"*Bring your own device*!": Adaptive IoT Device-type Fingerprinting using Automatic Behavior Extraction

Maxwel Bar-On[1], Katherine Patterson[1], Bruhadeshwar Bezawada[2], *Indrakshi Ray*[1] and Indrajit Ray[1]

Computer Science Department, Colorado State University

Mathematics & Computer Science Department, Southern Arkansas University

IoT Fingerprinting

- IoT security is challenging
  - Weak in-built security of devices, dynamic environment, lack of sys admin
- IoT device fingerprinting playing an increasingly important role
  - Identifying which devices are on the network
  - Checking device's expected behavior
- IoT fingerprinting uses machine learning models trained on known traffic patterns of devices
- Fingerprint model must constantly adapt to evolving IoT landscape
- *How can we fingerprint efficiently and accurately?*

Research Goals and Problem Statement

Goals:

- **Device Fingerprinting**: identifying a device by matching a meaningful representation of a device's communication patterns.
  - We assume the existence of a trained fingerprinting model on N devices.
- **Adaptation**: adapting, or reconfiguring, a fingerprinting model on N devices to identify P newly added devices
  - The adapted model must be capable of identify new devices with no loss of accuracy

Problem Statement:

- Adapt the fingerprinting model on N devices to N+P devices while keeping the cost of adaptation lesser than the cost of generating a fresh fingerprinting model on N+P devices.

Limitations of Existing Approaches

- Fingerprinting model must be retrained when new devices are added
- Existing model performs poorly if the original network features are insufficient for describing behaviors of new devices

Our Approach

- Two stage fingerprinting process
- *Behavior Extractor (BE)*: transformer-based behavior extraction
  - BE can be reused with no additional training
  - Conversational style BE trained on self-supervised general tasks
    - Anomaly locating, denoising, masked auto-encoding
- *Fingerprint Interpreter (FI)*: Neural network multiclassifier for identifying devices
  - FI needs to be retrained for new devices
  - Lightweight and efficient to train

Fingerprinting Identification

- Capture window of packets sent to/received from an unknown device
- Extract feature vectors from packet
  - Standard: Features extracted from each packet
  - Relative: Tries to capture correlations among packets in the window
- Use our architecture to predict the type of device

Standard Features

- 13 standard features
- Derived independently for each packet in a window
- Mix of categorical and numerical values
- Extracted from network, transport, and payload layers
- Feature design accommodates encrypted packet payloads

