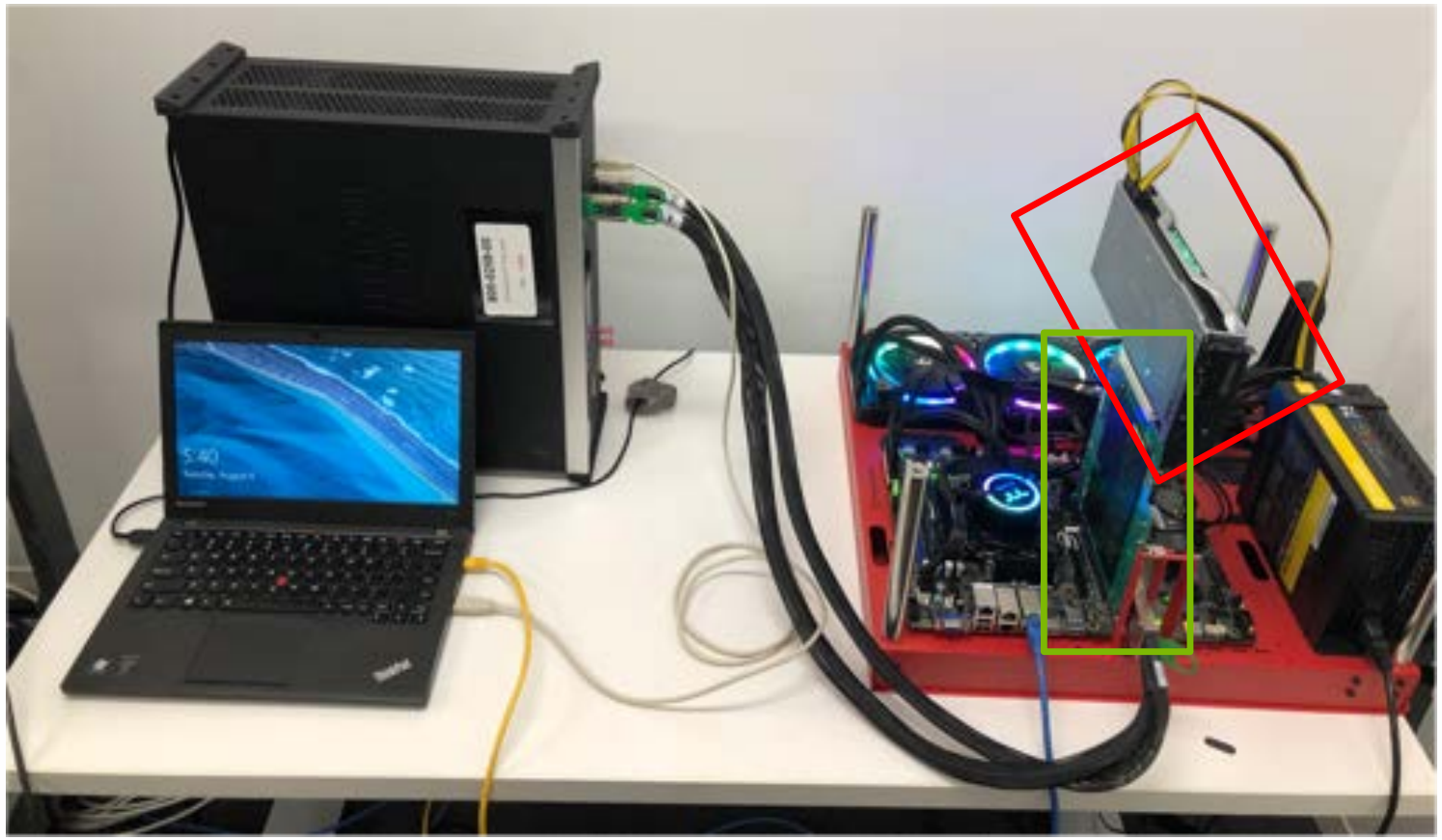


S3: Semantic Reconstruction

Attack Overview



PCIe Interception



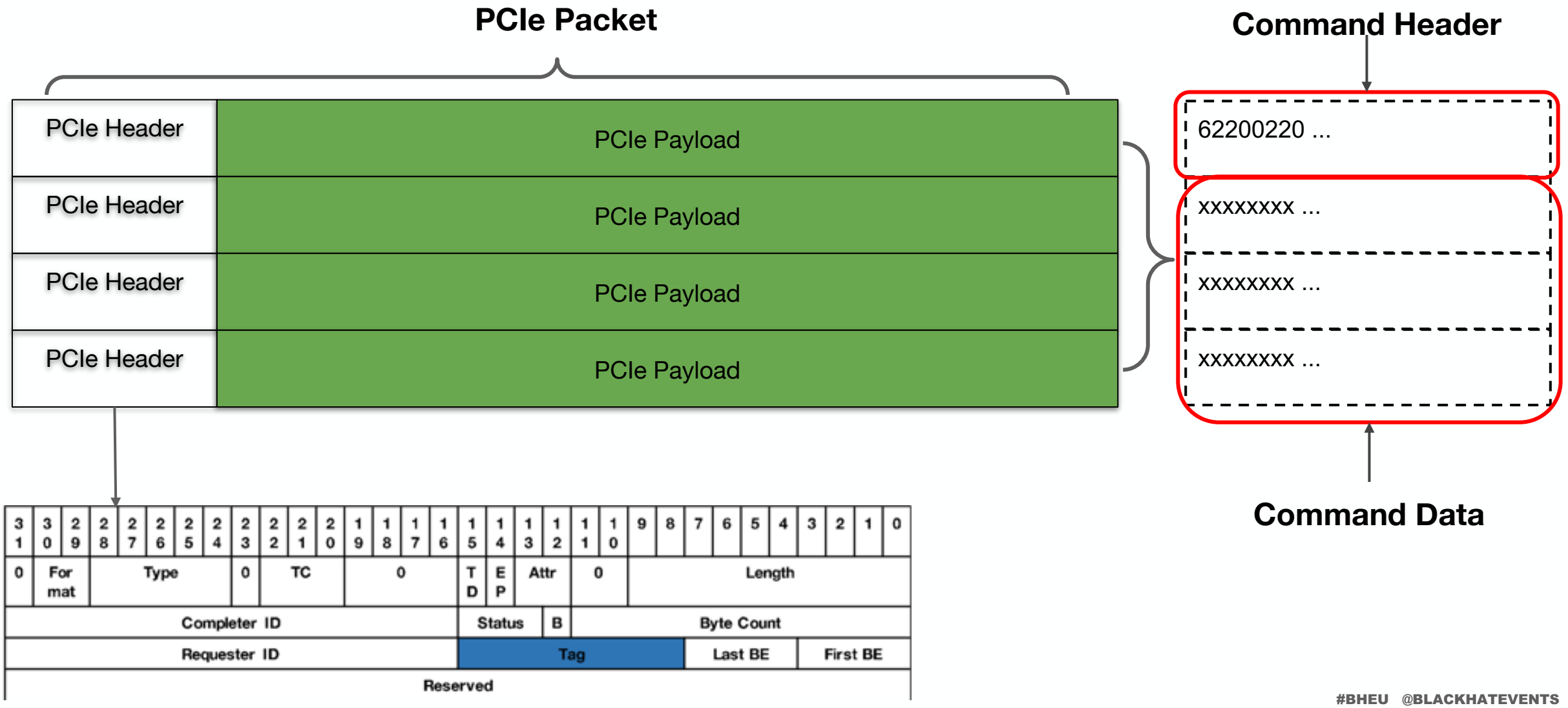
- Operating System: Ubuntu 16.04
- Tested GPU platform:
  - NVIDIA Geforce GT 730
  - NVIDIA Geforce RTX 1080 Ti
  - NVIDIA Geforce RTX 2080 Ti

