Traffic Analytics Development Kits (TADK): Enable Real-Time AI Inference in Networking Apps

Kun Qiu*, Harry Chang*, Ying Wang*, Xiahui Yu*, Wenjun Zhu*, Yingqi Liu*, Jianwei Ma*, Weigang Li*
Xiaobo Liu†, Shuo Dai†
*Intel Corporation, †ZTE Corporation
{kun.qiu,harry.chang,ying.wang,xiahui.yu,wenjun.zhu,yingqi.liu,jianwei.ma,weigang.li}@intel.com,
{liu.xiaobo2,dai.shuo}@zte.com.cn

***Abstract*—Sophisticated traffic analytics, such as the encrypted traffic analytics and unknown malware detection, emphasizes the need for advanced methods to analyze the network traffic. Traditional methods of using fixed patterns, signature matching, and rules to detect known patterns in network traffic are being replaced with AI (Artificial Intelligence) driven algorithms. However, the absence of a high-performance AI networking-specific framework makes deploying real-time AI-based processing within networking workloads impossible. In this paper, we describe the design of Traffic Analytics Development Kits (TADK), an industry-standard framework specific for AI-based networking workloads processing. TADK can provide real-time AI-based networking workload processing in networking equipment from the data center out to the edge without the need for specialized hardware (e.g., GPUs, Neural Processing Unit, and so on). We have deployed TADK in commodity WAF and 5G UPF, and the evaluation result shows that TADK can achieve a throughput up to 35.3*Gbps* per core on traffic feature extraction, 6.5*Gbps* per core on traffic classification, and can decrease SQLi/XSS detection down to $4.5\mu s$ per request with higher accuracy than fixed pattern solution.**

I. Introduction

Silicon and software technology advancements targeting AI inference have lowered the barrier (compute cost and R&D effort) to unleash the creativity and innovation of the network application developers on the use of AI-advanced techniques within their commercial solutions. Reports and analysis are projecting the use of AI in Enterprise SD-WAN deployments to increase from 5% in 2021 to 40% in 2025 [1].

Industry practices are introducing AI techniques using artificial intelligence (AI) and machine learning (ML) models across network analytics approach. Here are some examples of use cases: (1) **Traffic analytics:** Used to analyze encrypted network traffic, to identify anomalies within networks [2]; (2) **Malware Detection:** Detecting malicious traffic such as SQL injection or Cross-Site Script [3]; (3) **User Behavior analytics:** Detecting relationships, identifying anomalies, and conducting empirical assessments of security [4]–[6].

In order to support real-world workloads, an industry-standard framework for real-time AI traffic analytics has to meet the requirements for performance, accuracy, and scalability. Based on previous research and discussion with our customers and partners, we have identified several mutually challenging as follows:

- **High Throughput:** up to 3*Gbps* per core (rule-based level) for AI-based traffic classification [7], [8]
- **Low Latency:** $5 \sim 10\mu s$ per request for malicious traffic detection [9]
- **High Accuracy:** $\geq 95\%$ accuracy
- **Easy Deployment:** deploying without the need for specialized hardware (e.g., GPU, NPU, FPGA)
- **Easy Development:** module-based development like DPDK [10] and VPP [11]

To address above challenges, we have designed Traffic Analytics Development Kits (TADK), an industry-standard framework specific for AI-based networking workloads processing. TADK can provide real-time AI-based networking workload processing in networking equipment from the data center out to the edge without the need for specialized hardware [12]. Briefly speaking, TADK brings several advantages to AI-based networking processing:

1) **High Performance:** TADK provides highly-optimized library for real-time AI-based traffic analytics. We design several novel algorithms to increase the performance. From our benchmarking results, traffic classification can achieve up to 6.5*Gbps* per core, which can fully support real-time classification in most cases. Meanwhile, the overall pipeline of SQLi/XSS detection can achieve up to $4.5 \sim 6.1\mu s$ per HTTP request, which is 2x faster than the existing rule-based solution. Also, the accuracy of traffic classification and SQLi/XSS detection is $\geq 95\%$ in most cases [13]–[16].
2) **Easy Deployment:** The application developed with TADK does not rely on any specialized hardware. TADK fully utilizes modern CPU features such as **AVX512** to accelerate AI performance.
3) **Easy Development:** TADK offers a module-based development environment. Developers can implement their own AI-based traffic analytics application by combining TADK's modules like building block bricks [17].

