MULTA: Enabling Cost-efficient Multi-task Network Traffic Analysis

Cheng Guo*, Menghao Zhang*°, Guanyu Li*†, Mingwei Xu*
*Tsinghua University °Kuaishou Technology †Huawei Technologies Co., Ltd

***Abstract*—Machine learning based network traffic analysis has been widely used to combat malicious behaviors in various network scenarios. However, most of the applications are specialized for a single task and cannot cover a wide spectrum of security threats. Building a machine learning pipeline from scratch for each traffic analysis task is painstaking and time-consuming. And it is not practical to deploy several traffic analysis tasks together due to the high runtime overhead. In this paper, we propose MULTA to enhance the deployability of traffic analysis applications. MULTA is a framework that leverages Multi-Task Learning (MTL) – an emerging paradigm in machine learning – to enable the efficient parallel deployment of multiple traffic analysis tasks. Specifically, we devise a sequence-based traffic feature representation that is informative and general enough to serve a diverse range of tasks. Then we use the LSTM-based Multi-gate Mixture-of-Experts (LMMoE) model to learn task characteristics and propose a size tuning algorithm to optimize its performance. Considering different data curating methods, we also present two different training paradigms to construct MULTA. Our experiments show that MULTA can reduce the space occupancy of the model by up to 50% and accelerate the inference process by more than 100% in multi-task scenarios, which indicates better deployability in real-world networks.**

I. INTRODUCTION

Network traffic analysis refers to a class of tasks that infer sensitive information from network communication patterns to identify malicious behaviors. With the rapid deployment of data encryption technologies (e.g., SSL/TLS), traditional payload-based traffic analysis approaches are losing efficacy. In recent years, researchers are turning to machine learning (ML), especially deep learning, to glean useful information from encrypted traffic without touching packet payloads. These ML-powered traffic analysis applications have shown remarkably high detection accuracy in a wide range of tasks, such as website fingerprinting [1], [2], convert channel detection [3], [4], intrusion detection [5] and IoT device fingerprinting [6].

In real-world deployment scenarios, networks are usually facing complicated security situations and are threatened by diverse attack vectors, which drives operators to employ multi-dimensional detection and protection mechanisms. For example, IoT networks usually raise security concerns for their vulnerability to attacks. To prevent IoT devices from being exposed to most common security threats, an IoT gateway usually deploys Botnet detection as well as DoS detection applications [5]. Another example is the gateway of enterprise networks, which demands higher security requirements. Intrusion detection applications are always indispensable, and covert channel detection, together with fingerprinting-based systems, are also preferred to strengthen the security guarantee [7].

However, it is not easy to integrate multiple ML-based traffic analysis tasks together at a single deployment point. In a typical pipeline of ML, feature vectors are extracted from the raw traffic, and the ML model exploits them for training and inference. The features specified by each traffic analysis task are different, and the ML models they use also vary significantly. Therefore, if we simply put these tasks together, we need to design them separately and deploy several ML pipelines in parallel to get the results of each task. This is not a promising solution because of the following reasons. First, each ML pipeline is built from scratch, which needs task-specific expert knowledge to devise a representative feature extraction scheme and an effective ML model. Second, parallel ML pipelines require a huge amount of runtime resources, which raises high costs in real-world networks. In some scenarios (e.g., IoT gateway), the deployment platforms have strictly limited resources and cannot afford multiple traffic analysis applications. Even if the resource is not an intrinsic restriction, such as enterprise network gateways, the cost-efficiency of these traffic analysis tasks should also be considered carefully due to financial issues.

