RGB_Mem : At the Intersection of Memory Forensics and Machine Learning

Aisha Ali-Gombe, *Sneha Sudhakaran, Ramypandian Vijayakanthan, Golden G. Richard III*

Systems Security Labs (SySec)

Louisiana State University

July 2023 @ DFRWS

Motivation

Incident Response Steps

Step 1: Preparation

Step 2: Identification/Analysis

Step 3: Containment

Step 4: Eradication

Step 5: Recovery

Step 6: Lessons Learned

Traditional Malware Analysis Techniques

Static analysis

- examine program file
- extract data such as permissions, API calls, strings, resources and instructions
- detailed
- drawback – **time consuming and obfuscation**

Dynamic analysis

- runtime behavioral monitoring
- quick analysis, resilient to common obfuscation
- drawback – ***preconfigured environment requiring execution tracing, low-level system modification***

Memory Forensics

Clipboard data

Volatile registry branches

Network connections

Running processes

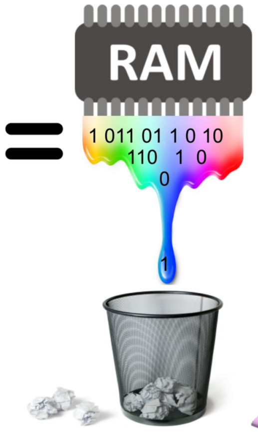
Open files

Encryption keys

Private browsing data

Kernel structures

Application structures



Userland Memory Forensics (UMF)

I stole the squirrels

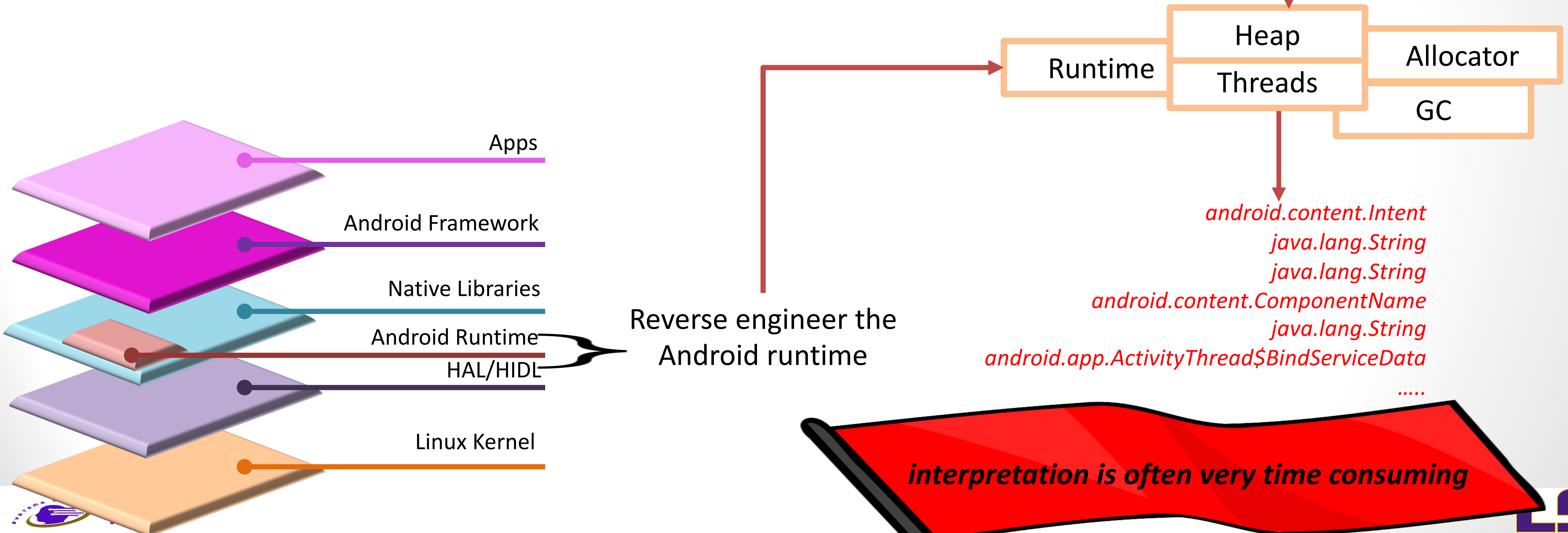
IP Header	sbcjkshuqeh qwnbNjhfiid

Kernel-land Memory Forensics

- post-mortem investigation of memory dump
- extract running processes and kernel data structures and modifications
- offline analysis – no system modification, resilient to obfuscation

DroidScraper (Ali-Gombe et. al, 2019)

