Malware Bytes

James Holt, Edward Raff

Over the past decade, the Advanced Computing Systems (ACS) office of the Laboratory for Physical Sciences (LPS) has explored a wide variety of approaches to analyzing and classifying binary files. One of the most significant findings, and a consistent theme of our work, is the surprising effectiveness of algorithms that look directly at the bytes of files. Previously, most approaches to applying machine learning (ML) to binary files first extracted some set of features, and used these features as input to an ML model. In contrast, we have built ML models that learn directly from the file itself, ingesting and learning from the file as a sequence of bytes.

One advantage of this is that it requires no domain-specific or file-type-specific knowledge, so the exact same approach and set of tools can be used on multiple file types. Because of this, we have been able to use the same tools, with no alteration or customization, to build classifiers for Windows executables, Linux executables, PDFs, Microsoft Office files, rich text files (RTFs), and others.

[Photo credit: iStock.com/Quardia]

AUTHENTICATED U.S. GOVERNMENT INFORMATION GPO

Historically, cybersecurity tasks concerning binary files, such as malware detection, analysis, triage, reverse engineering, disassembly, decompilation, etc., have required deep expertise and a low-level understanding of the structure of files and function of operating systems. The tools that have evolved to perform or assist with these tasks also require deep expertise to create, to use, and to keep up to date. The complexity of these files and systems, and therefore the degree of expertise needed to work with them, continues to grow, and this creates challenges in training people, in keeping the software tools current, and in computational cost of running the tools.

This challenge is not going away. Tools which embody deep knowledge and understanding are certainly still necessary. But, we can now complement them with no-domain-knowledge tools that can be rapidly adapted to new problems, providing agility, speed, and scale. In some cases they can reduce the need for more expensive high-domain-knowledge tools. Along these lines, ACS has developed a portfolio of byte-based algorithms, including compression-based file similarity metrics, a highly efficient n-gram algorithm, convolutional neural network-based file classification, n-gram and logistic regression (LR)-based file classification, and automatic signature generation.

In this article we will share some highs, lows, and points of interest from the journey our research has taken over the last decade, exploring different forms of ML for file classification, testing existing ML techniques, innovating new ones, creating new algorithms that yielded 100-fold performance gains, curating datasets, and delivering new capabilities. Along the way, we published many of our advances in academic papers. Often these were accompanied by code, allowing others to leverage and benefit from our new algorithms. As we go through these efforts, we will highlight the published papers for the reader who would like to go deeper.

The false negative/false positive trade-off

Antivirus (AV) products have been deployed and in use for decades. They were originally targeted at home consumers, which imposed early design constraints. The AV product needed to run fast enough to not get in the way (too much). If the AV slowed the system down so that it was no longer interactive, making users wait too long to load or use applications, they would forgo the product entirely. Essentially, there is a friction ceiling that AVs need to avoid to keep users happy.

This friction ceiling applies for both speed and accuracy. Imagine the AV was more likely to predict a benign file as malicious (i.e., false-positive or FP) than a malicious file as benign (i.e., false-negative or FN). This would mean every new benign application would have a larger chance of causing an alert to the user. This would result in interruption of their work, remediation, and whitelisting—that is, friction. Again, the user would forgo the AV. To prevent this, AVs are designed around minimizing false positives. These are reasonable design constraints, motivated by real-life use, and still relevant today. For this reason, these ideas and goals are heavily embedded in the current industry and academic cultures.

Unfortunately, systems designed this way do not adequately address the breadth of issues we see in the government. Some missions require that every file be inspected, analyzed, and a determination of risk made. In such a case, the FP/FN trade-off is not relevant, and the coarse "yes/no" returned by many AVs does not allow us to adjust this trade-off as we need [1]. Instead, what is desired is a score that can be used to rank all the files from most-likely malicious to least-likely. You can quantify this using the area under the curve (AUC), which measures the quality of a "ranking" of files by maliciousness. In other cases, like forensic investigations, there is an informed presumption that malware already exists and needs to be found. In this case, the motivations and costs are different. Minimal FN is the goal, so that if malware is actually on the system, it will be found, not missed, and evidence gathered. The costs of FPs are manageable in this instance, so the incentives are reversed. Our research has explored approaches for missions with varying FN/FP sensitivities.