To alleviate the problems caused by parallel ML traffic analysis pipelines, in this paper, we propose MULTA, a cost-efficient multi-task traffic analysis framework. MULTA is powered by Multi-Task Learning (MTL) – an emerging learning paradigm in machine learning, and can leverage the relevancy among multiple traffic analysis tasks to fuse them into a single model. Our key observation is that the traffic analysis tasks usually have substantial similarities, which makes it feasible to adopt the methodology of MTL. To implement MULTA, we generalize the traffic feature extraction process by proposing a sequence-based feature representation that can be exploited by different tasks. Then we use the LSTM-based Multi-gate Mixture-of-Experts (LMMoE) network to discover common knowledge among tasks and propose a size tuning algorithm to optimize its cost-efficiency. With these designs, multiple traffic analysis tasks can share a single feature extraction process and the same ML model, which means only one ML pipeline is required. Our design brings substantial benefits to real-world

The research is supported by the National Natural Science Foundation of China under Grant 62221003, China Postdoctoral Science Foundation (2022M720202), and Beijing Postdoctoral Research Foundation (2022-ZZ-078).

979-8-3503-1090-0/23/$31.00 © 2023 IEEE

deployment scenarios. First, the construction efforts of traffic analysis tasks can be significantly reduced, since every phase of the ML pipeline is general. Second, the information sharing of both feature representation and the LMMoE model promises a much lower runtime cost. In practice, network operators usually employ different methods to collect data for target tasks. To this end, we propose two paradigms – *Aurora* and *Rainbow* – to train our framework under different dataset layouts. We set up different experimental configurations to evaluate MULTA, and the results show that MULTA is highly effective with lower resource overhead.

II. Background and Related Work

A. ML-based Network Traffic Analysis

Network traffic analysis is the technique that extracts sensitive information from traffic communication patterns. In this paper, we focus on the works which leverage ML to discover implicit patterns from the encrypted traffic. Another type of work that relies on application-level semantics or payload inspection is out of our scope. Here we present several typical traffic analysis tasks.

- *Website fingerprinting* is a traffic analysis task that identifies the website user visits. Recent researchers extract sequence-based features from network connections and use a variety of deep learning models to conduct classification under different assumptions [1], [2].
- *Covert channel detection* aims to uncover the stealthy communications that parasitize legitimate traffic. Attackers usually exploit temporal and spatial features of a network transmission to build a covert channel, which leaves chances for researchers to devise ML-based detection algorithms by analyzing the packet timing and size properties [3], [4].
- *Intrusion detection* is vital for high-value networks since it can fight against malicious activities. By analyzing the traffic at various granularities, ML-based intrusion detection systems can discriminate between normal and abnormal network traffic [5].

Most of the existing works focus on the *detection efficacy* rather than the *deployability* of traffic analysis tasks. Although in recent years there have been efforts to leverage newly-emerging hardware to accelerate traffic analysis [8], [9], they have not addressed the issue of multi-task deployability. For a practical traffic analysis framework, the task scope it can cover, the system ease-of-use, and the resource overhead should all be given attention. We are the first to take these factors into consideration in ML-based network traffic analysis.

B. Multi-task Learning

Multi-task learning leverages the commonalities among a series of tasks to fuse them within a single ML model. The early works of MTL [10], [11] carry out the idea of *hard parameter sharing*, which uses a single shared neural network to hold the common knowledge of all tasks. These approaches rely on a strong similarity among tasks to harmonize them within a single model, and are sensitive to outliers since there is little headroom for task differences.

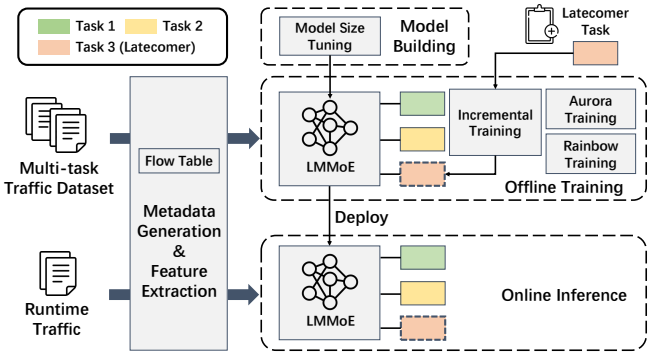

Fig. 1. MULTA architecture.

