Deep Learning for Adaptive Cyber Defense: Opportunities and Challenges

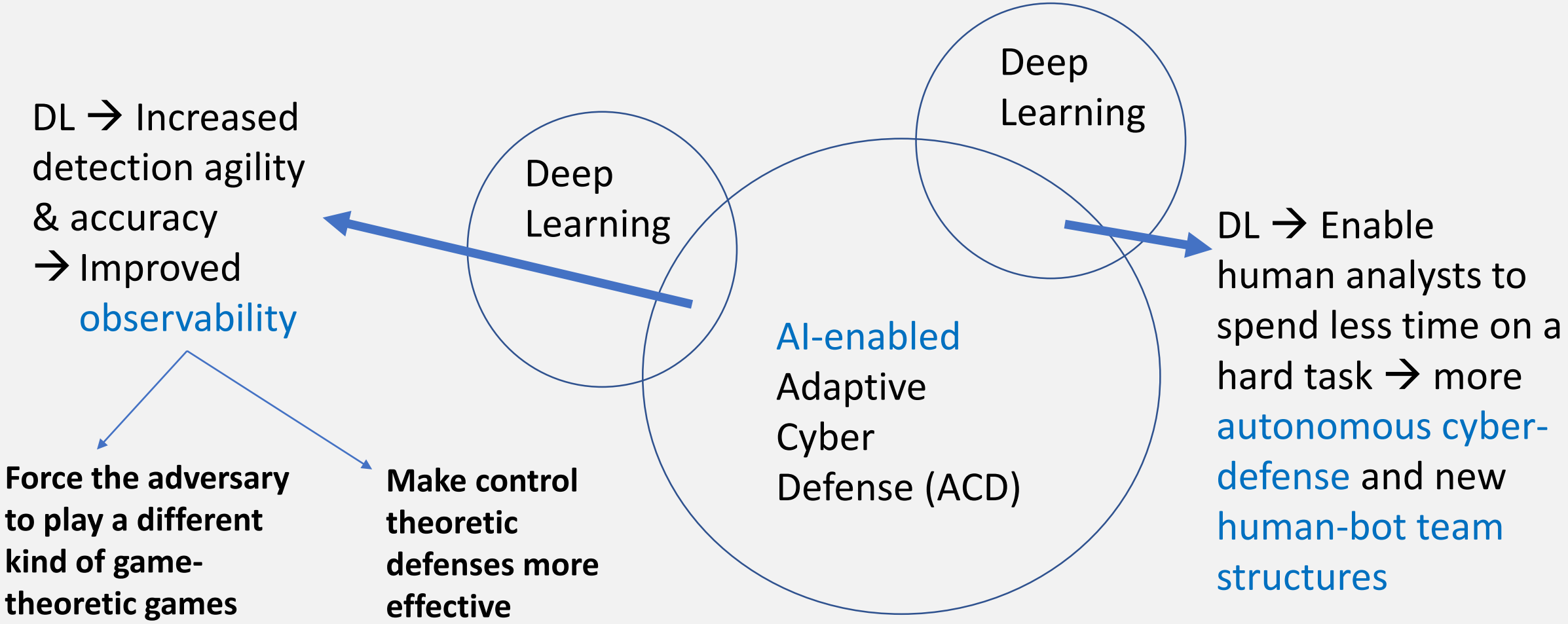
Peng Liu

Cyber Security Lab

Pennsylvania State University

pliu@ist.psu.edu

Two opportunities provided by Deep Learning in Adaptive Cyber Defense

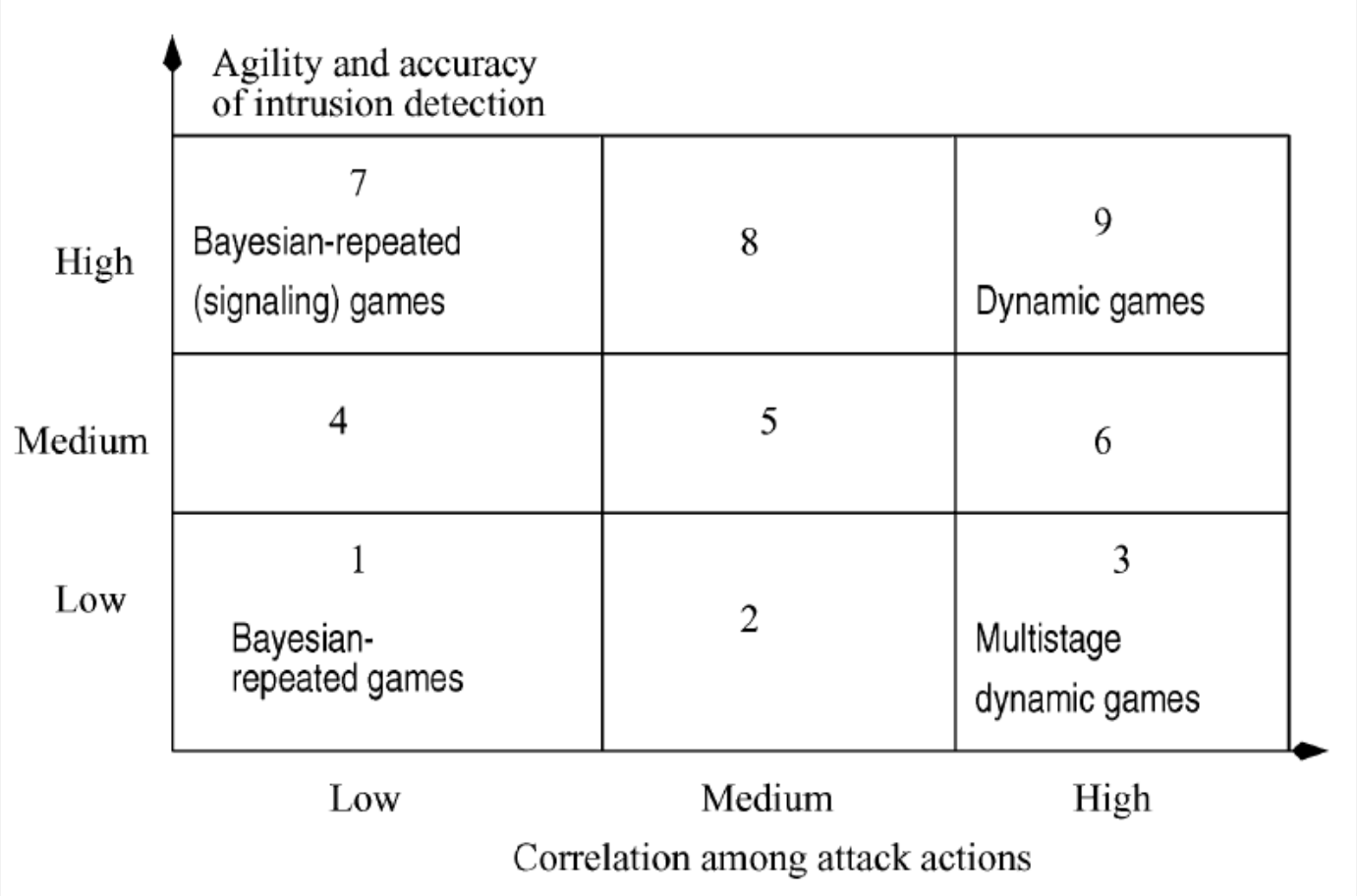