- GPU Programing Interface: CUDA 10.1
- GPU Driver: NVIDIA 418.43

- PCIe Interceptor: PCIe protocol Analysis

- DNN model: MNIST, VGG, and ResNet
- DNN platform: keras + tensorflow

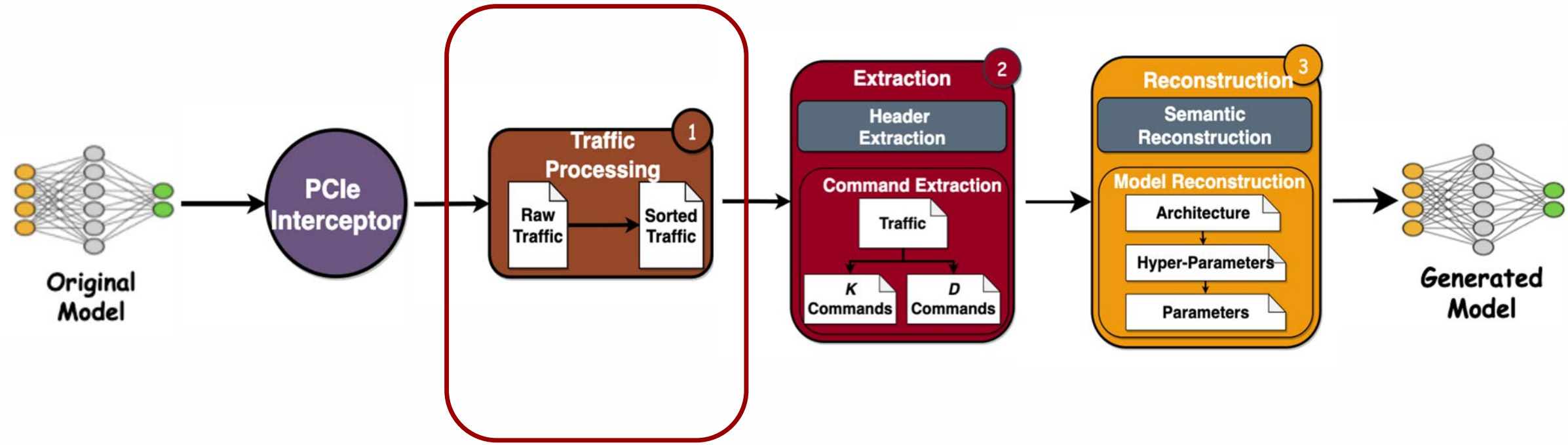
Terminologies



PCIe Packet Relationship

Address and Payload are connected through the same tag value

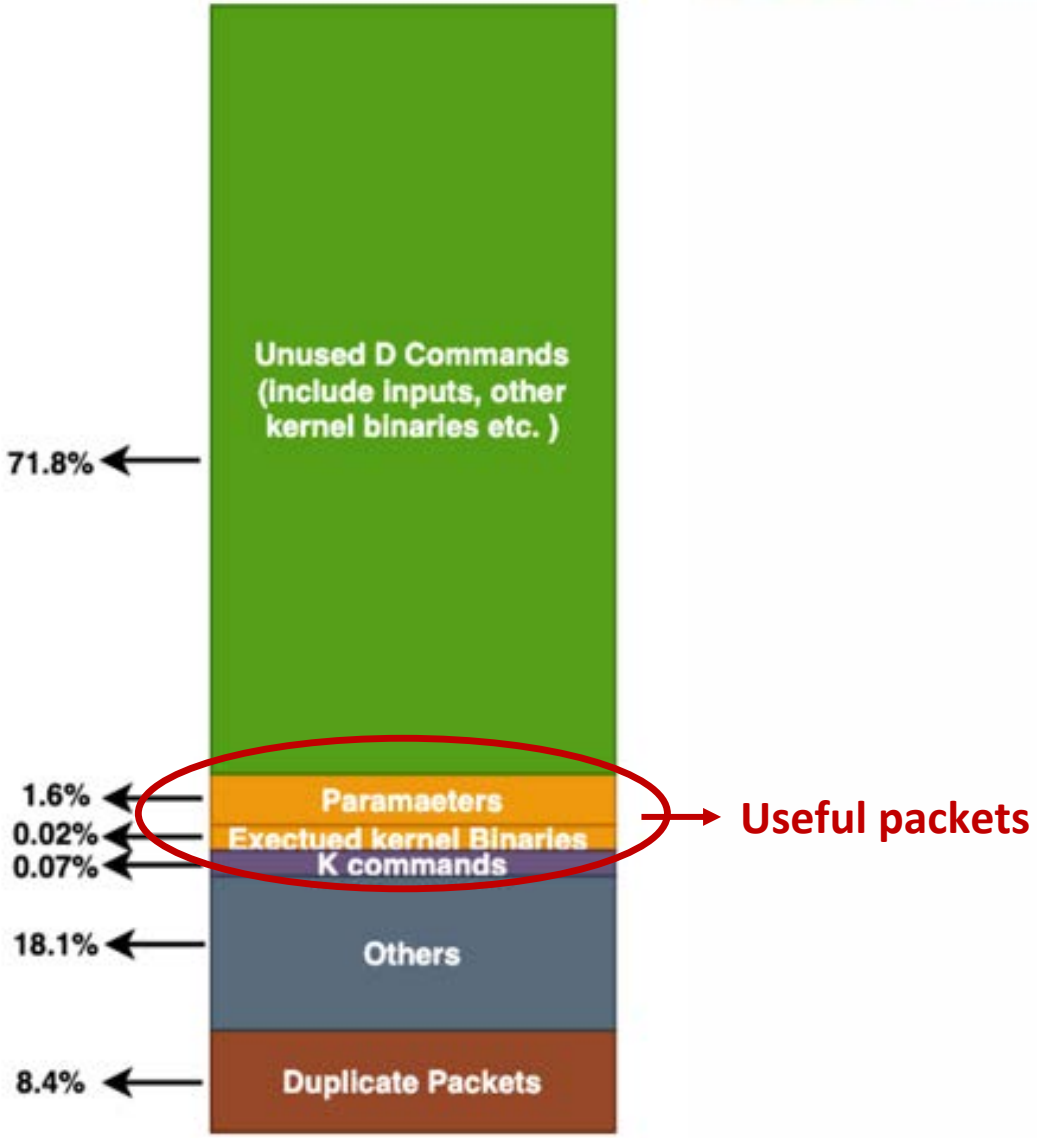
Attack Overview



Numerous Noises

- In a Traffic sample:
  - Number of PCIe packets: 1,077,757
  - Number of useful packets: ~20,000
  - 98.4% of the packets are noise

Focus on specific GPU Commands!