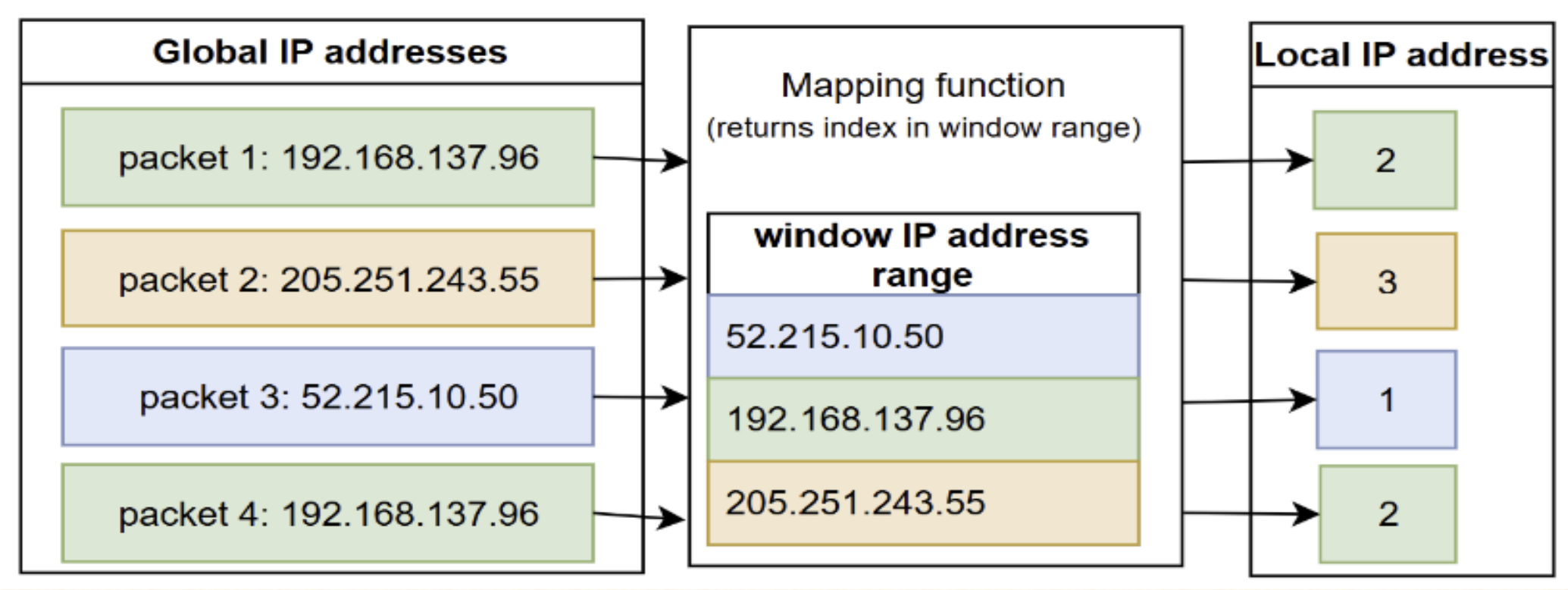
Feature	Explanation
Subnet Src	1 if source IP is in subnet else 0
Subnet Dst	1 if destination IP is in subnet else 0
Broadcast Dst	1 if destination IP is broadcast else 0
TLS	1 if packet includes TLS handshake header else 0
TCP	1 if packet includes TCP header else 0
UDP	1 if packet includes UDP header else 0
HTTP	1 if packet uses HTTP port else 0
SSL	1 if packet uses SSL port else 0
Common Src	1 if packet uses common source port else 0
Common Dst	1 if packet uses common dst port else 0
Header Length	length of transport-layer header (bytes)
Payload Length	length of payload (bytes)
Payload Entropy	information entropy of payload

Relative Features

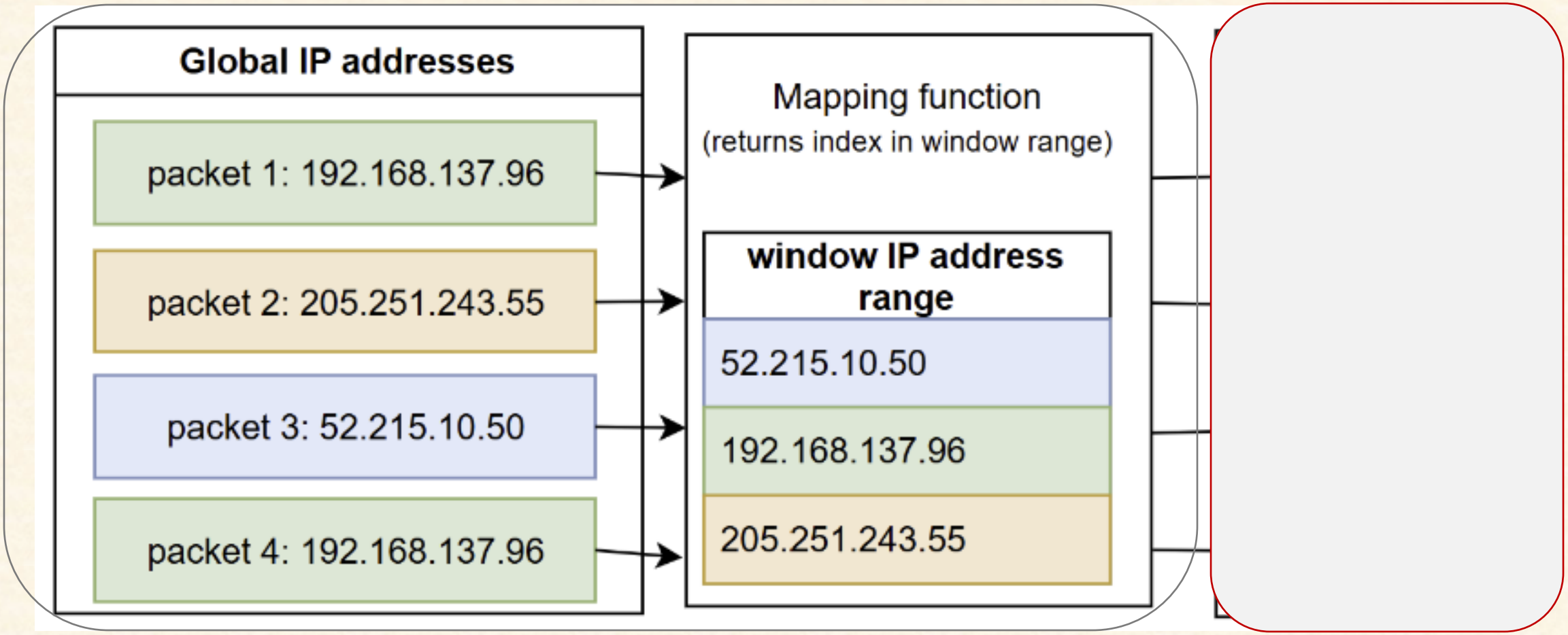
- Encoded representations of the endpoints of packets: <*Source IP address, destination IP address, source port, destination port*>
  - Useful for capturing relationship among packets and flows within a window
  - Relative features represented as a 15-dimensional embedding vector
- Generated using relative encoding mapping function
  - Maps each endpoint identifier field to a smaller scale that is local to the packets in a window
  - Localized endpoint identifiers are stored as one-hot encodings and compressed into 15-dimensional embedding vectors using an autoencoder
- Encoding on a smaller, local scale preserves relationships among packets and flows
  - Does not cause overfitting by reducing dimensionality or introduce overlap between different classes in the training data