- App-gnostic tool for in-memory data recovery and reconstruction



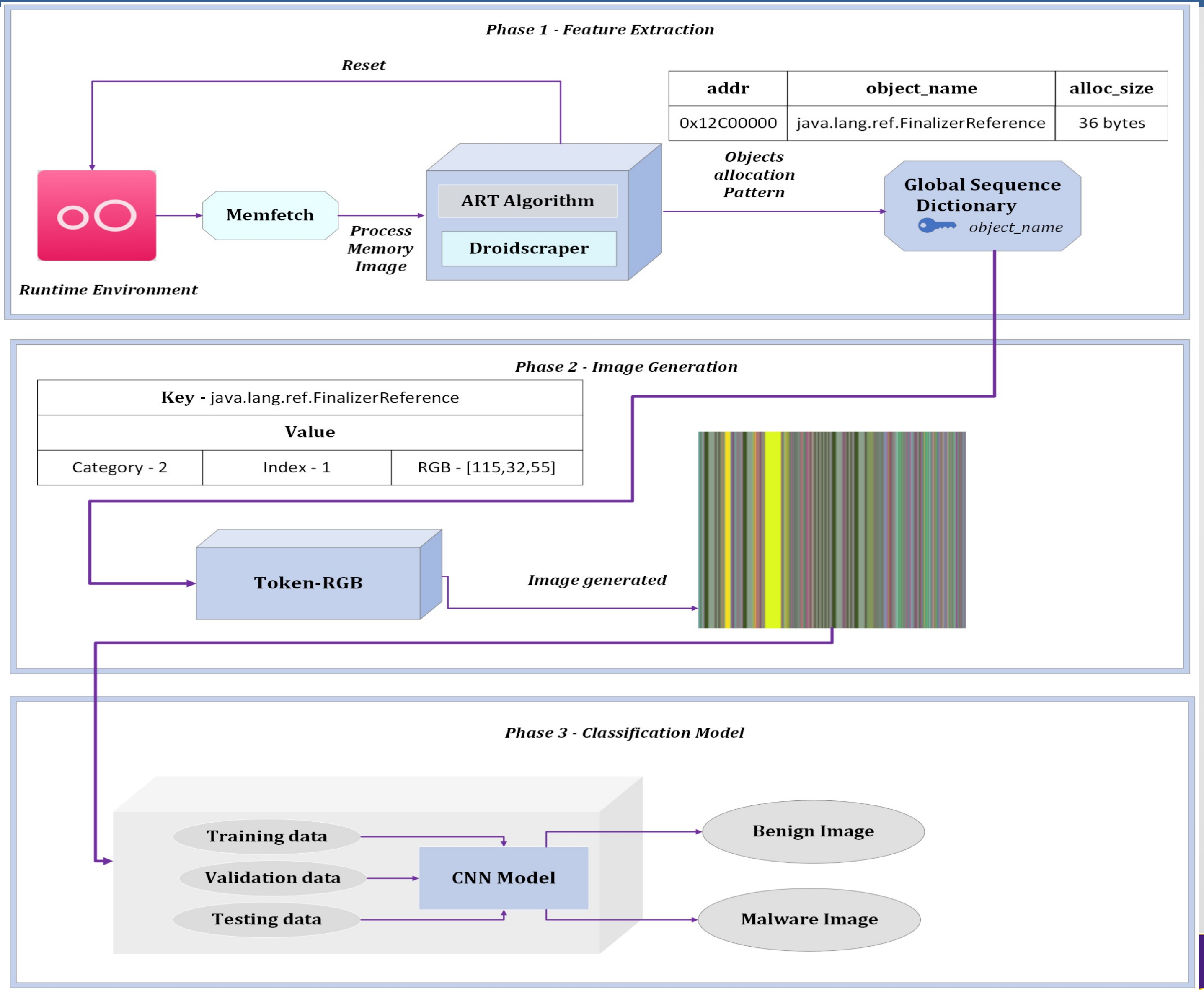
Research Questions

1. Can the recovered **in-memory artifacts** from memory forensics be used to generate **robust and uniquely identifiable features?**

2. Can these features be leveraged for **effective malware classification**?

RGB_Mem

- Automated Android malware classification engine
- Leverages Droidscraper to generate allocation patterns
- These patterns are processed into an RGB image representation and then passed to a CNN model as feature vectors
- Objectives –
  - overcome obfuscation, scalability, interpretation challenges of traditional techniques
  - develop effective classification model for Android



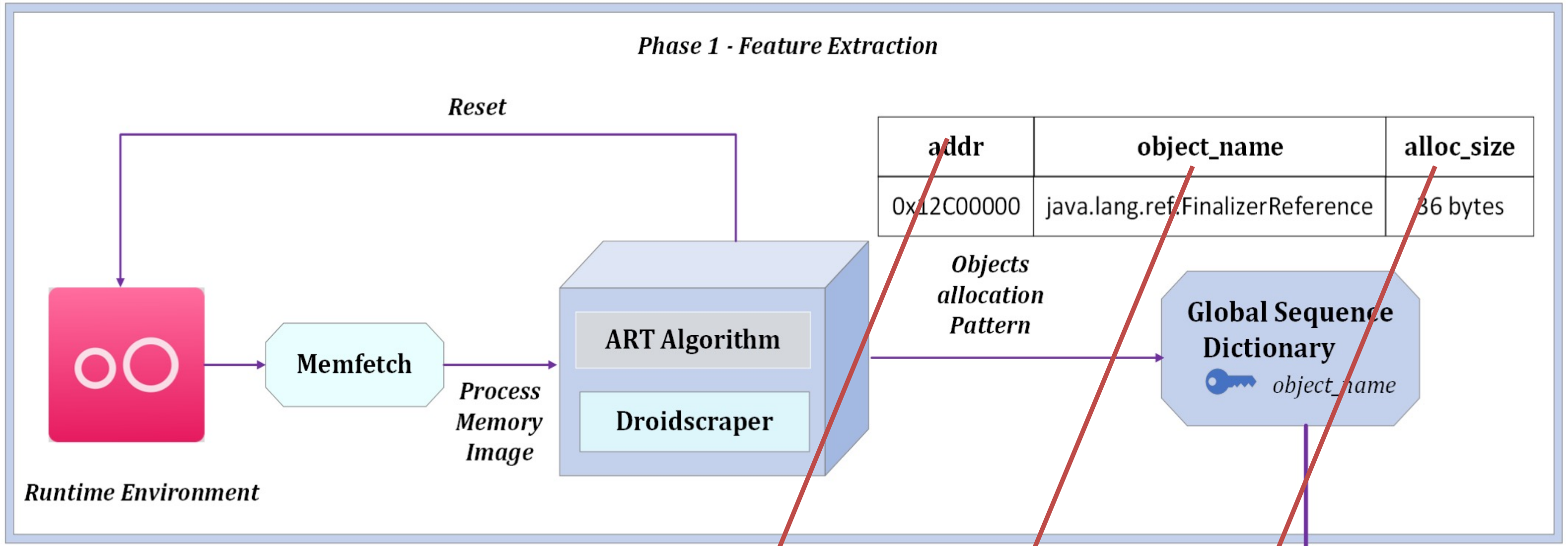
Android (ART) Region Space Allocator

```
inline mirror::Object* RegionSpace::Region::Alloc(size_t num_bytes, size_t* bytes_allocated,
size_t* usable_size, size_t* bytes_tl_bulk_allocated) {
    uint8_t* old_top; uint8_t* new_top;
    do {
        old_top = top_.LoadRelaxed()
        new_top = old_top + num_bytes;
    ….
    } while (!top_.CompareAndSetWeakRelaxed(old_top, new_top));
```

```
'RegionSpace' : [ 0xa8, {
    'ContinuousMemMapAllocSpace' : [0],
    'region_lock_': [56],
    'time_': [96],
    'num_regions_': [100],
    'num_non_free_regions_': [104],
    'regions_': [108],
    'non_free_region_index_limit_': [112],
    'current_region_': [116],
    'evac_region_': [120],
    'full_region_': [124],
    'mark_bitmap_': [164],
}
```

```
'Region' : [ 0x28, { 'idx_' : [0],
    'begin_': [4],
    'top_': [8],
    'end_': [12],
    'state_': [16],
    'type_': [17],
    'objects_allocated_': [20],
    'alloc_time_': [24],
    'live_bytes_': [28],
    'is_newly_allocated_': [32],
    'is_a_tlab_': [33],
    'thread_': [36],
}]
```