The rest of the paper is organized as follows: we first give the background and related work of AI-based traffic analytics in Section II. In Section III, we will give the overview design of TADK. Then, we will give some detail of our highly optimized feature extraction algorithms in Section IV, and we will evaluate TADK in two scenarios: traffic classification and

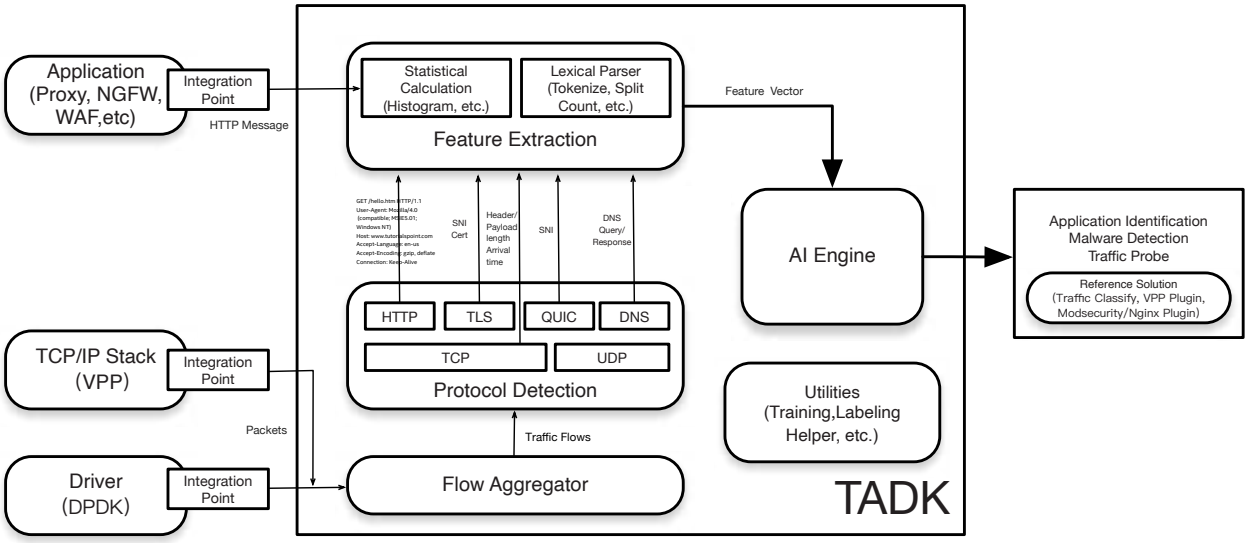


Fig. 1. The overall design of TADK.

SQLi/XSS detection in Section V. We conclude in Section VI.

II. Background and Related Work

A. Data Collection

A systematic survey has concluded a general pipeline of AI-based traffic analytics [18]. The first step is data collection [4], [5]. The AI-based solution needs historical data as the input source to train the model. However, capturing and labeling enough traffics is hard to conduct, mainly due to accuracy and privacy concerns. It is reported that 60% of research is using public non-encrypted traffic [18] and using DPI tools to label traffic. In order to cover this issue, TADK provides a labeling helper that can help users to label non-encrypted and encrypted traffic with only one click.

B. Feature Extraction

The next step is called feature extraction. The most common trend uses statistical-based features (e.g., inter-arrival time and packet size with the minimum, maximum and average metrics) since they can be used both on non-encrypted and encrypted traffic analytics [19]–[21]. However, some open-source feature extraction libraries [22] whose performance is as not good as TADK's library. Meanwhile, TADK can extract not only statistical features but also lexical features from encrypted traffic. It is proved that the combination of statistical and lexical features can significantly increase the accuracy. The flow extraction library of TADK has been utilized in AI traffic analytics [13]–[16]. TADK provides a tokenizer that is remarkably faster than existing solutions to extract lexical features.

C. AI Inference

At the last step, an AI model or an ensemble of models are needed for gathering analytics results [23], [24]. Both supervised and unsupervised methods are widely deployed in traffic analytics. Labeled datasets are used to train a supervised model such as SVM, decision tree, and random forest. Unsupervised models such K-Means are utilized in anomalous traffic detection. Meanwhile, most solutions use the unsupervised model to cluster encrypted traffics since labeling encrypted traffics [25]–[27] is difficult. In TADK, we provide an optimized random forest model for AI inference. We have compared a variety of models and found the random forest is well-balanced between accuracy and latency in traffic analytics workload.

III. The Overall Design of TADK

A. Core Libraries

TADK is composed of a series of core libraries, which are corresponding to feature extraction and AI inference steps we mentioned before. We show each component in Fig. 1. Flow aggregator is used to aggregate traffics from packets (e.g., real-time packets or packet traces from PCAP files) by 5-tuples. Protocol detection is used to identify protocols such as TCP, TLS, QUIC, and so on. Feature extraction, which is the competitiveness of TADK, has been well-designed to support real-time AI-based traffic analytics. We will describe some core algorithms in Section IV. AI engine is a wrapper of a high-performance random forest, which is based on Intel oneDAL [28]. Our AI engine supports both training and inferencing, including automatic feature reduction.