Overview

- Opportunity A: Deep Learning (DL) improves observability in ACD
- Opportunity B: Deep learning enables human analysts to spend less time on a task

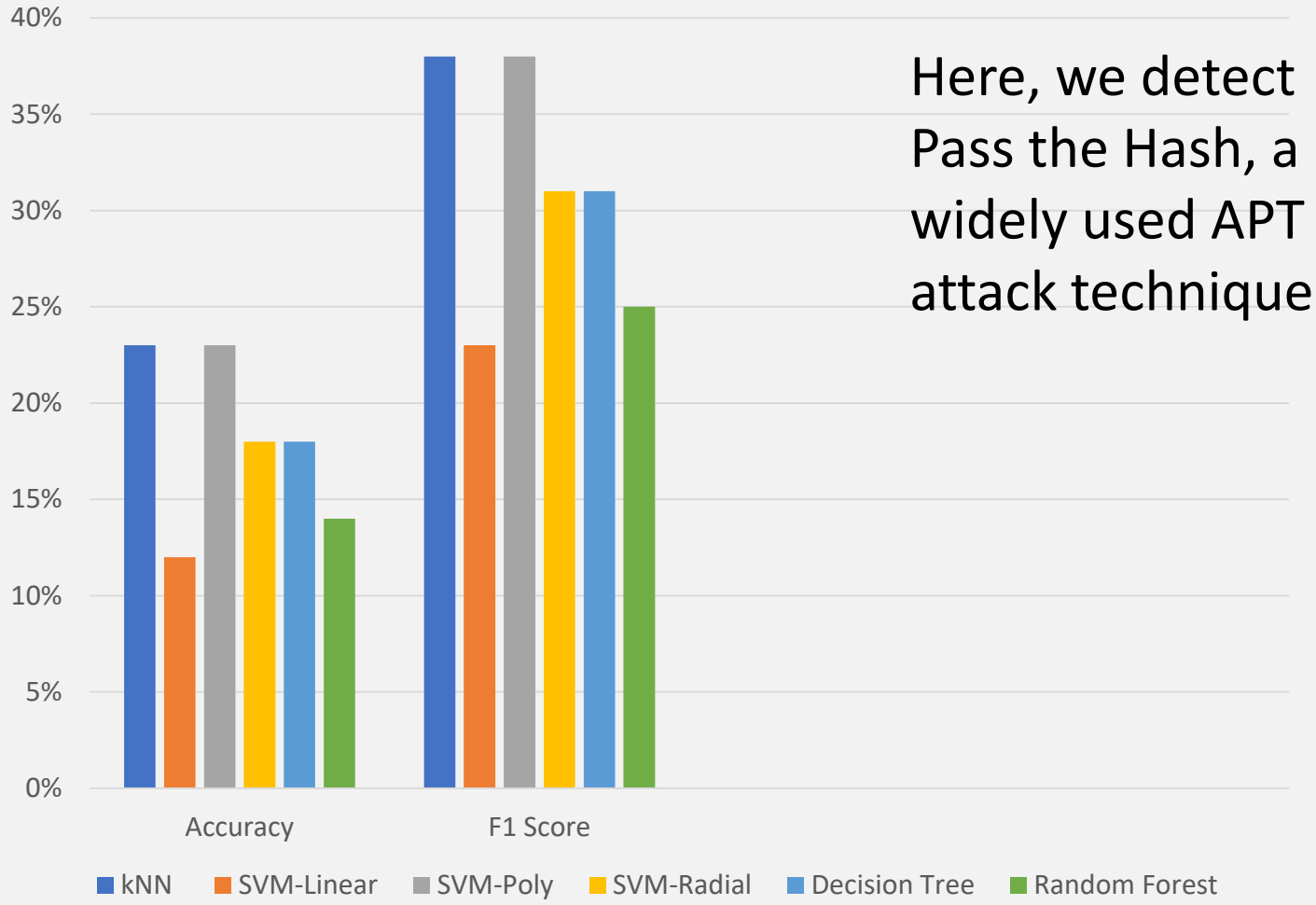
Impact of detection agility and accuracy on game-theoretic defenses