Dealing with the fire hose of files

AV software on an individual user's machine is one piece of a larger puzzle. Large AV vendors have millions of installations of their software, all sending suspicious files back to a central system for analysis. It may be useful to think of the user's AV as the front end, or the collector, and the central system as the back end. This central system will receive millions of files daily. From the view of the central system, this

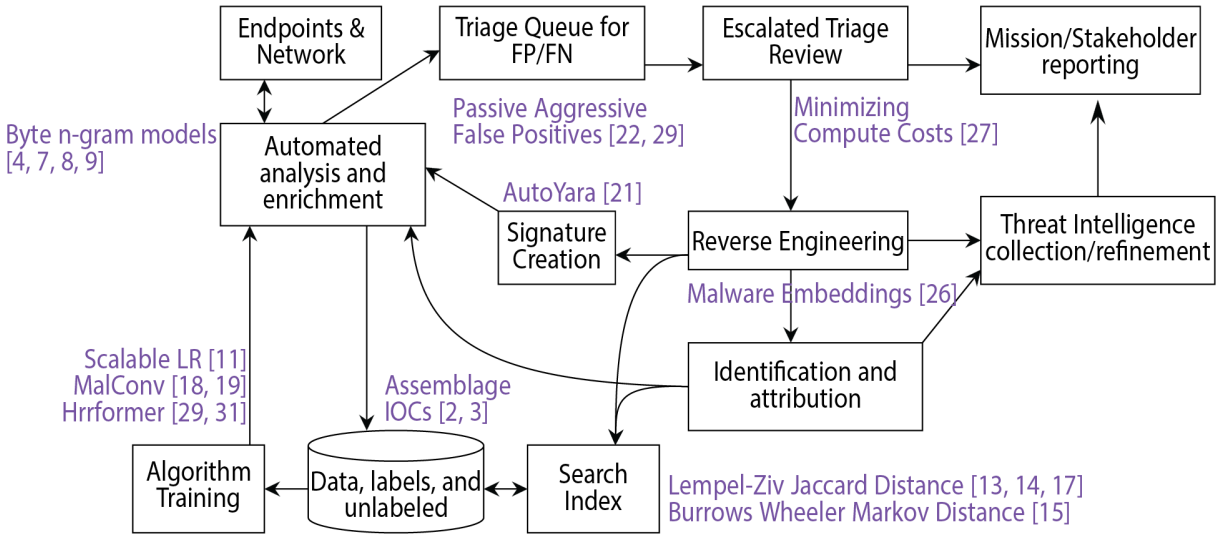

FIGURE 1. A simplified diagram of the kinds of interactions and stages necessary in supporting large scale file analysis. The purple text highlights selected research results, placed next to the stage it impacts.

fire hose of files must be triaged. Decisions must be made about whether and how to analyze each file, which analysis tools will provide the most relevant information, and how much computing and human effort to spend on each file. The US government faces similar issues, as it operates and defends millions of computers.

This large-scale analysis process can be thought of as a pipeline, including a variety of feature extraction and analytic processes which feed into decision-making. Often, analytics with increasing cost or levels of effort are applied to subsets of the files as they progress through the pipeline. Building capabilities that enhance both the analysis and decision-making parts of this pipeline has been a goal of our research.

Figure 1 is a simplified representation of parts of the file/malware analysis pipeline, showing how files or other information flows between stages. The purple text indicates papers we have published that are applicable to various stages.

Helping our academic and industry partners understand the government's perspective on this pipeline, and analytic needs associated with it, has also been a goal. We hope an increased understanding of this will motivate and incentivize additional research, resulting in new and improved capabilities for all cyber-defenders.

Deep file analysis

You've discovered some malware, what happens next? Sometimes you can just delete it, clean the machine it was on, and be done. Other times you may want to create a signature to automatically detect it, if it shows up again. If the malware was found on a high-value asset, you may want to investigate more deeply. How did this get onto this machine? What is the goal of this malware? How does it work? Who made it? How does the owner of the malware communicate with and control it?

The computers of the US government and defense contractors are targeted every day by actors from around the world. Some of these actors are highly motivated and well-resourced to come after our information. At times they even create custom-crafted malware that is specific and goal oriented. It is often critical to understand how we are being attacked, by whom, and what their motivations are. Better understanding of these factors informs our steps to remediate intrusions and prevent future ones through signatures and security updates.

Performing custom deep analysis on a specific file is a labor-intensive and primarily human-driven process. This work can take days or weeks for a single file, consuming analysts' time and creating a backlog. This slowness also interferes with a critical function of the analyst: noticing trends and making

connections. Is this malware similar to something we've seen before, or related to something we've seen before? That information could help reverse engineering go faster, avoid redundant work, and inform threat intelligence [2, 3]. Instead of treating all individual malware occurrences as isolated, understanding recurrence of related malware could help us identify a prolonged campaign.

