JAXED: Reverse Engineering DNN Architectures Leveraging JIT GEMM Libraries

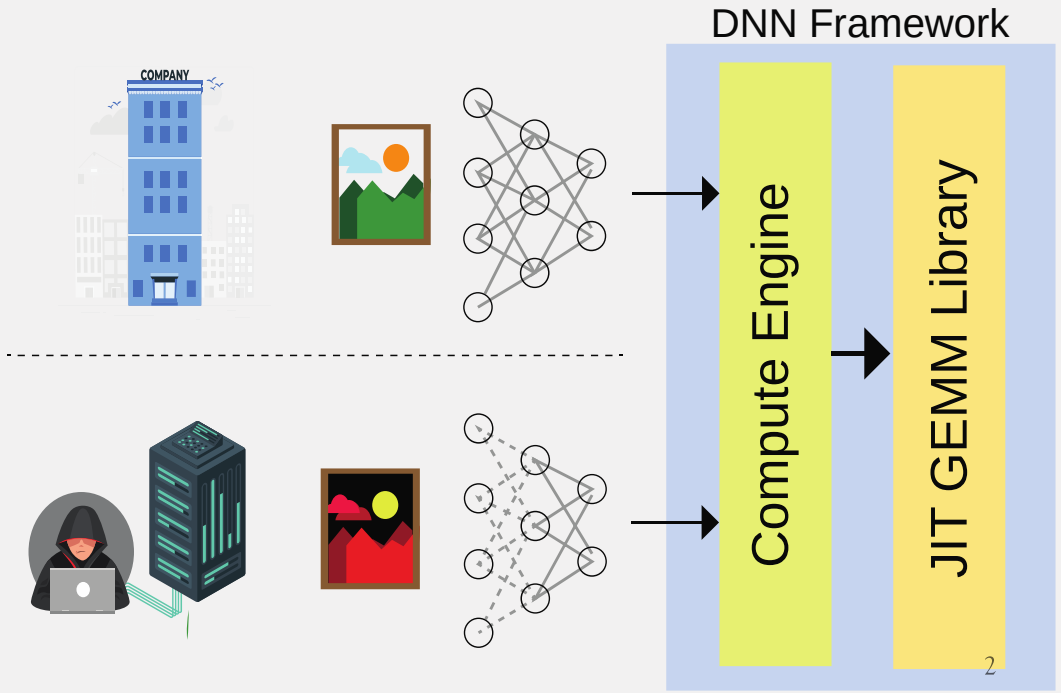
Malith Jayaweera, Kaustubh Shivdikar
Yanzhi Wang, David Kaeli
Northeastern University

Presentation Outline

1. Overview
2. Introduction
3. Threat Model
4. JIT Optimized GEMM Libraries
5. JAXED Attack
6. Results
7. Conclusion

JAXED Overview

- DNN (Deep Neural Network) model hyperparameter extraction attack
- Exploiting a novel side channel exposed through JIT optimized GEMM (General Matrix Multiplication) libraries



Deep Learning

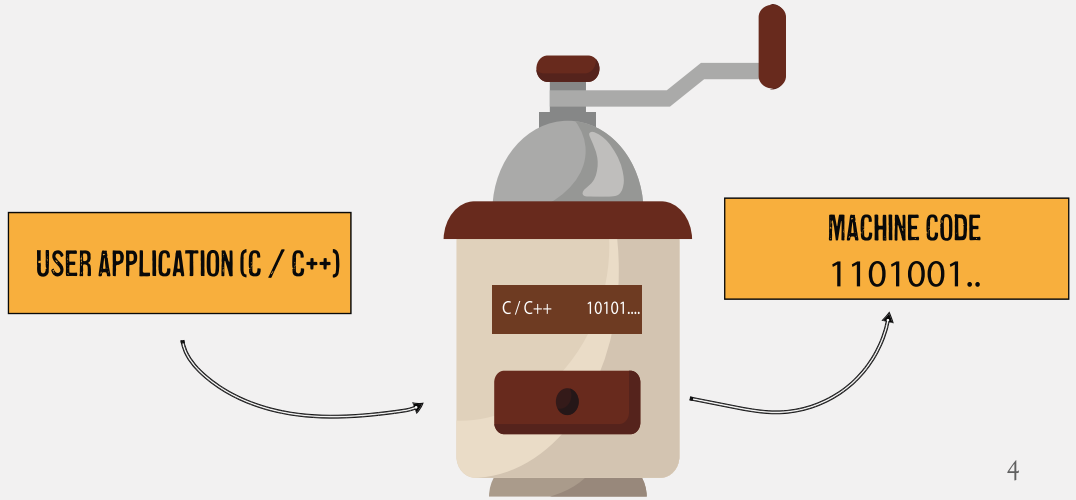
- DNNs have created many business opportunities

- Increasingly, companies treat DNN model hyperparameters as intellectual property

Convolution Operations

- DNNs consist of convolution operators which are computed through GEMM libraries

- Recently there has been a trend to incorporate JIT optimized GEMM libraries to ML frameworks.



PyTorch

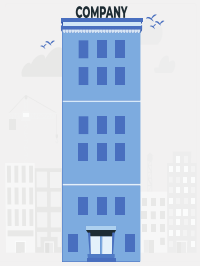
JIT Optimization Exposes a Side Channel!

- Generates code at runtime to exploit aggressive optimizations

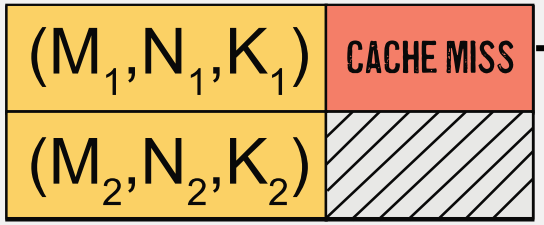
- A code cache is maintained to amortize the cost of code generation over many execution iterations

JIT Optimization Exposes a Side Channel!