In the context of defending multi-stage attacks in enterprise networks…



P. Liu, et al., "Incentive-Based Modeling and Inference of Attacker Intent, Objectives, and Strategies," ACM TOPS, 2005

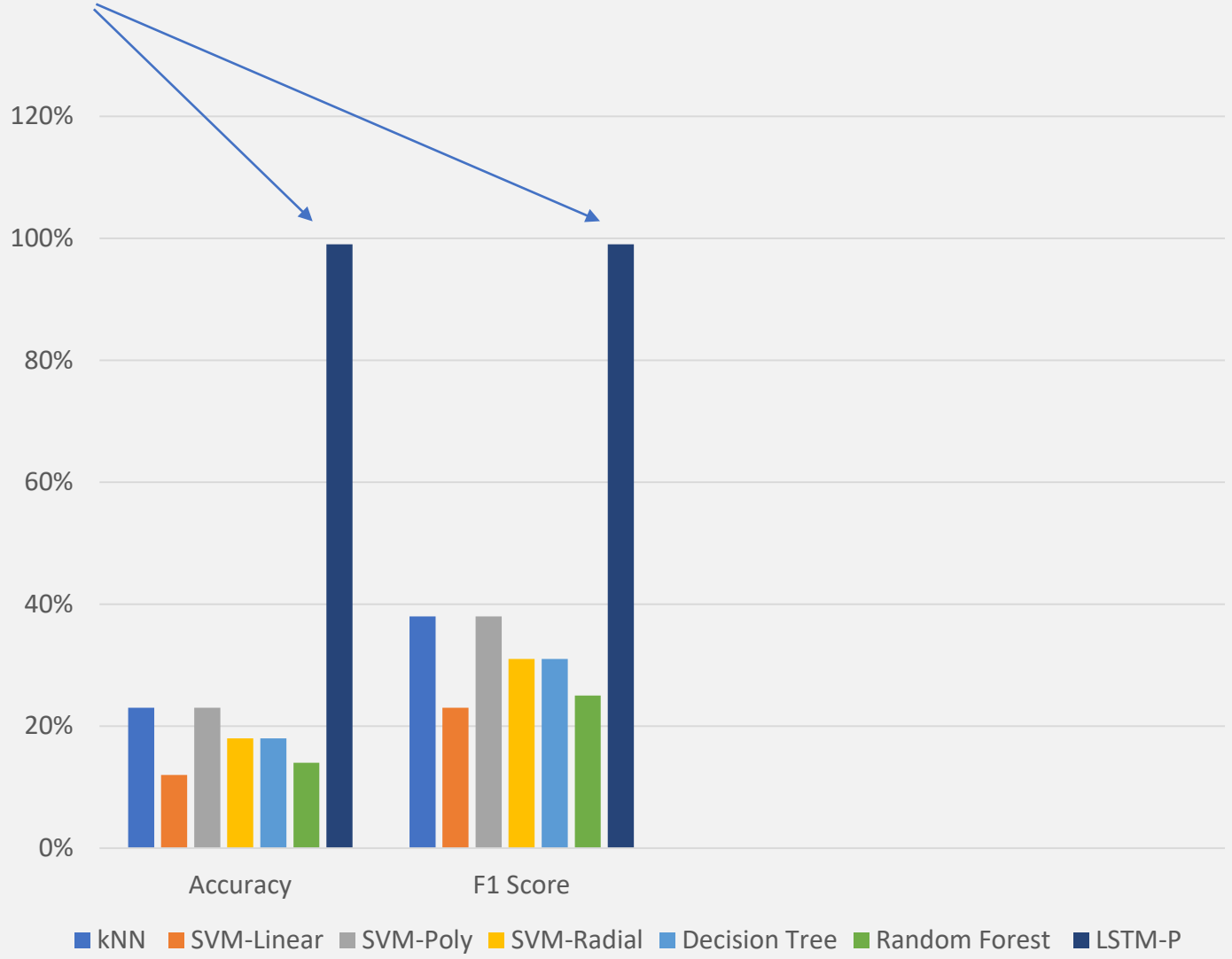
Traditional Machine Learning is not very effective in detecting attacks



Here, we detect Pass the Hash, a widely used APT attack technique

- **Heavily dependent on feature engineering**
- **Requires a lot domain knowledge (e.g., the relation of the features to the attacks of interest)**