Our foray into malware detection

Starting in 2012, the ACS team began looking at malware detection as a way to apply our ML research to real-world problems. At the time this seemed eminently reasonable. After a review of the academic literature, it seemed many published papers reported over 99 percent accuracies using byte n-grams. Given the reported results, it seemed like the easiest way to get started. Other approaches like dynamic analysis, disassembly, and similar techniques required extensive infrastructure, making both training and deployment more challenging. The reported accuracies in the literature were also not meaningfully different, so why pay this high price?

Replicating the data collection, algorithm training, and design processes allowed us to produce initial results. Significant software engineering work was done to make the process as fast as possible, rewriting research-grade Python code into Java gave us 100-1000 times speedups, and made it easy to deploy in a variety of mission environments. We had great initial success in our lab environment, so we decided it was time for field testing with our mission partners. As our first trial deployment reached its conclusion, we got harsh feedback: *the system was not useful, everything was labeled as malware, and they did not want to continue using the tool.*

This is how we first learned of the considerable gap between academic knowledge on the subject of malware detection and the practical application of such knowledge. The crux of the issue was that all our benign data came from a clean installation of Microsoft Windows, and our model would learn to literally look for the bytes of "Copyright Microsoft Corporation" as its method of determining "malicious" versus "benign." In reality, we had developed a "from Microsoft or not" detector. This underscores the importance of knowing your data, and training with representative data, as an ML model can only learn from the data you give it.

It was clear that we needed to get better-quality benign data for our research efforts. It was also clear that nontrivial parts of academic literature and culture around the application of ML for malware had misunderstandings of what would generalize to practice, and what tools and volumes of data were needed.

Byte-based models

Given the cost of feature engineering and domain knowledge, our team decided to explore just how far byte-based models could be pushed. This led to three different families of techniques that we have advanced over the past decade: 1) byte n-gram-based models, which have been surprisingly effective for malware detection; 2) using compression algorithms (similar to those used in a .zip file) for ML, which we have found especially useful for information retrieval situations; and 3) scaling and accelerating deep-learning techniques to handle malware data.

All three techniques have the broad benefit that we can apply them to any file format or data type, provided we are able to obtain sufficiently representative data (both benign and malicious) for said file format. This has allowed our small team to adapt and stay relevant to mission interests despite gaps in our domain expertise for many platforms that our mission partners are interested in. Each is also useful from a deployment and product life cycle perspective: because we work from raw bytes, there is no parsing involved. This means we never have parsing errors in production, a common issue as malware authors intentionally subvert the fact that the specification doesn't always match the actual implementation for a given file loader. This also avoids highly cumbersome dependencies. Many projects depend on the parsing capabilities of LIEF, IDA Pro, Ghidra, or other software with significant domain-expert knowledge. While valuable, that also means these tools are constantly evolving as malware actors adapt to detection approaches. This continuous evolution results in a significant technical burden of keeping tools current.

Byte n-grams

When we started working on byte n-grams with higher-quality data, one early win became very important for people using our algorithms and code. Deployment was fast. The model size and the code to run it was only a few megabytes. The model

essentially contained a list of byte 6-grams, and corresponding weights indicating if the byte pattern was positively/negatively impacting the malware decision [4]. This was low profile, meaning it could be deployed on almost any platform. Running the model could be done using a small buffer, streaming the data into main memory and checking if bytes occurred, incrementing/decrementing the cumulative score accordingly, and then returning the answer. Using rolling hash functions means you could perform the initial load of a file into memory or transfer it to another location and get a benign/malicious prediction essentially "for free" because this code was faster than the data transfer process.

Our first deployments were in situations where speed was the critical factor to deployment and mission success. Anything that took more than 100 milliseconds would be too slow to run on all the incoming data. Others were mission partners that needed to go on "fly away" missions, carrying all the computing they could use with them, and our tool was fast and easy to add. More deployments came as some missions used an ensemble of products to make a prediction, and our byte-based model was unusual compared to the standard AV approaches. This is valuable, as ensembles improve their predictive performance when the members of the ensemble are different (or, uncorrelated in their errors).

These benefits do not come for free though. The first cost is a decrease in accuracy compared to many industry products, which was expected. Industry AV products are built using vast amounts of data, and many person-years of labor analyzing and signaturing specific malware. AVs, by virtue of their presence on the host system, also have the ability to observe more than just the file, including behavioral patterns. Our byte-based models do not have these advantages, but they address different strategic needs than AVs.

The real costs come in algorithmic issues. Creating byte n-grams is expensive. We were using a cluster of 12 machines over three weeks to perform the n-gram mining on around 400,000 files[a]. Our method of choice, after considerable testing, was L_1 penalized logistic regression, because it could perform feature selection as a part of model training (the fewer features we have, the smaller the model for deployment, and the faster it is).

