GPT-like Pre-Training on Unlabeled System Logs for Malware Detection

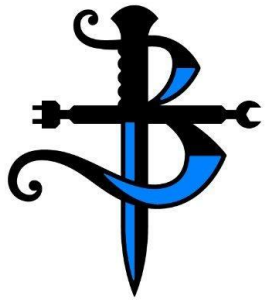
Dmitrijs Trizna

Sr. Security Software Engineer @ Microsoft
Doctoral Researcher @ UniGE

Luca Demetrio

Assistant Professor @ UniGE

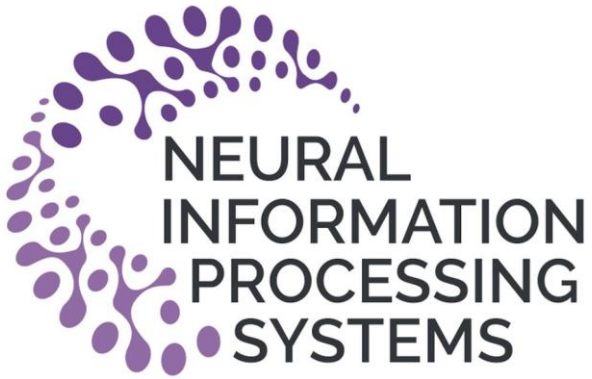
Brief Bio



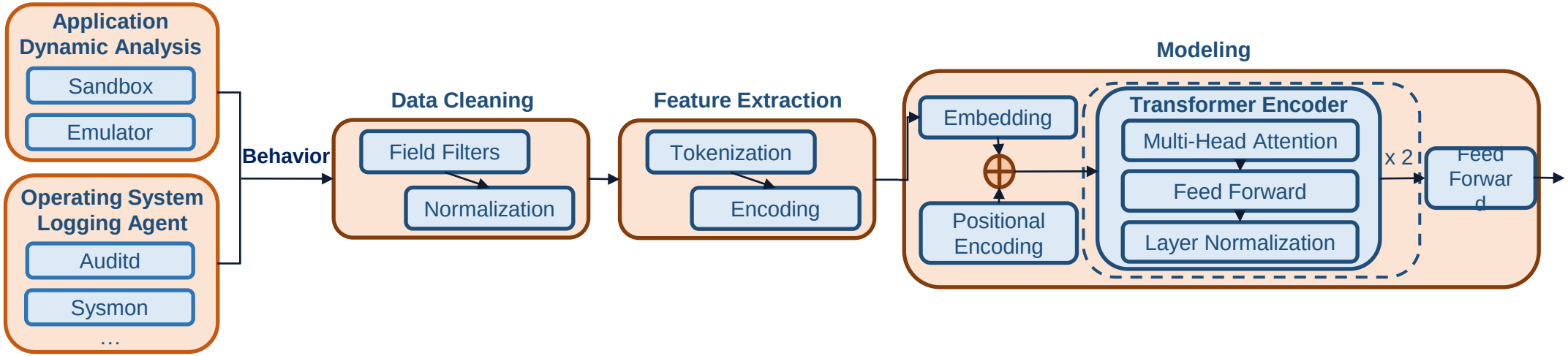
pralab / **secml_malware** Public

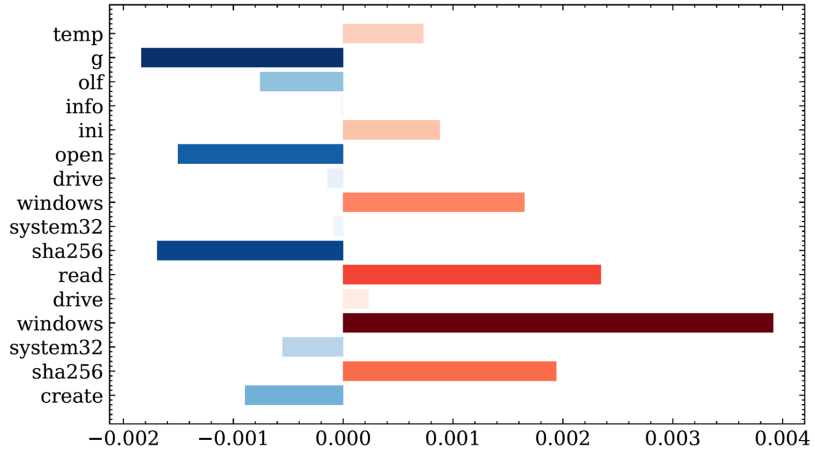
Create adversarial attacks against machine learning Windows malware detectors

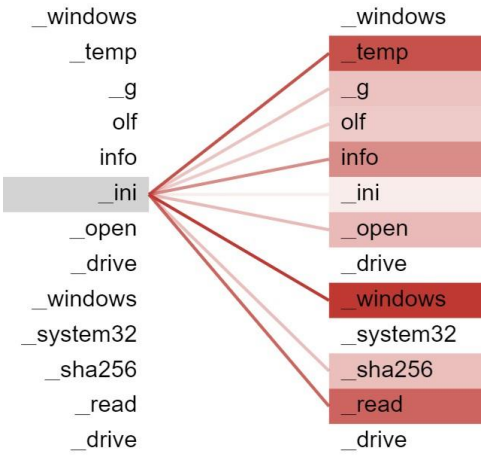
secml-malware.readthedocs.io/



Take-home message:
Transformers can model (and understand) system logs!



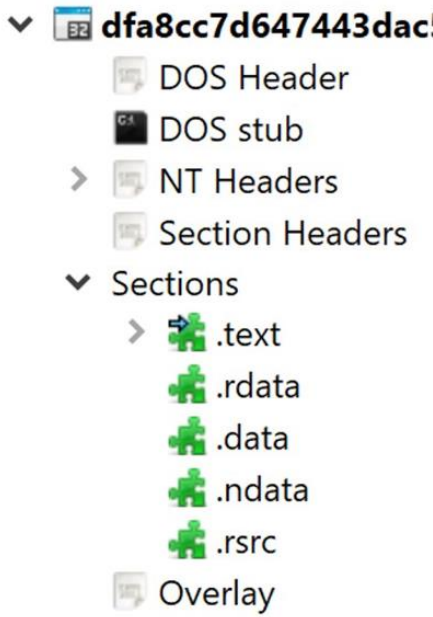




Roadmap

- Malware detection with machine learning
- Main advances in AI that lead to GPT success
- Behavior malware modeling with Transformers
- Preliminary Results and Explainability

(Static) Malware Classification with ML Models



Static features (import, exports, sections…)

Supervised ML

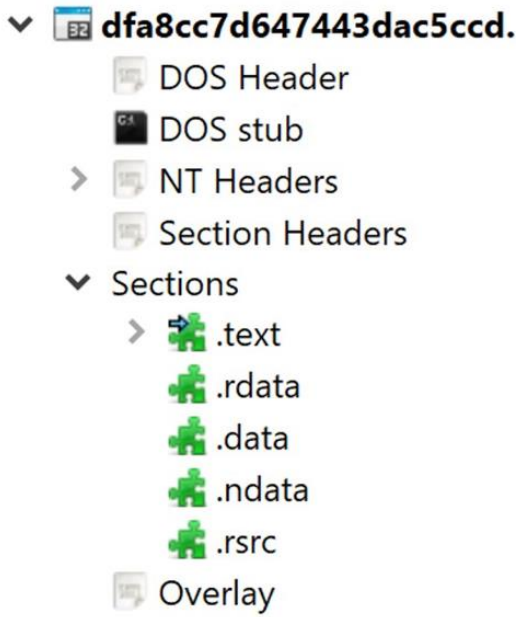
p(malicious) = 0.951

Human or "greedy" labels:

Malicious

Benign

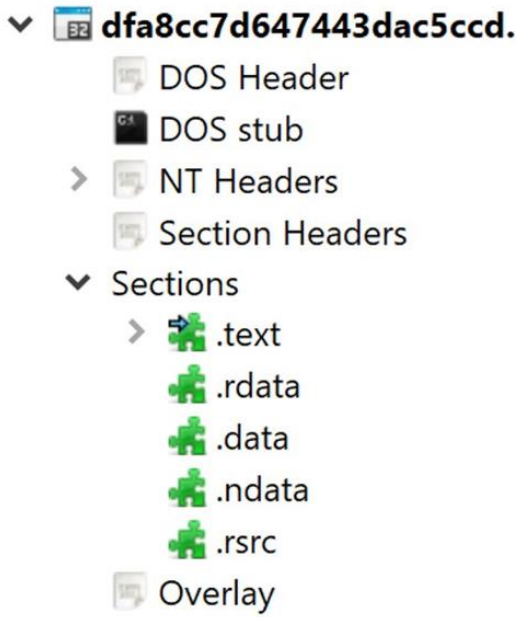
(Static) Malware Classification with ML Models




```
71  class ByteEntropyHistogram(FeatureType):
72      ''' 2d byte/entropy histogram based loosely on (Saxe and Berlin, 2015).
73      This roughly approximates the joint probability of byte value and local entropy.
74  125  class SectionInfo(FeatureType):
75  126      ''' Information about section names, sizes and entropy.  Uses hashing trick
76  127      to summarize all this section info into a feature vector.
77  128      '''
78  195   class ImportsInfo(FeatureType):
    129  196       ''' Information about imported libraries and functions from the
    130  197       import address table.  Note that the total number of imported
    131  198       functions is contained in GeneralFileInfo.
         199       '''
```

[*] Anderson, H. S., & Roth, P. (2018). **EMBER**: An Open Dataset for Training Static PE Malware Machine Learning Models. https://doi.org/10.48550/arXiv.1804.04637

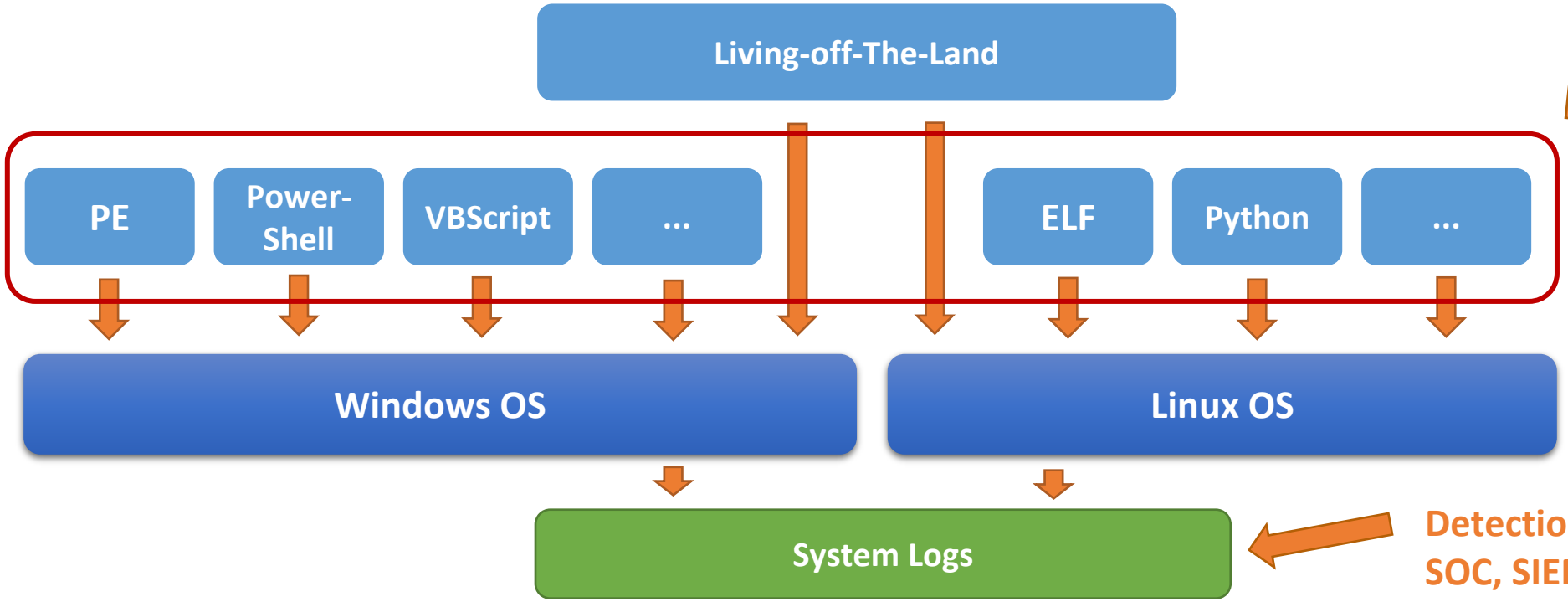
Limitations of Static Analysis



Meaningful but not complete
PE structure is informative, but it provides limited information on the real functionality of the analysed sample (and there are plenty of ways to execute code not contained in PE files)

Why is it malicious?
Extracted features are not human-readable (think about histograms), decisions are difficult to explain

Easy to restructure at will
Plenty of tools for packing and obfuscating to avoid detection
(also, someone said minimal adversarial attacks?)