RGB_Mem Phase 1 – Feature Extraction

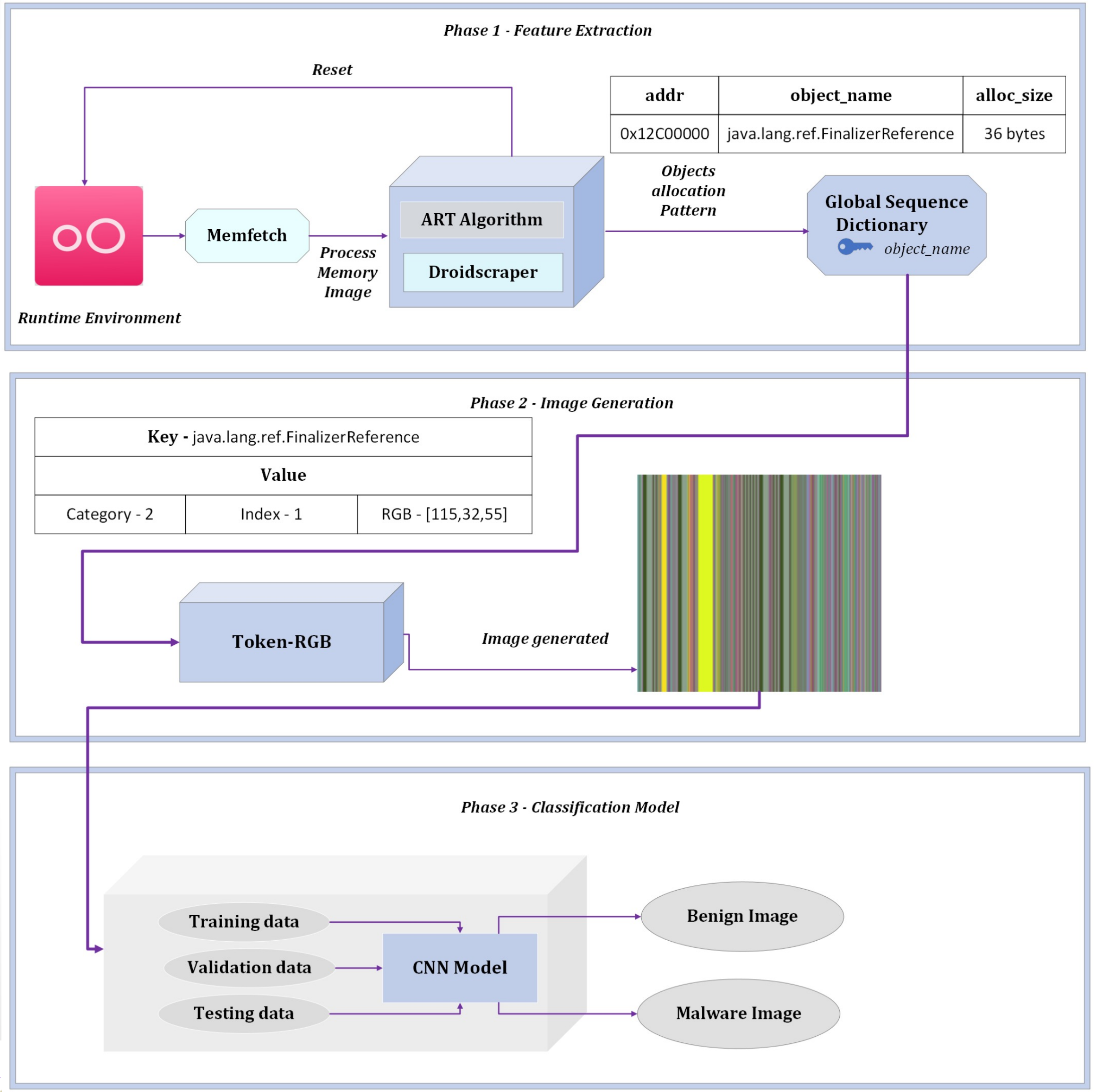



```
Address 0x12c83298
Address 0x12c832a0 [Ljava.util.concurrent.RunnableScheduledFuture; 8
Address 0x12c832b0 [Ljava.util.concurrent.RunnableScheduledFuture; 12
Address 0x12c832c0
Address 0x12c832c8 [Ljava.util.concurrent.RunnableScheduledFuture; 8
Address 0x12cc0000 [Ljava.lang.Class; 17
Address 0x12cc0020 [Ljava.lang.Class; 17
Address 0x12cc0040 [Ljava.lang.Object; 13
Address 0x13280000 java.lang.String 20
Address 0x13280018 java.lang.String 16
Address 0x13280028 android.net.NetworkInfo 44
Address 0x13280058 java.lang.String 25
Address 0x13280078 java.lang.String 25
Address 0x13280098 java.lang.String 29
Address 0x132800b8 java.io.File 24
Address 0x132800d0 java.io.File 24
Address 0x132800e8 [C 118
```

Algorithm 1 Resolving object allocation patterns for the memory snapshots

$corpus \leftarrow List()$
for $mem \in mem1, mem2....., memn$ **do**
  $runtime \leftarrow Droidscraper.getRuntime(mem)$
  $heap \leftarrow Droidscraper.getHeap(meme)$
  $regions \leftarrow getRegion(runtime, heap)$
  $sort(regions, regions.index)$
  for $region \in regions$ **do**
    $P\_region \leftarrow List()$
    $endOfRegion \leftarrow region.size$
    $current \leftarrow seekRegion()$
    while $current \neq endOfRegion$ **do**
      $objAddr \leftarrow current + region.addr$
      $classOff = read(4)$
      $name, size \leftarrow resolve(classOffset)$
      $P\_region \leftarrow [objAddr, name, size]$
      $corpus \leftarrow P\_region$
      $current \leftarrow current + size$
    end while
  end for
end for

RGB_Mem Phase 1 – Sequence Dictionary

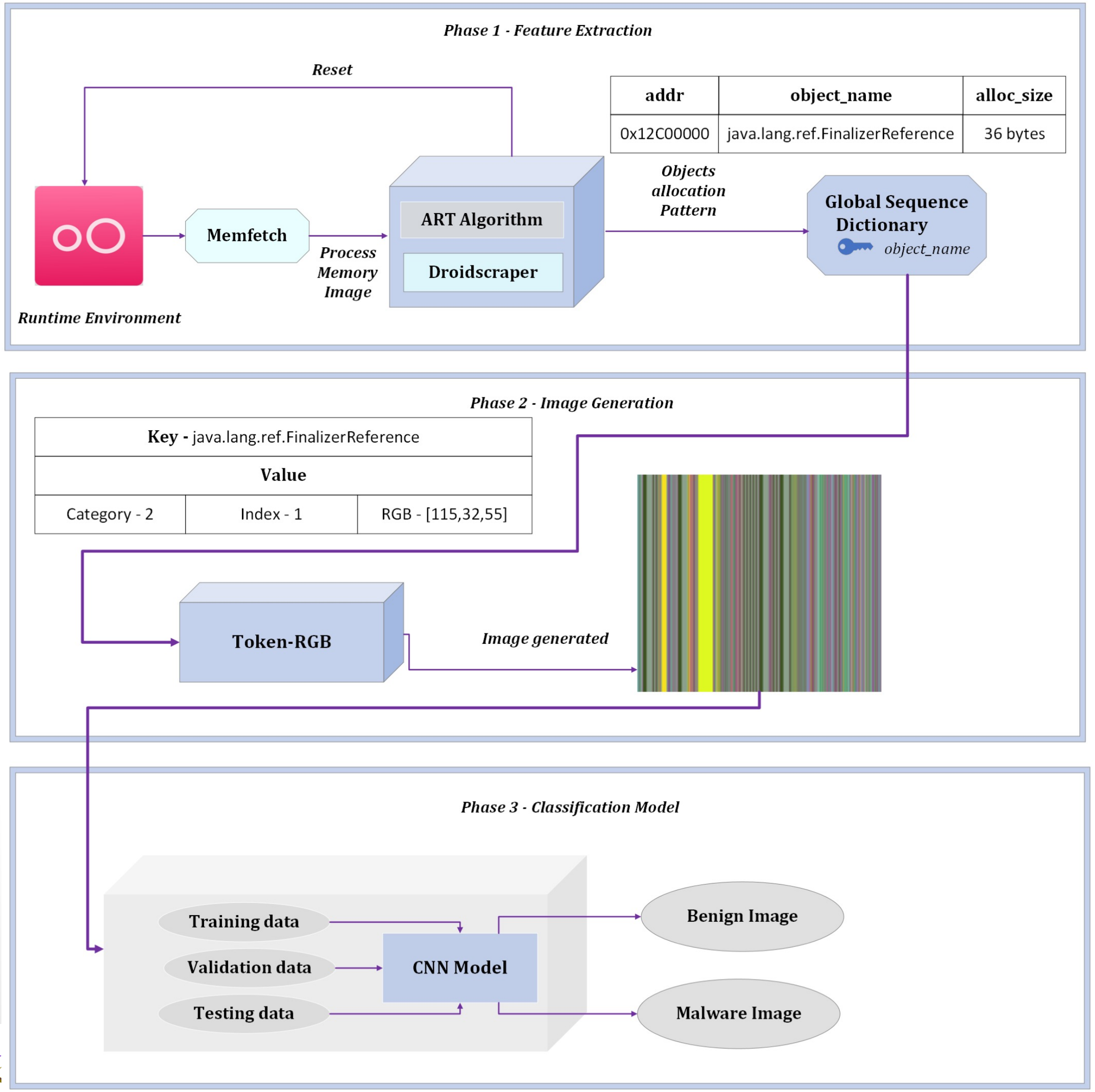


Algorithm 2 Creating the sequence dictionary

```
index ← 0
for l in list1,list2.....,listn do
    cat ← l.getCat()
    for object in list do
        if ! sequenceDict[object] then
            rgbVal ← randRGB(index)
            sequenceDict[object] ← (cat, index, rgbVal)
            index ← index + 1
        else if sequenceDict[object].getCat() ≠ cat then
            sequenceDict[object] ← cat
        end if
    end for
end for
```