This led to years of research problems to tackle. First was discovering why byte n-grams failed to perform as well as we expected, and noting that the "Copyright Microsoft" problem had gone unchallenged in academic literature for decades [4]. We learned that byte n-grams are not 99.9 percent accurate; but surprisingly, they perform better than disassembly-based n-grams [5] and have more robustness to alterations than many AV products [6]. This increased our confidence in them. Addressing the large computing resources required to generate n-grams for a dataset, we developed new algorithms, including our KiloGrams algorithm, which reduced the $O(256^n)$ complexity down to just $O(n)$ [7, 8, 9]. This allowed us to explore larger byte n-grams. Our current constraint to additional scaling is the memory required to train logistic regression models on multi-terabyte datasets. We continue research on how to scale these computations [10, 11].

Compression-based similarities

A second direction we have explored is using compression algorithms to measure the similarity between arbitrary byte sequences. This is inspired by Kolmogorov complexity and originally developed as the normalized compression distance (NCD) [12]. To understand this, consider the function $K(\cdot)$, which takes an input x, which has a length of $|x|$, and returns the length of the smallest program that can reproduce the value of x. So, imagine you have the input x = *aaaaaaaaaa*. Then K(x) would return a program that loops 11 times, returning the value of "a" each time. This function is thus the perfect compressor. The ratio $|K(x)|/|x|$ tells you how compressible a sequence x is. You can extend this idea by giving the function inputs, so K(x|y) would be the shortest program that produces x as output, given y as input. There is always the option to ignore the input, so we get $K(x|y) \leq K(x)$.

Computing a perfect K(x) is impossible, but we have general-purpose compression algorithms like zip, bzip, lzma, etc., which approximate K. If we represent our favorite compression algorithm, C(x), which computes the size of the string x after being compressed, we can compute a normalized compression distance (NCD):

$$NCD(x,y) = \frac{C(x+y) - \min\{C(x), C(y)\}}{\max\{C(x), C(y)\}}$$

This gives us a way to measure the similarity between arbitrary byte sequences. This is useful for

a. This was also after years of extensive performance optimization of our n-gramming software.

malware because very few ML techniques can scale to the size of an executable. For example, a 200 megabyte (MB) program is not unreasonable, but this is larger than the entire CIFAR100 dataset. But we can and do run compression algorithms without issue on large files, and this trick requires no training data. It is also useful for forensic applications: it may be the case that a recovered hard drive has corrupted data, and so only fragments of the file exist. Was the drive corrupted because of malware or just hardware failure? If you can match a fragment of a file to known destructive malware, you can answer the question. Similarly, an in-progress download may be cut short by other defenses, but you want to see if the fragment of the download matches any known data. Using compression similarity, we can still do these tasks, even though domain-knowledge-based parsing becomes impossible.

Unfortunately, NCD does not scale up. You must run a compression search for every distance computation you want to perform, which is painfully slow. However, this inspired us to try to optimize the idea behind NCD: repurposing compression algorithms for ML.

This started by noting that Lempel-Ziv-based compression algorithms, which build compression dictionaries, tend to work best for malware detection. We took the compression dictionary created by the compression process, and treated it as a set of objects representing the file. Then we used set similarity measures like the Jaccard distance to measure the distance between two files. This way we could build the compression dictionary once per file, and re-use it in distance computations. Combined with min-hashing to make Jaccard distances faster, and with provably bounded error, the Lempel-Ziv Jaccard distance (LZJD) was born [13]. LZJD can compare any two sequences of bytes (e.g., files) and produce a similarity score. LZJD led to early success in mission deployments as it allowed the same kind of functionality as hashing digests such as ssdeep and sdhash, while being many times faster to search and more accurate [14]. With an eye to scaling up to even larger datasets, we applied the same trick using the bzip compression technique. Instead of Lempel-Ziv, bzip uses the Burrows-Wheeler transform and run-length encoding, which we connected to being spiritually similar to a Markov model: you assume that the current token is sufficient to predict the next token. This led to the Burrows-Wheeler Markov distance (BWMD). It functions similarly to LZJD, and we were able to scale to searching millions of files in seconds [15]. We also extended LZJD to create feature vectors usable for ML and added the ability to over-sample the minority classes to help handle class imbalance that often occurs in family detection work [16, 17].

Deep learning over long sequences

As mentioned, a 200 MB executable is not unusual. But processing that byte-by-byte means a 200,000,000 step long time series classification task. Most recurrent neural networks and long short-term memories consider 5,000 steps to be "very long," and only recently transformers have attempted to get to around 32,000 steps—at a nontrivial cost to accuracy.