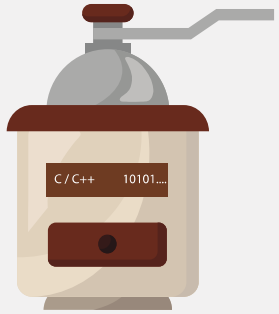
①



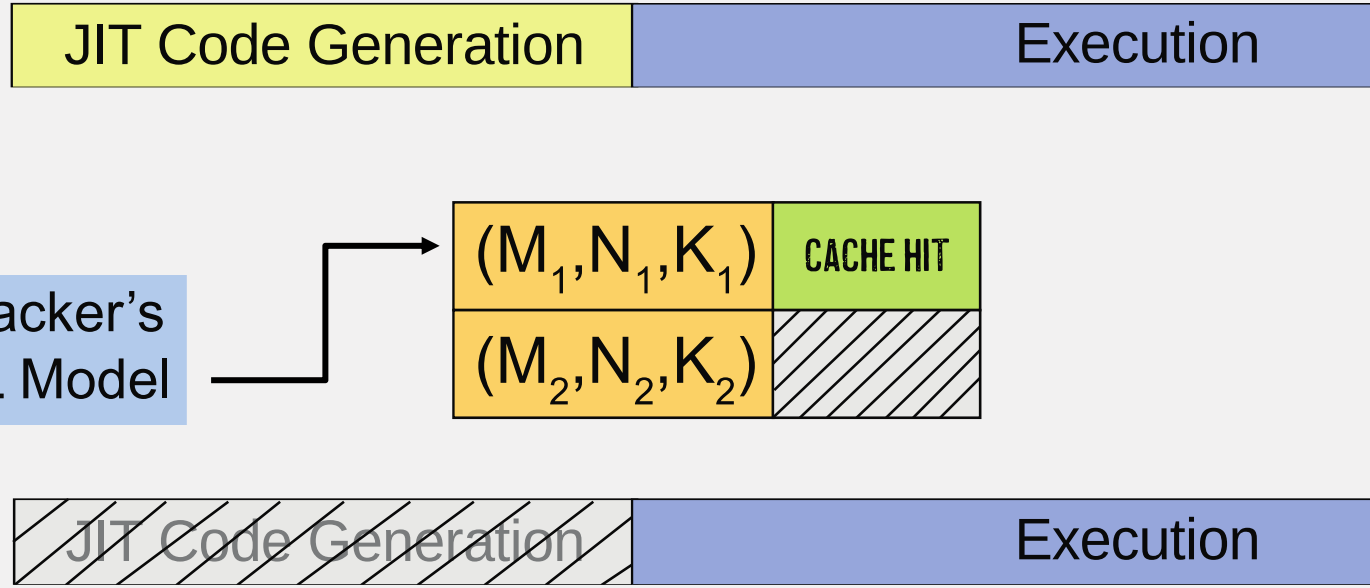
Victim's ML Model



triggers



JIT COMPILATION



②

Attacker's ML Model

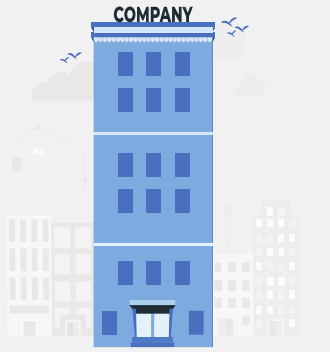
Attack Scenario

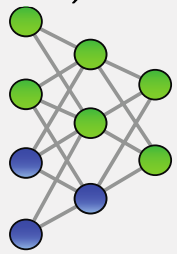
- Read only access to a web API allows the attacker to discover the availability of a side channel

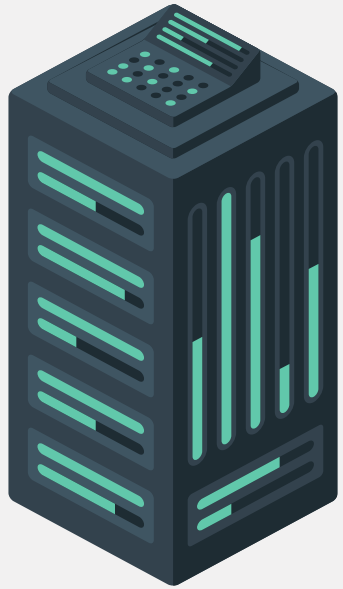
- Read and write access to a web API allows the attacker to perform an exploratory attack to extract model hyperparameters

Attack Scenario (READ only API access)

Attack Scenario (READ + WRITE API access)