Relative Features: Encoding Process

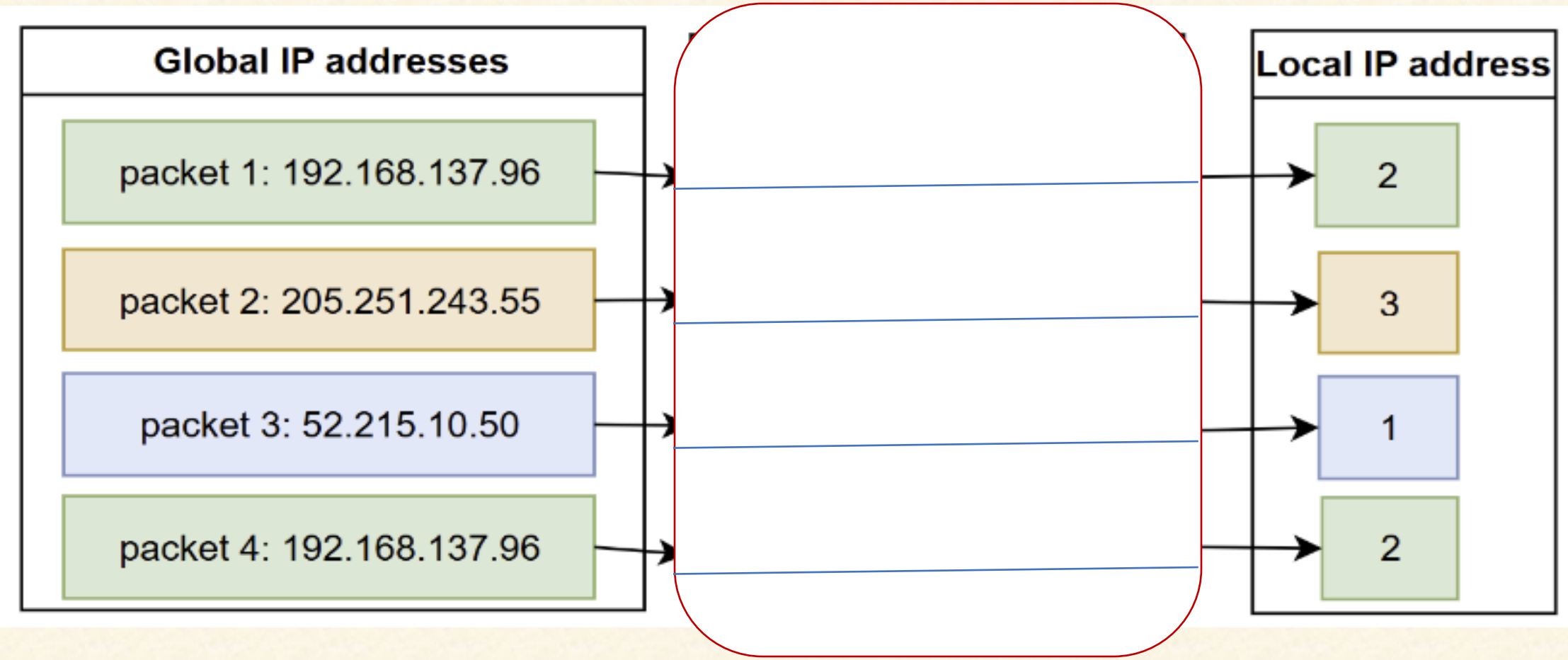
- Mapping function returns the index of each endpoint identifier field in the sorted list of unique values for the field within the window