When we first attempted to tackle this problem back in 2016, we turned to convolutional networks as an approach to tackling the scale of a single data point. Through significant exploration, we found that the canonical best practices did not work for malware data. We had to make our model shallower, with wider convolutions, and eschew normalization layers to get improved accuracy. Our first result, MalConv, could process 1,000,000 steps, a significant advancement over the previous best possible number[18]. But that still wasn't enough. A few years later, we cracked the 200,000,000 time step barrier for effectively length-invariant convolutional neural-network-based training [19].

While there have been barriers to deploying the MalConv architecture operationally, it has been a significant advancement that other organizations have built on. Academically, we have seen uses of MalConv for authorship identification, tool-chain identification, and other reverse engineering tasks, with similar deployments in the national labs. MalConv has also been one of our most successful efforts in influencing academia to focus more effort on problems that will have a real-world impact. Notably, a significant amount of work in adversarial attacks and defense possibilities has been developed using MalConv, providing us with valuable information about potential weaknesses, and has allowed us to develop new non-negative techniques to mitigate some of these attacks [20]. The research community's work exploring adversarial ML attacks on MalConv has provided insights into vulnerabilities and potential defenses of a class of ML methods which would have taken us years to work out on our own. This

symbiotic information benefit is one of the major reasons we push to continue working publicly with academic partners.

Hybrid full-stack

The "domain-knowledge free" approach our team started has invigorated significant academic research that helps us better understand what capabilities and risks are possible in the near future, while producing valuable, fast, low technical debt solutions to satisfy a variety of Agency missions. But this approach is perhaps most valuable as the first-level analysis/automated triage. Eventually, more technically sophisticated resources are required. Our goal over the past few years has been to leverage what we've learned and begin to adapt it toward hybrid solutions, forward integrating it into the set of common tasks that take up valuable analyst time.

Many of the techniques we have developed have been deployed in the government directly or via corporate partners who have leveraged the research. They address pieces of the big picture in figure 1. Another application of our extremely efficient n-gramming techniques is *AutoYara,* a technique for taking a handful of files known to be related and automatically producing a Yara signature that can be run with standard tools. Compared to human analysts doing this work, we found AutoYara could produce satisfying signatures for 84 percent of their work queue. This is especially helpful when human analysts have items flowing into their work queue faster than they can work them. Top human analysts produce better signatures, but AutoYara can be scaled to handle the volume that humans cannot [21].

Cyber environments vary widely. For ML models that are used in production, one may need to adapt the model to the individual environment that they are operating in—because models trained on a global population of benign and malicious programs often don't quite fit the idiosyncrasies of a specific environment. We developed a *passive aggressive* approach to patching a deployed malware detector to allow it to be adapted or tuned to a local environment with limited data [22].

Working with collections of files that may number in the hundreds of millions creates many storage and computational challenges. For any single task that is known a priori, one can extract the necessary features from each file and then discard the file. But we don't always know what information in a file will be important to future analyses. Therefore, creating a compact representation of a file, which contains information that can be used for multiple tasks, such as malware detection, family classification, and malware attribute prediction, is desirable. In ML lingo, can we use metric learning to derive low-dimensional embeddings that are useful for downstream tasks? We explored this direction, learning embeddings using CAPA [23, 24] and Endgame Malware Benchmark for Research (EMBER) [25] features, and experimented with different kinds of loss functions. This direction has potential, but more work remains to be done [26]. Similarly, we've used ML to predict what CAPA will find, with the ability to abstain from predictions when uncertain. This allows us to *minimize compute* costs by running expensive analytics only when necessary and can save multiple compute years of time if expensive virtual machines are used in the second phase [27].

Cognitive science-inspired learning

Cognitive science, the general study of how the brain and human cognition work, played a fundamental and important role in early artificial intelligence research. Current work in deep learning has its origins in neural networks inspired by the brain, though it has diverged much since that original spark. One of our research directions has been to look at tools that have found utility in cognitive science and try to move them back into ML. The hope is that this could allow us to obtain benefits, while being different from existing approaches, and enhance the diversity of our models.

The holographic reduced representation (HRR) is one particular method our team has extended. Originally developed in 1995 [28], it found significant use in cognitive science for its ability to perform symbolic artificial intelligence on a biologically plausible substrate. If $\mathrm{F}(\cdot)$ represents the Fourier transform, and $\mathrm{F}^{-1}(\cdot)$ its inverse, and *a, b, x, y* are all vectors in a d-dimensional space, it was shown that you can develop a binding operation $a \otimes b = \mathrm{F}^{-1}(\mathrm{F}(a) \odot \mathrm{F}(b))$. This has unique properties where you can distinguish between items added and bound together using an inversion operator †, so a "statement" $S = a \otimes b + x \otimes y$ can be constructed, and you can approximately determine "what was bound with x" by computing $y \approx S \otimes x^{\dagger}$. It turns out that this is sufficient to perform symbolic artificial intelligence,

but all the operations are being done with real-valued vectors and in a compressed representation.