Traffic Processing

- PCIe Bus Snooping Device: PCIe protocol Analysis

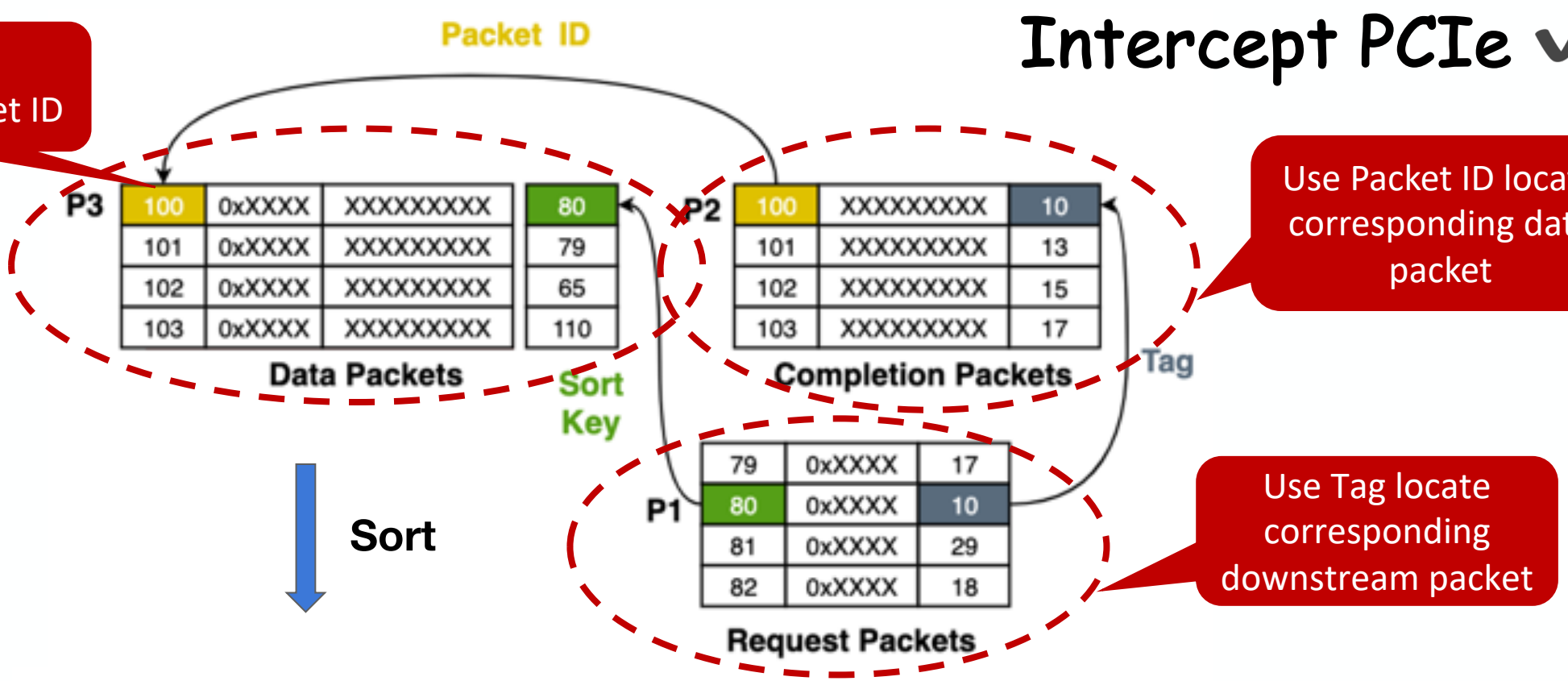
Packet	Dir	Type	Length	Address	Payload
168	R→	MRd64/CplD (SC)	16	00000008:1B521080	00000000 0DD06A81 00000000 0DD16A81 00000000 0DD26A81 00000000 0DD36A81 ...
169	R→	MRd64/CplD (SC)	16	00000008:1B521040	00000000 0D5AAE82 00000000 0D5BAE82 00000000 0DD8B281 00000000 0DD9B281 ...
170	R→	MRd64/CplD (SC)	16	00000008:1B5210C0	00000000 0D94977E 00000000 0D95977E 00000000 0D96977E 00000000 0D97977E ...
171	R→	MRd64/CplD (SC)	32	00000008:1B521100	00000000 0D549581 00000000 0D559581 00000000 0D569581 00000000 0D579581 ...
172	R→	MRd64/CplD (SC)	16	00000008:1B521180	00000000 0D10D082 00000000 0D11D082 00000000 0D12D082 00000000 0D13D082 ...
173	R→	MRd64/CplD (SC)	16	00000008:1B521200	00000000 0D1C157E 00000000 0D1D157E 00000000 0D1E157E 00000000 0D1F157E ...
174	R→	MRd64/CplD (SC)	16	00000008:1B5211C0	00000000 0D908E81 00000000 0D918E81 00000000 0D928E81 00000000 0D938E81 ...
175	R→	MRd64/CplD (SC)	16	00000008:1B52124	00000000 0D507681 00000000 0D517681 00000000 0D527681 00000000 0D537681 ...
176	R→	MRd64/CplD (SC)	32	00000008:1B521280	00000000 0D28D17B 00000000 0D29D17B 00000000 0D2AD17B 00000000 0D2BD17B ...
177	R→	MRd64/CplD (SC)	32	00000008:1B521300	00000000 0DF46B81 00000000 0DF56B81 00000000 0DF66B81 00000000 0DF76B81 ...
178	R→	MRd64/CplD (SC)	32	00000008:1B521400	00000000 0DEC637E 00000000 0DED637E 00000000 0DEE637E 00000000 0DEF637E ...
179	R→	MRd64/CplD (SC)	32	00000008:1B521380	00000000 0DF8AE82 00000000 0DF9AE82 00000000 0DFAAE82 00000000 0DFBAE82 ...
180	R→	MRd64/CplD (SC)	32	00000008:1B521480	00000000 0D2C8781 00000000 0D2D8781 00000000 0D2E8781 00000000 0D2F8781

Out of order!