AV/EDR vendors & DFIR only

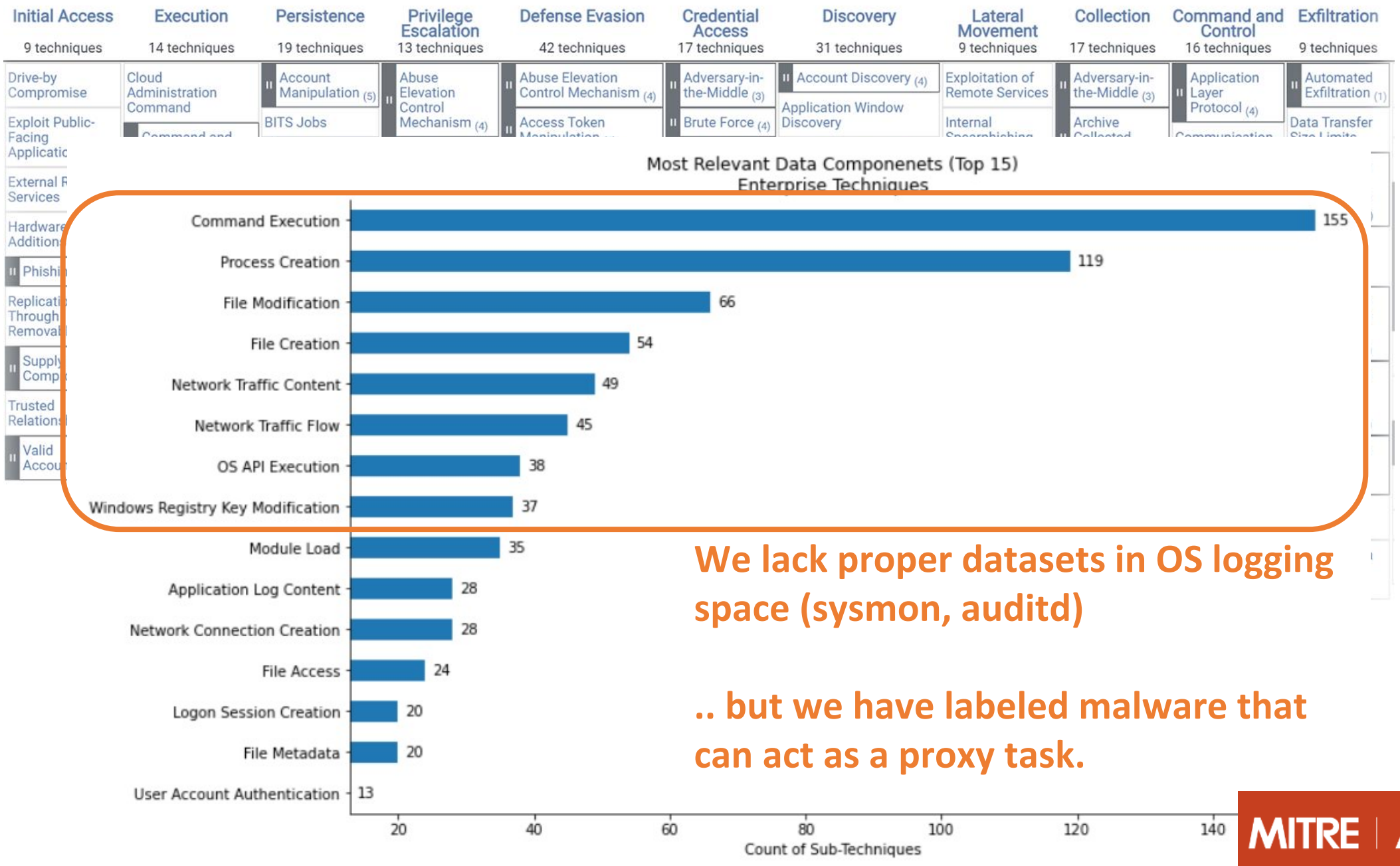
Detection Engineering SOC, SIEM


```
{
  "UserName": "NT AUTHORITY\\NETWORK SERVICE",
  "ParentProcessId": "10504",
  "ProcessId": "2780",
  "ProcessLuid": "{824a0b1e-b9f6-481c-af09-ceb0bf6414f9}",
  "CommandLine": "\"\"C:\\ProgramData\\Microsoft\\Windows Defender\\platform\\4.18.2207.
  "ParentProcess": "MpCmdRun.exe",
  "Process": "MpCmdRun.exe",
  "SubjectLogonId": "996",
  "SessionId": "0",
  "ProcessStartTime": "2022-10-26T15:40:33.745514Z",
  "ParentProcessLuid": "{67c36c2b-aa1a-4e1d-b4bf-1a11a9af156f}",
  "Sha256Hash": "749FA9AC3B7DD44938983FD04E01870181E88D5BB65CB1322BF4867A7C23B453"
}
```


current folder.

p://7-zip.org/


```
{
    "process": {
        "parent": {
            "process": {
                "pid": "23463",
                "title": "/bin/bash/<appcontainer>",
                "name": "bash",
                "executable": "/bin/bash",
                "ppid": "23459",
                "start_time": "2022-06-13T07:37:19.1800000Z"
            }
        },
```



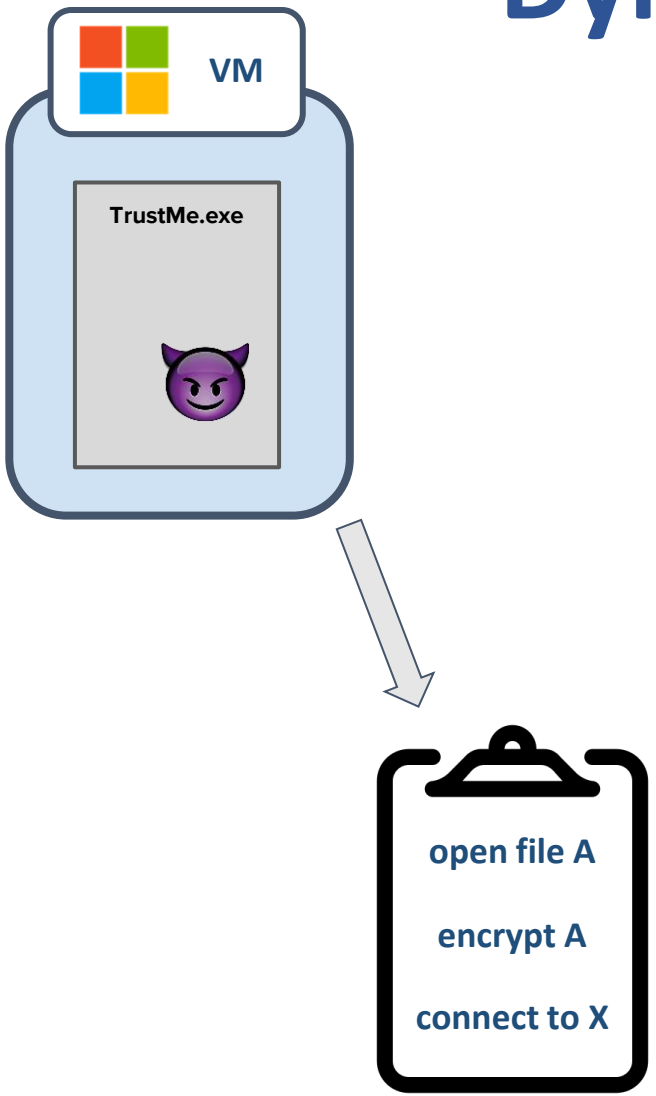
We lack proper datasets in OS logging space (sysmon, auditd)

.. but we have labeled malware that can act as a proxy task.

MITRE | ATT&CK®

[*] https://github.com/mitre-attack/attack-datasources

Dynamic analysis on the rescue



Chain of events
Run program inside protected isolated environment, take note of every observable action of the program

Human-readable reports
The analysis outputs a textual report that specifies the timeline of all the triggered events

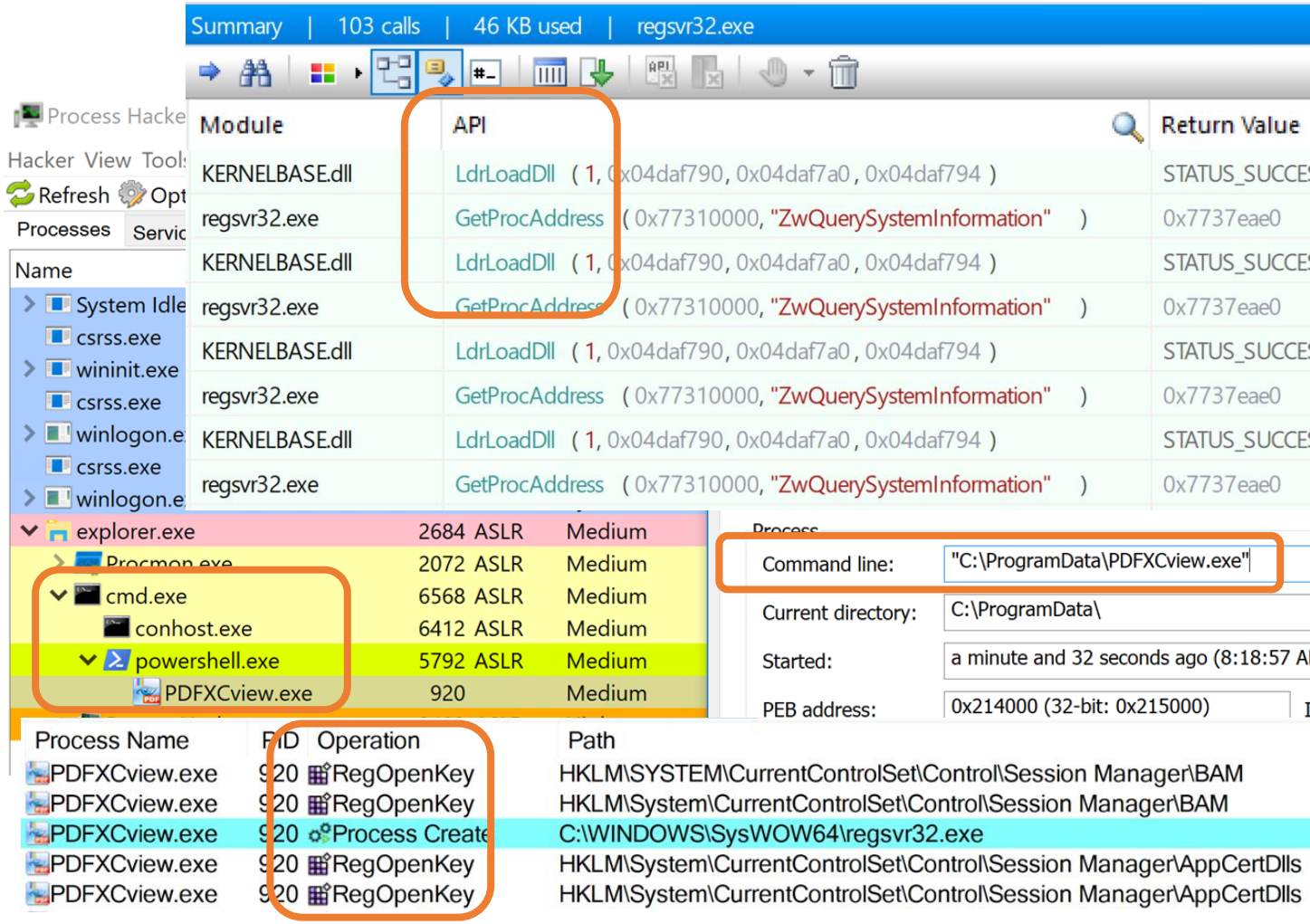
Circumventing obfuscation
Even if samples are packed or obfuscated, at some point the functionality will be manifested through interactions with the underlying OS