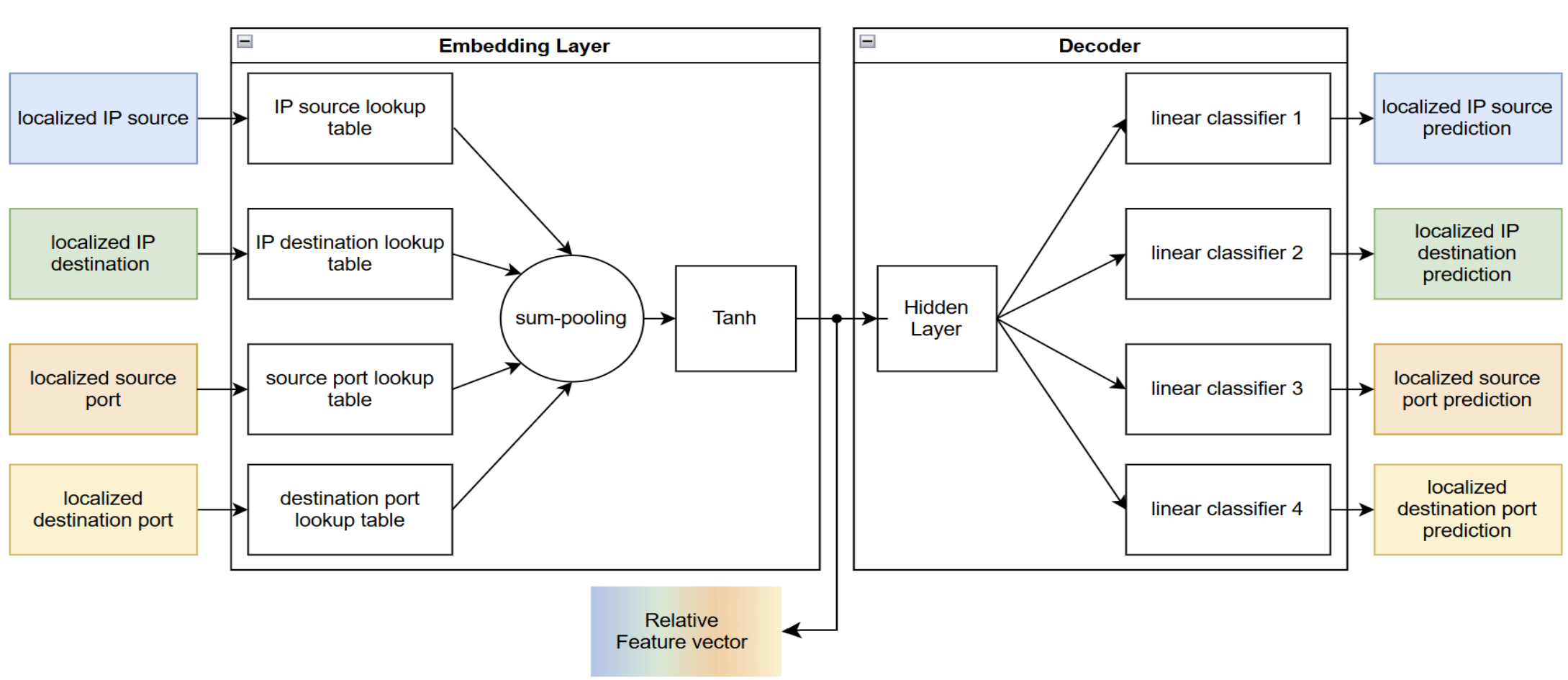
Relative Features: Encoding Process Step 1