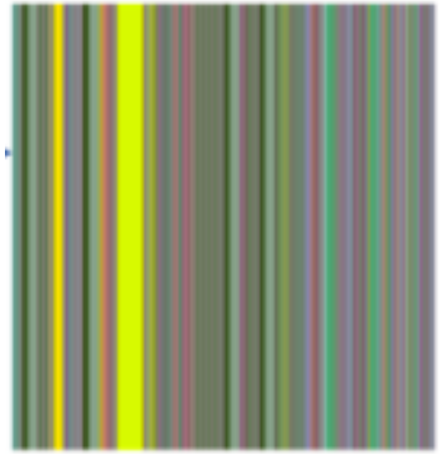
(java.lang.String, $Index_1$, RGB_1),
(java.lang.Float, $Index_2$, RGB_2), ...,
(java.lang.T hread, $Index_{100}$,RGB_{100}),...,
($Object_n$,$Index_n$,RGB_n

RGB_Mem Phase 2 – Image Generation



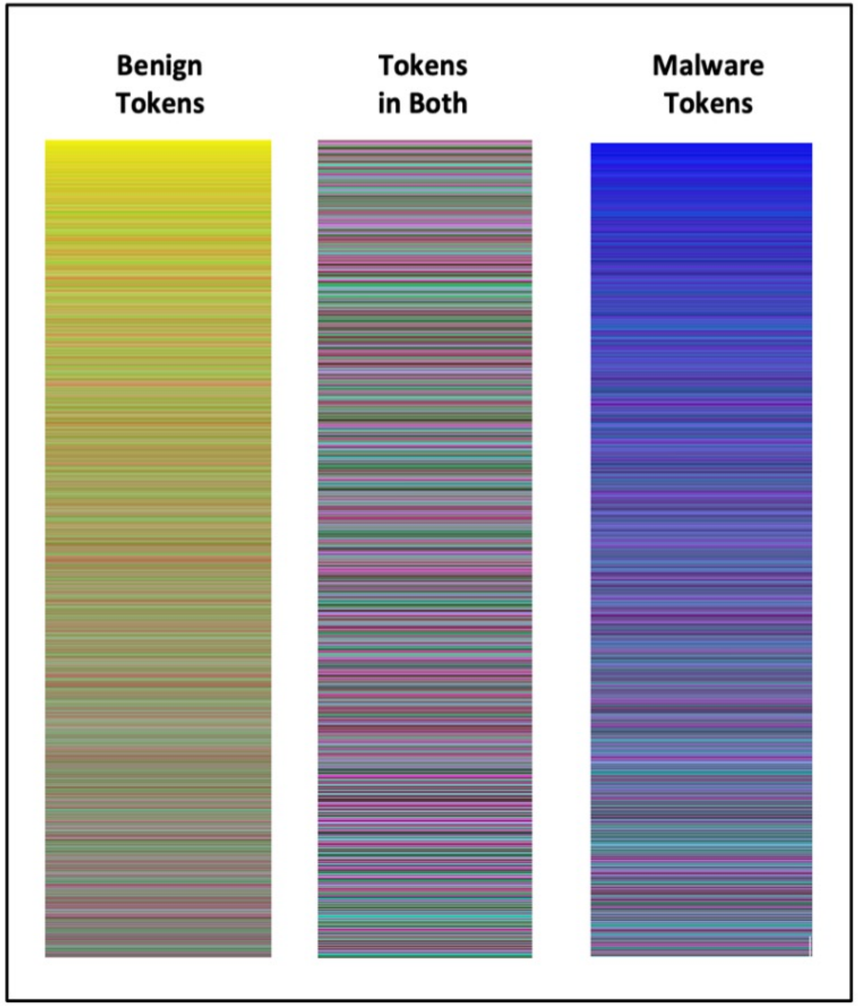
$Pattern_a \rightarrow$ [[0x0,java.lang.String, 12], [0xC, java.lang.Thread, 36], [0x30, java.lang.String, 24], [0x48, java.Lang.Float, 24]],