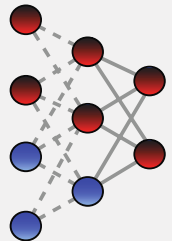


{model: “user_dnn”, data: img1.png}

① {model: “att_dnn_0”, data: “h4x0r.png”}

② {action: “modify_dnn”, param: “conv2d”}

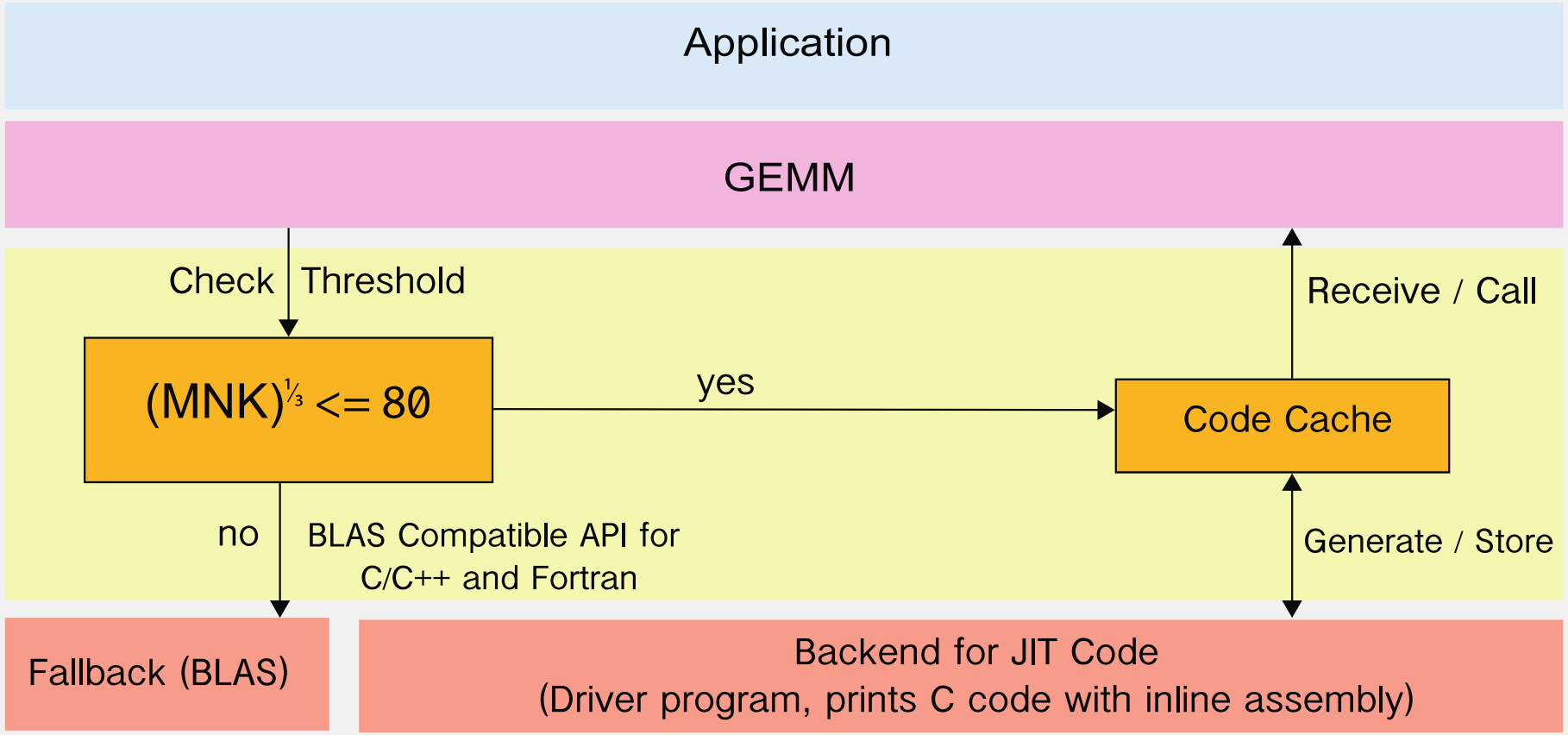
③ {model: “att_dnn_1”, data: “h4x0r.png”}



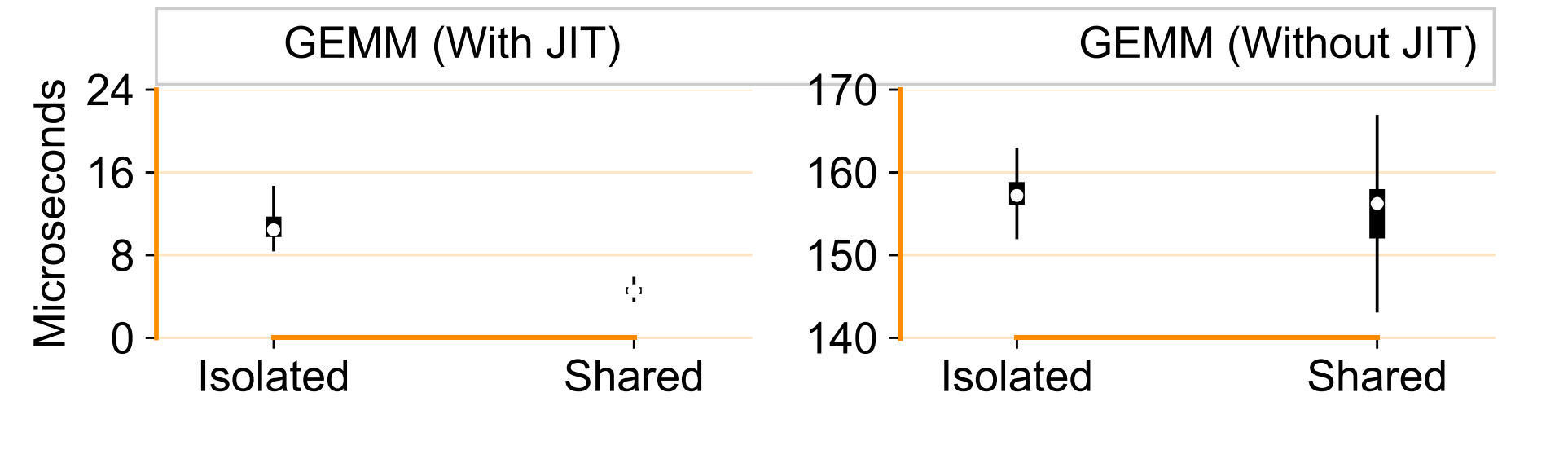
Attack Model

- **Black-box Access** - The victim's DNN model hyperparameters and weights are not visible to the adversary
- **Shared Platform** – ML framework (and accompanying GEMM library) is shared between the victim's DNN model
- **Timing Information** - The attacker should be able to gain fine-grained timing details for her own DNN model