Behavioral Properties

- dfa8cc7d647443dac5ccd.
  - DOS Header
  - DOS stub
  - NT Headers
  - Section Headers
  - Sections
    - .text
    - .rdata
    - .data
    - .ndata
    - .rsrc
  - Overlay

→ **Sandbox** →

→ **Emulator** →



Speakeasy Dataset

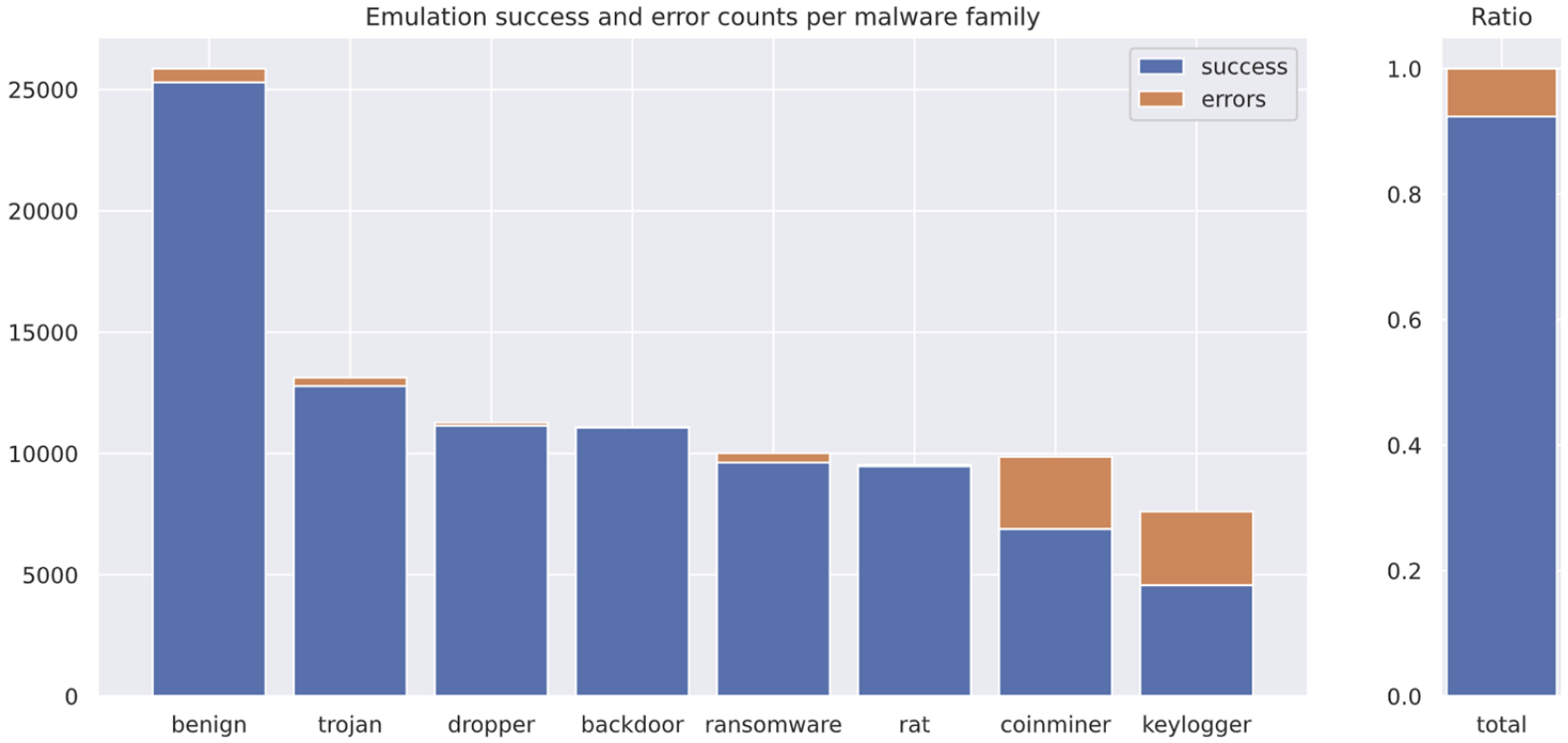
Emulation at its core
Speakeasy is a great product from Mandiant with ongoing R&D

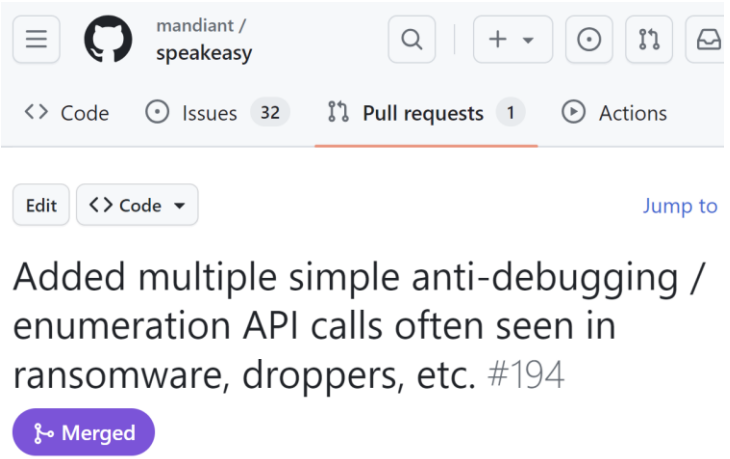
Cheap, fast, and precise
Emulation is fast thanks to **Unicorn** and **QEMU**, that leverage native implementations.
Also, results are very close to real execution, with low error rate

Perfect tool for creating a dataset of behavioral traces!

Our data – behavior reports of:

~70k malware samples over 7 malware types
~25k clean samples





https://www.kaggle.com/datasets/dmitrijstrizna/quo-vadis-malware-emulation

How data looks like with Speakeasy


```
"traffic": [
    {
        "server": "www.iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com",
        "proto": "tcp.http",
        "port": 80,
        "headers": 0
    }
],
"registry_access": [
    {
        "event": "open_key",
        "path": "HKEY_LOCAL_MACHINE\\Software\\Microsoft\\IE4"
    }
],
"file_access": [
    {
        "event": "create",
        "path": "C:\\Windows\\temp\\golfinfo.ini",
        "open_flags": [
            "OPEN_EXISTING"
        ],
        "access_flags": [
            "GENERIC_READ"
        ]
    },
    {
        "event": "write",
        "path": "C:\\Windows\\temp\\golfinfo.ini",
        "data": "sqyyr83/zv/H/9H/yv/L/9H/zP/O/9H/zv/J/8r/////////////",
        "size": 512,
        "buffer": "0x9000"
    },
```

```
"apis": [
    {
        "pc": "0x41ec8f",
        "api_name": "KERNEL32.GetSystemTimeAsFileTime",
        "args": [
            "0x1211fd8"
        ],
        "ret_val": null
    },
    {
        "pc": "0x41ec9b",
        "api_name": "KERNEL32.GetCurrentProcessId",
        "args": [],
        "ret_val": "0xe8bdc"
    },
```


[*] https://github.com/mandiant/speakeasy

Human readable != Easy to model


```
{
   "pc": "0x411c3c",
   "api_name": "KERNEL32.GetProcAddress",
   "args": [
      "0x77000000",
      "EncodePointer"
   ],
   "ret_val": "0xfeee0004"
},
{
   "pc": "0x411c5a",
   "api_name": "kernel32.EncodePointer",
   "args": [
      "0xfeee0003"
   ],
   "ret_val": "0xfeee0004"
},
{
   "pc": "0x41cb1a",
   "api_name": "KERNEL32.InitializeCriticalSectionAndSpinCount",
   "args": [
      "0x4e4568",
      "0xfa0"
   ],
   "ret_val": "0x1"
},
{
   "pc": "0x41cb1a",
   "api_name": "KERNEL32.InitializeCriticalSectionAndSpinCount",
   "args": [
      "0x4e4580",
      "0xfa0"
   ],
   "ret_val": "0x1"
},
```


Complex “language” to learn

Tight structure, many “words”, mixing data types like strings, int, and pointers

Data is still noisy

Activities can be numerous, and real behavior is hidden among them

Not everything is necessary

Plenty of “words” that are not really useful to machine learning models

Does progress in AI provide techniques for cyber-security telemetry modeling?

Let me tell you a story of how GPT works…

AI advancement Nr.1:

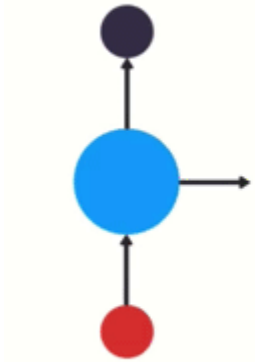
Neural Network Architectures

AI in 2010s: Convolutional NNs and Recurrent NNs

1989
(LeCun et al.)

1997: LSTM
(Hochreiter and Schmidhuber)

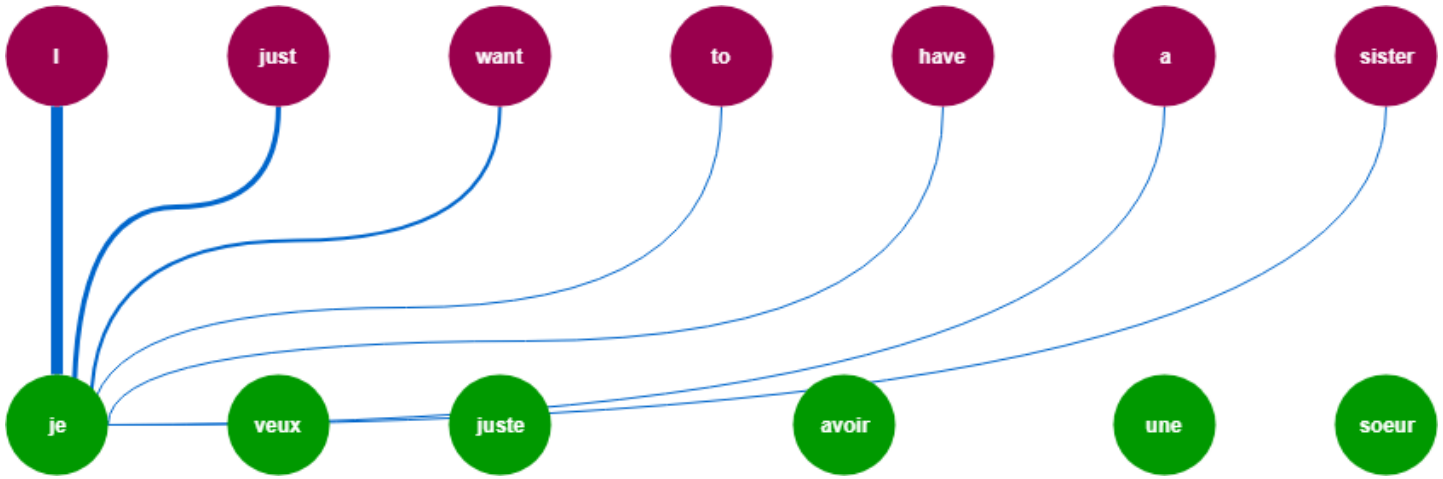
2010-2017:
CNNs and RNNs dominate AI



AI 2020s:
Attention and Transformers

2017: Google
"Attention is All You Need"