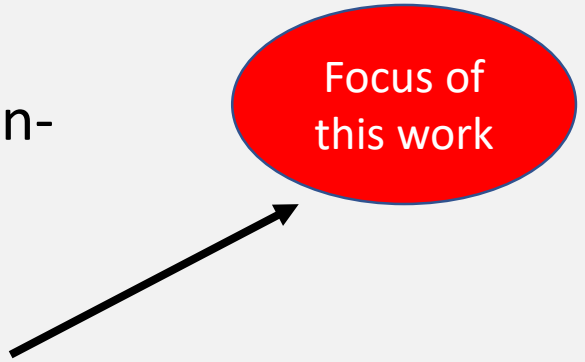
DL makes a difference



- One major advantage of DL is that it makes learning algorithms less dependent on feature engineering
- Another major advantage of DL is that it could achieve high classification accuracy with minimum domain knowledge

Importance of logic-flaw-exploiting network attacks

- APT attacks leverage 5 main categories of **R2L** (remote to local) techniques
  - Spear Phishing
  - Cross-site scripting and Drive-by download
  - Memory corruption attacks (e.g., buffer overflow, return-to-Libc, ROP)
  - Backdoor (SolarWinds)
  - **Logic-flaw-exploiting (LFE) network attacks (e.g., PtH)**



However, it is very challenging to detect LFE network attacks!

Traditional network attack detection techniques

- Signature-based
- Rule-based

} Need constant updates & May not always be available

- Anomaly detection-based → Tend to raise false alarms

Public network attack data sets

KDD99, NSL-KDD, UNSW-NB15, CICIDS2017, CICIDS-2018, etc.

They are not really suitable for applying DL in detecting LFE network attacks!

- Do not include LFE attacks
- Unbalanced
- Lack of variations
- Coarse labeling

A new opportunity:

Deep learning could be applied to achieve **accurate**, **signature-free**, and **low-false-alarm-rate** detection!

Two major challenges:

--- data sets

--- neural network architecture search space is large

New method (JCS 2021)

- We propose an end-to-end approach to detect the LFE network attacks
  - The end-to-end approach means it starts with acquiring data and ends with detecting attacks using the trained neural networks.
- We address two challenges:
  - Data generation: **penetration testing + protocol fuzzing** → a new approach
  - Neural network training: identify fields of interest and do model selection