Sort Payload

Intercept PCIe ✓

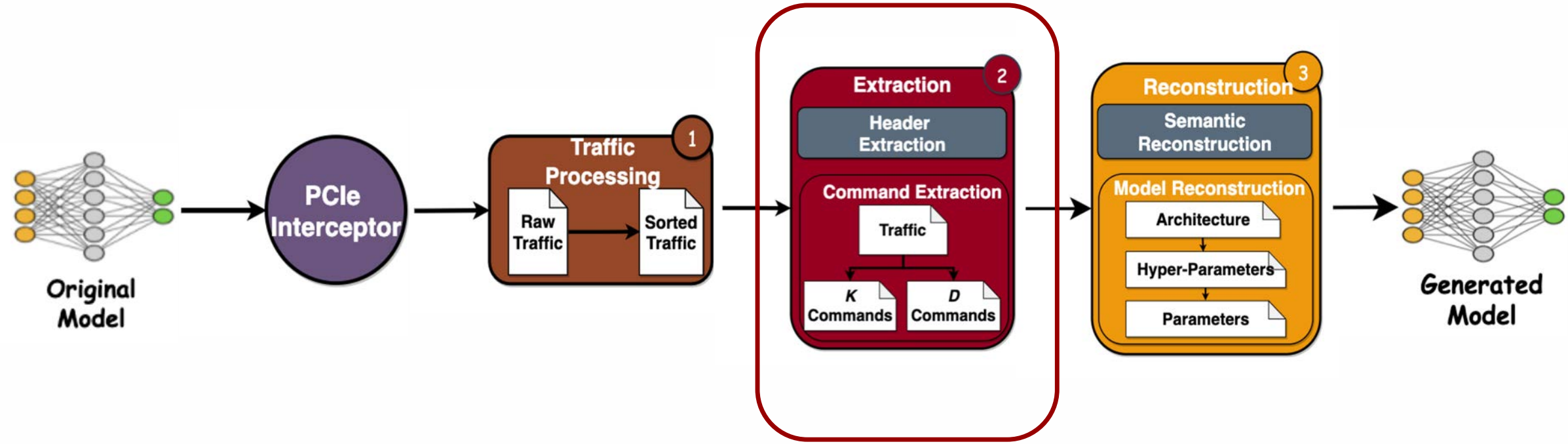
Sort by the upstream packet ID



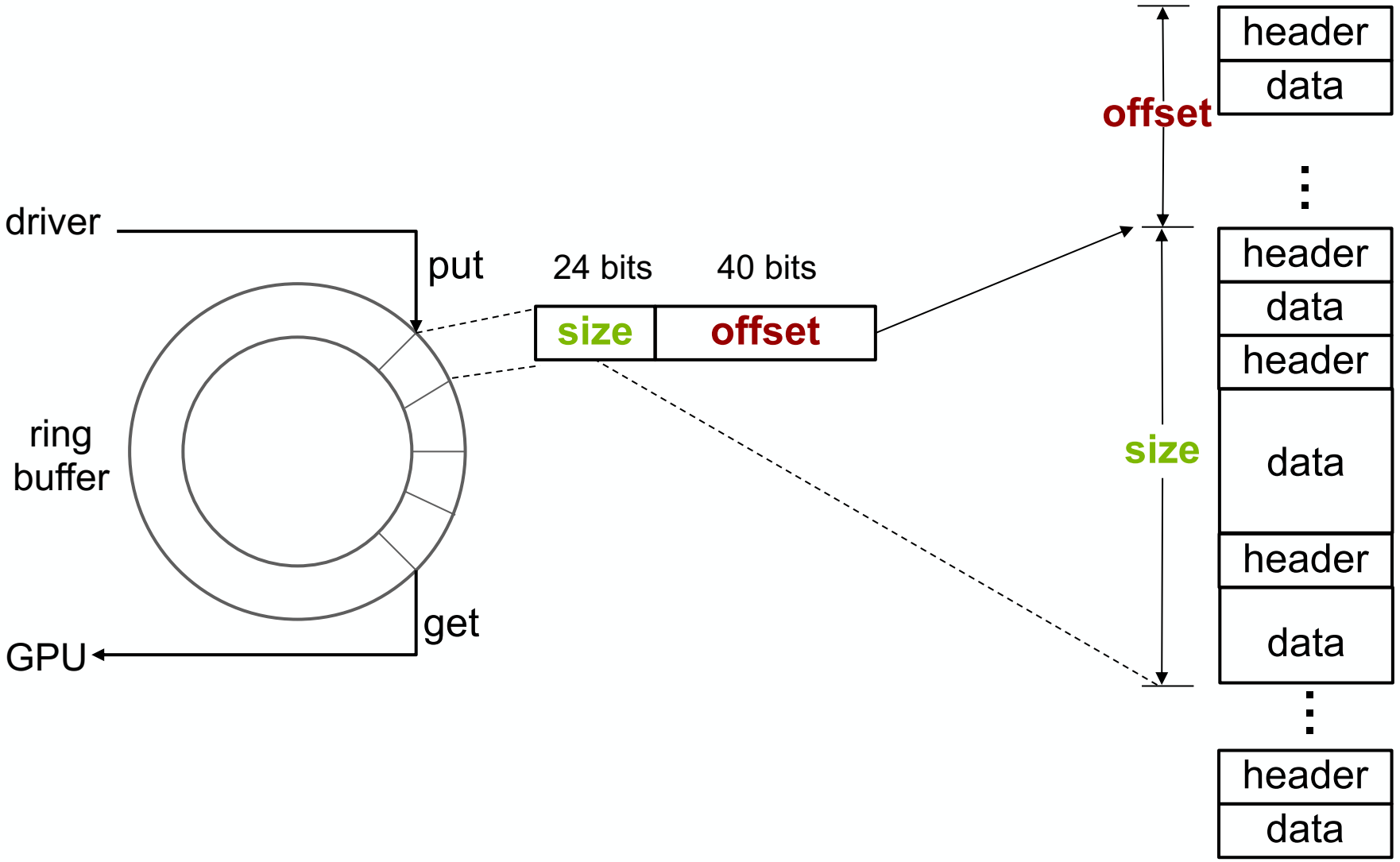
Use Packet ID locate corresponding data packet

Use Tag locate corresponding downstream packet

Attack Overview



GPU Command


driver
put
24 bits
40 bits
size
offset
ring
buffer
get
GPU
offset
size
header
data
header
data
header
data
header
data
header
data

Header Extraction

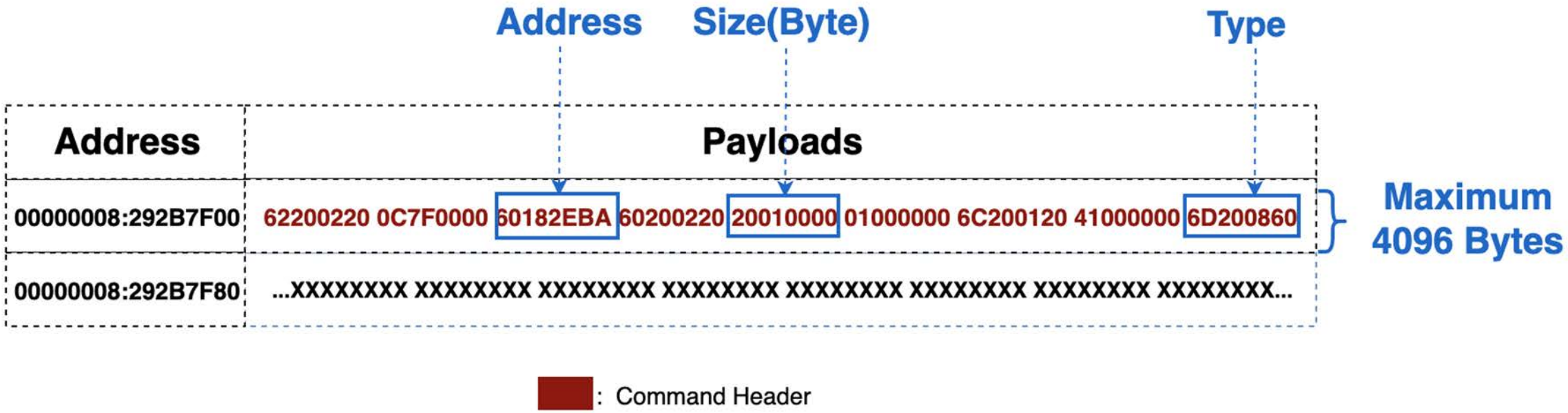
- GPU is controlled by CPU using commands. Command types includes:
  - GPU Initialization Command
  - Synchronization Command
  - Data Movement Command
  - Kernel Launch Command

- We only need **Data Movement Commands**, noted as **D commands**, and **Kernel Launch Commands**, noted as ***K* commands**.

- Identify headers of ***D* cmds** and ***K* cmds** through offline training.

Extract Header From PCIe Packet

Example of a GPU command header

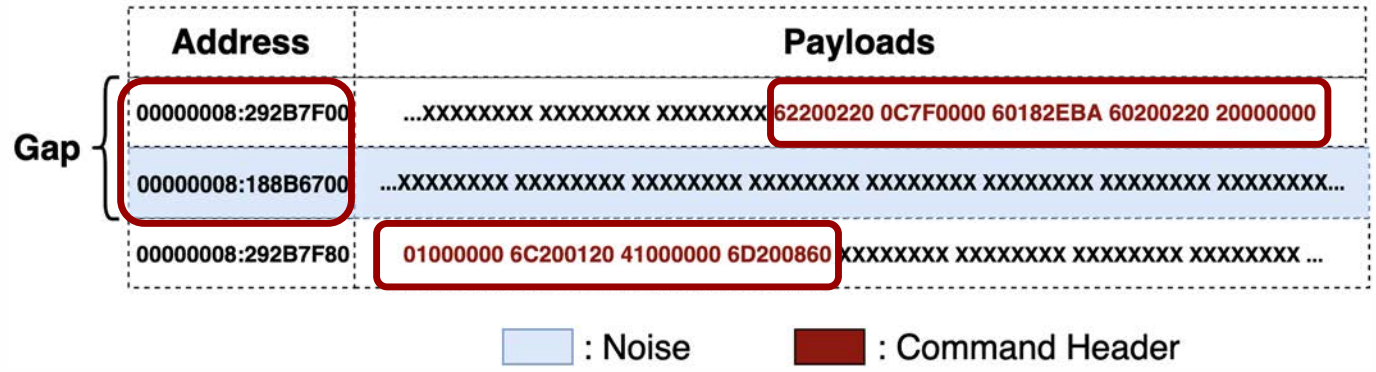


Command Extraction

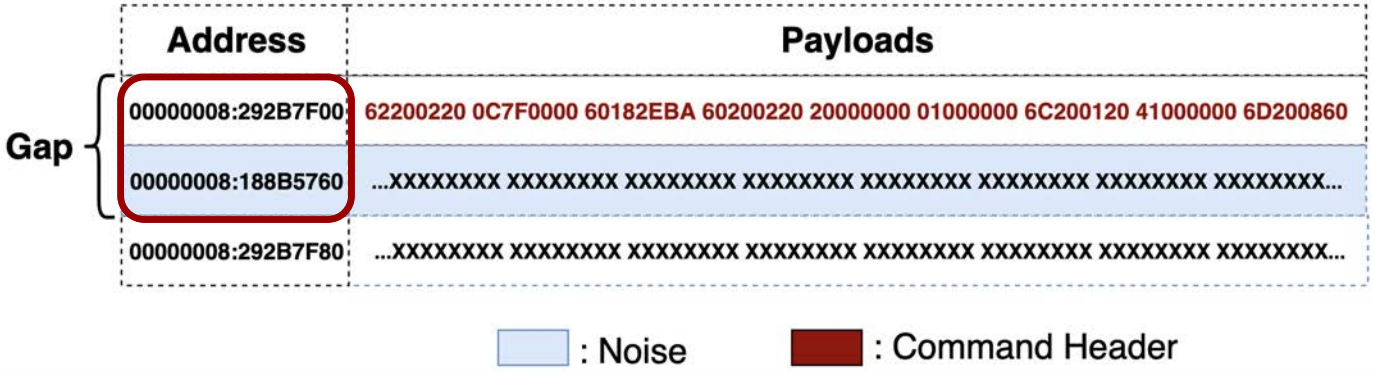
How to handle noise?