$Image_a \rightarrow$ $(RGB_1, RGB_{100}, RGB_1, RGB_2)$

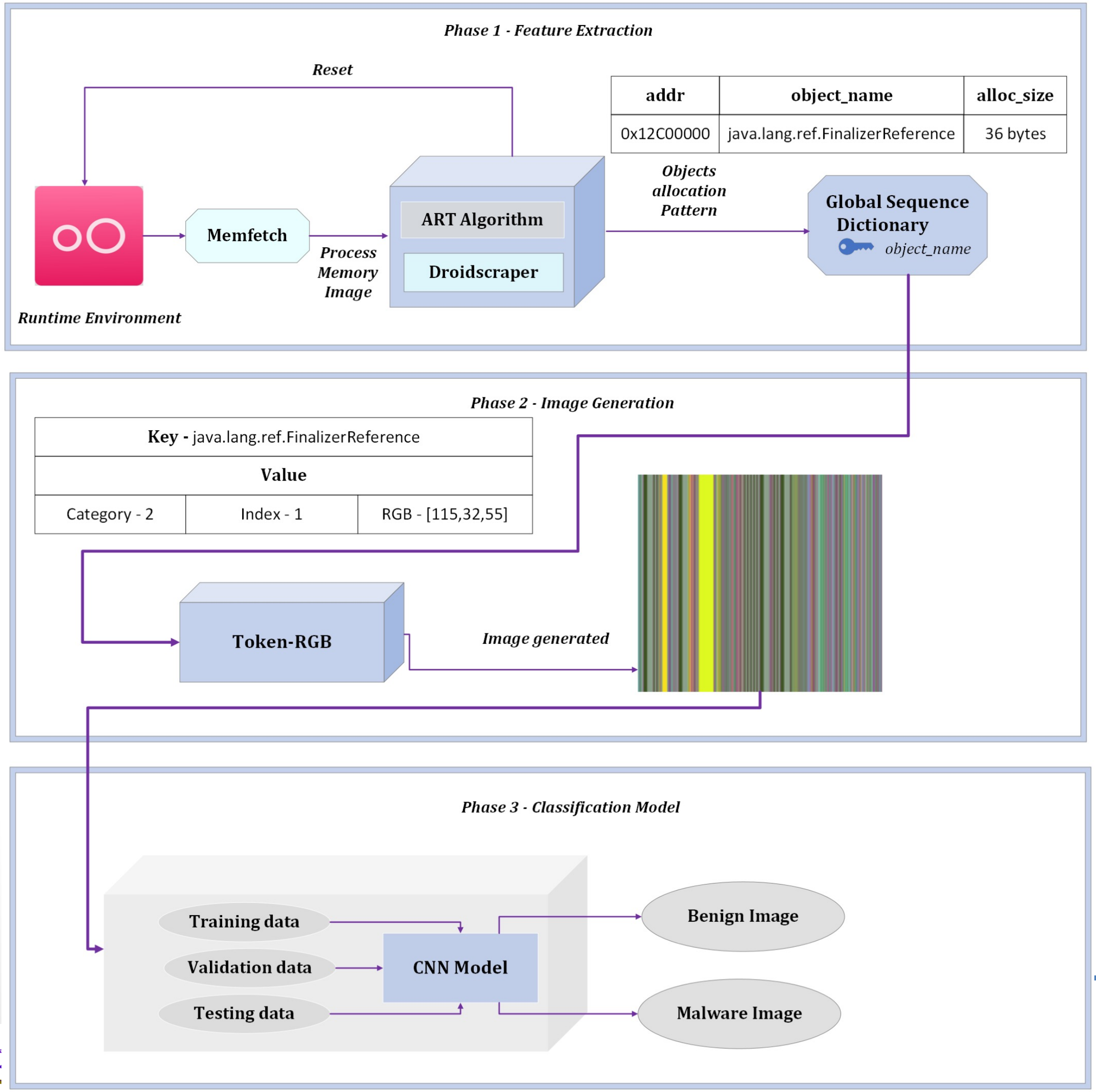


RGB_Mem Dataset Sequence Dictionary

- Dataset = 1411 memory images (823 malware and 588 benign)
- Size of sequence dictionary = 18,659 unique objects
  - Malware-only objects = 6986
  - Benign-only objects = 9191
  - Overlapping objects = 2479

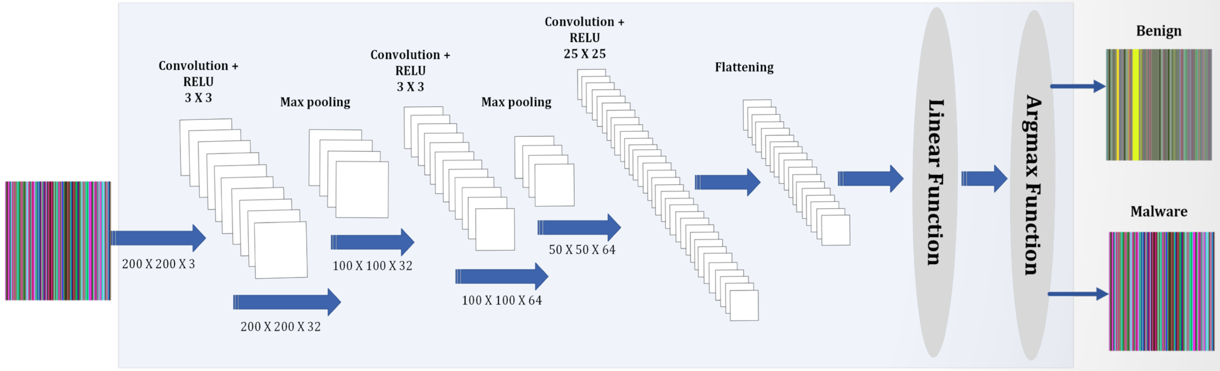


RGB_Mem Phase 3 – Classification Model



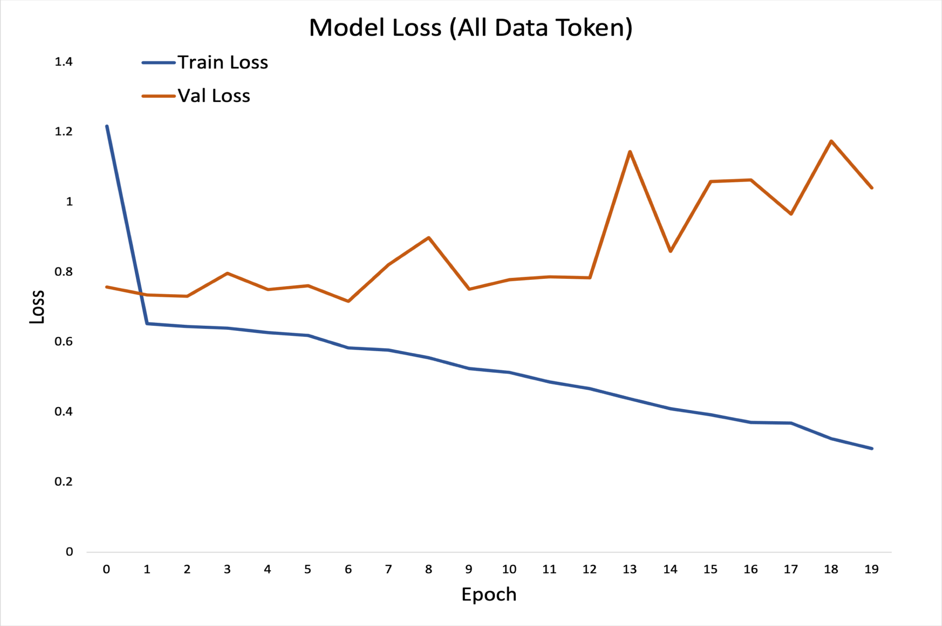
CNN - Model

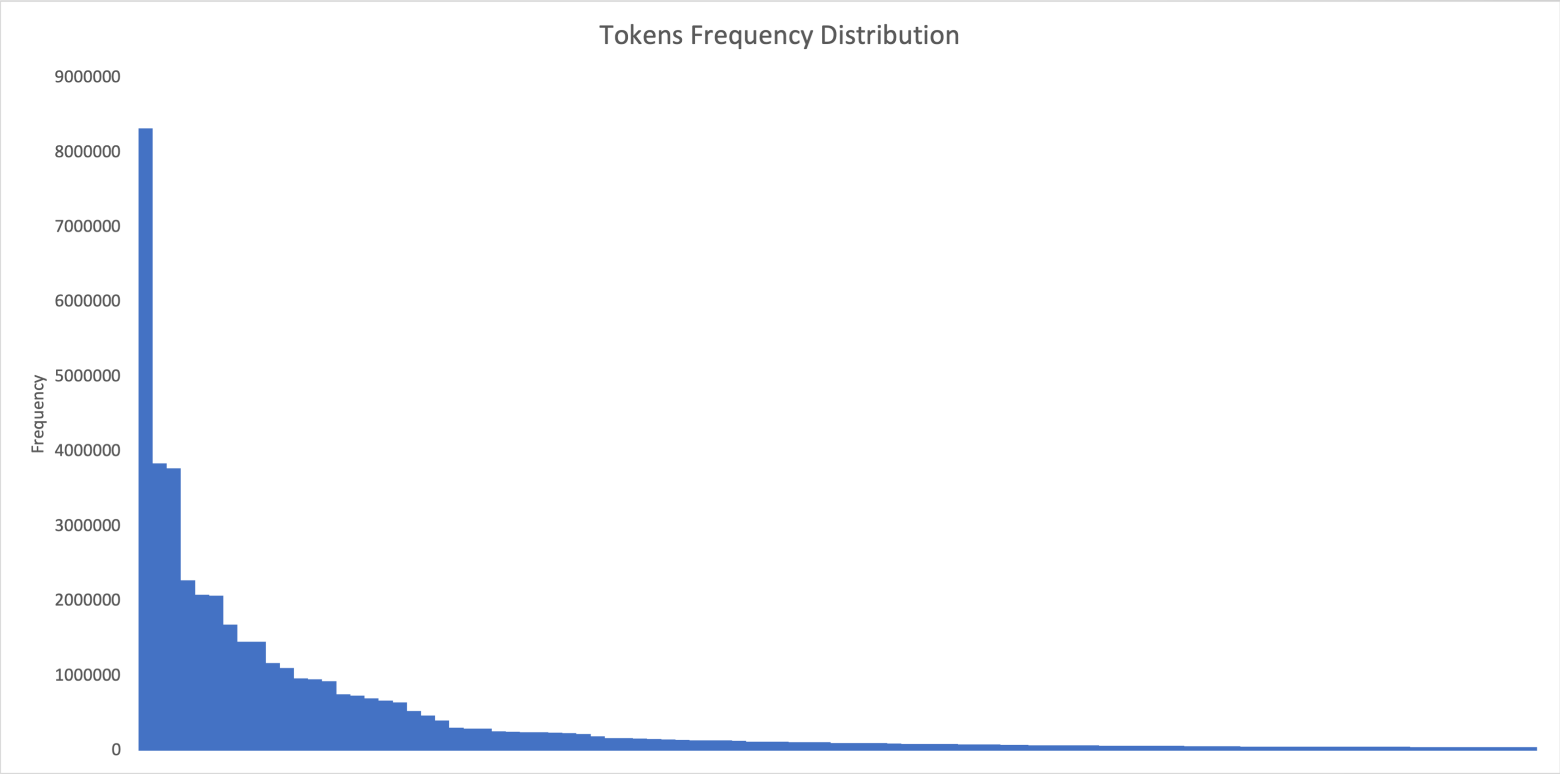
Data split - 70% training, 16% validation, and 14%