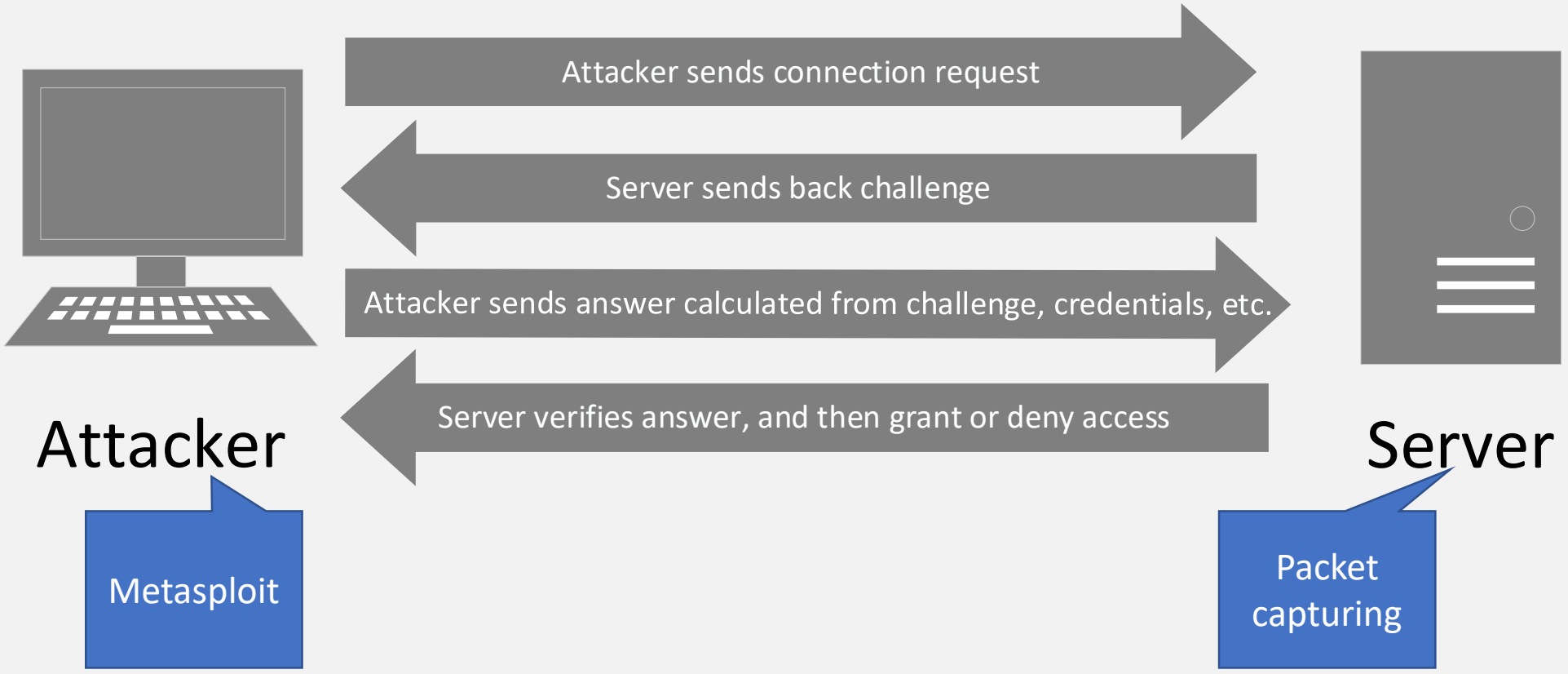
New approach for data generation

Our approach combines penetration testing and protocol fuzzing

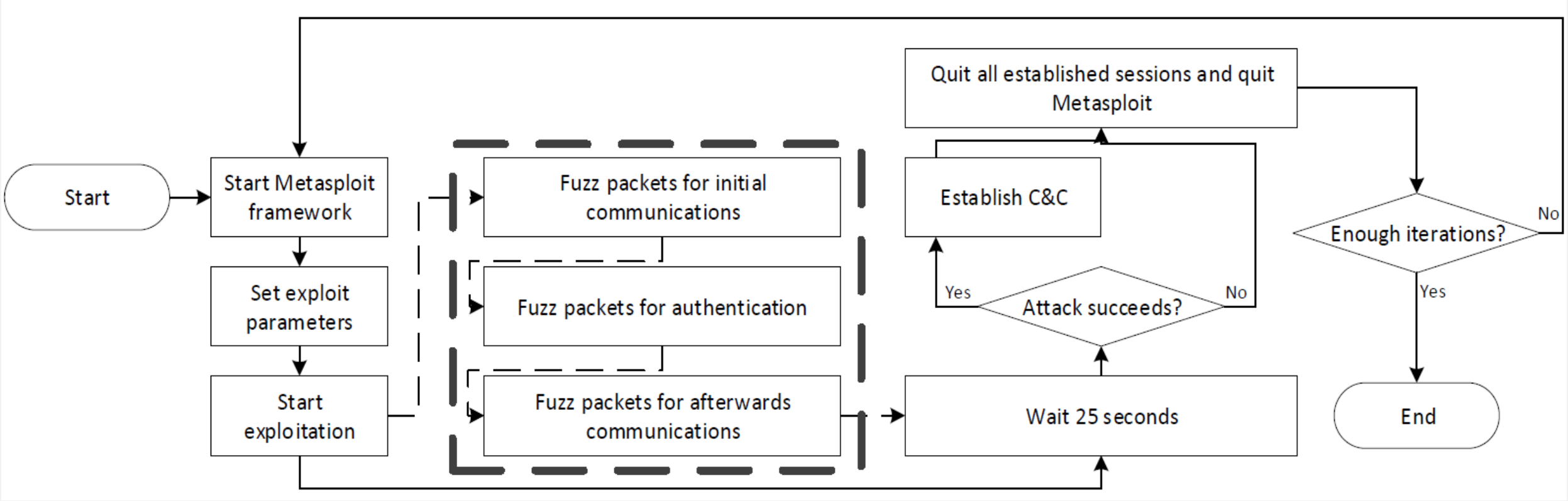
- Fuzzing inside Metasploit → change the contents of network packets → the values of some fields in the packets
- Find out which fields are able to be fuzzed
- ARP poisoning, DNS cache poisoning, PtH.

```
Result: BList, which stores fields to fuzz
input AList of all available fields;
initialize an empty BList to store fields to fuzz;
foreach field in AList do
    fuzz field;
    fuzz all fields in BList;
    launch the attack for hundreds of times;
    count successful attacks and calculate success rate;
    if attack success rate is over 50% then
        add field to BList;
    end
end
```