We can extract commands through identified command headers.

Command Extraction



Command Header Noise: when a command header has been split into two packets, the noise may appear between these two packets.

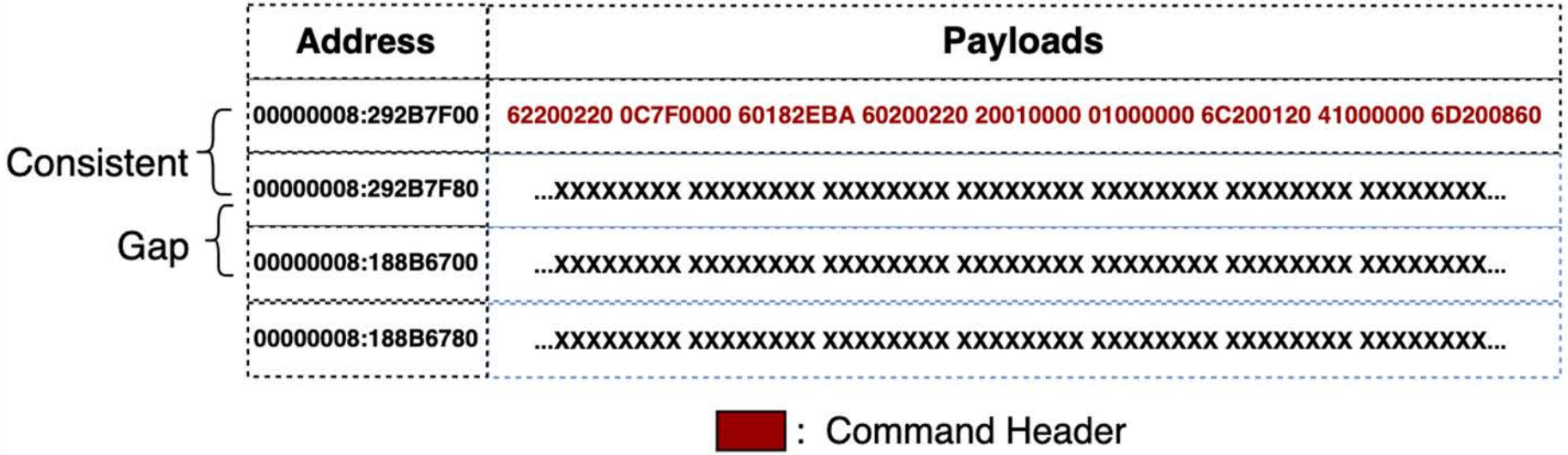


Command Data Noise: noise packet appeared among command data packets.

Check address consistency !

However, only check address consistency is not enough…

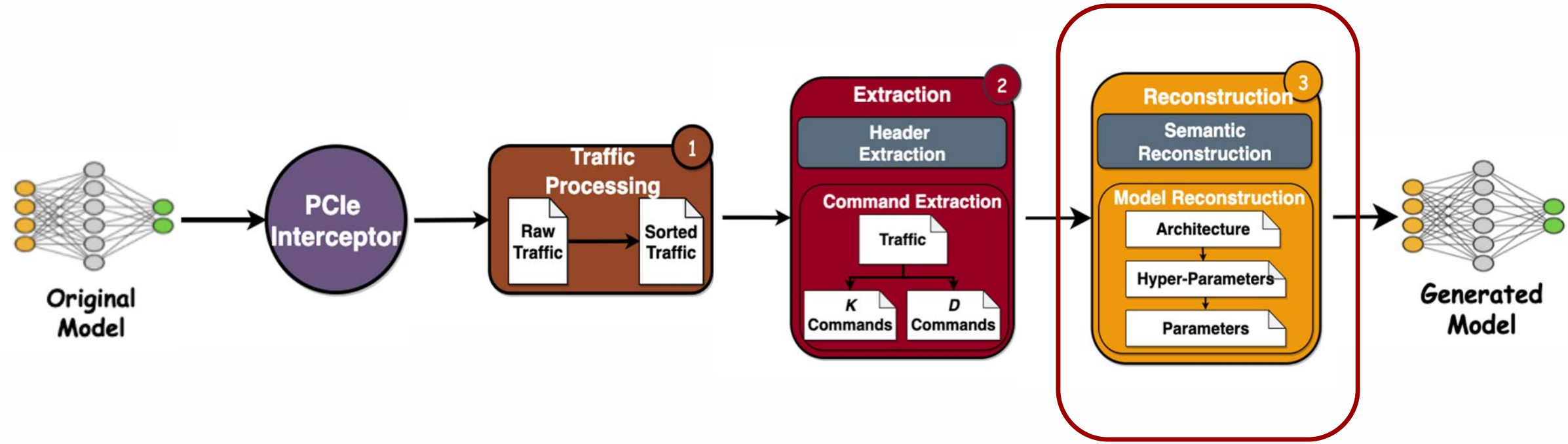
Command Extraction (cont.)