B. Utilities

TADK brings some useful utilities for training. The data cleaner and labeling helper provide an one-click solution for traffic labeling. The user only needs to capture one or several packet traces (e.g., PCAP files) as input of the labeling helper, and the helper will cluster these packet traces into several clusters. Each cluster will have a labeling tip. The only work

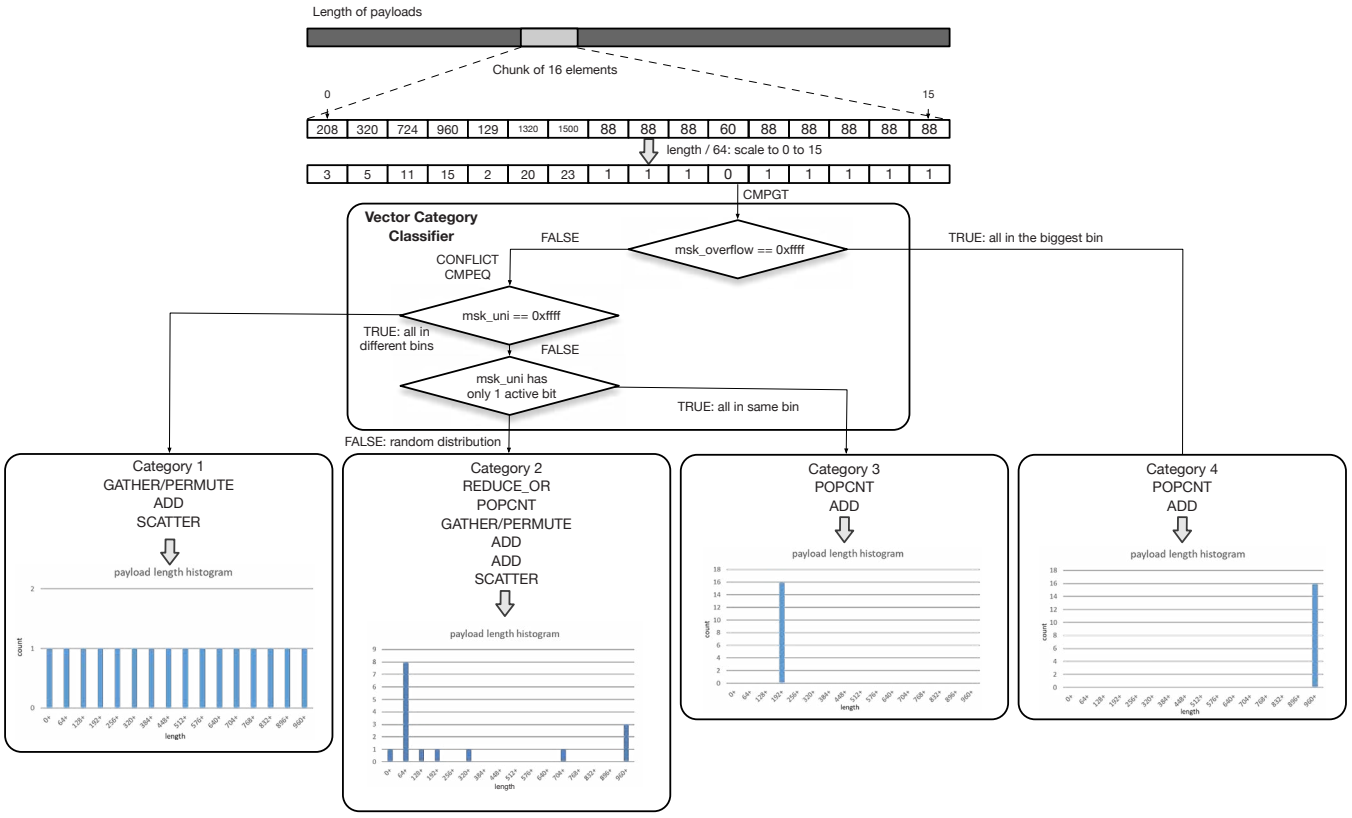


Fig. 2. The workflow of Advanced Vector Calculation with Vector Category Classifier

for the user is to label each cluster with tips and use labeled traffic to train a model.

C. Reference Solutions

TADK provides some samples to show the reference usage of TADK core libraries. The traffic classification sample can monitor network traffic and identify different applications in encrypted traffic. Either packet traces (PCAP files) or real-time traffic can be the input of the traffic classification sample. The SQL injection (SQLi)/Cross-Site Script (XSS) detection sample can detect whether the payload of HTTP traffic contains malicious code. TADK also provides a VPP plugin for the traffic classification sample and ModSecurity plugin for the SQLi/XSS detection sample. With these plugins, users can directly integrate AI-based solutions into their existing pipeline without any modification. We give the integration points in Fig. 1.

IV. Feature Extraction

A. SIMD-based Histogram

Histogram, such as the distribution characteristic of TCP packet header length, payload length, and arriving time intervals, etc., are mostly used as statistical features. Thus, designing an efficient histogram algorithm is a critical issue. We take histogram calculation of TCP packet payload length as an example to illustrate the detailed implementation. A buffer of lengths of packets as a shown example in Fig. 2 used to store the payload length of each packet in a network flow (for simplicity, 16 packets are considered here). The purpose of the histogram is to count the number of each element in the buffer belonging to a specific bin.