Researchers seek for more flexible sharing mechanisms in recent studies. *Soft parameter sharing* shares the network structures in a moderate manner, which enables a better way to model the complicated relationship of multiple tasks. Lu et al. [12] propose a greedy method to discover the hierarchical grouping relationship among tasks during the training process. Misra et al. [13] introduce cross-stitch units to learn an optimal split of network layers. Instead of sharing model structures directly, Duong et al. [14] and Yang et al. [15] choose the method of regularization. They add constraints to network layers to bound the distance between their parameters, using L2 and trace norm respectively. The MMoE model [16] we leverage in this paper is a variant of the vanilla MoE model [17]. MMoE employs an ensemble of experts to perform inference, just like MoE, and adds task-specific gates to realize a weighted voting mechanism on the results of experts. We introduce MMoE to multi-task traffic analysis for its flexibility in modeling complicated task relationships.

III. Multi-task Traffic Analysis

In this section, we present how to build our multi-task traffic analysis framework MULTA to meet the requirements of real-world deployment. We illustrate the overall architecture of MULTA in Fig. 1. Given a set of target tasks, we first perform a size tuning step to build a suitable LMMoE model. In the training phase, the traffic feature representations are extracted from the dataset. The LMMoE model exploits these features to learn the characteristics of multiple tasks under different training methodologies. When MULTA gets deployed, it performs multi-task online inference on the runtime traffic.

A. Informative Feature Representation

To minimize the feature extraction cost of the multi-task traffic analysis pipeline, we seek a general traffic feature representation that can serve as many traffic analysis tasks as possible, and preserve the inherent information of the tasks for ML training and inference. In some other application areas of multi-task learning, the input features of tasks are naturally identical, such as CV [13] and NLP [18]. However, traffic analysis presents significant differences as the feature sets claimed by different traffic analysis tasks show a huge variety. For instance, website fingerprinting applications usually prefer a sequence of packet directions, and intrusion detection

TABLE I
THE PACKET FEATURE METADATA USED BY TRAFFIC ANALYSIS APPLICATIONS

Traffic Analysis Application		Packet Feature Metadata			
		Size	*Interval*	*Direction*	*Headers*
Website fingerprinting	AWF [1]			✓	
	TF [2]			✓	
Botnet detection	PeerShark [19]	✓	✓		
	ODCLA [20]	✓	✓		✓
Covert channel detection	MPTD [3]	✓	✓	✓	
	NPOD [21]	✓	✓	✓	✓
Intrusion detection	Kitsune [5]	✓	✓	✓	
	HELAD [22]	✓	✓	✓	
Application fingerprinting	CAAB [23]	✓		✓	
	RSAI [24]	✓		✓	

applications often rely on a range of coarser-grained but more complicated statistical features. Besides, there are also tasks that require semantic information derived from packet header fields. Because of such diversity and specificity, devising a proper feature set for a given traffic analysis task typically involves specialized expert knowledge and careful tuning.

Our design of general traffic feature representation is inspired by an extensive investigation on state-of-the-art traffic analysis applications. We observe that four kinds of fundamental packet feature metadata are vital to profile the behavior of network traffic. *Packet size* and *packet time interval* are two key elements that can describe the spatial and temporal characteristics of the traffic. *Packet direction* is a strong representation of the bidirectional communication pattern, while some specific *header fields* provide crucial information on the protocol semantic level. Based on the above packet-level metadata, traffic analysis applications usually compute a set of high-level features (e.g., max, mean, etc.). The metadata picked by different traffic analysis applications are summarized in Table I. To make our design general, we choose to use the neural network to automatically learn discriminating high-level features from the feature metadata. Previous works also proved that the front-part packets of flows are sufficient to distinguish abnormal traffic effectively [25], which enables the early detection of malicious behaviors. Therefore, we form our traffic feature representation as a sequence of metadata extracted from the first several packets of a network flow. Specifically, our traffic feature representation F is denoted as follow

$$F = \{M_k\}_{k=1,2,\ldots,n} \tag{1}$$

where M_k is the metadata unit of k'th packet, composed by

$$M_k = \{s_k, i_k, d_k, h_k\} \tag{2}$$

where s_k, i_k and d_k denote the size, time interval and direction of the packet, and h_k denotes the selected header fields.