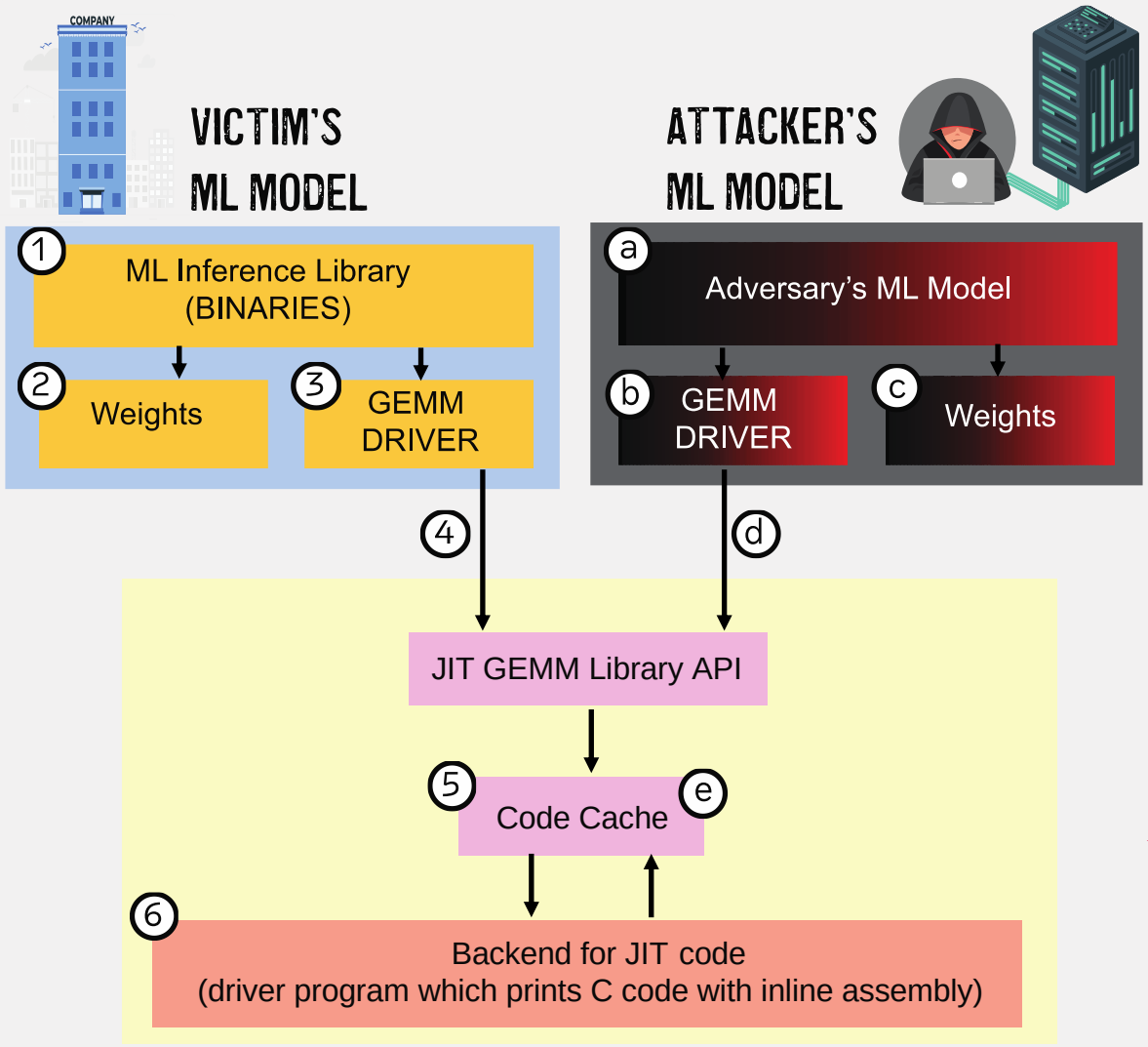
Example: LIBXSMM



Timing Difference



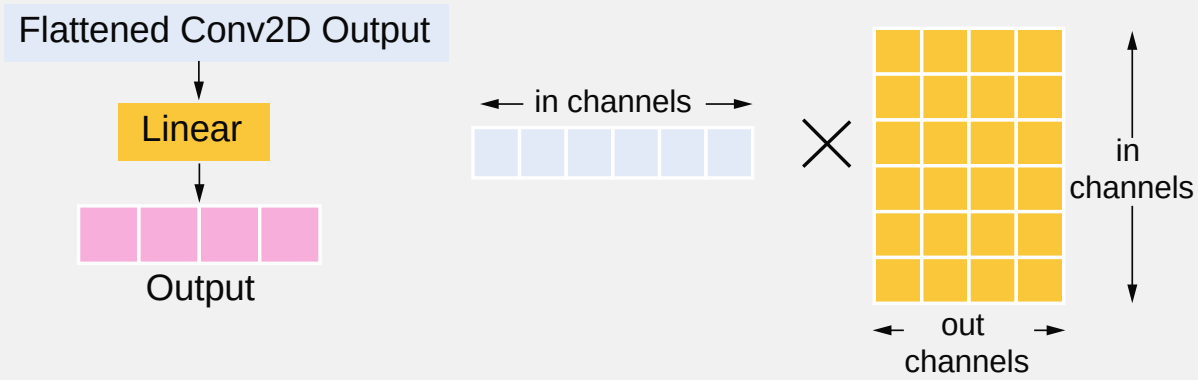
JAXED Attack



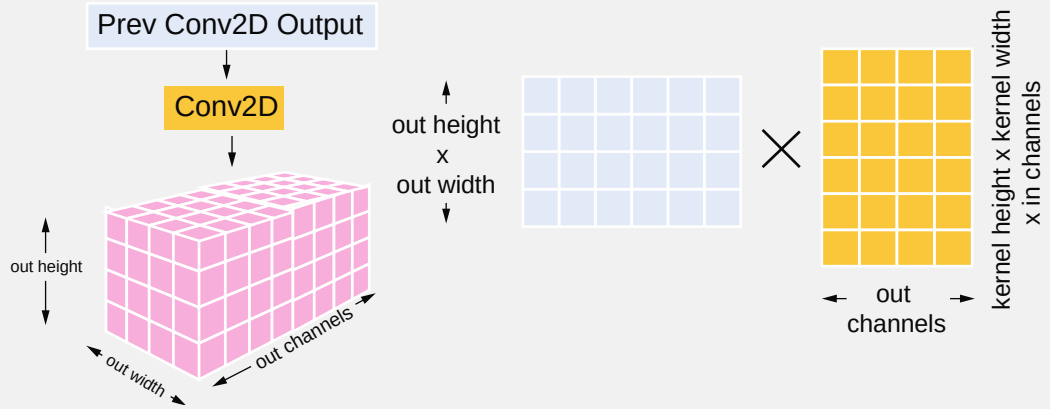
Does not trigger JIT!