1) Existing Solution: Scalar Calculation (SC) is a widely utilized method. It has been implemented in most feature extraction libraries. SC is a loop-based method, which means they use huge amounts of loop and branch (it has to process and count each element one by one) for the histogram. In order to cover the disadvantage, a loop-free design such as a SIMD-based algorithm has been proposed.

2) Advanced Vector Calculation: We propose a SIMD-based algorithm called Advanced Vector Calculation (AVC). As shown in Fig. 2, we separate the input traffic into 4 categories:

1) **Category 1:** All elements are in different bins.
2) **Category 2:** All elements are random distribution.
3) **Category 3:** All elements are in one bin (except the biggest bin).
4) **Category 4:** All elements are in the biggest bin.

Since each category needs a different algorithm to calculate, we also propose a Vector Category Classifier (VCC) to identify the category of input data. In order to prevent VCC to be an overhead of histogram calculation, we use only up to 3 instructions to identify the category, which is also shown in Fig. 2. We give define SIMD intrinsics in TABLE I.

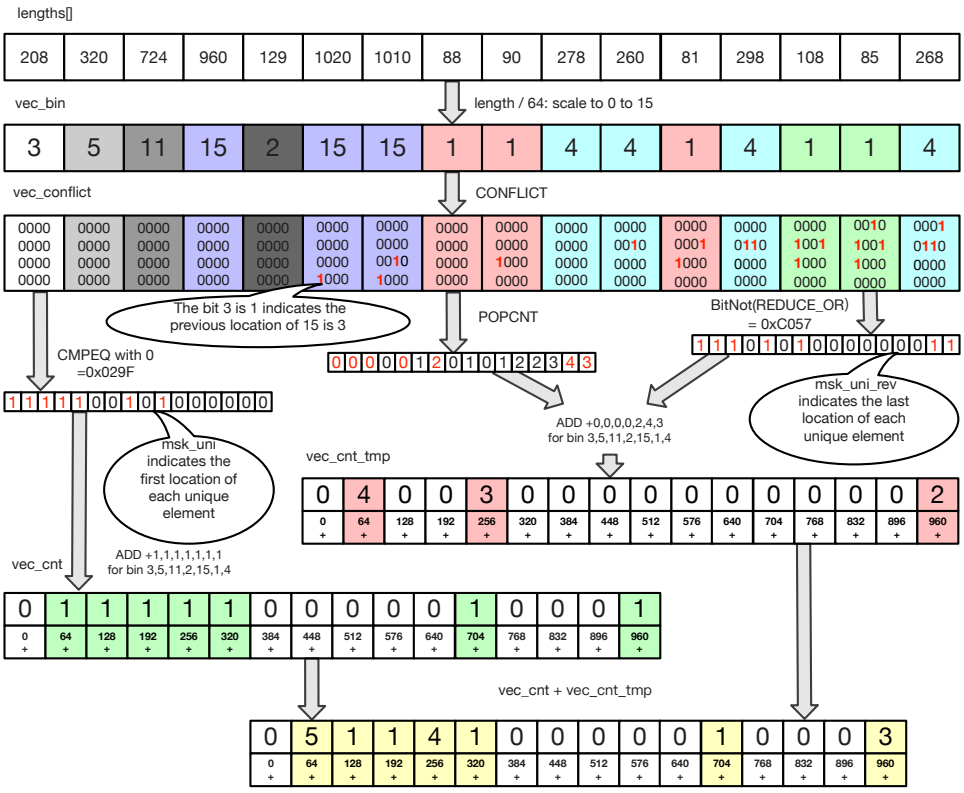


Fig. 3. Calculating histogram of category 2

TABLE I
DEFINITION OF SIMD INTRINSICS [29]

Notation	Description
CMPGT($\vec{a}$, $\vec{b}$)	Compare $\vec{a}$ with $\vec{b}$ for greater-than
CONFLICT($\vec{a}$)	Test each element of $\vec{a}$ for equality
REDUCE_OR($\vec{a}$)	Reduce each element in $\vec{a}$ by bitwise OR
CMPEQ($\vec{a}$, $\vec{b}$)	Compare $\vec{a}$ with $\vec{b}$ for equal
POPCNT($\vec{a}$)	Count the number of logical 1 bits in each element in a
ADD($\vec{a}$, $\vec{b}$)	Add $\vec{a}$ with $\vec{b}$
GATHER($\vec{a}$, b) PERMUTE($\vec{a}$, b)	Load b to $\vec{a}$ with a specific order
SCATTER($\vec{a}$, b)	Store $\vec{a}$ to b with a specific order

Briefly speaking, we first use a CMPGT to identify whether each element are larger than the biggest bin. If all elements is larger than the biggest bin, it is category 4. Then, we use CONFLICT to compute *vec_conflict* and *msk_uni* for checking whether each element is unique. If all elements are unique , it is category 1. At last, we can simply check whether the *msk_uni* with only one active bit. If there is only one bit in the *msk_uni*, it is category 3, otherwise, it is category 2.