PtH introduction



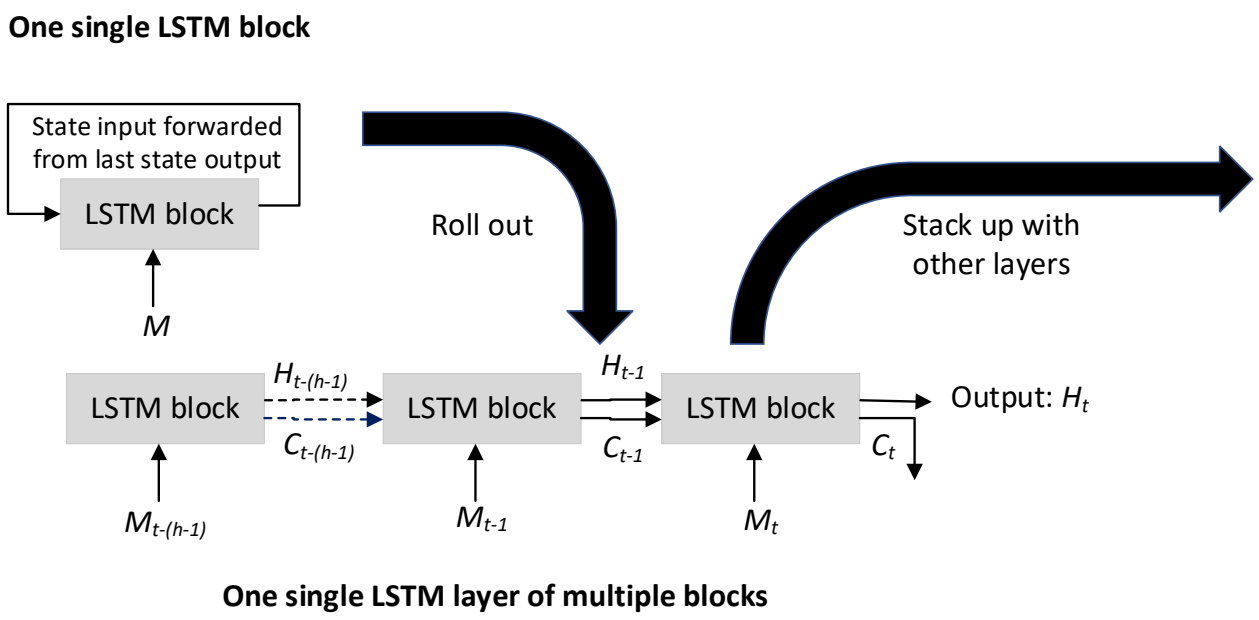
PtH data generation

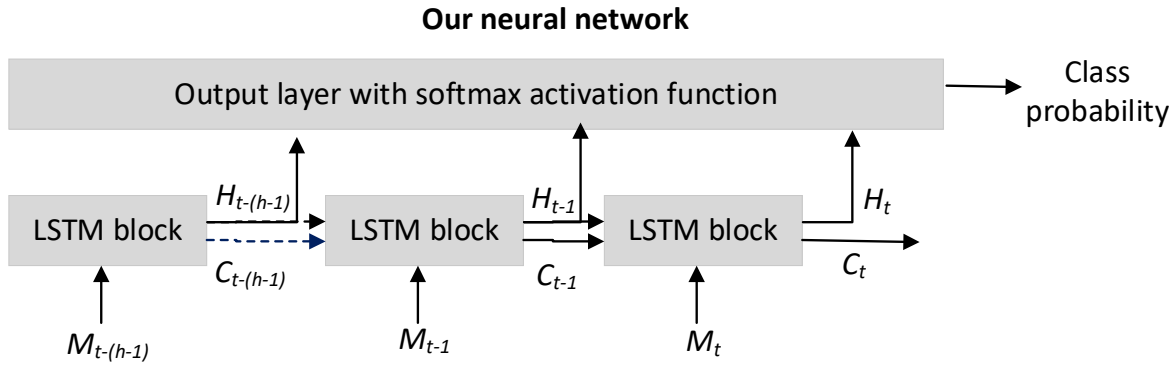


PtH detections – key insights

1. Each data sample should be **a sequence of packets**, rather than an individual packet.
2. Differences between benign and malicious exist in the **field values** of the SMB/SMB2 layer.

LSTM model for PtH detection





M stands for input; H stands for each LSTM block's output; and C stands for each LSTM block's state output. Subscripts stand for the time points, in which h stands for the window size.

Model selection

		Model classification	
		Positive (malicious)	Negative (benign)
Ground truth	Positive (malicious)	True positive (TP)	False negative (FN)
	Negative (benign)	False positive (FP)	True negative (TN)

$$Acc = \frac{TP + TN}{TP + FN + FP + FN}$$

$$F1 = \frac{TP}{TP + 0.5 * (FP + FN)}$$

$$DR = \frac{TP}{TP + FN}$$

$$FPR = \frac{FP}{FP + TN}$$

Metrics:

- Accuracy (Acc), F1 score (F1), detection rate (DR), and false positive rate (FPR)
- Best-performing: $P = \frac{Acc+F1}{2}$
- Best-detecting: $D = \frac{DR+1-FPR}{2}$ or $D = DR$ (if there is no negative data sample and FPR cannot be calculated)

Data set description

- Fuzzing data set: **80% as the training set, and 20% as the test set**.
  - **4-fold cross-validation** to avoid over-fitting.
- Unfuzzed data set (**real attack set**) → verify whether trained detection models can detect real attacks.

Attacks	Set	Size	Benign to malicious ratio
PtH* (best-performing)	Training	3932	1.364:1
	Test	983	1.329:1
	Real attack	214	0:1
PtH* (best-detecting)	Training	2556	0.974:1
	Test	640	0.839:1
	Real attack	192	0:1

PtH: Best-performing models

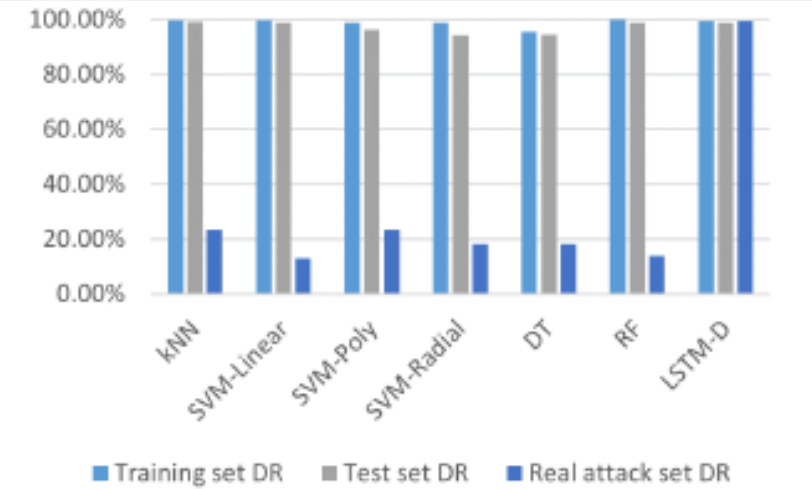
DL or ML	Model type	Training set		Test set		Real attack set	
		Acc	F1	Acc	F1	Acc	F1
DL	LSTM-P	98.45%	0.9865	98.07%	0.9831	98.96%	0.9948
ML	kNN	96.77%	0.9682	96.53%	0.9658	23.44%	0.3797
	SVM-Linear	96.89%	0.9694	96.72%	0.9674	13.02%	0.2304
	SVM-Poly	97.75%	0.9779	94.69%	0.9479	23.44%	0.3797
	SVM-Radial	98.07%	0.9810	93.72%	0.9378	18.23%	0.3084
	DT	94.70%	0.9467	95.44%	0.9533	18.23%	0.3084
	RF	100.00%	1.0000	97.99%	0.9798	14.06%	0.2466