This traffic feature representation establishes a simple yet cost-effective feature extraction phase for MULTA. It is easy to be deployed even on an edge platform with limited resources (e.g., IoT gateway). And the feature form of sequence-based metadata is also an informative traffic feature representation that can be shared among several tasks.

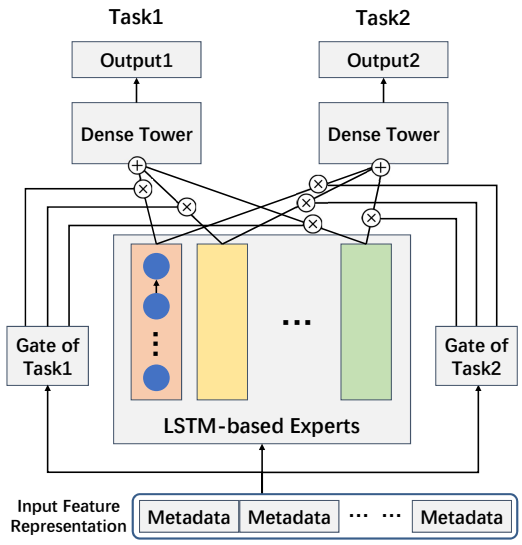

Fig. 2. The LMMoE used in MULTA.

B. Cost-efficient LSTM-based Multi-gate Mixture-of-Experts

MULTA employs LSTM-based MMoE network to seek better cost-efficiency without harming the detection efficacy in multi-task scenarios. MMoE [16] is proposed to model complicated relationship among multiple tasks. Along with the idea of ensemble learning, the cornerstone of MMoE comprises a set of expert networks, which hold common and individual knowledge of tasks. The MMoE model we used is shown in Fig. 2. The experts learn different aspects of the hidden representation from the multi-task training data, thus producing differentiated outputs. And the outputs of experts are weighted by a series of task-specific gates. The ensemble of experts, along with the gates, takes the traffic feature as input. Since our traffic feature representation encodes the sequential order of packets intrinsically, we use LSTMs as the expert networks to exploit the temporal information of the metadata sequence (i.e., the LSTM-based MMoE, LMMoE). For the output of the model, there is a simple header for each task connected to the gated experts, which is composed of towered dense layers and an output layer.

MMoE enables a moderate parameter sharing paradigm among the target tasks. Benefiting from the flexibility of multiple independent experts, different tasks can hold different hidden representations inside the network. Suppose that the tasks are aggregated into several clusters in the latent space, MMoE can preserve the adjacency by parameter sharing within an expert, and keep enough tolerance for task variety among the isolated experts. The gates are trained to selectively combine the results of experts together and produce task-specific insights for the output headers, allowing tasks to consult experts differently.

Multi-task traffic analysis is a cost-intensive scenario that requires the ML model to hold its efficacy even with limited size. However, we usually do not have quantitative knowledge about the internal relevance of traffic analysis tasks, which prevents us from determining the size of the model *a priori*. Enlightened by the structure of MMoE, we propose an

Algorithm 1: The model size tuning on a task set

Input:
A task set S_t with cardinality N,
Expected single-task accuracy a_{target},
Acceptable multi-task accuracy drop a_{drop},
Expert size initial search point s_{start}.
Output:
Expert size s_e,
Experts number n_e.

begin
/* Search for expert size s_e. */
$s_e \leftarrow 0$;
for *task in* S_t **do**
$s \leftarrow s_{start}$;
$lb \leftarrow 0$, $ub \leftarrow s_{start}$;
do
$a \leftarrow accuracy\_of$(LSTM, s, $task$);
if $a < a_{target}$ **then**
$lb \leftarrow ub$, $s \leftarrow 2 * ub$, $ub \leftarrow 2 * ub$;
else
$s \leftarrow (lb + ub)/2$;
end
while $a < a_{target}$;
if $ub < s_e$ **then**
continue; // Pruning.
end
$bisearch$(LSTM, s, (lb, ub), $a \geq a_{target}$);
// Pruning when $ub < s_e$.
$s_e \leftarrow max(s, s_e)$;
end
/* Search for experts number n_e. */
$n_e \leftarrow N/2$;
$bisearch$(LMMoE, n_e, $(1, N)$,
$a \geq a_{target} - a_{drop}$);
end