Although it is easy to calculate the histogram in categories 1, 3, and 4 with up to 3 instructions, designing an algorithm for category 2 is the most challenging work. Thus, we propose a novel algorithm to calculate the histogram in category 2. We also give an example in Fig. 3. Algorithm 1 shows the pseudo-code of AVC and VCC. We evaluate our proposed AVC for histogram calculation. AVC can achieve up to 11.73x, 4.38x, 1.33x and 1.47x faster than the existing solution in categories 1,2,3,4 respectively.

B. DFA-based Tokenization

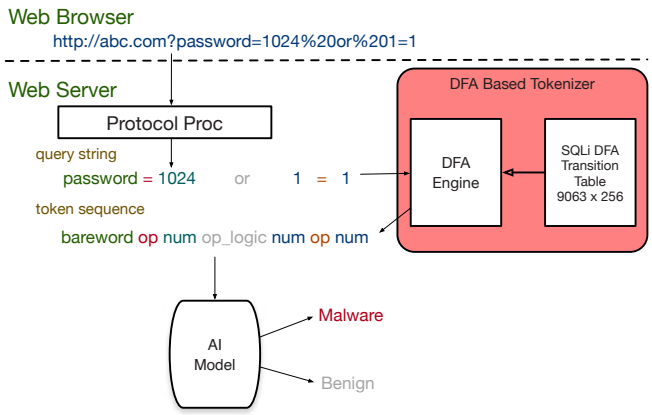


Fig. 4. An example of SQLi detection with DFA-based tokenizer

Most AI-based traffic analytics (e.g, Next Generation Web

Algorithm 1: Advanced Vector Calculation (AVC)

Input: *length* the buffer of payload length
Output: *hist* the histogram of payload length

Reset *hist*
Load *length* to *vec_len*
vec_bin $\leftarrow \frac{vec\_len}{64}$
msk_overflow $\leftarrow$ CMPGE(*vec_bin*, $\vec{15}$)
if *msk_overflow = 0xffff* **then**
    Category 4: all in the biggest bin
    hist[15] $\leftarrow$ *hist*[15] + 16
    return *hist*
Remove elements that is larger than 15 in *vec_bin*
vec_conflict $\leftarrow$CONFLICT(*vec_bin*)
Use CMPEQ to convert *vec_conflict* to *msk_uni*
if *msk_uni = 0xffff* **then**
    Category 1: all in different bins
    GATHER(*vec_cnt*, *hist*)
    vec_cnt_added $\leftarrow$ ADD(*vec_cnt*, $\vec{1}$)
    SCATTER(*hist*,*vec_cnt_added*)
else if *msk_uni BitAnd (msk_uni - 1)* = 0 **then**
    Category 3: all in the same bin
    hist[*vec_bin*[0]] $\leftarrow$ *hist*[*vec_bin*[0]] + 16
else
    Category 2: random distribution
    msk_uni_rev $\leftarrow$ *BitNot* REDUCE_OR(*vec_conflict*)
    vec_popcnt $\leftarrow$ POPCNT(*vec_conflict*)
    GATHER(*vec_cnt*, *hist*)
    vec_cnt_tmp $\leftarrow$ ADD(*vec_cnt*, $\vec{1}$)
    vec_cnt_added $\leftarrow$ ADD(*vec_cnt_tmp*, *vec_popcnt*)
    SCATTER(*hist*,*vec_cnt_added*)

Application Firewalls) needs tokenization to convert lexical features (string-based information) into vectors as the input of the AI-model. For lexical features, most tokenizers (e.g., OpenNMT) are branch-based, which means they use huge amounts of IF-ELSE for tokenizing. A branch-based solution is easy to implement, but it is unfriendly to the CPU's pipeline, and it may increase the number of cache misses. Thus, TADK uses a DFA-based tokenizer and provides a generator that can convert an easy-to-code profile into a specific DFA. We give an example of SQLi detection with a DFA-based tokenizer in Fig. 4. We also propose a training video [30] to describe how the tokenizer works.

1) Generator: In order to support multiple language/file formats, we propose a generator that can generate DFA from user-defined profiles. We defines a DFA profile language which can be easily maintained by our customers, and easy to extend to add new tokens for emerging threats, and to support more use cases. The generator also includes a DFA compiler to compile the user-defined profile into its corresponding DFA transition table. The DFA transition table is directly used by the Tokenizer.

Algorithm 2: DFA engine

Input: T the transition table
V the input string
A the state table
Output: R the accept state

$S \leftarrow$ inital state
for $i \in$ *the numbers of character C divide by V* **do**
    $S \leftarrow T[S][C]$
    if S *is accept state* **then**
        output $A[S]$

2) Tokenizer: The DFA transition table describes the transition behavior under every state and input character. Algorithm 2 shows how the DFA engine works. The engine does simple transitions in the main loop which makes it very fast.