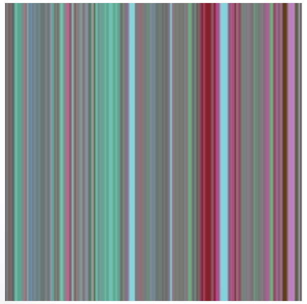


Is the Model Optimal?

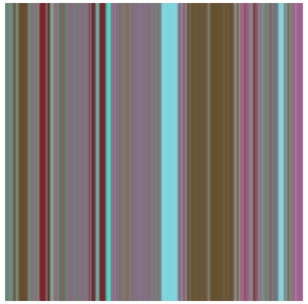
Optimization Learning Curve







Benign

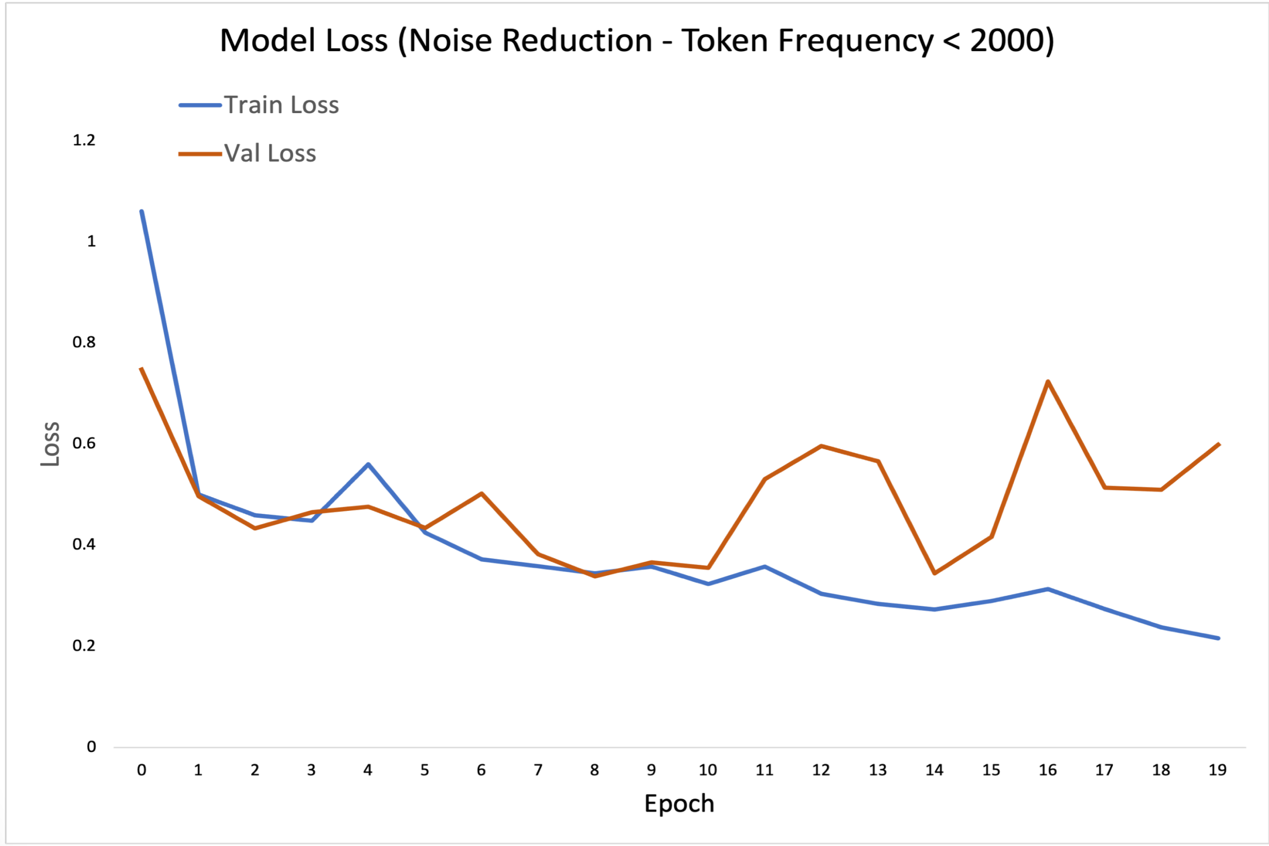


Malware

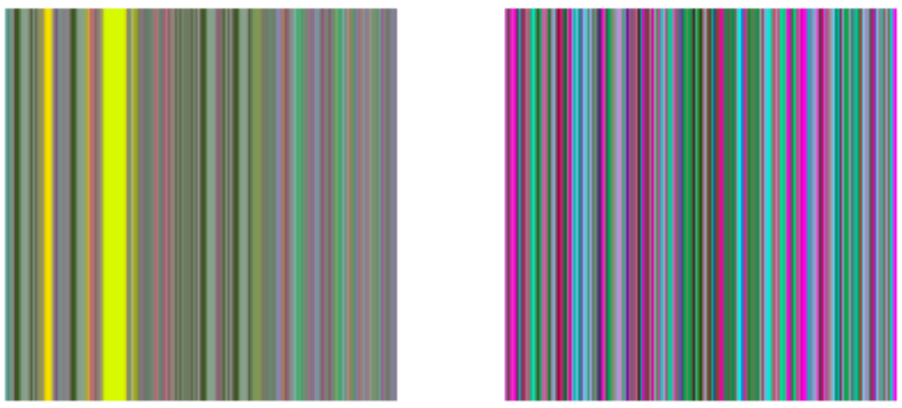
This indicates that the model could not learn from the training dataset mainly due to noisy and redundant data