Relative Features: Encoding Process Step 2



Relative Features: Compression



Relative Features: Illustration using Attention Values

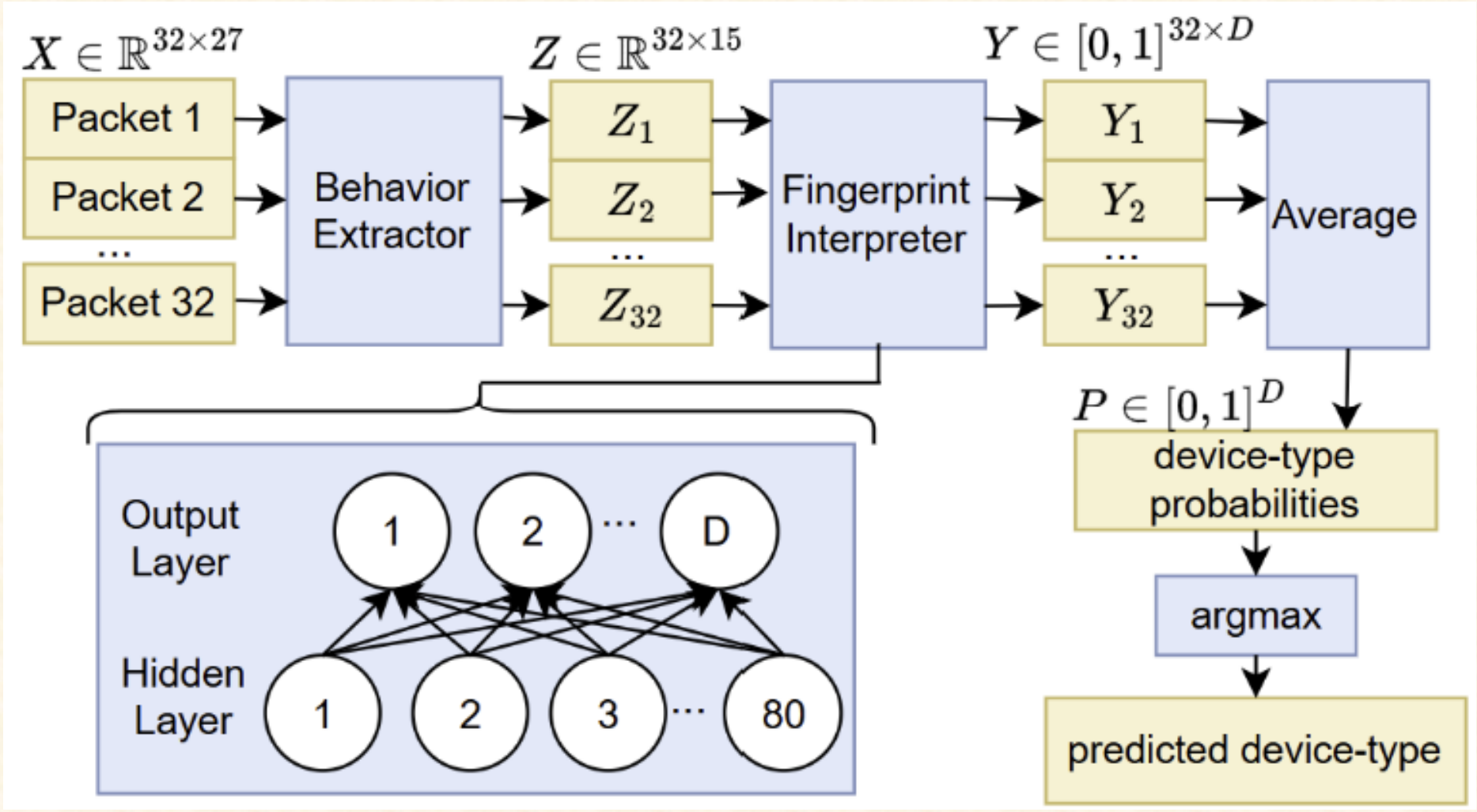
- $R\left(P^{(1)} : (192.168.137.96,\ 205.251.243.55,\ 42000,\ 443),\ S\right) = [1, 2, 3, 2]$
- $R\left(P^{(2)} : (205.251.243.55,\ 192.168.137.96,\ 443,\ 42000),\ S\right) = [2, 1, 1, 3]$
- $R\left(P^{(3)} : (192.168.137.96,\ 205.251.243.55,\ 45000,\ 137),\ S\right) = [1, 2, 4, 1]$
- $R\left(P^{(4)} : (192.168.137.96,\ 205.251.243.55,\ 42000,\ 443),\ S\right) = [1, 2, 3, 2]$
- $R\left(P^{(5)} : (192.168.137.96,\ 205.255.4.123,\ 40000,\ 137),\ S\right) = [1, 3, 2, 1]$