V. Evaluation

A. Environment

We implement TADK using GCC 7.5. Since TADK has been deployed in several scenarios, such as WAF or 5G User Plane Function (UPF), we have different CPU and RAM environment. The 5G UPF uses ZTE 5300G4X, which is based on Intel Xeon Gold 6330N CPU (Icelake) with 512GB DDR4 RAM. Other evaluation is based on Xeon Gold 6148 CPU (Skylake) and Intel Xeon Platinum 8358 CPU (Icelake) with 32G DDR4 RAM. We integrate our reference traffic classification sample into ZTE 5G UPF to test its throughput and accuracy. We use IXIA as a traffic generator to generate traffic to test the maximum throughput with zero packet loss.

B. Data

Since we choose random forest as our AI inference model, we evaluated the accuracy of random forest in both traffic classification and malware detection. In traffic classification, we have collected top applications in China (BAIDU, TMALL, BILIBILI, TENCENT, TOUTIAO, KUAISHOU, QQ, HUOSHAN, QQNEWS, YOUKU, WECHAT) from the real-world, for both training and inferencing. In malware detection, we use SQLMAP [31] for SQLi and XSSTRIKE [32] for XSS to gather data for both training and inferencing. We also choose some public data for inferencing.

C. Traffic Classification

1) Accuracy: We give the confusion matrix of the model that can classify 9 applications in Fig. 5. All precision and recalls are larger than 90%, and the average precision, recall, and f1-score are $0.936, 0.926, 0.918$. From the evaluation result we can see that the accuracy for traffic classification is sufficient for most scenarios. We also train a model to classify WeChat image transfer traffic and WeChat video transfer traffic, which are UDP traffic. We prepare 70 image transfer flows and 100 video transfer flows to train, and we give the accuracy detail in TABLE II. The average precision, recall, and f1-score are $0.883, 0.884, 0.883$.

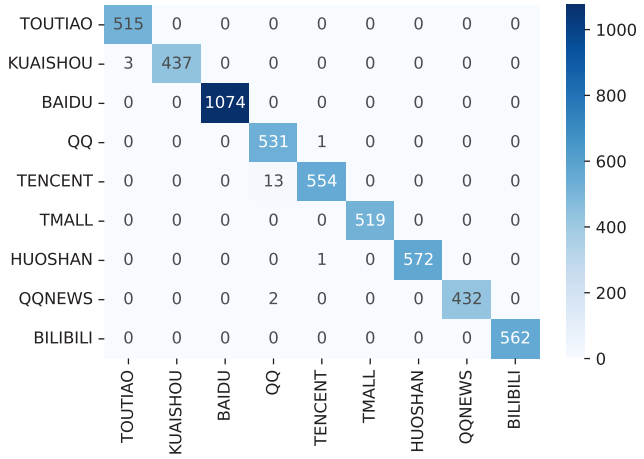


Fig. 5. The confusion matrix of 9 applications

TABLE II
CLASSIFY WECHAT VIDEO AND IMAGE TRANSFER

Class	Precision	Recall	F1-score	Flows
WECHAT Video	0.903	0.875	0.889	32
WECHAT Image	0.862	0.893	0.877	28

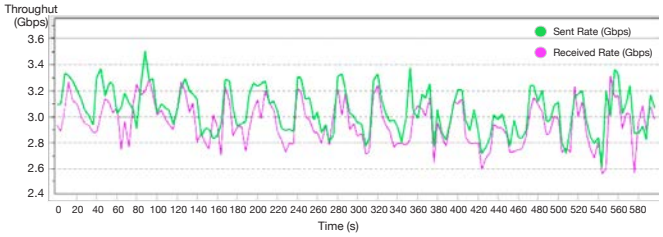
2) Performance: We test our latency with the model that can classify 2 applications (train and test by WECHAT with 1524 flows and YOUKU with 1551 flows). From Table III we can see that our latency can achieve $10.7\mu s$ per flow, which is sufficient for most scenarios. Moreover, we also test the latency of feature extraction for DNS, HTTP and TLS in Table III. The average packets of DNS, HTTP, and TLS are 2, 8 and 13. With the POPCNT instruction and the new architecture, the latency has been reduced significantly. The reason why TLS has lower latency than HTTP is TLS has less lexical features to extract.