The HRR presented a new direction to build more efficient algorithms with potentially new capabilities. We decided to explore this direction, and we have had some success. First was the task of determining how to modernize HRRs within a differentiable framework so that we could use modern PyTorch and JAX libraries [29]. This enabled multiple new directions. First, we leveraged the symbolic properties of HRRs to build a neural network that performed most of its work on a third-party, and slightly untrusted, computer. This allowed for reducing the amount of local compute resources while hiding the nature of the data from the third party via a kind of pseudo-encryption. While not as strong as encryption, these *Connectionist Symbolic Pseudo Secrets (CSPS)* were several thousand times faster than the existing cryptographic options [30]. More recently we used HRRs to tackle the computational bottleneck of transformers. Our malware data has sequences in excess of $T = 200{,}000{,}000$ time steps, and normal transformers have $O(T^2)$ computational complexity. Using HRRs, we brought this down to just linear $O(T)$ cost by inventing the HRRformer, while simultaneously improving accuracy [31].

In both cases, we were unable to fully solve the problem of interest, which is often the case in research; however, both cases significantly informed our computational needs for the future. The CSPS informs the possibility of using the floating point capabilities to perform at least some limited set of obfuscation when full cryptography may not be required or feasible. The HRRformer brought us closer to the computational throughput needed to tackle our long sequence malware goals—now we need an additional two orders of magnitude improvement, where before we needed seven. Our research thus transformed an unlikely avenue for computing systems to impact future missions into something with realistic potential in the next few years.

Which ML techniques really work?

When considering future hardware designs, the time from conception to completion is measured in years. For this reason, it is critical that we design algorithmic primitives we know will be useful several years into the future, and discover which do not scale to our desired needs [32]. Working in malware detection research helps us to ensure that is the case, by narrowing the focus to ML techniques that generalize.

This is an insight we have developed over time through hard-fought progress. Most ML and deep-learning research occurs in computer vision, natural language processing, and speech/signal processing. In all of these domains, there is a shared fundamental property: things near each other are related to each other. This is the prior of spatial locality, which is a core foundation of convolutions, recurrent neural networks, and positional embeddings. When this prior is too strongly violated, many of the "best" techniques that the community "knows" work begin to fall apart. In pushing these boundaries, we separate what is only conditionally useful from what truly generalizes to new and different problems.

For example, we recently turned to multiple instance learning (MIL) as a technique that makes the ML match how analysts do their job: something is benign by default, and only becomes malicious if something malicious is detected. There are no "positive" features in this regard, or at least, there shouldn't be. Yet we found that many current deep MIL algorithms fail to maintain this invariant and cause us unexpected and unsatisfying results [33]. Similarly, we must be careful with our data, and so have spent time studying how to evaluate [34] and construct [35] new and more informative datasets so that we, and the community at large, confidently know what we are measuring.

This process has helped us further refine the kinds of fundamental deep-learning building blocks that we care about for designing future hardware. It informed our big-picture view on what algorithmic issues we can work around and what are still roadblocks, compute versus memory trade-offs, and the types of primitives and operations we find consistently useful in malware and the broader ML problem spaces.

Conclusion

Our team has made significant advances in scaling, accelerating, and parallelizing ML techniques which can be applied to malware detection and a host of other domains and problems. We function as an interface between academic research, industry, and government mission users, following the latest research, influencing the directions of academic research and commercial investments in research and development, advancing the state of the art with

our own work, and bringing the best of all of these to bear on critical mission problems. And we look forward to doing this for years to come.

References

[1] Nguyen AT, Raff E, Nicholas C, Holt J. "Leveraging uncertainty for improved static malware detection under extreme false positive constraints." In: *IJCAI-21 1st International Workshop on Adaptive Cyber Defense;* 2021.

[2] Pingle A, Piplai A, Mittal S, Joshi A, Holt J, Zak R. "Relext: Relation extraction using deep learning approaches for cybersecurity knowledge graph improvement." In: *Proceedings of the 2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining,* New York, NY; 2020: pp. 879–886. Association for Computing Machinery. Available at: https://doi.org/10.1145/3341161.3343519.

[3] Piplai A, Mittal S, Joshi A, Finin T, Holt J, Zak R. "Creating cybersecurity knowledge graphs from malware after action reports." *IEEE Access*. 2020;8:211691–211703. doi: 10.1109/ACCESS.2020.3039234.