Releases "Transformer" model



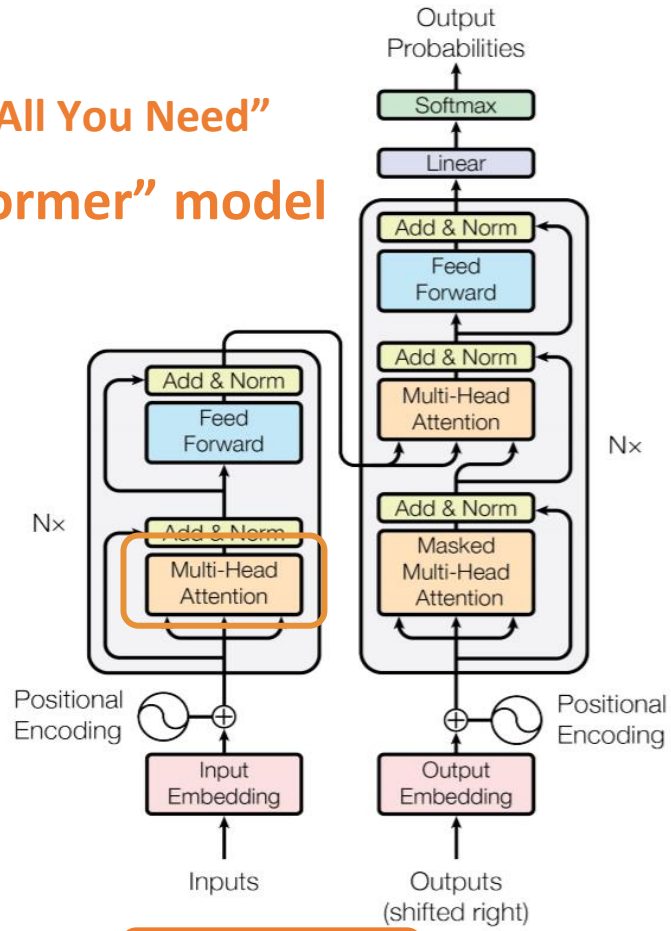


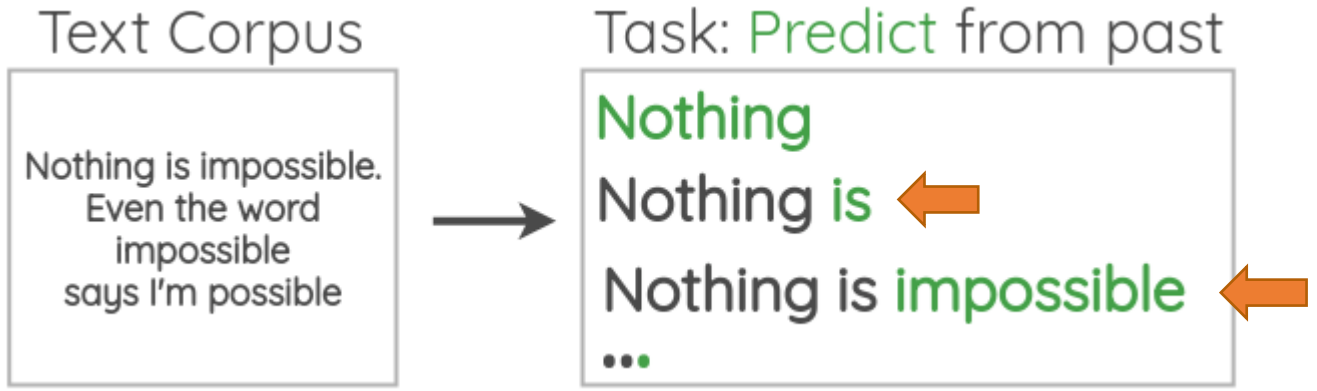
Figure 1: The Transformer - model architecture.

AI advancement Nr.2:

Self-Supervised Language Modeling

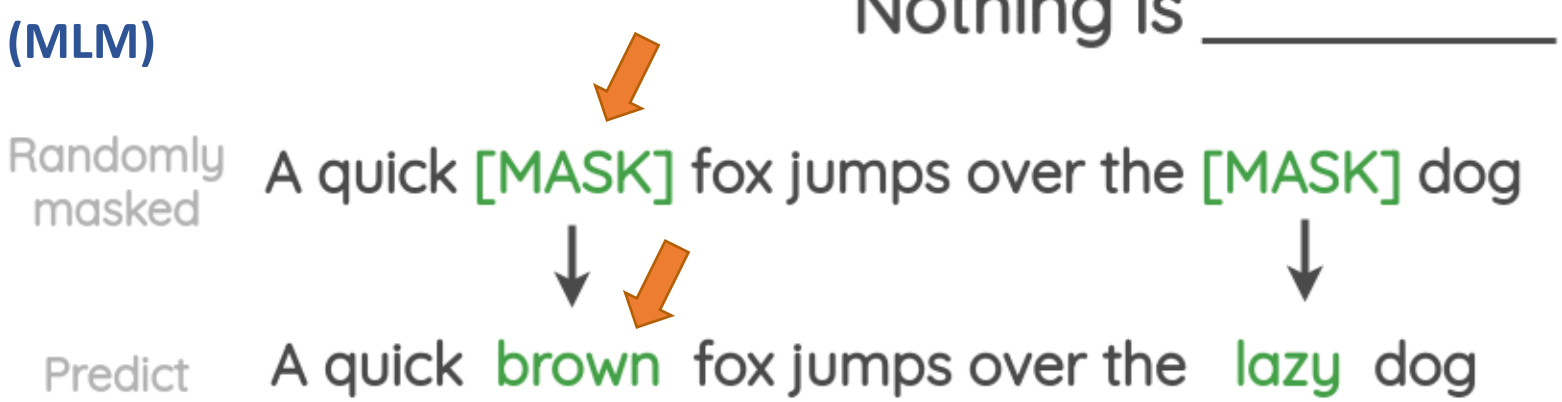
GPT-like – Autoregressive

- Predict what comes next based on context:

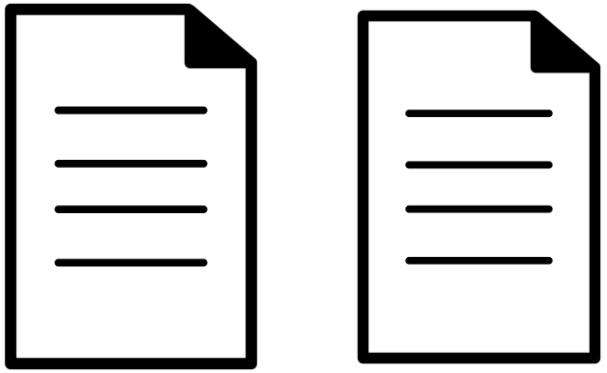


BERT-like – Masked Language Model (MLM)

- Predict what tokens are masked:



[img source] https://amitness.com/2020/05/self-supervised-learning-nlp/



Unlabeled corpus:
books, wiki, reddit

Labeled data:
- Question-Answering
- User recommendations
- Malware & benign-ware
- …

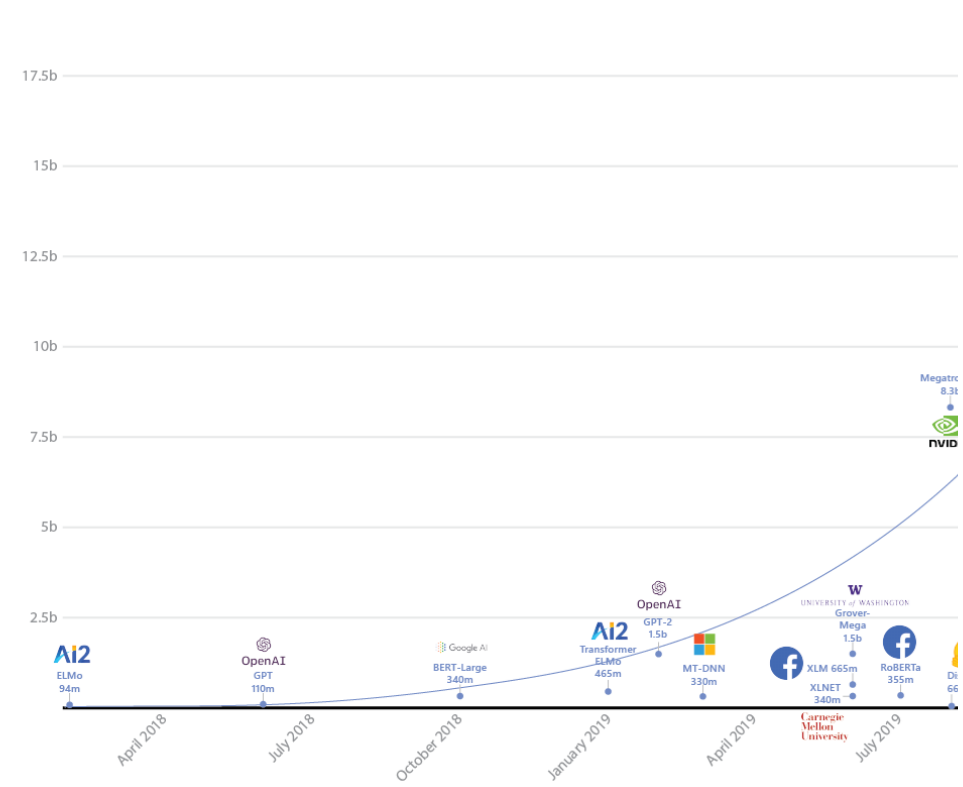
Pre-Training
via
Self-supervised
learning

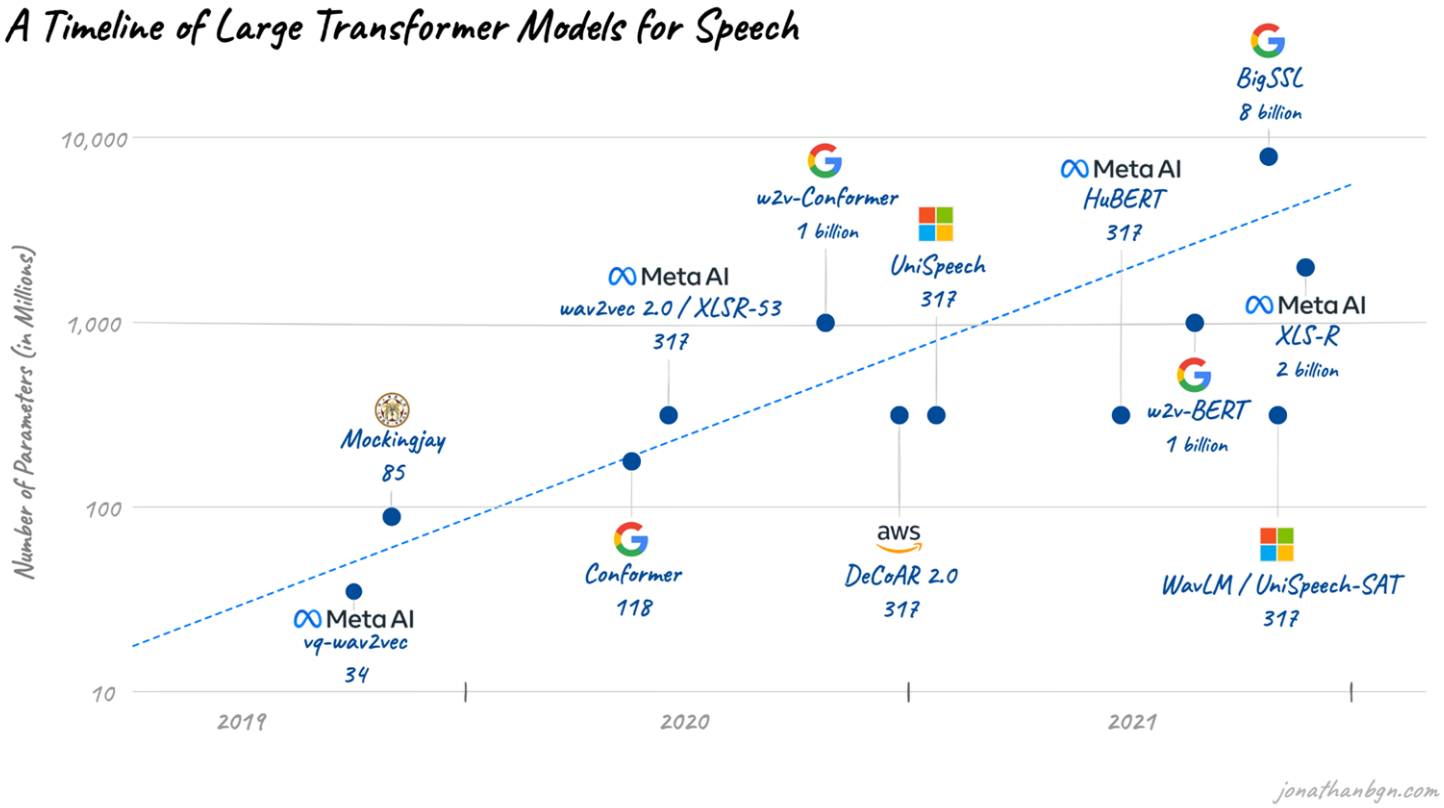
Fine-Tuning
via
Supervised learning

Application,
e.g.
p(malware) = ?