Inference through Matrix Multiplication

- Fully Connected Layer

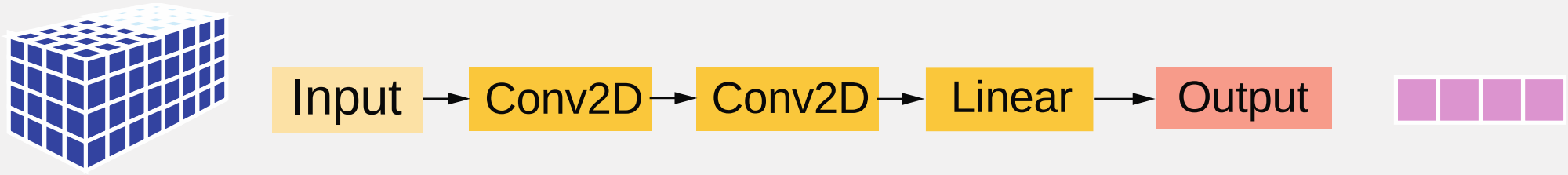


- Convolution Layer

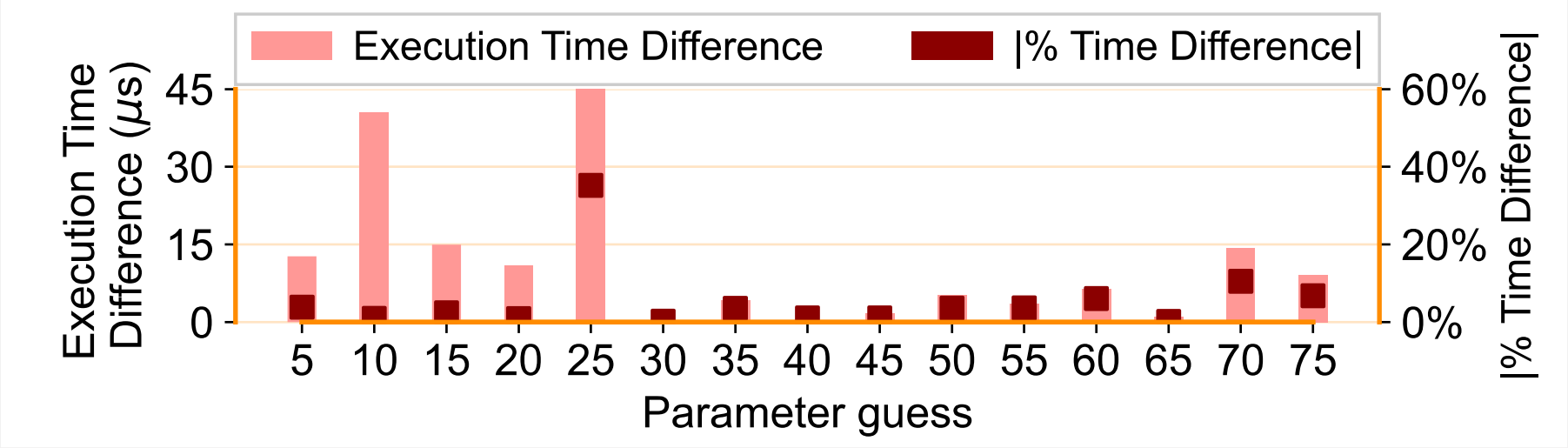


Last Layer is Often a Fully Connected Layer!