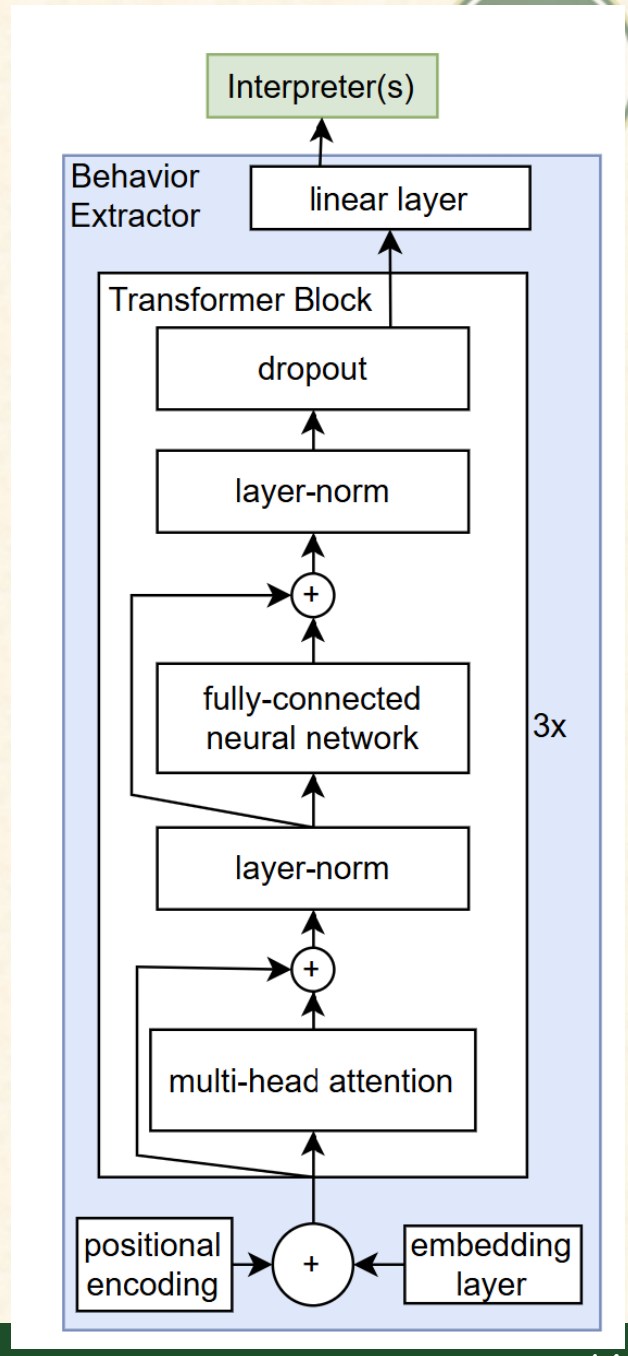
- Window of five packets (S)
  - R(packet, S): relative feature mapping function

- Attention values between packets in S reflect the relationships between the various packets/flows