[4] Raff E, Zak R, Cox R, Sylvester J, Yacci P, Ward R, Tracy A, McLean M, Nicholas C. "An investigation of byte N-gram features for malware classification." *Journal of Computer Virology and Hacking Techniques.* 2018;14:1–20. Available at: https://doi.org/10.1007/s11416-016-0283-1.

[5] Zak R, Raff E, Nicholas C. "What can N-grams learn for malware detection?" In: *2017 12th International Conference on Malicious and Unwanted Software;* 2017: pp. 109–118. Available at: https://doi.org/10.1109/MALWARE.2017.8323963.

[6] Fleshman W, Raff E, Zak R, McLean M, Nicholas C. "Static malware detection & subterfuge: Quantifying the robustness of machine learning and current anti-virus." In: *2018 13th International Conference on Malicious and Unwanted Software;* 2018: pp. 1–10. Available at: https://doi.ieeecomputersociety.org/10.1109/MALWARE.2018.8659360.

[7] Raff E, Nicholas C. "Hash-grams: Faster N-gram features for classification and malware detection." In: *Proceedings of the ACM Symposium on Document Engineering 2018,* New York, NY; 2018. Association for Computing Machinery. Available at: https://doi.org/10.1145/3209280.3229085.

[8] Raff E, McLean M. "Hash-grams on many-cores and skewed distributions." In: *2018 IEEE International Conference on Big Data;* 2018: pp. 158–165. Available at: https://doi.org/10.1109/BigData.2018.8622043.

[9] Raff E, Fleming W, Zak R, Anderson H, Finlayson B, Nicholas C, Mclean M. "Kilograms: Very large N-grams for malware classification." *Learning and Mining for Cybersecurity;* 2019. Available at: https://eda.rg.cispa.io/events/lemincs19/papers/paper_raff_etal.pdf.

[10] Lu F, Raff E, Holt J. "A coreset learning reality check." In: *Proceedings of the AAAI Conference on Artificial Intelligence,* 37; 2023. doi: 10.1609/aaai.v37i7.26074.

[11] Raff E, Sylvester J. "Linear models with many cores and CPUs: A stochastic atomic update scheme." In: *2018 IEEE International Conference on Big Data;* 2018: pp. 65–73. Available at: https://doi.org/10.1109/BigData.2018.8622172.

[12] Li M, Chen X, Li X, Ma B, Vitanyi PMB. "The similarity metric." *IEEE Transactions on Information Theory.* 2004;50(12):3250–3264. Available at: https://doi.org/10.1109/TIT.2004.838101.

[13] Raff E, Nicholas C. "An alternative to NCD for large sequences, Lempel-Ziv Jaccard Distance." In: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining,* New York, NY; 2017: pp. 1007–1015. Association for Computing Machinery. Available at: https://doi.org/10.1145/3097983.3098111.

[14] Raff E, Nicholas C. "Lempel-Ziv Jaccard Distance, an effective alternative to ssdeep and sdhash." *Digital Investigation.* 2018;24:34–49. Available at: https://doi.org/10.1016/j.diin.2017.12.004.

[15] Raff E, Nicholas C, McLean M. "A new Burrows Wheeler transform Markov Distance." In: *Proceedings of the AAAI Conference on Artificial Intelligence 34(04);* 2020: pp. 5444–5453. Available at: https://doi.org/10.1609/aaai.v34i04.5994.

[16] Raff B, Nicholas C. "Malware classification and class imbalance via stochastic hashed LZJD." In: *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security,* New York, NY; 2017: pp. 111–120. Association for Computing Machinery. Available at: https://doi.org/10.1145/3128572.3140446.

[17] Raff E, Aurelio J, Nicholas C. "PyLZJD: An easy to use tool for machine learning." In: Calloway C, Lippa D, Niederhut D, Shupe D, editors. *Proceedings of the 18th Python in Science Conference;* 2019: pp. 101–106. Available at: http://dx.doi.org/10.25080/Majora-7ddc1dd1-00e.

[18] Raff E, Barker J, Sylvester J, Brandon R, Catanzaro B, Nicholas C. "Malware detection by eating a whole EXE." *AAAI Workshop on Artificial Intelligence for Cyber Security;* 2017. Available at: https://cdn.aaai.org/ocs/ws/ws0432/16422-75958-1-PB.pdf.

[19] Raff E, Fleshman W, Zak R, Anderson HS, Filar B, McLean M. "Classifying sequences of extreme length with constant memory applied to malware detection." In: *Proceedings of the AAAI Conference on Artificial Intelligence 35(11);* 2021:9386–9394. Available at: https://doi.org/10.1609/aaai.v35i11.17131.