AI 2020s: Attention and Transformers

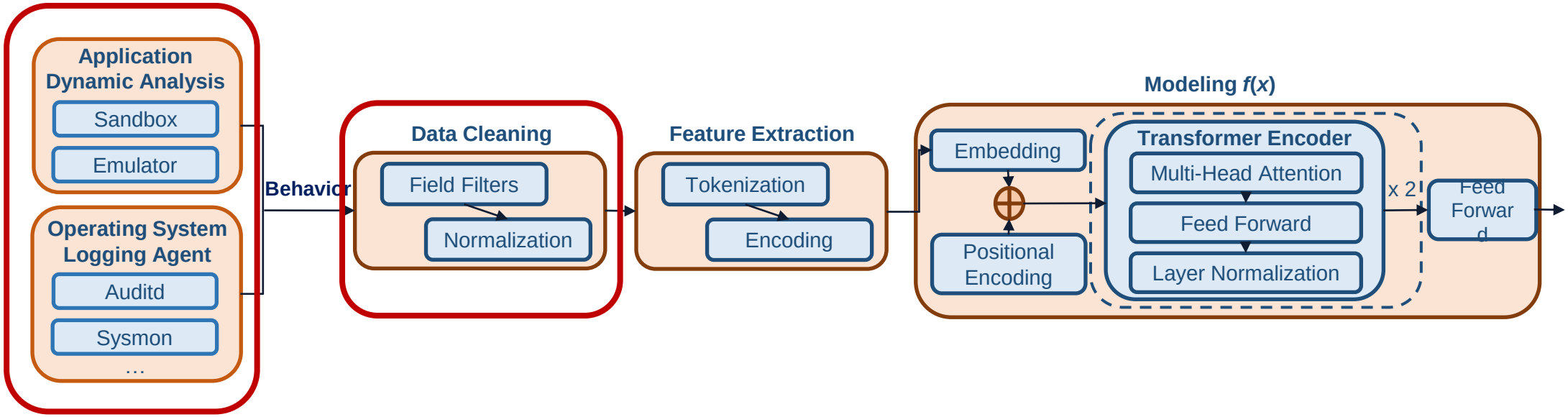
2018-2023: Scaling up + Engineering





How this applies to security?

Behavioral Log Modeling with Transformer

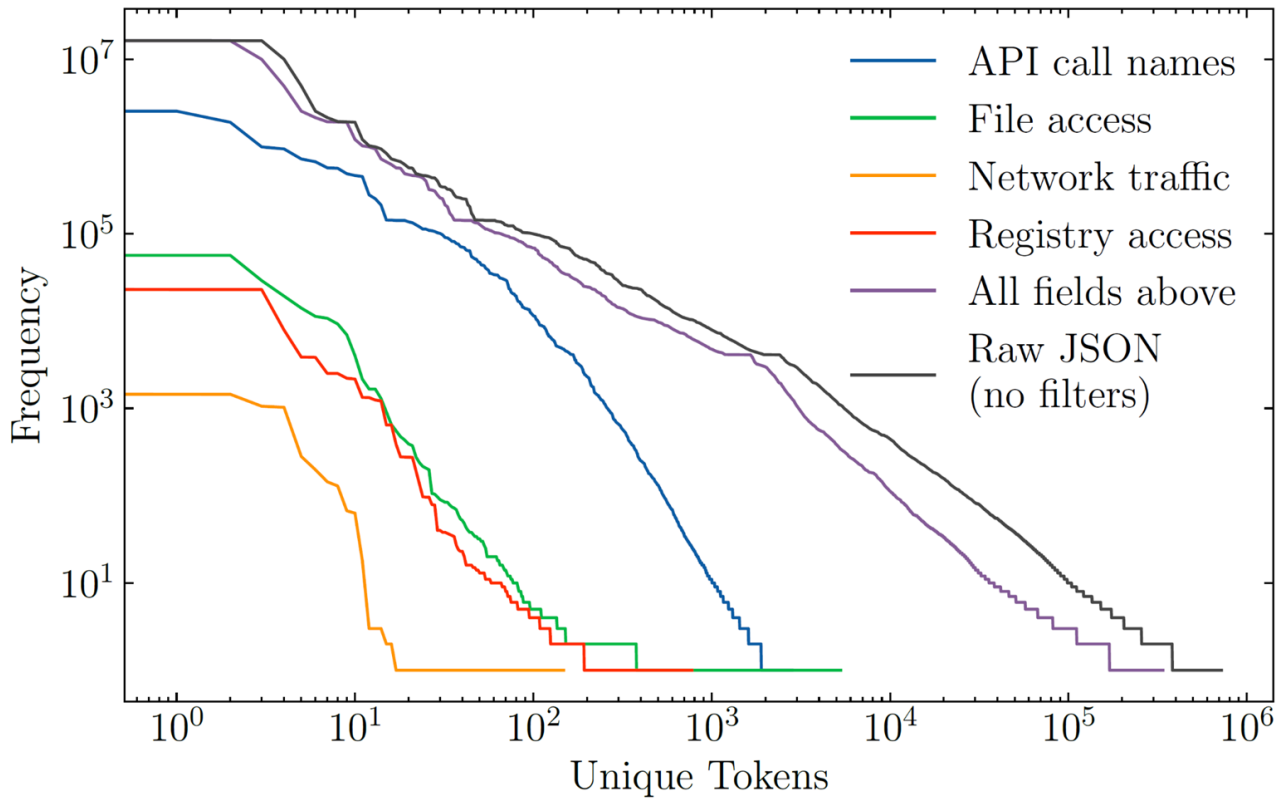


Data Cleaning – Filters


```
{
    "ep_type": "module_entry",
    "start_addr": "0x409a16",
    "ep_args": [
        "0x4020",
        "0x4030",
        "0x4040",
        "0x4050"
    ],
    "apihash": "fb8c06ac28f07f903a1ea7a2450f5e862
    "apis": [
        {
            "pc": "0x409a49",
            "api_name": "MSVCRT.__set_app_type",
            "args": [
                "0x2"
            ],
            "ret_val": null
```


Too many fields:

1. lengthy sequences – too much data for model

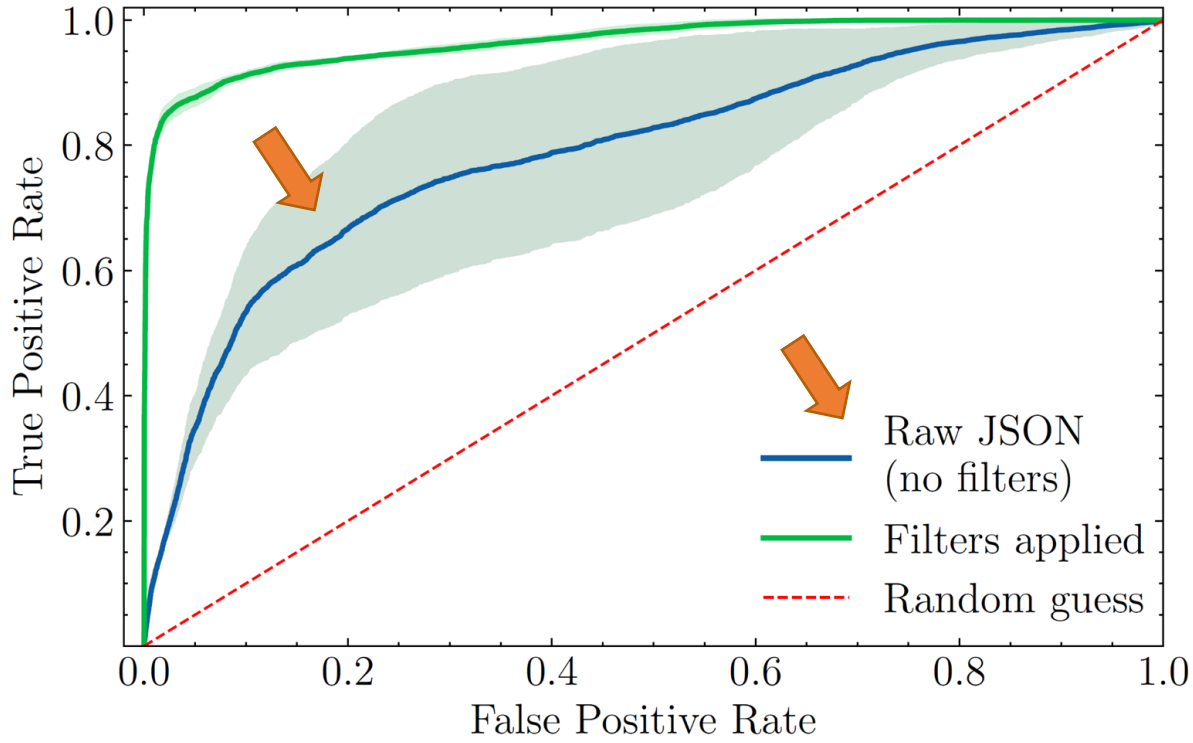


Data Cleaning – Filters


```
{
    "ep_type": "module_entry",
    "start_addr": "0x409a16",
    "ep_args": [
        "0x4020",
        "0x4030",
        "0x4040",
        "0x4050"
    ],
    "apihash": "fb8c06ac28f07f903a1ea7a2450f5e862
    "apis": [
        {
            "pc": "0x409a49",
            "api_name": "MSVCRT.__set_app_type",
            "args": [
                "0x2"
            ],
            "ret_val": null
```


Too many fields:

1. lengthy sequences – too much data for model
2. non-essential values – irrelevant for task (overfitting)



Data Cleaning – Filters


```
{
    "ep_type": "module_entry",
    "start_addr": "0x409a16",
    "ep_args": [
        "0x4020",
        "0x4030",
        "0x4040",
        "0x4050"
    ],
    "apihash": "fb8c06ac28f07f903a1ea7a245
    "apis": [
        {
            "pc": "0x409a49",
            "api_name": "MSVCRT.__set_app_
            "args": [
                "0x2"
            ],
            "ret_val": null
```


Too many fields:

1. lengthy sequences – too much data for model
2. non-essential values – irrelevant for task (overfitting)

```
40  SPEAKEASY_RECORD_FIELDS = [
41      'file_access.event',
42      'file_access.path',
43      'network_events.traffic.server',
44      'network_events.traffic.port',
45      'registry_access.event',
46      'registry_access.path',
47      'apis.api_name',
48      'apis.args',
49      'apis.ret_val',
50  ]
```

We preserve only:

- API calls
- File access
- Network events
- Registry access

Data cleaning – Normalization

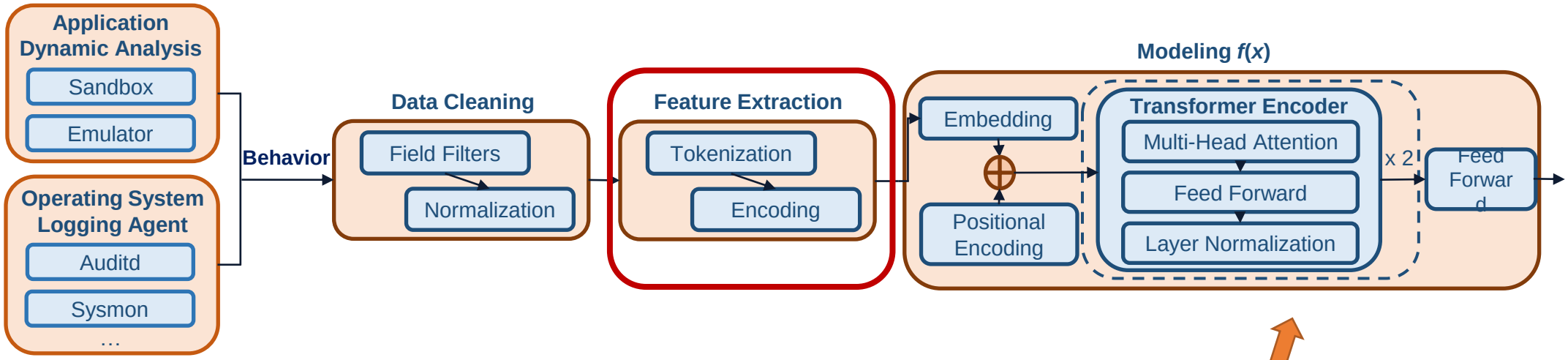
```
"api_name": "KERNEL32.GetModuleFileNameA",
"args": [
    "0x400000",
    "C:\\Windows\\system32\\45eda9efc948db2506eed18ec2b9387e4117d87933df49a9e1bf8a3852ce9685.dat",
    "0x104"
],
"ret_val": "0x58"
```

"<drive>\windows\system32\<sha256>.dat"

```
"network_events": {
    "dns": [
        {
            "query": "4wf4791276.qicp.vip",
            "response": "185.212.33.4"
        },
        {
            "query": "work.contoso.com",
            "response": "127.0.0.1"
        }
```

```
"dns": [
    {
        "query": "4wf4791276.qicp.vip",
        "response": "<public_ip>"
    },
    {
        "query": "<domain>",
        "response": "<lopback_ip>"
    }
```

Behavioral Log Modeling with Transformer



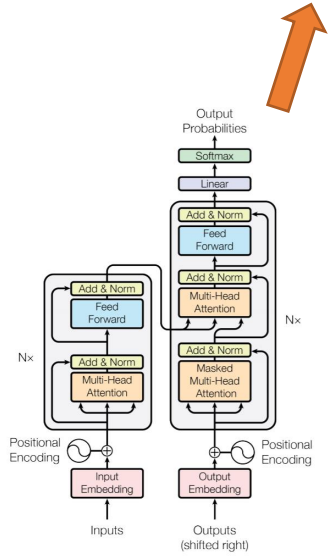


Figure 1: The Transformer - model architecture.

Tokenization & Encoding

Input text is first split into pieces. Can be characters, word, "tokens":

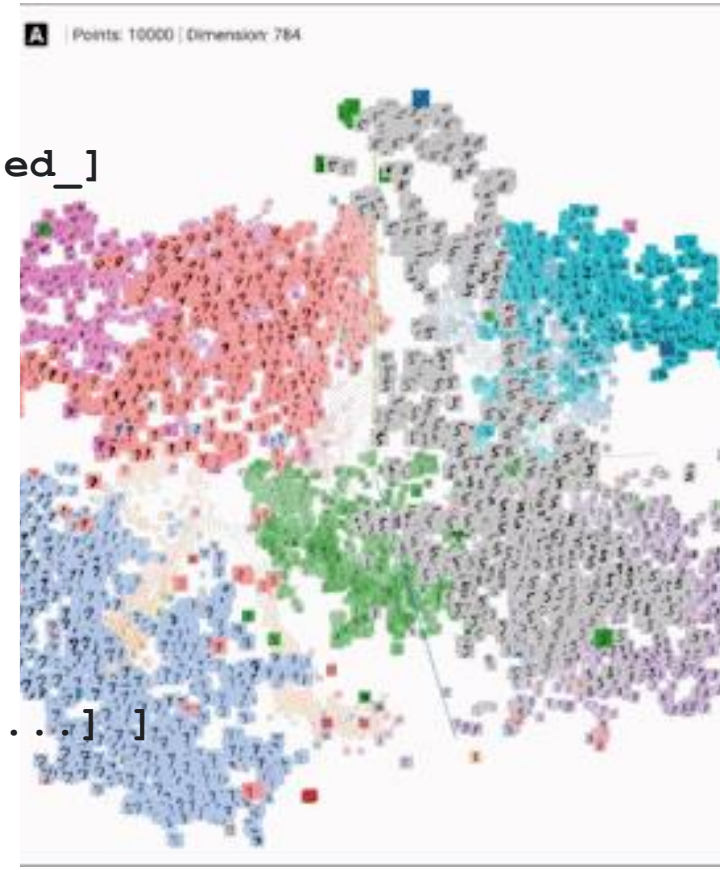
```
"The detective investigated" -> [The_] [detective_] [invest] [igat] [ed_]
```

Tokens are indices into the "vocabulary":

```
[The_] [detective_] [invest] [igat] [ed_] -> [3 721 68 1337 42]
```

Each *vocab* entry corresponds to a learned multi-dimensional vector.

```
[3 721 68 1337 42] -> [ [0.123, -5.234, ...], [...], [...], [...], [...] ]
```



[img source] https://ai.googleblog.com/2016/12/open-sourcing-embedding-projector-tool.html

Tokenization & Encoding

```
{
"file_access":
[
    {
        "event":"open",
        "path":"<drive>\\windows\\system32\\<sha256>.dat"
    },
    {
        "event":"read",
        "path":"<drive>\\windows\\system32\\<sha256>.dat"
    }
],
"network_events.traffic":
[
    {
        "server":"www.iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com",
        "port":80
    }
],
"apis":
[
    {
        "api_name":"MSVCRT.__set_app_type",
        "args":[
            "0x2"
        ],
        "ret_val":null
    },
    {
        "api_name":"MSVCRT.__p__fmode",
```

```
[
"file_access",
"open",
"<drive>",
"windows",
"system32",
"<sha256>.dat",
"read",
"<drive>",
"windows",
"system32",
"<sha256>.dat",
"network_events.traffic",
"www.iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com",
"80",
"apis",
"msvcrt.__set_app_type",
"0x2",
"null",
"msvcrt.__p__fmode",
```

Whitespace tokens:

```
['kernel32.getprocaddress', '0x1000', 'null', '0x77000000',
'kernel32.isbadreadptr', '0xfa0', '0xfeee0001',
'kernel32.tlsgetvalue', 'kernel32.initializecriticalsectionex',
'kernel32.flsgetvalue', 'kernel32.heapalloc', '0x100',
'kernel32.leavecriticalsection', 'kernel32.entercriticalsection',
'kernel32.getlasterror', '0x46f0', 'kernel32.setlasterror',
'0x45f0', 'kernel32.encodepointer', '0x46d0']
```

Byte-Pair Encoding (BPE) tokens:

```
['er', '00', 'ne', '32', '32.', '_k', 'l32.', 'erne', '_kerne',
'_kernel32.', '_0x1', '_0x0', 'et', 'ad', 'al', 'ti', 'get',
'000', '_0x7', 're', 'ec', '_0x4', '_0xf', 'in', 'oc', '_0x77',
'iti', 'ee', '_0xfee', '_0xfeee', 'ss', 'pr', 'one', 'tr',
'0000', 'proc', 'ter', 'getproc', 'sec', 'dre', 'cal', 'on',
'secti', 'addre', 'address', 'getprocaddress', 'riti', 'ritical',
'riticalsecti', 'le']
```

Behavioral Log Modeling with Transformer

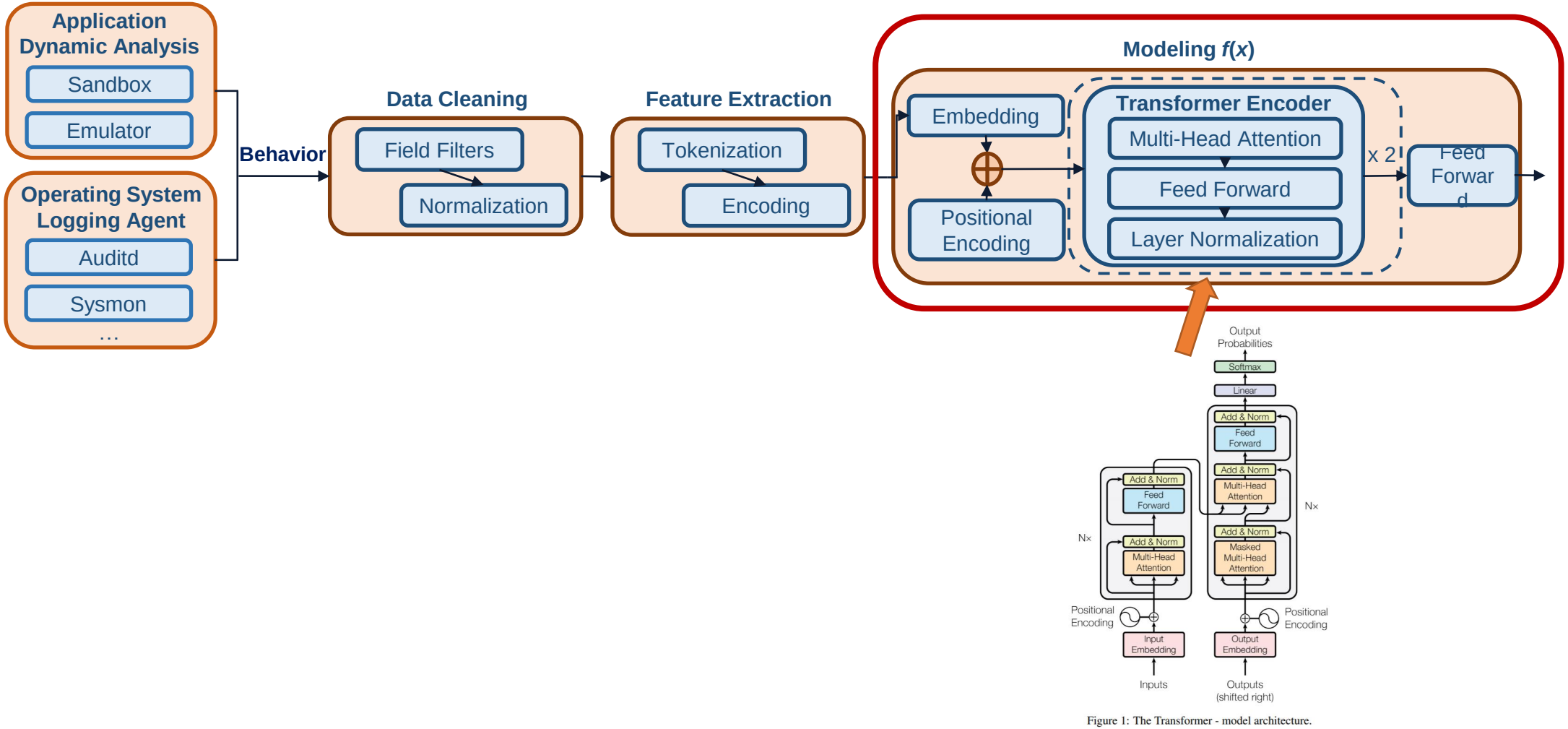
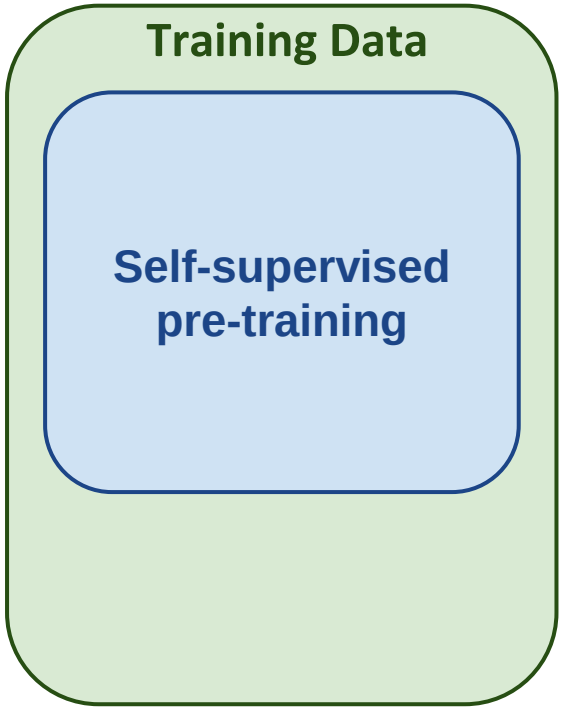


Figure 1: The Transformer - model architecture.