	$P^{(1)}$	$P^{(2)}$	$P^{(3)}$	$P^{(4)}$	$P^{(5)}$
$P^{(1)}$	0.4538	0.0083	0.0614	0.4538	0.0226
$P^{(2)}$	0.0171	0.9317	0.0171	0.0171	0.0171
$P^{(3)}$	0.0950	0.0129	0.7021	0.0950	0.0950
$P^{(4)}$	0.4538	0.0083	0.0614	0.4538	0.0226
$P^{(5)}$	0.0397	0.0146	0.1080	0.0397	0.7979

Architecture





Behavioral Extractor Training

- BE trained on self-supervised machine learning tasks
- BE is updated by back-propagating loss through the self-supervised interpreter
- We evaluate several BE training procedures
  - Each procedure includes one or more self-supervised tasks
  - For each task, we initialize a separate interpreter
  - Simultaneously train the interpreters and the BE until all tasks in the procedure have converged

Self-Supervised Tasks

- Masked autoencoding
  - Apply mask to BE input features, randomly remove 40% of features
  - Train model to reconstruct missing features
- Denoising
  - Add random noise to BE inputs by adding values in the range (-0.15, 0.15) to each value
  - Train model to reconstruct original features
- Anomaly locating
  - Randomly insert anomalous value in place of one packet in each window
  - Train model to predict relative position of anomaly for each packet X
  - Predict 0, 1, or 2 depending on whether X is before, or is, or after the anomaly

BE Training Procedures

- L + M: anomaly locating and mask encoding
- Denoiser: only denoising
- L + D: anomaly locating and denoising
- M + D: masked autoencoding + denoising
- Mask: only masked autoencoding

Fingerprint Interpreter

- Fully connected neural network multiclass classifier
  - Only one hidden layer
- Training
  - Fingerprint interpreter is trained on encoded traffic generated by BE to identify the IoT devices corresponding to each sample
- Identifying devices
  - Traffic sample collected from unknown devices
  - Standard and relative features extracted from samples
  - BE extracts behaviors from traffic features and produces encoded samples
  - Fingerprint identifier identifies device

Adding New Device

- Traffic from new device is encoded using original BE
- Encoded traffic from new device is combined with the encoded traffic from the initial devices and the combined traffic is used to train the FI
- Original FI is replaced by the new FI

Experimental Evaluation

- Data Sets
  - Colorado State University, Canadian Institute of Cybersecurity, University of New South Wales
- Static Model
  - Use it as a baseline
  - Static model is formed by combining the BE + FI are trained together as one component
  - When new devices are introduced, the static models are retrained from scratch

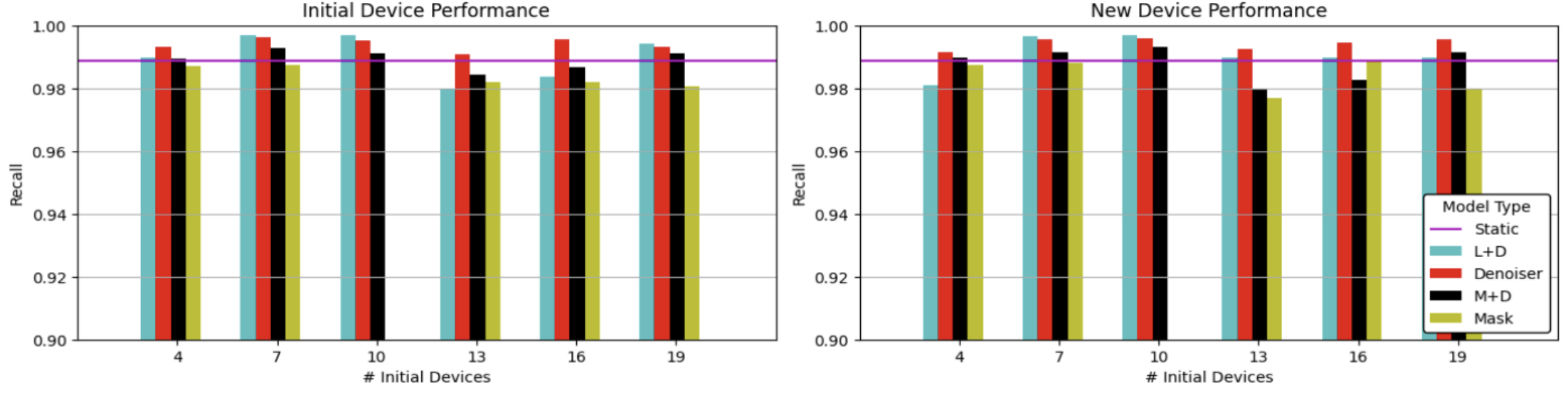
Results: Efficiency

- Adapt time: time required to train the FI
- Denoiser model was the best

	Adaptable Models					Static Model
	L+M	Denoiser	L+D	M+D	Mask	
GPU	384	60	67	303	391	4,698
CPU	22,921	3,619	4,021	18,095	23,323	78,513