- Adversary can easily start from the last layer and backtrack

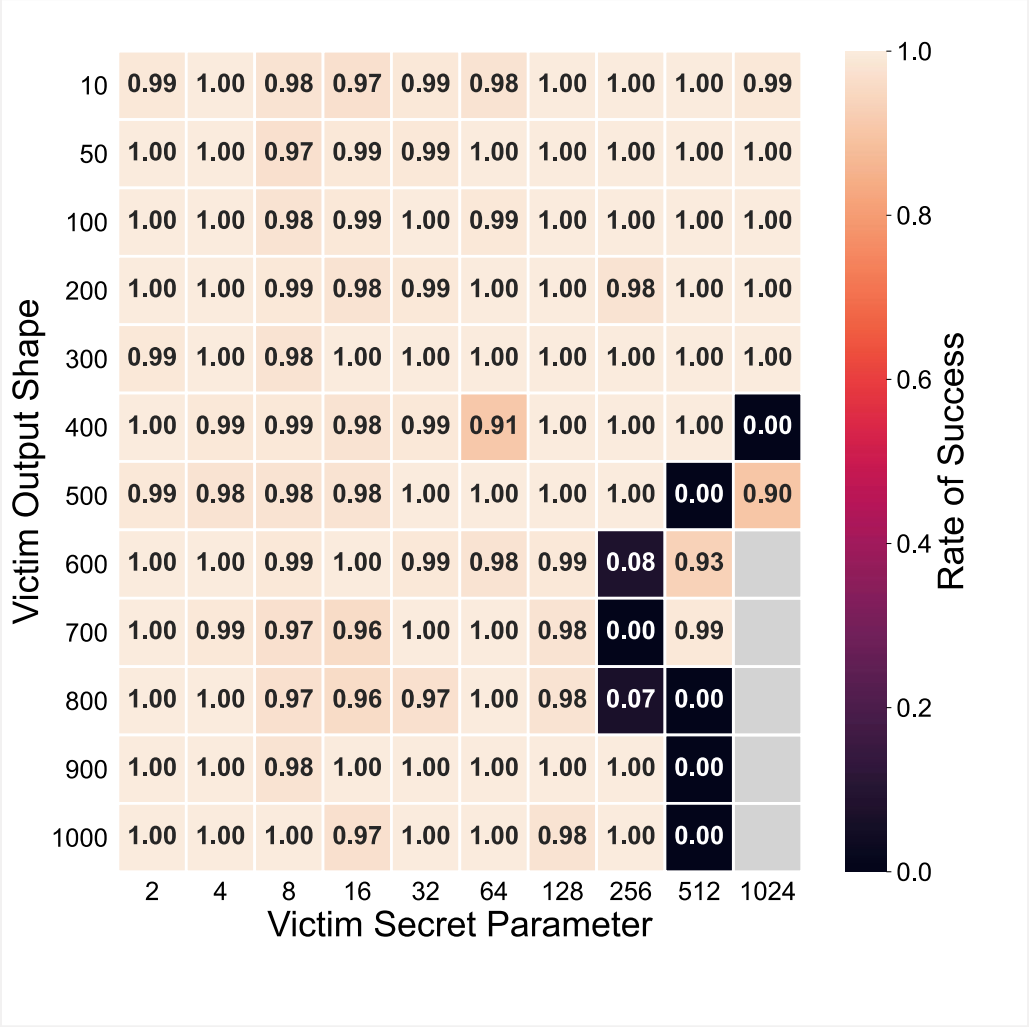


Attacking Last FC Layer

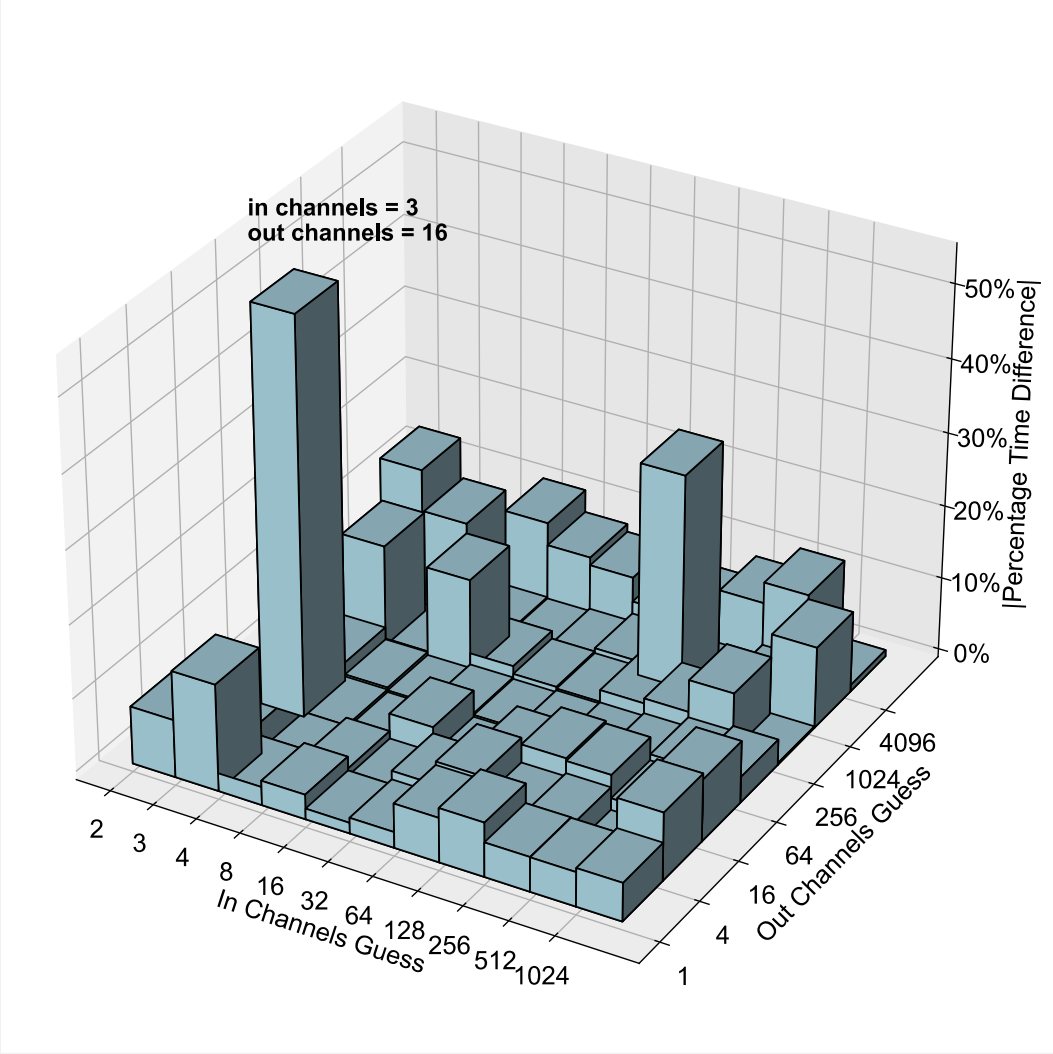


Time difference is largest when the victim's secret parameter (25) is the same as the adversary