TABLE III
LATENCY PER FLOW

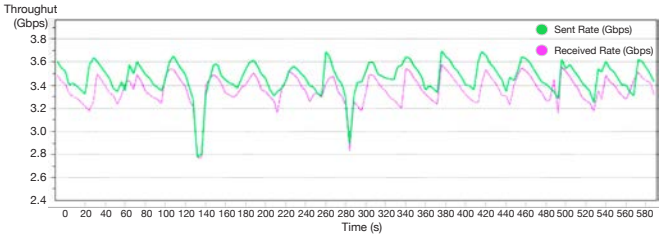
	Traffic Classification		Feature Extraction		
Architecture	WECHAT	YOUKU	DNS	HTTP	TLS
Skylake	$12.9\mu s$	$15.0\mu s$	$1.3\mu s$	$3.3\mu s$	$2.5\mu s$
Icelake	$10.7\mu s$	$12.2\mu s$	$0.9\mu s$	$2.6\mu s$	$2.0\mu s$
Reduction	17%	19%	31%	21%	20%

We also test the throughput with YOUKU. The average packets per flow is 20 and more than 99% flows are HTTP and TLS flows. The throughput is $6.5Gbps$ $(1,629kpps)$ per core. Since the average packets per flow is 28 in Internet [33], we can estimate our throughput can achieve $9.1Gbps$ in most cases. Moreover, the throughput of feature extraction can achieve $35.3Gbps$.

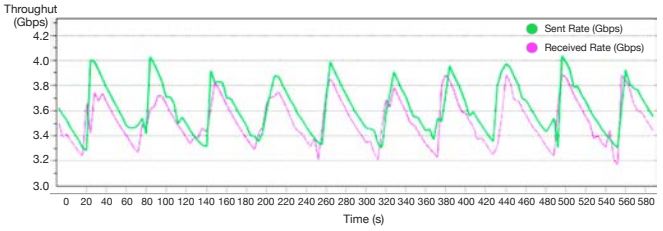
3) Throughput in 5G UPF: We test our throughput with models that can classify 3, 5, and 9 applications in 5G UPF respectively in Fig. 6. The maximum throughput is $3.78Gbps$ $(618kpps)$ with 5 applications and can get $3.39Gbps$ $(515kpps)$ and $3.58Gbps$ $(599kpps)$ with 3 and 9 applications. The result shows that the performance will not reduce with the increasing number of applications. The reason why the throughput in 5G UPF cannot achieve our aforementioned throughput and it has jitter is our naïve flow table implementation and other integration overhead.



(a) QQ, MEITUAN, QQNEWS



(b) QQ, TOUTIAO, HUOSHAN, KUAISHOU, QQNEWS



(c) BAIDU, TMALL, BILIBILI, TENCENT, TOUTIAO, KUAISHOU, QQ, HUOSHAN, QQNEWS

Fig. 6. Throughput Evaluation with IXIA

D. Malware Detection

1) Accuracy: We implement a ModSecurity plugin for SQLi/XSS with TADK. We compare our plugin with the well-utilized *libinjection* [9] in the same server environment (Nginx with ModSecurity). We set an attacking client with SQLMAP and XSSTRIKE to generate traffic to test the accuracy. TADK's plugin has higher accuracy (100% for SQLi and 99.8% for XSS) than the libinjection and it has fewer false positives.

2) Latency: We evaluate the latency of SQLi/XSS plugin in Table IV. TADK's latency is 50% less than libinjection. In conclusion, The AI-based solution has lower latency than a rule-based solution in SQLi/XSS that makes real-time AI-based malware detection possible.

VI. CONCLUSION

In this paper, we proposed TADK as a solution to address real-time AI-based networking workloads processing. The evaluation result shows that the application implemented

TABLE IV
LATENCY PER REQUEST

Plugin	SQL injection	Cross-Site Script
libinjection	$14.4\mu s$	$8.9\mu s$
TADK	$6.1\mu s$	$4.5\mu s$

with TADK can meet the requirements for real-time performance ($4.5\mu s$ per request on malware detection, 6.5*Gbps* and 35.3*Gbps* per core on traffic classification and feature extraction), accuracy ($\geq 95\%$), and scalability without any specialized hardware. We have deployed our solution in WAF and 5G UPF and we have evaluated it for real-world usage. We are currently working with our partners to improve the reliability and missing features (e.g., GQUIC) required for real-world deployment, and will be examined and used by the public and community eventually.

REFERENCES

[1] (2022) SASE, AI Fuel SD-WAN Winners. [Online]. Available: https://www.sdxcentral.com/articles/news/sase-ai-fuel-sd-wan-winners/2021/09/?utm_source=sendgrid&utm_medium=email&utm_campaign=website

[2] T. T. Nguyen and G. Armitage, "A survey of techniques for internet traffic classification using machine learning," *IEEE communications surveys & tutorials*, vol. 10, no. 4, pp. 56–76, 2008.

[3] A. Dainotti, A. Pescape, and K. C. Claffy, "Issues and future directions in traffic classification," *IEEE network*, vol. 26, no. 1, pp. 35–40, 2012.

[4] P. Megyesi, G. Szabó, and S. Molnár, "User behavior based traffic emulator: A framework for generating test data for DPI tools," *Computer Networks*, vol. 92, pp. 41–54, 2015.

[5] S. Molnár, P. Megyesi, and G. Szabó, "Multi-functional traffic generation framework based on accurate user behavior emulation," in *2013 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2013, pp. 13–14.

[6] L. Vassio, I. Drago, and M. Mellia, "Detecting user actions from HTTP traces: Toward an automatic approach," in *2016 International Wireless Communications and Mobile Computing Conference (IWCMC)*. IEEE, 2016, pp. 50–55.