Model Optimization - Recursive Feature Elimination

- Dimensionality Reduction with Token Frequency < 2000



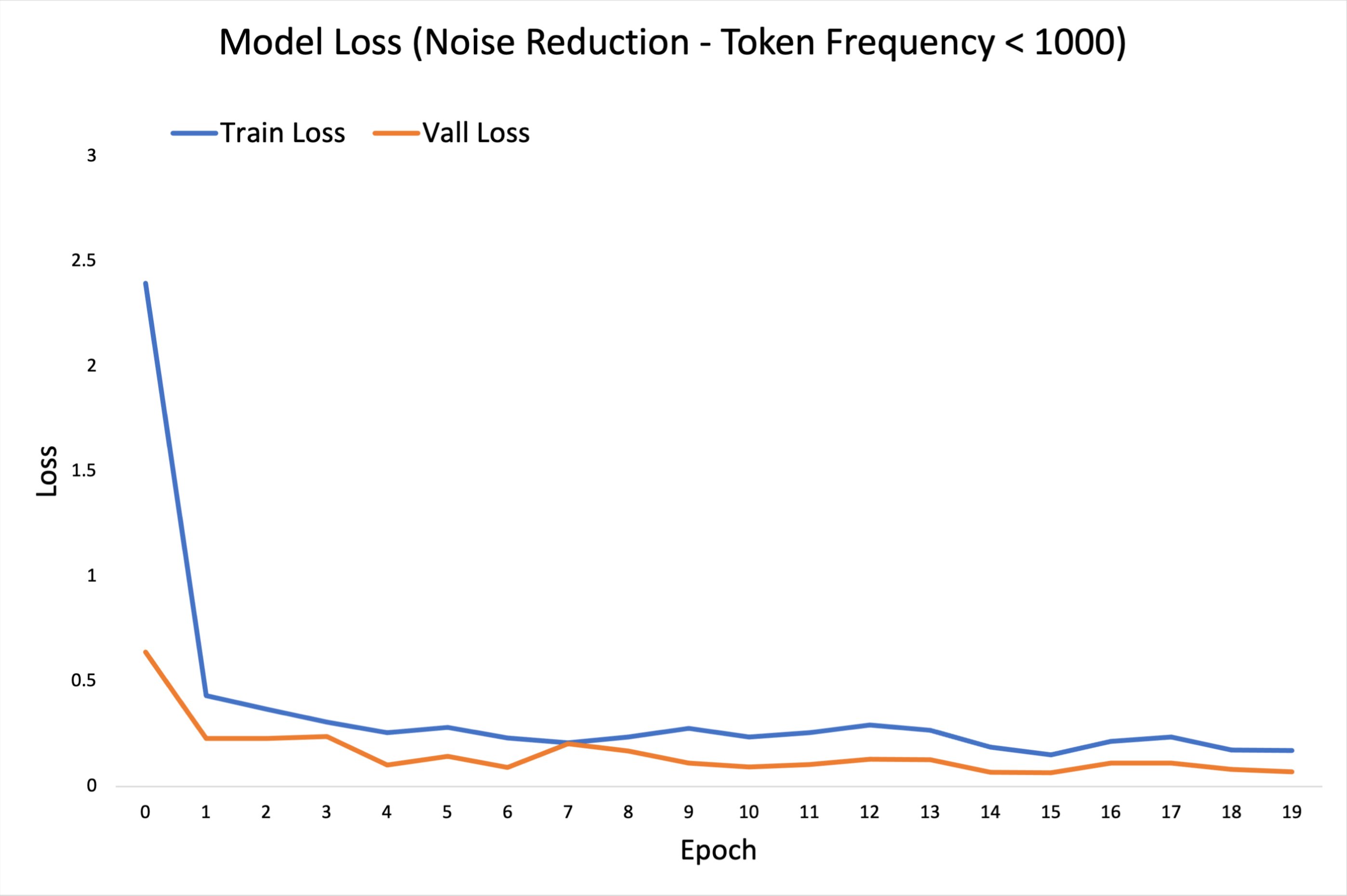
Sequence dictionary = 17,917
1740 are overlapping tokens



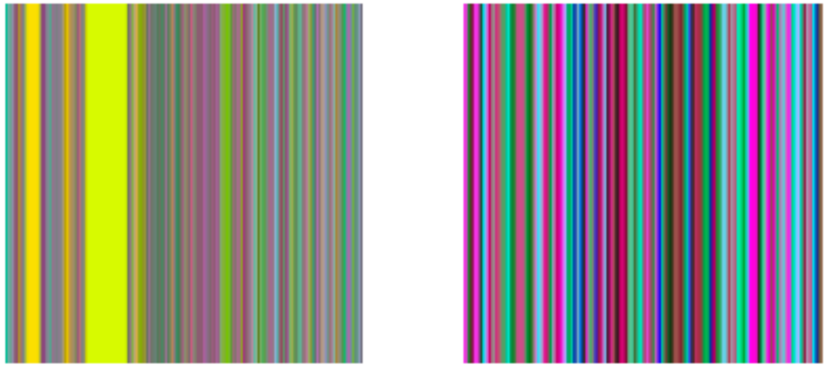
(a) Benign (b) Malware

Model Optimization - Recursive Feature Elimination

- Dimensionality Reduction with Token Frequency < 1000



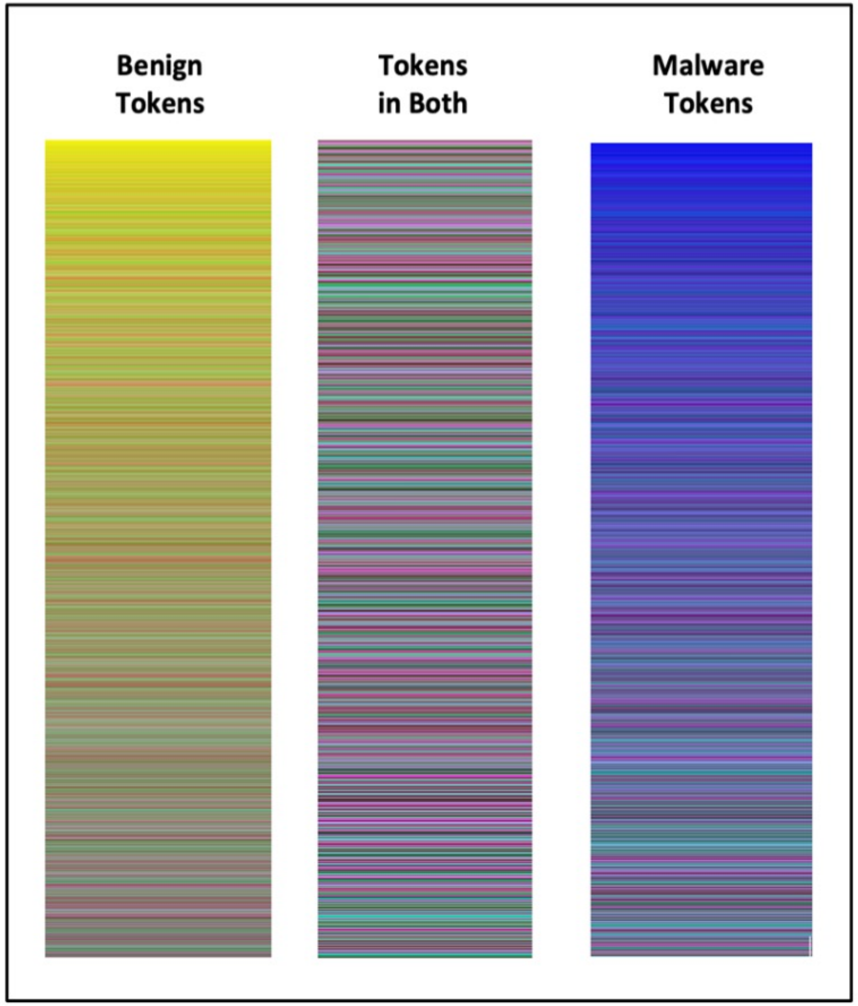
Sequence dictionary = 17,666
1489 are overlapping tokens