Introduce MAX_SCAN_DISTANCE

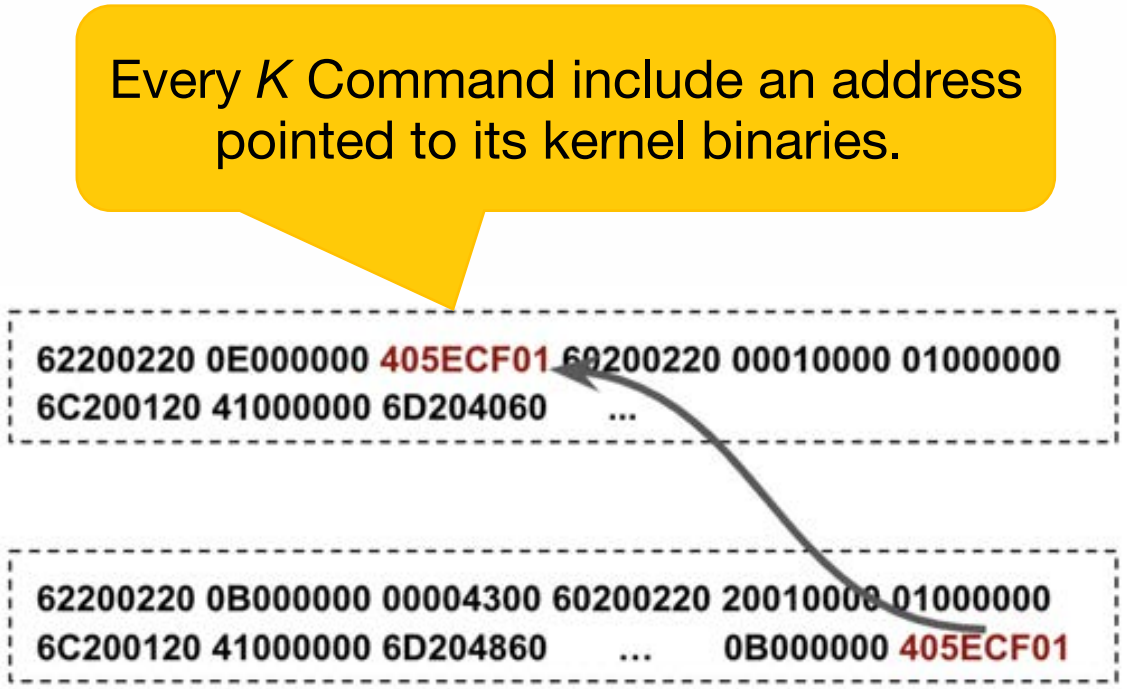
Command Extraction ✓

Attack Overview



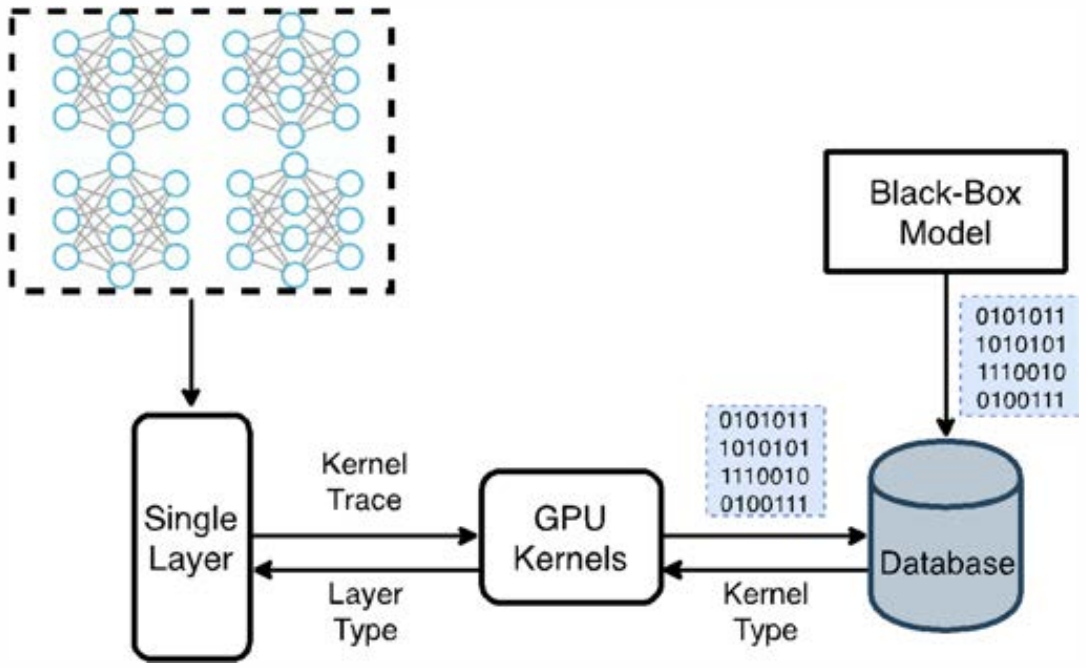
Semantic Reconstruction Challenges

- **Layer type information is lost in PCIe traffic**
- Every layer is completed through several kernels.
- Every Kernel is launched through a ***K* command.**
- Every *K* Command include an address pointed to its kernel binaries.

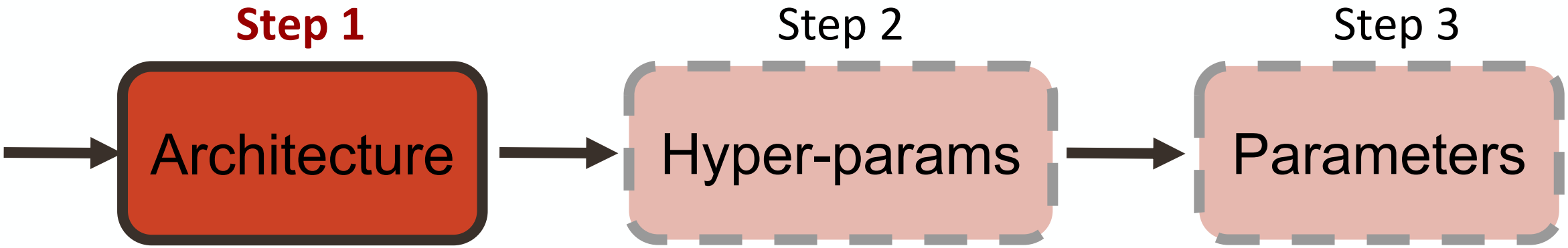


Offline Database Generation

- **Determine kernel types of kernel binaries**
  - ➢Extract corresponding kernel binaries from PCIe traffic.
  - ➢Obtain kernel trace of each layer.
- **Determine kernels of layers**
  - ➢Repeat inference procedure on different models.
  - ➢Obtain kernel trace of each layer.
- **Put <kernel binaries, layer types> pairs into database.**

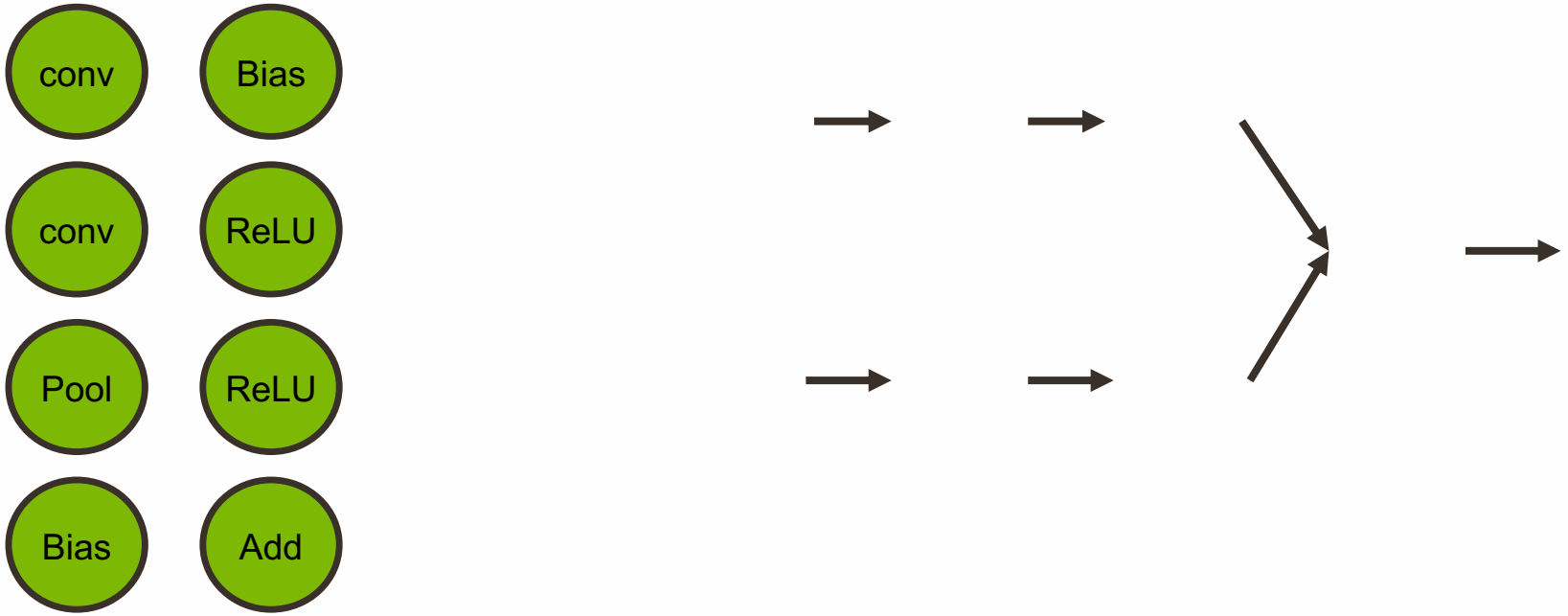


Model Reconstruction



Extract Architecture

- Build Data Flow Graph
  - Treat kernel as vertice
  - Treat Input address and output address as flow-from and flow-to
- Substitute kernel vertice as DNN layers



Model Reconstruction

Extract Hyper-Parameters

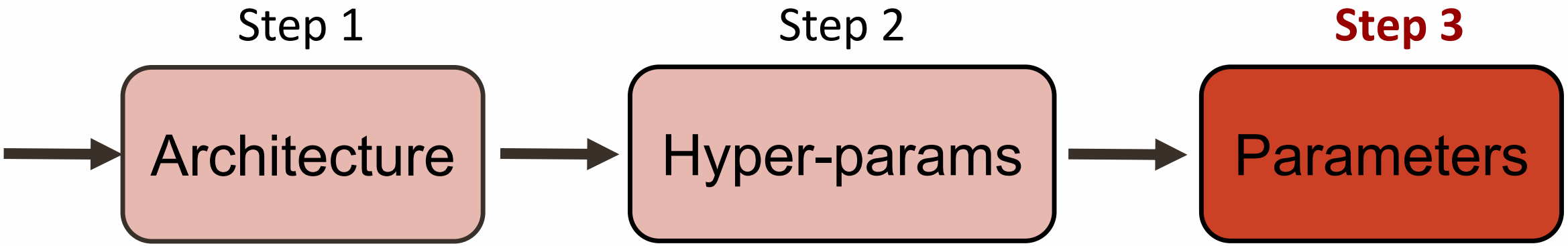
- **Directly obtained from PCIe**
  - ➢ Fetch from command data, use identified offset.
  - ➢ eg. Kernel size, Strides, Filters, etc.