[20] Fleshman W, Raff E, Sylvester J, Forsyth S, McLean M. "Non-negative networks against adversarial attacks." In: *The AAAI-19 Workshop on Artificial Intelligence for Cyber Security;* 2018.

[21] Raff E, Zak R, Munoz GL, Fleming W, Anderson HS, Filar B, Nicholas C, Holt J. "Automatic Yara rule generation using biclustering." In: *Proceedings of the 13th ACM Workshop on Artificial Intelligence and Security,* New York, NY; 2020: pp. 71–82. Association for Computing Machinery. Available at: https://doi.org/10.1145/3411508.3421372.

[22] Raff E, Filar B, Holt J. "Getting passive aggressive about false positives: Patching deployed malware detectors." In: *2020 International Conference on Data Mining Workshops;* 2020: pp. 506–515. doi: 10.1109/ICDMW51313.2020.00074.

[23] Mandiant. CAPA. Available at: https://github.com/mandiant/capa/.

[24] Ballenthin W, Raabe M. "capa: Automatically Identify Malware Capabilities." 2020. Mandiant. Available at: https://www.mandiant.com/resources/blog/capa-automatically-identify-malware-capabilities.

[25] Anderson HS, Roth P. "EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models." 2018; ArXiv. Available at: https://arxiv.org/abs/1804.04637.

[26] Rudd EM, Krisiloff D, Coull S, Olszewski D, Raff E, Holt J. "Efficient malware analysis using metric embeddings." *Digital Threats: Research and Practice.* Accepted August 2023. Available at: https://doi.org/10.1145/3615669.

[27] Nguyen AT, Zak R, Richards LE, Fuchs M, Lu F, Brandon R, Munoz GL, Raff E, Nicholas C, Holt J. "Minimizing compute costs: When should we run more expensive malware analysis?" In: *Proceedings of the Conference on Applied Machine Learning in Information Security;* 2022. Available at: https://www.camlis.org/andre-nguyen-2022.

[28] Plate TA. "Holographic reduced representations." *IEEE Transactions on Neural Networks.* 1995;6(3):623–641. doi: 10.1109/72.377968.

[29] Ganesan A, Gao H, Gandhi S, Raff E, Oates T, Holt J, McLean M. "Learning with holographic reduced representations." In: *Advances in Neural Information Processing Systems;* 2021. Available at: https://proceedings.neurips.cc/paper_files/paper/2021/file/d71dd235287466052f1630f31bde7932-Paper.pdf.

[30] Alam MM, Raff E, Oates T, Holt J. "Deploying convolutional networks on untrusted platforms using 2D holographic reduced representations." In: Chaudhuri K, Jegelka S, Song L, Szepesvari C, Niu G, Sabato S, editors. *Proceedings of the 39th International Conference on Machine Learning, vol. 162 of Proceedings of Machine Learning Research;* 2022: pp. 367–393. Available at: https://proceedings.mlr.press/v162/alam22a.html.

[31] Alam MM, Raff E, Biderman S, Oates T, Holt J. "Recasting self-attention with holographic reduced representations." In: Krause A, Brunskill E, Cho K, Engelhardt B, Sabato S, Scarlett J, editors. *Proceedings of the 40th International Conference on Machine Learning, vol. 202 of Proceedings of Machine Learning Research;* 2023: pp. 490–507. Available at: https://dl.acm.org/doi/10.5555/3618408.3618431.

[32] Raff E, McLean M, Holt J. "An easy rejection sampling baseline via gradient refined proposals." In: *26th European Conference on Artificial Intelligence,* 2023. Available at: https://ebooks.iospress.nl/pdf/doi/10.3233/FAIA230483.

[33] Raff E, Holt J. "Reproducibility in Multiple Instance Learning: A case For algorithmic unit tests." In: *Advances in Neural Information Processing Systems;* 2023. Available at: https://proceedings.neurips.cc/paper_files/paper/2023/hash/2bab8865fa4511e445767e3750b2b5ac-Abstract-Conference.html.

[34] Patel T, Lu F, Raff E, Nicholas C, Matuszek C, Holt J. "Small effect sizes in malware detection? Make harder train/test splits!?" In: *Proceedings of the Conference on Applied Machine Learning in Information Security;* 2023.

[35] Joyce RJ, Raff E, Nicholas C, Holt J. "MalDICT: Benchmark datasets on malware behaviors, platforms, exploitation, and packers." In: *Proceedings of the Conference on Applied Machine Learning in Information Security;* 2023.