(a) Benign (b) Malware

RGB_Mem Dataset

- Dataset = 1411 memory images (823 malware and 588 benign)
- Size of sequence dictionary = 17,666 unique objects
  - Malware-only objects = 6986
  - Benign-only objects = 9191
  - Overlapping objects = 1489



RGB_Mem Evaluation

- Goals
  - T_01: Android malware detection based on known features - can the model correctly classify an app whose allocated objects were part of the sequence dictionary?
  - T_02 - Android malware detection based on unknown features - can the model correctly classify an app in which some of its uniquely allocated objects are not part of the sequence dictionary?
  - T_03 - Comparative analysis with state-of-the-earth Android malware classification techniques - how effective is the proposed approach compared to existing methods

T_01 - Android malware detection based on known features

- Maintain dimensionality reduction with object frequency < 1000
- Goal is to generate the sequence dictionary with all tokens from all dataset
  - Size of sequence dictionary = 17,666 unique objects
    - Malware-only objects = 6986
    - Benign-only objects = 9191
    - Overlapping objects = 1489
- Dataset = 1411 memory images (823 malware and 588 benign)

Table 1: Confusion Matrix for R0

[[97	3]
[4	70]]

Accuracy = 95.98%
F1-score = 95.24%
Precision = 95.89%
Recall rate = 94.59%.

T_02 - Android malware detection based on unknown features

- Maintain dimensionality reduction with object frequency < 1000
- Goal is to generate the sequence dictionary with only tokens from the training set
  - Size of sequence dictionary = 13,458 unique objects
    - Malware-only objects = 5750
    - Benign-only objects = 6400
    - Overlapping objects = 1308
- Dataset = 1411 memory images (823 malware and 588 benign)

Table 2: Confusion Matrix for R1

$$\begin{bmatrix} 86 & 14 \\ 13 & 61 \end{bmatrix}$$

Accuracy = 84.48%
F1-score = 81.88%
Precision = 81.33%
Recall rate = 82.43%.

T_03 - Comparative Analysis

Tool	Accuracy	F1-Score	Precision	Recall
DexRay	79.30	83.49	85.90	81.21
RGB_Mem Known features	95.98	95.23	95.89	94.49
RGB_Mem Unknown features	84.48	81.88	81.3	82.48

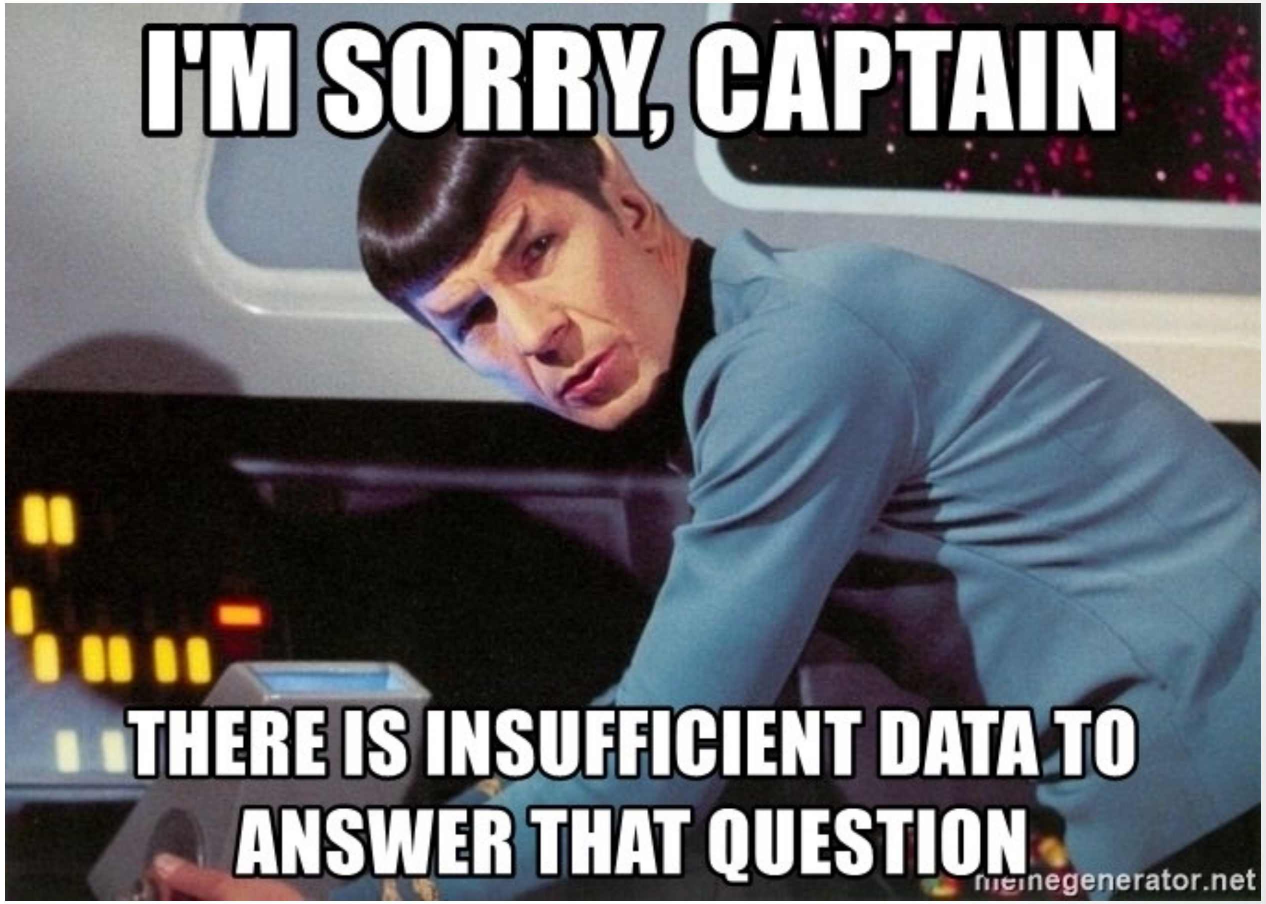
Table 3: Comparative Analysis with DexRay

Summary

- In-memory forensics artifacts can be leveraged for robust feature engineering
- These features can be used for an effective malware classification
- RGM_Mem could potentially aid incident response on Android involving malware

RGB_Mem Limitations and Future Work

- Size of dataset (impact on accuracy metric)– currently no repository for memory images
  - Increase dataset
- Learns better from known features
  - Continual reinforcement learning
- Features from object allocation relation (graph) instead of pattern



Acknowledgement

This work is supported by the U.S. National Science Foundation under the awards -

NSF-CRII-SaTC-1850054

THANK YOU! QUESTIONS!

aaligombe@lsu.edu
@aishagombe