Address	Payloads
00000008:C6ED0800	62200220 0B000000 00512D01 ... 6C200120 41000000 6D202061
00000008:C6ED0880	...XXXXXXXX XXXXXXXX 03000000 03000000 XXXXXXXX...

kernel size

- **Infer**
  - ➢ Check if related kernel exist
  - ➢ eg. Activation function, Use_bias

Model Reconstruction



Extract Parameters

- Parameter includes weights and bias.
- Intuitively, parameters are easier to obtain since all parameters must be transmitted through PCIe.
- However, the data movement commands are not used only for transfer parameters. **How to distinguish weights from all data movement commands?**

Naïve Idea:

Assume all parameters are in range(-1,1), if all data of a command meet this criteria, we can regard this command as a parameter transfer command

Extract Parameters

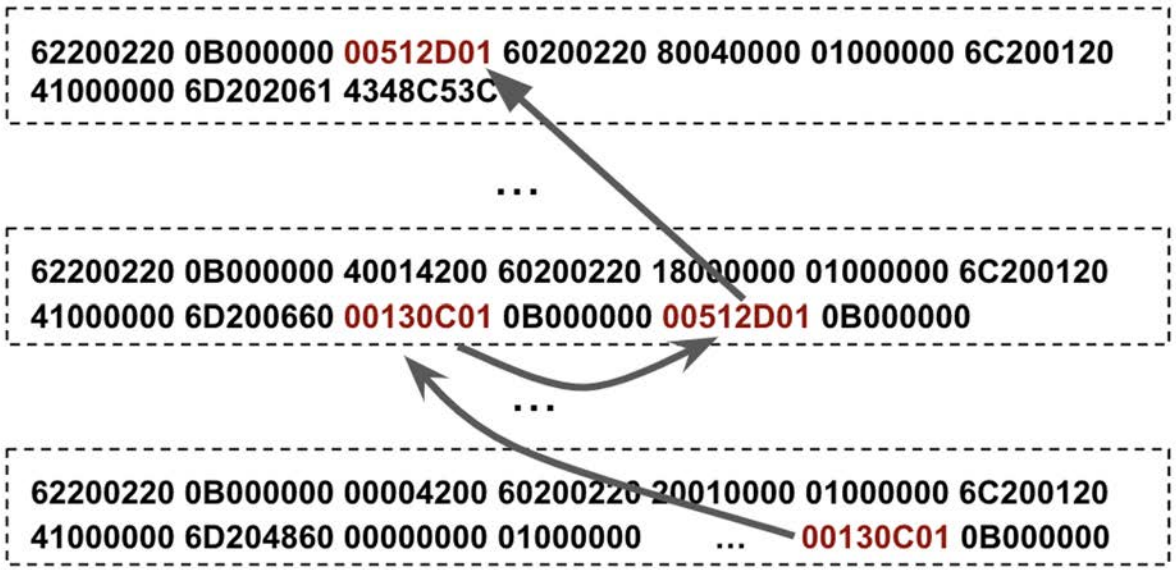
Q: If we can find kernel binaries address as an parameter, why can't we get the weights address as well?

***A*: There is no direct connection between kernel launch command and weights address.**

Extract Parameters

- **Observation:**
  - Weights and bias are not called by kernels directly, they are first called by

    CudaMemcpyDeviceToDevice()
  - This API is also use **kernel launch command** to complete.



Details in our paper https://arxiv.org/abs/2006.12784

Extract Parameters

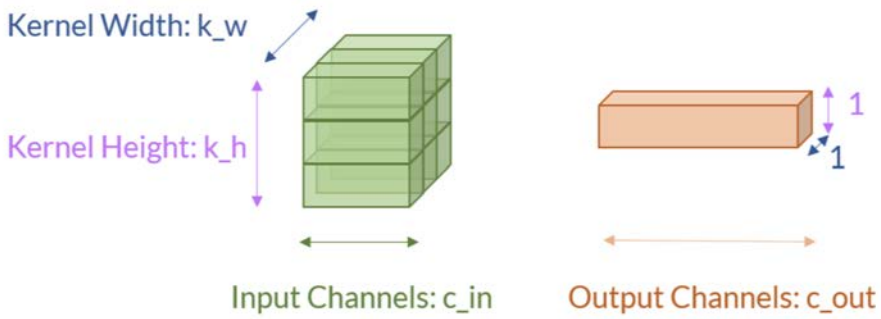
- While extracting command, only check unread packets, after extracted a packet, mark this packet as readed.

- Use the amount of parameters as a junction between weights and layers. The total number of weights of convolution layer could be calculated as:

$$\#weights = k_w * k_h * c_{in} * c_{out} \qquad \#bias = c\_cout$$

For dense layer, it could be calculated as:

$$\#weights = c_{out} * c_{in} \qquad \#bias = c\_cout$$



Model Reconstruction ✓

Multiple Platforms

Different header type

Differences

Command Header

Kernel Binary

Different assembly

Special Cases

Part of binary code is dynamic

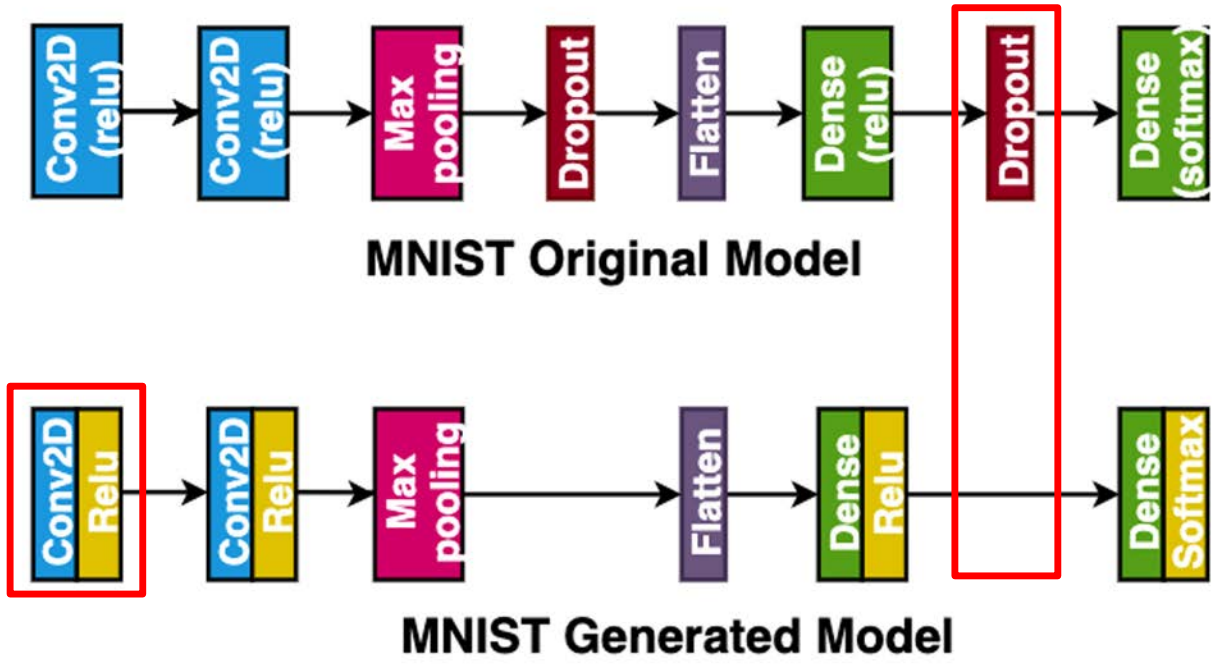
Dynamic Kernel Binary (1080 Only)

Primary Kernel Conflict (2080 Only)