Evaluation – How Well Self-Supervised Pre-Training Works?

1. Choose large part of training dataset to act as unsupervised corpus

2. Pre-train model using Masked Language Model (MLM)



Jan 2022
~70k samples

Apr 2022
25k samples

Masked Language Modeling (MLM)

```
[
    "file_access",
    "create",
    "<drive>",
    "windows",
    "temp",
    "golfinfo.ini",
    "<mask>",
    "<drive>",
    "windows",
    "temp",
    "<mask>",
    "create",
    "<drive>",
    "windows",
    "temp",
```

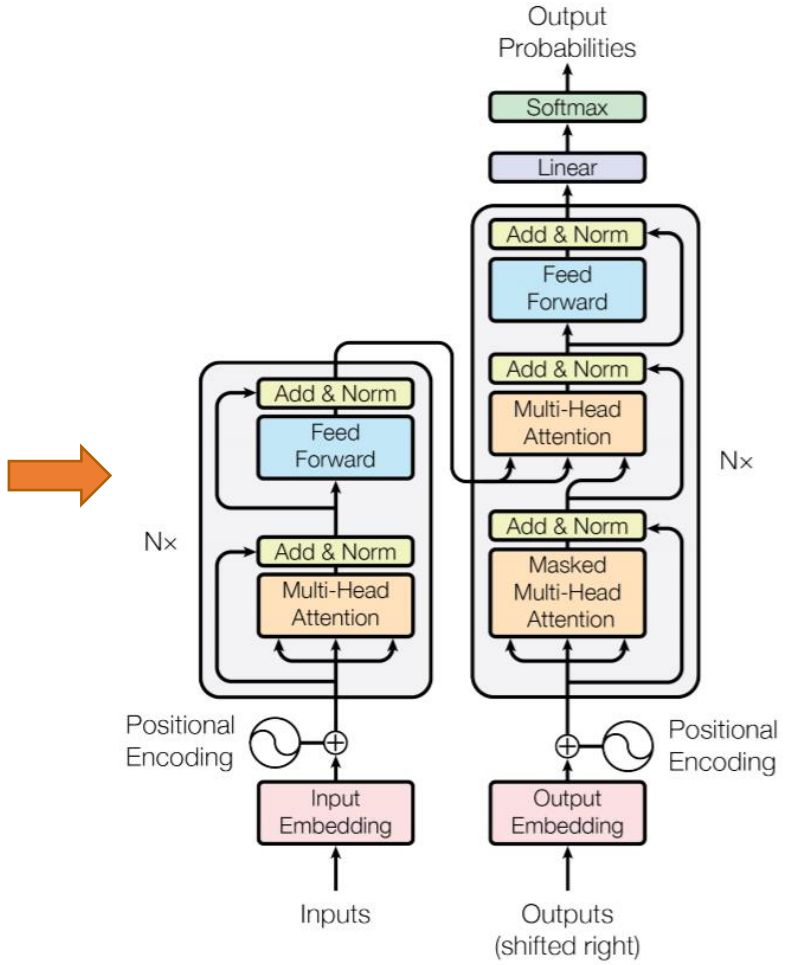


Figure 1: The Transformer - model architecture.

Which tokens are masked?

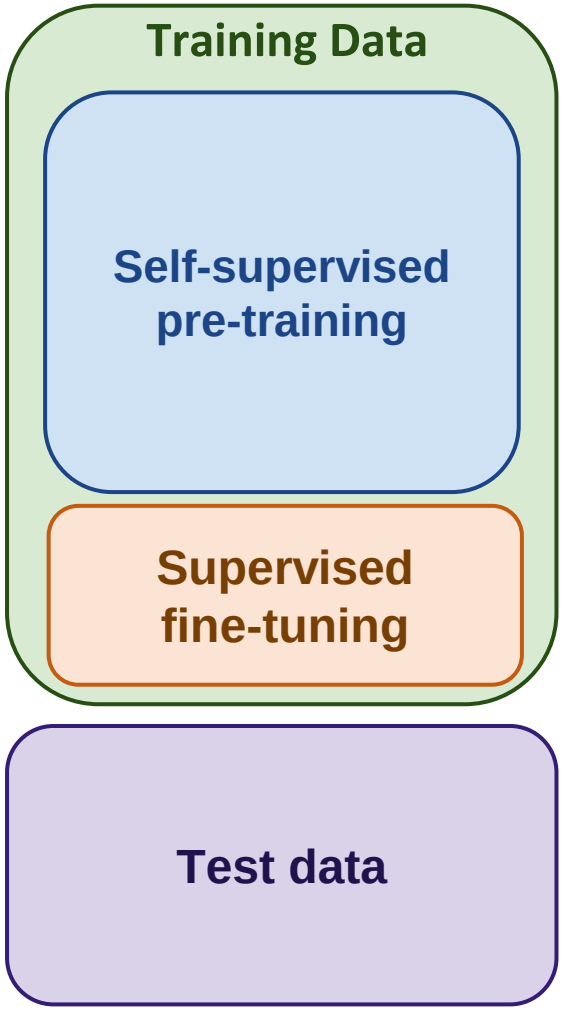
Evaluation – How Well Self-Supervised Pre-Training Works?

1. Choose large part of training dataset to act as unsupervised corpus

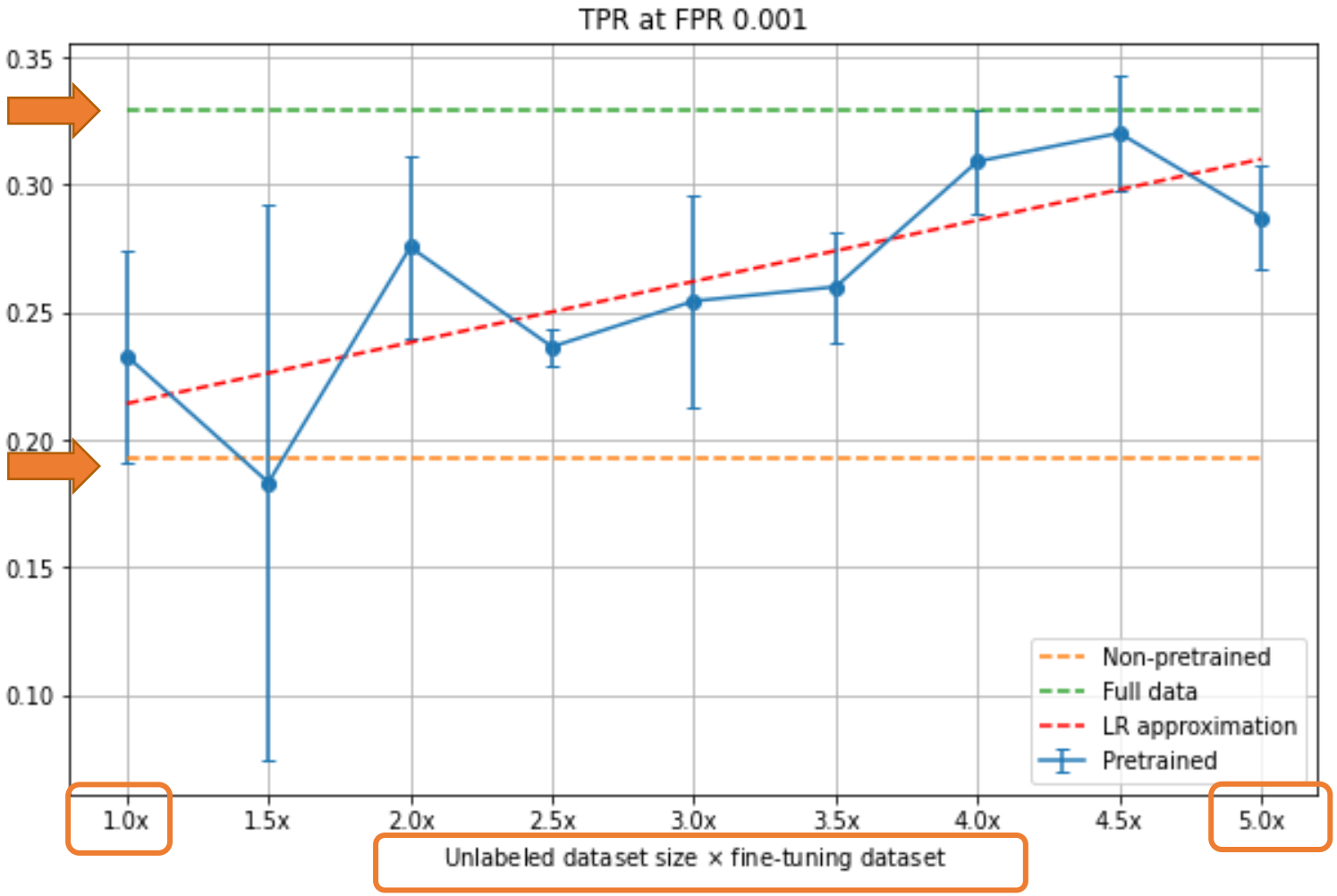
2. Pre-train model using Masked Language Model (MLM)

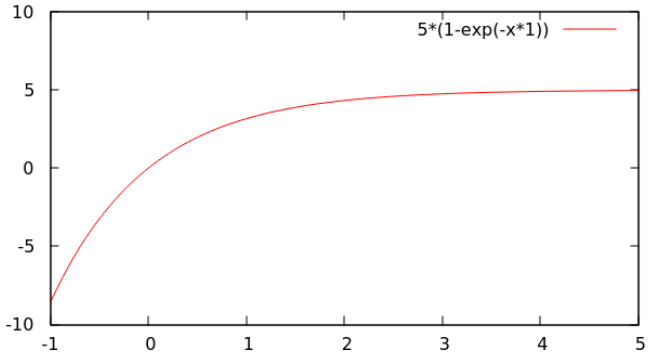
3. Fine-tune model on small supervised dataset