DL models are substantially better than traditional ML models, especially on real attack set.

PtH: Best-detecting models

- LSTM model achieves highest DR.
- LSTM model achieves comparatively low FPR

DL models are better than ML models, especially on real attack set.



(b) PtH DR.

(e) PtH FPR.

Deep Learning for ACD: Challenges

- (C1) Imbalanced data
- (C2) Deep neural networks (DNN) are hard to explain
- (C3) Generalization ability cannot be taken for granted: people are **concerned** with using different neural networks to detect different network attacks
- (C4) DNN models have an extended attack surface: poisoning attack, adversary examples, etc.

We proposed “dual-threat adversary attack” in one no-going work

Overview

- Opportunity A: Deep Learning (DL) improves observability in ACD

- Opportunity B: Deep learning enables human analysts to spend less time on a task

Why identify critical data in server programs

- ACD needs to defend not only control-flow hijacking, but also **data-oriented attacks**
- In data-oriented exploits, attackers use memory errors to modify the non-control, **security-critical data** to affect the program execution
- Turing complete attacks
- Can help attackers achieve similar malicious goals

```
1  int setup_env( ... ) { ...
2    aclp = login_check_limits(conf, FALSE, TRUE, &i); ...
3    if (c == NULL && aclp == 0) { ...
4      goto auth_failure;
5    } ...
6  }
7  // login_check_limits ->check_limit ->check_limit_deny
8  int check_limit_deny(config_rec *c, ...) { ...
9    if (check_user_access(c->subset, "DenyUser")) {
10     return 1;
11   } ...
12 }
```

Code 1: An example of critical data. `aclp` determines whether the program accepts current user authentication or not. Attackers may corrupt this variable to bypass blacklist.

Existing works heavily rely on human efforts

- Different from control data that are easy to detect based on the data type (e.g., function pointers) and instructions (e.g., jump, call and return), critical data are mainly defined using the program-specific, high-level semantics

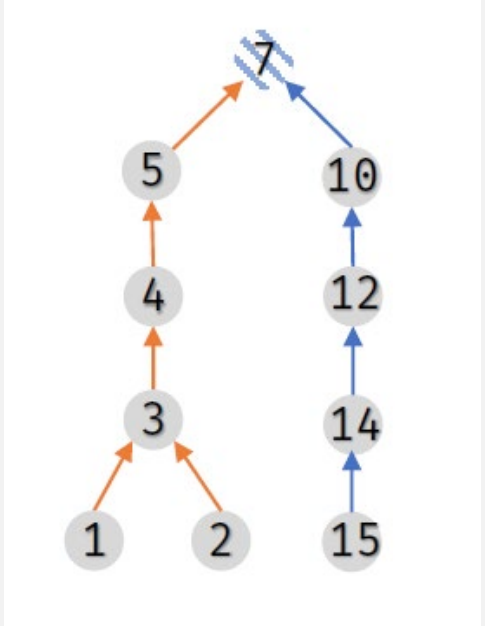
- Previous works on data-oriented exploits mainly rely on tedious human efforts to identify critical data

New method (arXiv:2108.12071)

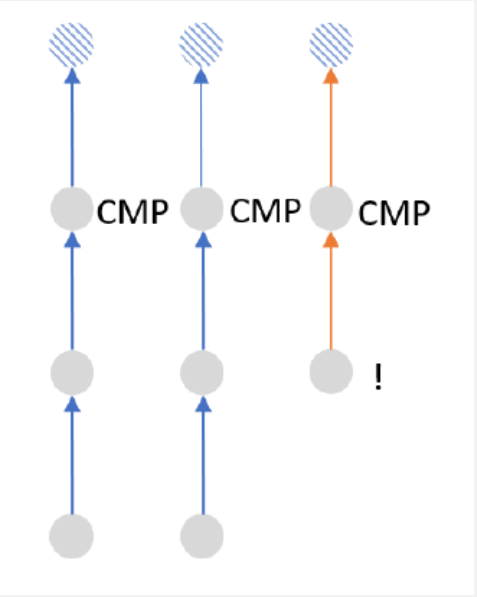
- It identifies critical data in an automated way
- It uses a deep neural model to identify critical data
  - Critical data are often associated with a particular data-flow pattern
  - Although a human analyst can write data-flow patterns to identify specific critical variables, such rules usually have very limited generalization ability, due to diversity among the data-flow patterns
  - In contrast, deep neural models have recently demonstrated remarkable generalization ability in recognizing useful patterns in several application domains such as computer vision and NLP
- It is the first work that applies DL to identify critical data in program binaries