Kernel is not enough

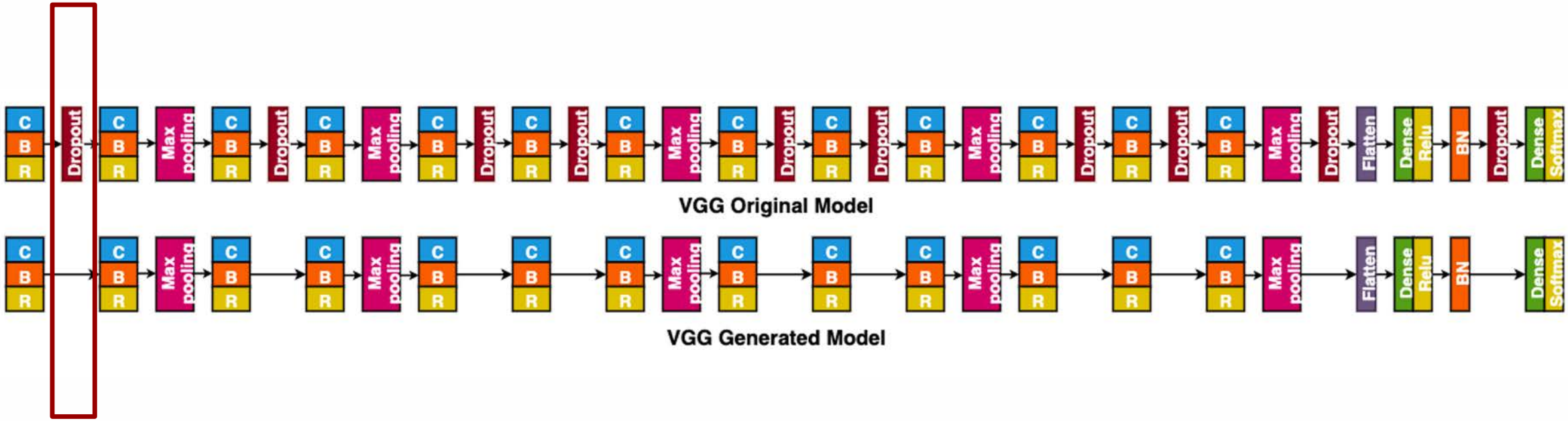
Evaluation

MNIST Architecture Comparison



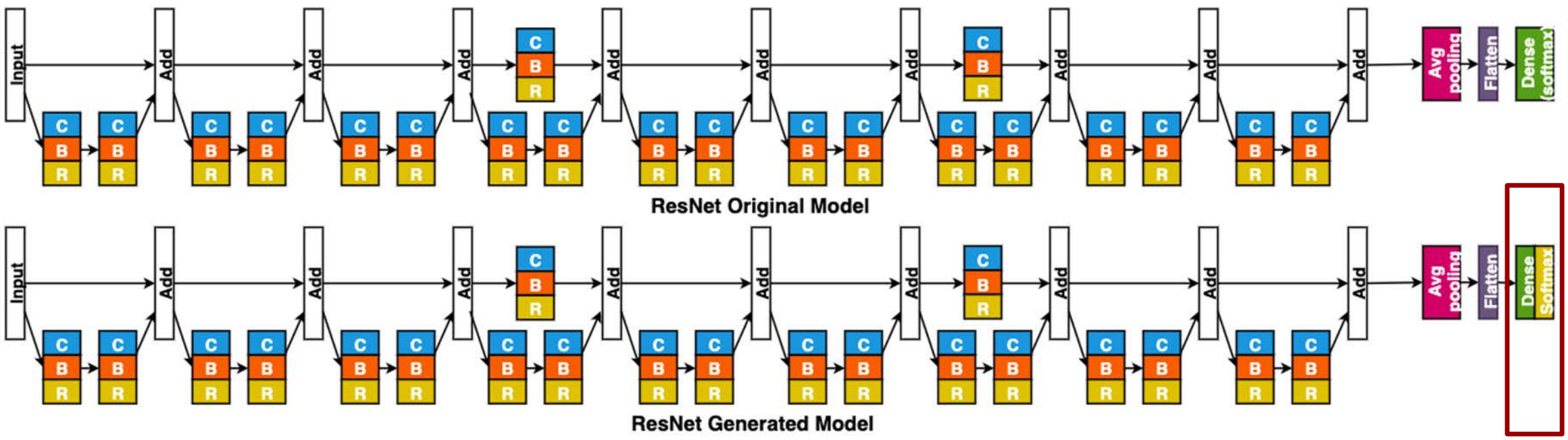
Evaluation

VGG Architecture Comparison



Evaluation

ResNet Architecture Comparison



Evaluation

Victim Models

	MNIST	VGG16	ResNet-20
Number of Layers	8	60	72
Number of Parameters	544,522	15,001,418	274,442
Datasets	mnist	cifar10	cifar10
Input Shape	(28,28,1)	(32,32,3)	(32,32,3)

Evaluation

Identity Evaluation

Metrics	Model	Original	Reconstructed		
N/A	N/A	N/A	GT 730	1080 Ti	2080 Ti
Accuracy	MNIST	98.25		5%	98.25%
	VGG	93.59		9%	93.59%
	ResNet	91.45%	91.45%	91.45%	91.45%
Inference Times(s)	MNIST	2.24	2.39	2.52	2.38
	VGG	65		3	61
	ResNet	20		0	21

Same Accuracy

Same Efficiency

Evaluation

Performance Evaluation

	MNIST			VGG			ResNet		
Platform	GT 730	1080 Ti	2080 Ti	GT 730	1080 Ti	2080 Ti	GT 730	1080 Ti	2080 Ti
# of D Commands	25,680	28,590	24,342	27,287	27,677	24,931	28,433	28,518	25,577
# of K Commands	216	139	181	903	628	793	1011	886	988
# of Completion Packets	1,077,756	2,244,115	2,959,613	4,284,946	2,615,895	3,354,411	975,257	2,052,657	2,717,451
Generation Time (min)	5	8	11	17	11	12	6	9	10

Demo

Countermeasure Discussions

Hardware:

- GPU encryption
- PCIe encryption

Software:

- command/data level obfuscation
- Offload partial task onto CPU
- privacy preserving machine learning

Take Away

The first successful attack could fully steal DNN models.

Support multi-platforms.

Very hard to defense.

All GPUs are not safe under this attack.

References

[1] Hu, Xing, et al. "Neural Network Model Extraction Attacks in Edge Devices by Hearing Architectural Hints." *arXiv preprint arXiv:1903.03916* (2019).

[2] Yan, Mengjia, Christopher Fletcher, and Josep Torrellas. "Cache telepathy: Leveraging shared resource attacks to learn DNN architectures." *arXiv preprint arXiv:1808.04761* (2018).

[3] Hua, Weizhe, Zhiru Zhang, and G. Edward Suh. "Reverse engineering convolutional neural networks through side-channel information leaks." *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. IEEE, 2018.

[4] Yun Xiang and Zhuangzhi Chen and Zuohui Chen and Zebin Fang and Haiyang Hao and Jinyin Chen and Yi Liu and Zhefu Wu and Qi Xuan and Xiaoniu Yang."Open DNN Box by Power Side-Channel Attack." *arXiv preprint arXiv:1907.10406* (2019)

[5] Duddu, Vasisht, et al. "Stealing Neural Networks via Timing Side Channels." *arXiv preprint arXiv:1812.11720* (2018).

[6] Wei, Lingxiao, et al. "I know what you see: Power side-channel attack on convolutional neural network accelerators." *Proceedings of the 34th Annual Computer Security Applications Conference*. ACM, 2018.

[7] Wang, Binghui, and Neil Zhenqiang Gong. "Stealing hyperparameters in machine learning." *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018.

[8] Oh, Seong Joon, Bernt Schiele, and Mario Fritz. "Towards reverse-engineering black-box neural networks." *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*. Springer, Cham, 2019. 121-144.

Hermes Attack: Steal DNN Models In AI Privatization Deployment Scenarios

Yueqiang Cheng*, Yuankun Zhu#, Husheng Zhou$
*Baidu Security, #University of Texas at Dallas, $Vmware