4. Evaluate how well it performs on test data compared to model with no pre-training



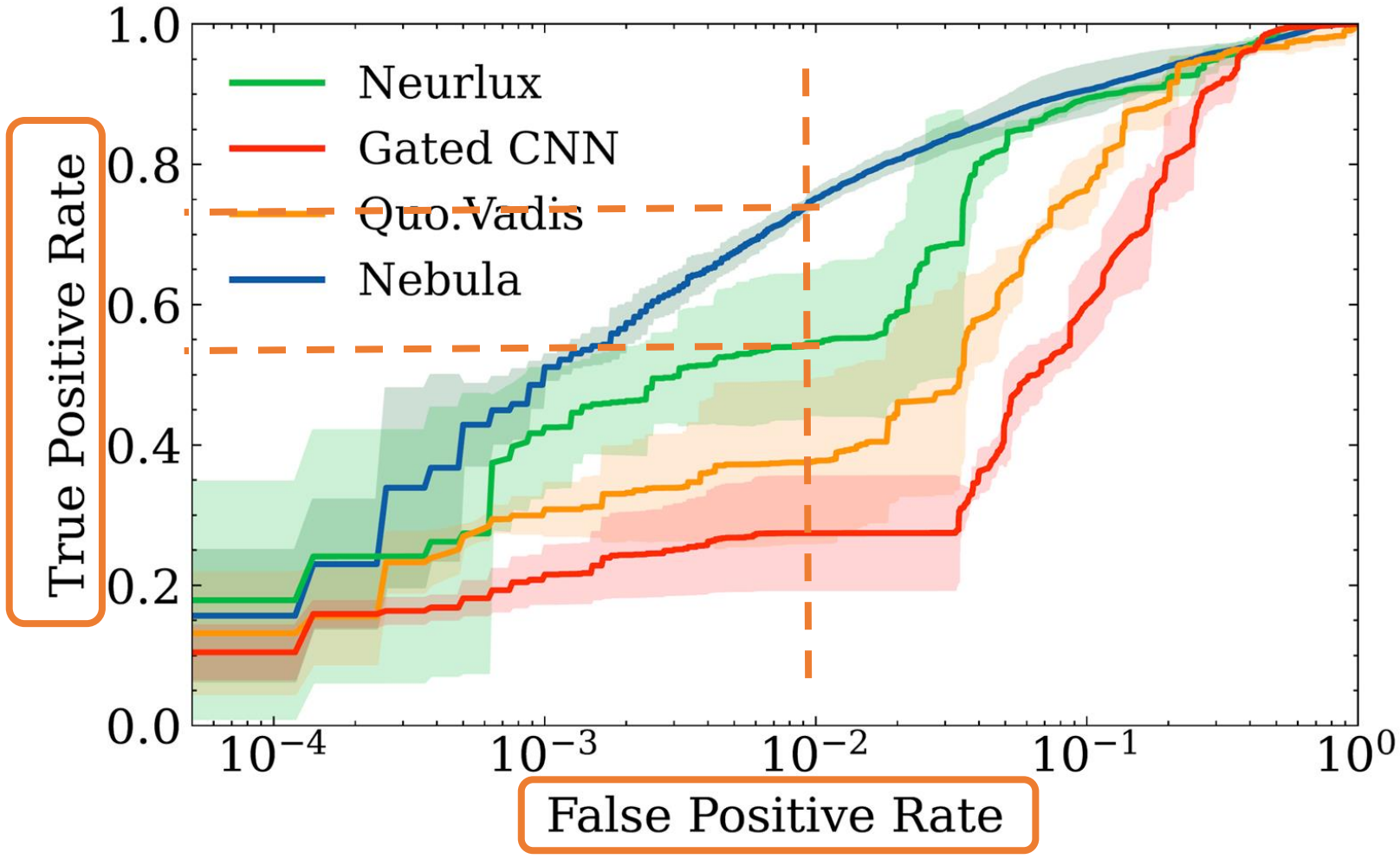
Preliminary results on pre-training:



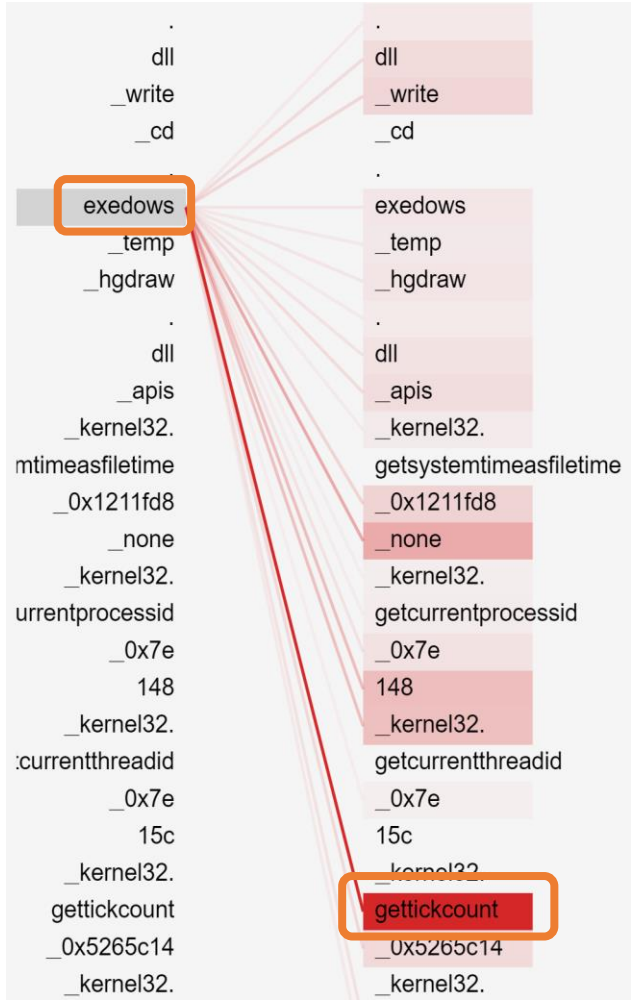


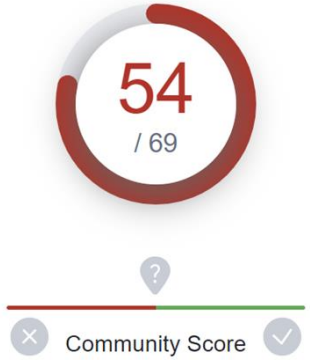
The way to utilize Tera-Bytes of unlabeled system logs!?

Comparison with Existing Techniques:



Explainability (1/3):
Transformer Attention Show What It Learns





54 security vendors and no sandboxes flagged this file as malicious

0009064322cdc719a82317553b805cbbc64230a9212d3b8aad1ea1b78d3bf10a

nyxuy.exe

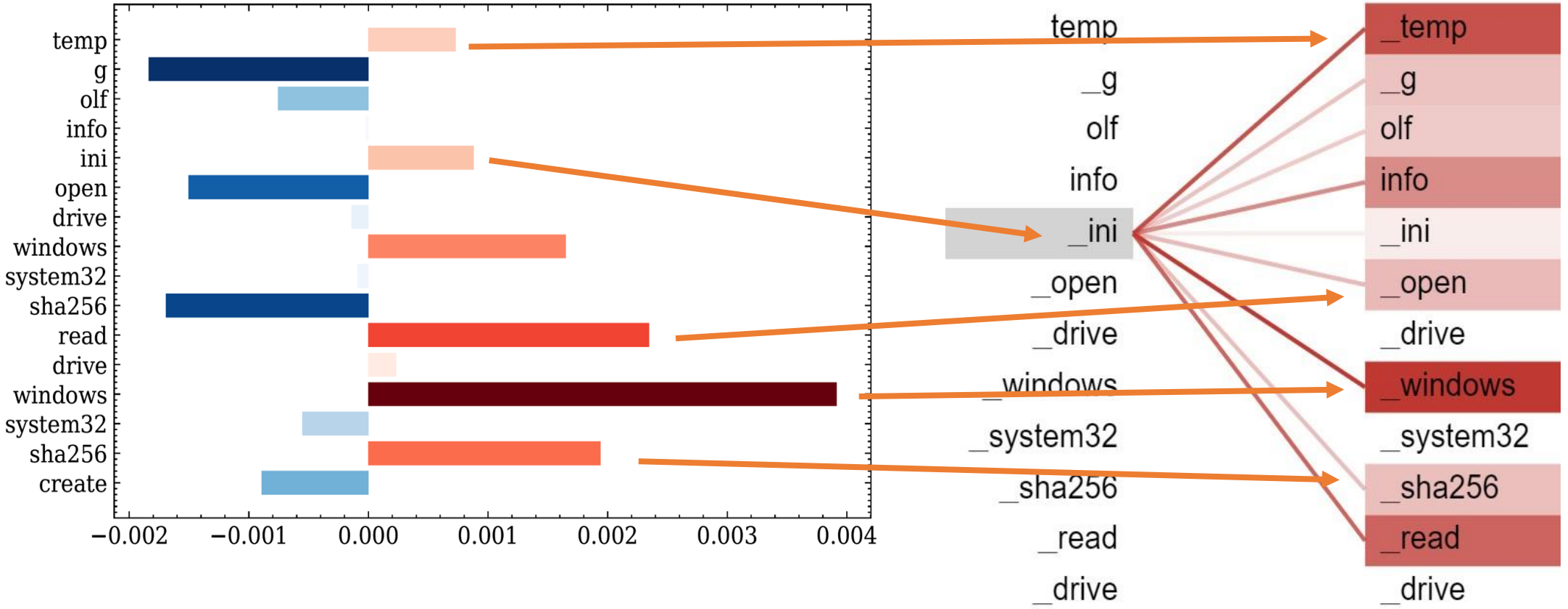
peexe ov

```
bool IsDebugged(DWORD dwNativeElapsed)
{
    DWORD dwStart = GetTickCount();
    // ... some work
    return (GetTickCount() - dwStart) > dwNativeElapsed;
}
```

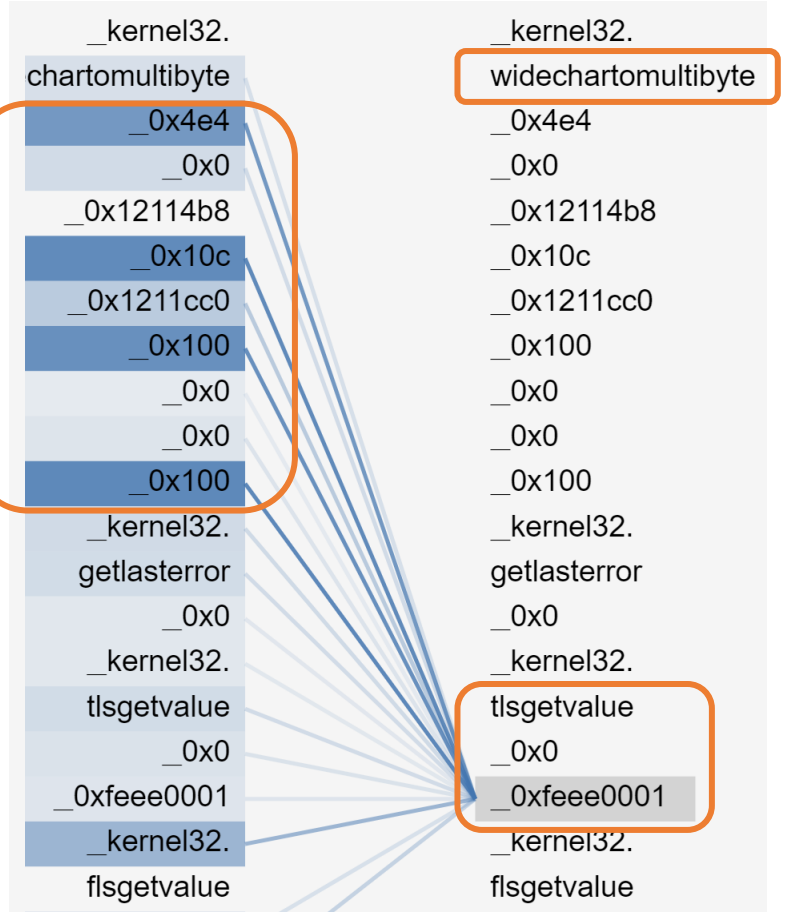
```
rule:
  meta:
    name: check for time delay via
    namespace: anti-analysis/anti-debugging/debugger-detection
    author: michael.hunhoff@fireeye.com
    scope: function
    mbc:
      - Anti-Behavioral Analysis::Debugger Detection::Timing/Delay Check GetTickCount [B0001.032]
    examples:
      - Practical Malware Analysis Lab 16-03.exe_:0x4013d0
  features:
    - and:
      - count(api(kernel32.GetTickCount)): 2 or more
```

Explainability (2/3):
Explanation Consistent Between Various Techniques

SHapley Additive exPlanations (SHAP)



Explainability (3/3):
Can Yield Indicators-of-Compromise (IoC) ?



Syntax

C++ Copy

```
int WideCharToMultiByte(
  [in]            UINT                                CodePage,
  [in]            DWORD                               dwFlags,
  [in]            _In_NLS_string_(cchWideChar)LPCWCH  lpWideCharStr,
  [in]            int                                 cchWideChar,
  [out, optional] LPSTR                               lpMultiByteStr,
  [in]            int                                 cbMultiByte,
  [in, optional]  LPCCH                               lpDefaultChar,
  [out, optional] LPBOOL                              lpUsedDefaultChar
);
```

Summary

- Realistic exploitation detection considers wide scope of attack vectors through log analysis
- Industry lacks efficient AI modeling of this data
- Transformer architectures rivals Convolutional Neural Networks in this domain:
  - Global vs Local semantics
- Unsupervised pre-training in this domain has huge potential
  - *Now*: Security industry relies on manual / supervised methods

Thank you for your attention!

Medium