Varying Victim's Secret Parameter



Attacking a Conv2d Layer



DNN Attacks

TABLE II: Last Fully Connected Layer Attack Results.

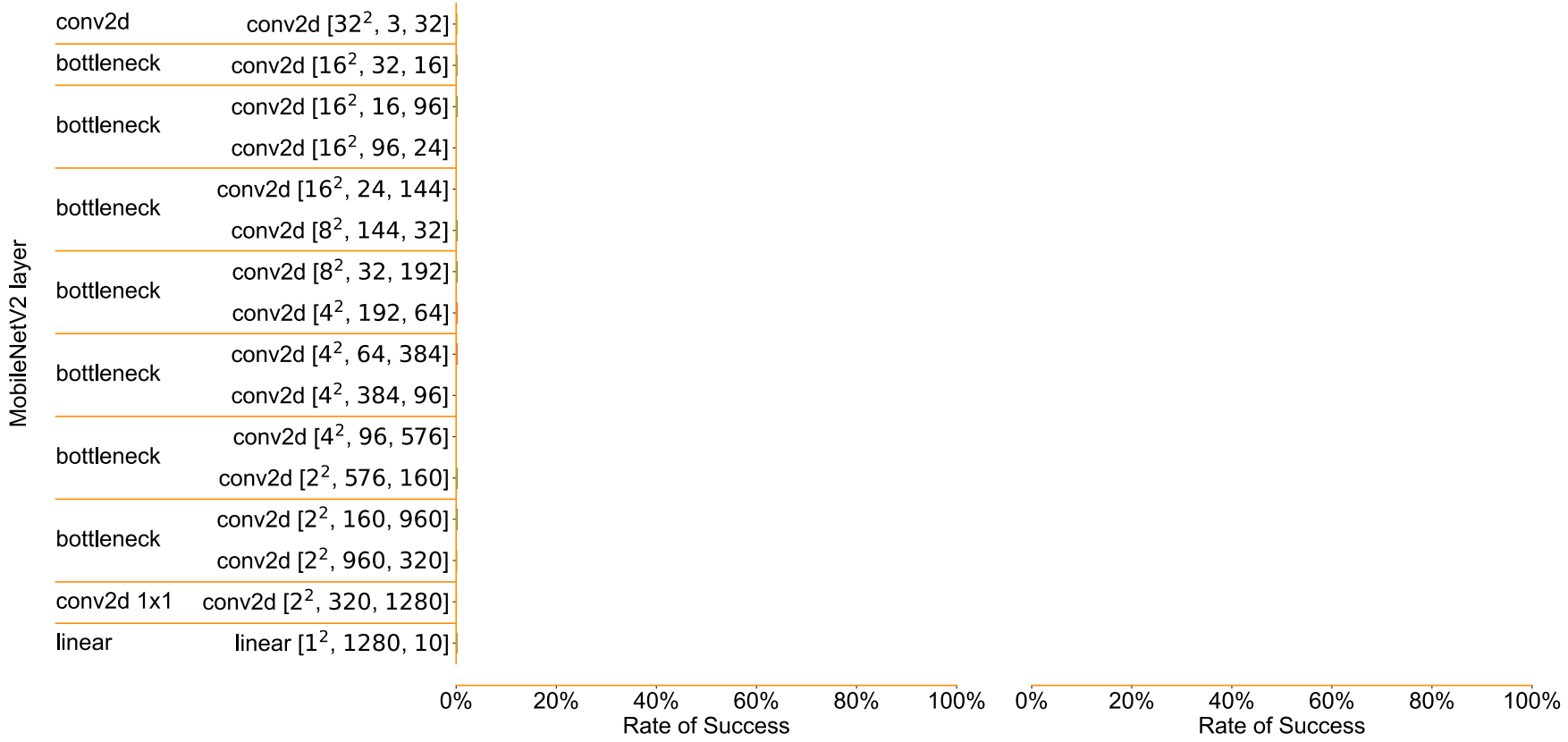
DNN	Input Channels	Output Channels	Success rate
ResNet-18	512	10	99%
ResNet-34	512	10	99%
ResNet-50	2048	10	99%
VGG16	4096	10	99%
VGG19	4096	10	100%
MobileNetV2	1280	10	99%
DenseNet121	1024	10	100%
DenseNet169	1664	10	100%
Inception v3	2048	10	100%
GoogLeNet	1024	10	99%

End-to-end DNN Attack with MobileNetV2

TABLE III: MobileNetV2 Architecture. Each line describes a set of 1 or more identical layers, repeated n times. All layers in the same set have the same number of output channels - c. The first layer of each set has a stride s and all others use stride 1. Expansion factor t is used for spatial convolution.