New insights

- We found that traditional DFGs lead to low accuracy (62%, F1 score of 0.57), since the learned graph neural networks (GNN) can only learn “short” data-flow paths
  - The intuitive GNN models are not very capable
- Combination of newly defined data-flow trees and tree-LSTM enables one to learn “long” data-flow paths
  - The non-intuitive tree-LSTM models work well
- Measured control dependencies (i.e., how many basic blocks are influenced) are an important feature

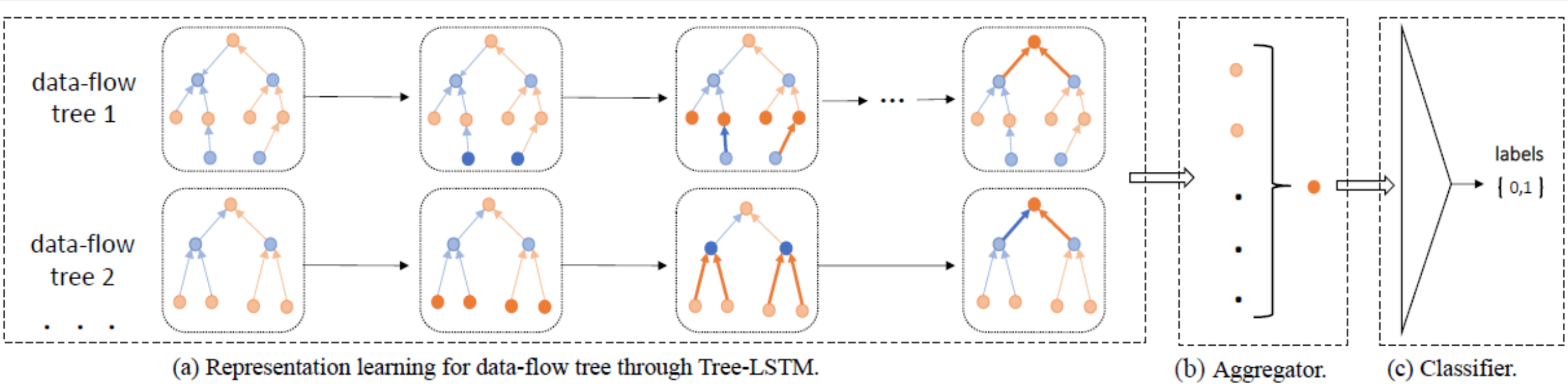


True Positive



True Negative

Workflow of the new method



(a) Representation learning for data-flow tree through Tree-LSTM. (b) Aggregator. (c) Classifier.

Evaluation results

Dataset: 6 programs (proftpd, nginx, etc.)
117 critical variables are labeled; 104 non-critical variables; 145K nodes; 133K edges

- Successfully identified 562 out of 615 critical variable usages
- 55 non-critical variable usages were misclassified as critical

TABLE II: Evaluation results on our model and other baselines.

Design	*Model*	*Accuracy*	*Precision*	*Recall*	*F1*
Ours	Tree-LSTM	0.8747	0.9108	0.9138	0.9123
Baseline 1	RNN	0.7254	0.8287	0.6342	0.7185
Baseline 2	LSTM	0.7465	0.8484	0.6871	0.7593
Baseline 3	MLP	0.2590	0.3900	0.3821	0.3859
Baseline 4	ConvGNN	0.5465	0.5071	0.3580	0.4197
Baseline 5	BRGCN	0.6200	0.5849	0.5553	0.5698

Deep Learning for ACD: Additional Challenges

- (C5) Insights on “which data structure is most appropriate to represent the raw data” are hard to be automatically obtained or learned
- (C6) When recognizing sophisticated patterns, DL could suffer from quite a few false positives

Overview

- Opportunity A: Deep Learning (DL) improves observability in ACD
- Opportunity B: Deep learning enables human analysts to spend less time on a task
- Future directions

Direction 1: Transfer learning

- Two main motivations
  - Imbalanced data is an outstanding issue in cybersecurity industry
  - Professionals want to avoid training a model from scratch when dealing with a new kind of attack or a new kind of cybersecurity task

- Solution ideas
  - Domain adaptation could be very useful
  - Pre-trained models (e.g., CodeBERT) could be very useful

Direction 2: Auto AI for autonomous defenses

- Current solutions need a security expert to manually work on certain things
  - During the data preprocessing phase, which data structure to use is a critical factor.
  - In many cases, a **new** data structure is manually envisioned based on experts' domain knowledge
- How to make the learning procedure more automated?

Direction 3: Hierarchical learning

- Lower level learning agents (e.g., DNNs) are performing perception tasks (e.g. pattern recognition)
- Higher level learning agents are performing comprehension tasks (e.g., Bayesian reasoning)
- However, the research community has limited understanding about how cross-level learning could be conducted

Direction 4: Human-bot teaming

- On one hand, human analysts and AI bots (e.g., DNNs) are complementary
  - AI bots can achieve very high throughput in information processing, but human analysts have cognition bottleneck
  - Human analysts can leverage past experiences to conduct unconscious reasoning (e.g., intuitions, gut reactions, guesses, and insights), but AI bots cannot
- On the other hand, they speak different “languages” and often do not “understand” each other

Questions?

Thank you!