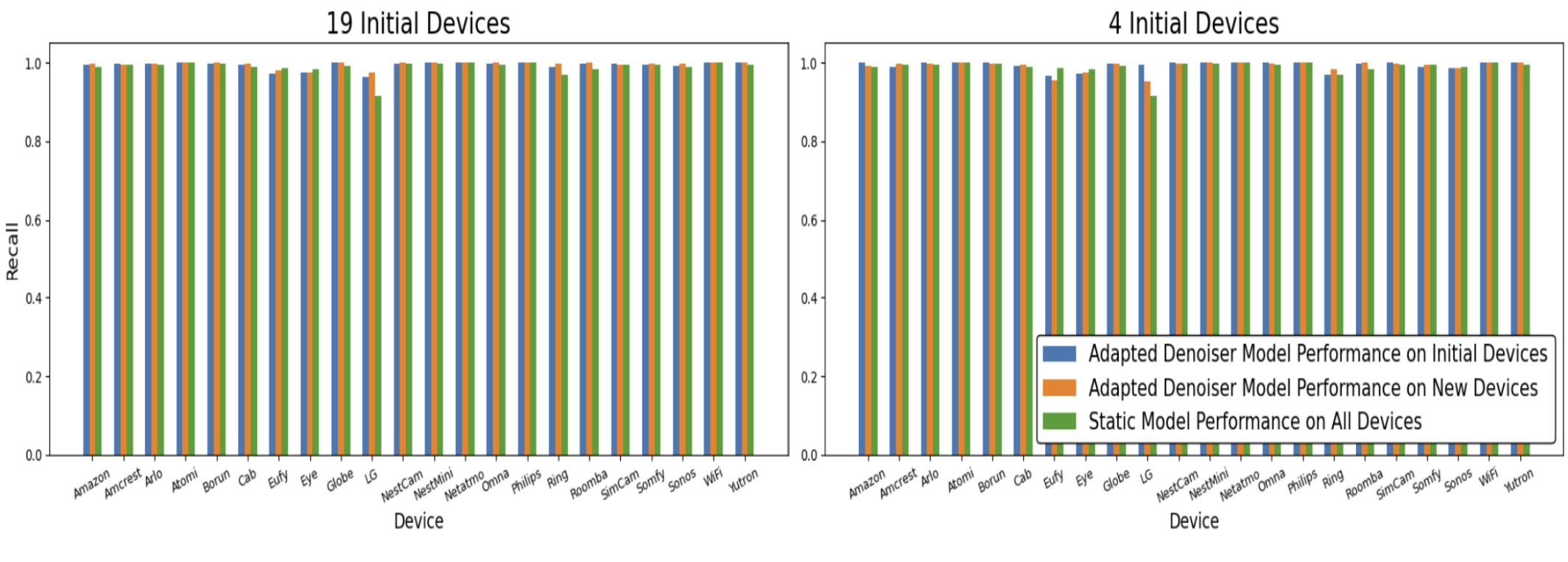
Results: Performance

- Static model always has the same number of initial devices because it does not have multiple training stages
- Denoiser model always outperforms static models



Results: Denoiser Model

- Best performing across all devices



Conclusion

- Developed an IoT fingerprinting method based on a bi-component architecture
  - once trained component (BE) and light-weight retraining FI component
- Introduced concept of relative features
- Evaluated five different procedures for training the BE in terms of adaptation efficiency and performance
  - 60 second adapt time with Denoiser-trained BE on GPU
  - over 98% average Recall on new and initial devices

Future Work

- Transformers training costs are high, so look at reducing them
- Demonstrate using larger networks
- Address behavior drift in machine learning models
- Address adversarial attacks