Input	Operator	t	c	n	s
$32^2 \times 3$	conv2d	-	32	1	2
$16^2 \times 32$	bottleneck	1	16	1	1
$16^2 \times 16$	bottleneck	6	24	2	1
$16^2 \times 24$	bottleneck	6	32	3	2
$8^2 \times 32$	bottleneck	6	64	4	2
$4^2 \times 64$	bottleneck	6	96	3	1
$4^2 \times 96$	bottleneck	6	160	3	2
$2^2 \times 160$	bottleneck	6	320	1	1
$2^2 \times 320$	conv2d 1×1	-	1280	1	1
-	dropout 0.2	-	-	-	-
$1 \times 1 \times 1280$	linear	-	10	-	-

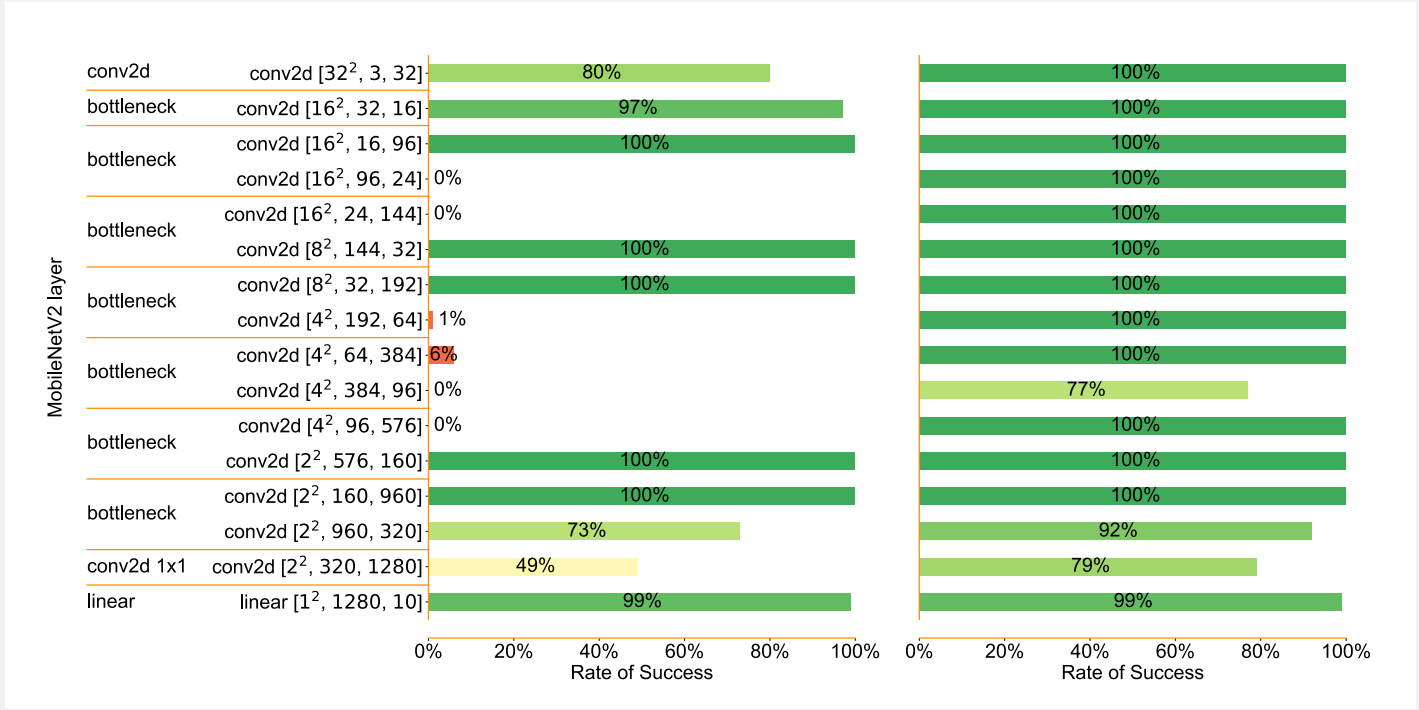
MobileNetV2



The Top-1 success rate (left) and the Top-3 success rate (right).

MobileNetV2 (Detailed Results)

Layer	c	f	R0	R1	R2
conv2d [32^2, 3, 32]	**3**	**32**	80%	20%	0%
	320	512	20%	72%	5%
	576	128	0%	6%	60%
conv2d [16^2, 96, 24]	16	96	100%	0%	0%
	144	24	0%	68%	32%
	96	**24**	0%	32%	68%
conv2d [16^2, 24, 144]	32	16	100%	0%	0%
	24	**144**	0%	100%	0%
	512	576	0%	0%	59%
conv2d [4^2, 192, 64]	32	16	99%	0%	0%
	192	**64**	1%	99%	0%
	576	1,024	0%	1%	97%
conv2d [4^2, 64, 384]	16	96	94%	6%	0%
	64	**384**	6%	94%	0%
	144	24	0%	0%	92%
conv2d [4^2, 384, 96]	16	96	100%	0%	0%
	96	24	0%	86%	14%
	384	**96**	0%	14%	63%
conv2d [4^2, 96, 576]	32	16	100%	0%	0%
	96	**576**	0%	87%	13%
	24	144	0%	13%	87%
conv2d [2^2, 960, 320]	**960**	**320**	73%	13%	6%
	24	1,280	3%	13%	11%
	32	1,280	0%	1%	1%
conv2d [2^2, 320, 1280]	**320**	**1280**	49%	21%	9%
	4	1,280	7%	3%	4%
	3	1,280	1%	0%	4%



Preventive Measures

- User Awareness
- Explicit Code Cache Flushing
- Modifying Library Design

Conclusion

- Novel timing attack on JIT-optimized GEMM libraries, successfully extracting model hyperparameters.

- Our work will educate both library developers and model users and motivate new security research in JIT-optimized GEMM libraries