[7] X. Wang, Y. Hong, H. Chang, K. Park, G. Langdale, J. Hu, and H. Zhu, "Hyperscan: A Fast Multi-pattern Regex Matcher for Modern CPUs," in *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, 2019, pp. 631–648.

[8] (2022) DPI Benchmarking. [Online]. Available: https://www.intel.com/content/dam/www/public/us/en/documents/solution-briefs/atom-c2000-hyperscan-pattern-matching-brief.pdf

[9] (2022) libinjection. [Online]. Available: https://github.com/client9/libinjection

[10] (2022) DPDK. [Online]. Available: http://www.dpdk.org/

[11] (2022) VPP. [Online]. Available: https://wiki.fd.io/view/VPP

[12] (2022) Huastart uCPE with TADK integration (in Chinese). [Online]. Available: https://www.intel.cn/content/www/cn/zh/communications/ai-sd-wan.html

[13] O. Barut, Y. Luo, T. Zhang, W. Li, and P. Li, "Multi-Task Hierarchical Learning Based Network Traffic Analytics," in *ICC 2021-IEEE International Conference on Communications*. IEEE, 2021, pp. 1–6.

[14] "NetML: A Challenge for Network Traffic Analytics," 2020. [Online]. Available: https://arxiv.org/abs/2004.13006

[15] O. Barut, R. Zhu, Y. Luo, and T. Zhang, "TLS Encrypted Application Classification Using Machine Learning with Flow Feature Engineering," in *2020 the 10th International Conference on Communication and Network Security*, 2020, pp. 32–41.

[16] O. Barut, M. Grohotolski, C. DiLeo, Y. Luo, P. Li, and T. Zhang, "Machine learning based malware detection on encrypted traffic: A comprehensive performance study," in *7th International Conference on Networking, Systems and Security*, 2020, pp. 45–55.

[17] (2022) Intel AI Networking Solution Guide. [Online]. Available: https://networkbuilders.intel.com/solutionslibrary/ai-technologies-unleash-ai-innovation-in-network-applications-solution-brief

[18] F. Pacheco, E. Exposito, M. Gineste, C. Baudoin, and J. Aguilar, "Towards the deployment of machine learning solutions in network traffic classification: A systematic survey," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 2, pp. 1988–2014, 2018.

[19] J. J. Davis and A. J. Clark, "Data preprocessing for anomaly based network intrusion detection: A review," *computers & security*, vol. 30, no. 6-7, pp. 353–375, 2011.

[20] J. J. Davis and E. Foo, "Automated feature engineering for HTTP tunnel detection," *computers & security*, vol. 59, pp. 166–185, 2016.

[21] A. K. Marnerides, A. Schaeffer-Filho, and A. Mauthe, "Traffic anomaly diagnosis in Internet backbone networks: A survey," *Computer Networks*, vol. 73, pp. 224–243, 2014.

[22] (2022) Cisco Joy. [Online]. Available: http://github.com/cisco/joy

[23] L. Peng, B. Yang, Y. Chen, and Z. Chen, "Effectiveness of statistical features for early stage internet traffic identification," *International Journal of Parallel Programming*, vol. 44, no. 1, pp. 181–197, 2016.

[24] R. Alshammari and A. N. Zincir-Heywood, "Identification of VoIP encrypted traffic using a machine learning approach," *Journal of King Saud University-Computer and Information Sciences*, vol. 27, no. 1, pp. 77–92, 2015.

[25] K. Goseva-Popstojanova, G. Anastasovski, A. Dimitrijevikj, R. Pantev, and B. Miller, "Characterization and classification of malicious web traffic," *Computers & Security*, vol. 42, pp. 92–115, 2014.

[26] M. C. Belavagi and B. Muniyal, "Performance evaluation of supervised machine learning algorithms for intrusion detection," *Procedia Computer Science*, vol. 89, pp. 117–123, 2016.

[27] K. Lalitha and V. Josna, "Traffic verification for network anomaly detection in sensor networks," *Procedia Technology*, vol. 24, pp. 1400–1405, 2016.

[28] Intel oneDAL. [Online]. Available: https://github.com/oneapi-src/oneDAL

[29] (2022) Intel intrinsics guide. [Online]. Available: https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html

[30] (2022) Real Time AI Inferencing & Traffic Analytics Development Kit (TADK). [Online]. Available: https://networkbuilders.intel.com/real-time-ai-inferencing-traffic-analytics-development-kit-tadk-overview-training-video

[31] (2022) SQLMAP. [Online]. Available: https://sqlmap.org/

[32] (2022) XSStrike. [Online]. Available: https://github.com/s0md3v/XSStrike

[33] M.-S. Kim, Y. J. Won, H.-J. Lee, J. W. Hong, and R. Boutaba, "Flow-based characteristic analysis of Internet application traffic," in *Workshop Chair*, 2004.