function $accuracy\_of(model, size, task)$:
/* train *task* on *model* sized *size* and get the final accuracy. */

function $bisearch(model, size, range, cond)$:
/* conduct binary search to find minimal *size* in *range* that satisfies *cond* by training *model*. */

optimization algorithm to automatically tune the size of the model for MULTA. In the LMMoE model used by MULTA, we need to specify two critical hyper-parameters that control its volume, i.e., the size of an LSTM expert and the number of LSTM experts. They have different effects in the context of multi-task traffic analysis. An LSTM expert is basically a stand-alone component in MMoE, and its size determines the capability to handle intra-task knowledge. On the contrary, the number of LSTM experts is tuned to adjust the tolerance of the inter-task differences. There is a trade-off we should notice. To retain enough detection efficacy, the values of these parameters should not be too low. But higher values also incur the expansion of model size as well as costs. Searching exhaustively for the best values of these parameters is laborious. Our algorithm aims to efficiently find the near-optimal hyper-parameters considering the balance between detection efficacy and cost.

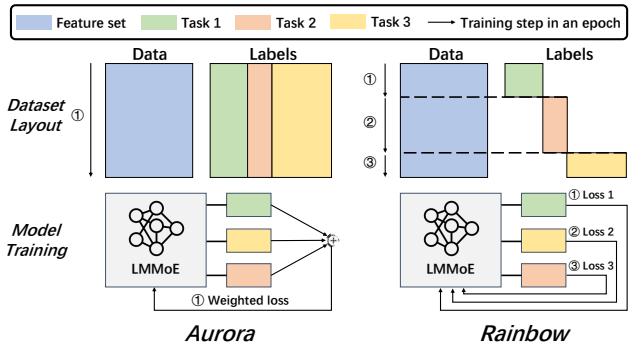

Fig. 3. Two dataset layouts and corresponding training paradigms.

The algorithm is substantially based on an observation that the task whose complexity can be handled by a single expert will at least not perform worse under multiple experts. Then we can decouple the size of a single expert and the number of experts. First, we search for the minimal size of an LSTM expert that can still hold the most informative task. Given an expected single-task accuracy, we train the tasks respectively on an auxiliary single LSTM expert model. For each task, we use binary search to find the smallest size of the LSTM expert that can meet the accuracy requirement. To further reduce the search overhead, the recorded lower bound of previous tasks can be leveraged to prune the search space in the search procedure of subsequent tasks. Once the size of an LSTM expert is determined, we conduct a second step to find the set of experts with minimal cardinality but can still cover the task relationships. In this step, we use the LMMoE and fix the size of a single LSTM expert to the value found previously. We train all the tasks simultaneously and use binary search to find a minimal number of experts that guarantee the accuracy of every task. We limit the upper bound in this step to the number of tasks in order to stay cost-efficient while leaving enough headroom. More details are presented in Algorithm 1. Notice that the task may be not able to reach the target accuracy in the first step, the algorithm can early stop when the accuracy doesn't rise along with the expert size.

C. Flexible Training Paradigms

We propose two paradigms to train MULTA, namely *Aurora* and *Rainbow*. They can cope with two typical deployment scenarios. As the layouts of datasets vary, the model training approaches also diverge. We illustrate these two cases in Fig. 3.

In *Aurora*, the tasks are tightly coupled by a fused dataset. Every data item in the dataset is multi-labeled by all tasks. This is a typical way to establish a multi-task traffic analysis dataset. For example, a network operator may dump raw traffic from the monitor point, and label them using a variety of analysis tools. This kind of methodology promises better consistency between the model training and deployment contexts. In the training phase of *Aurora*, the entire dataset is fed to the model,

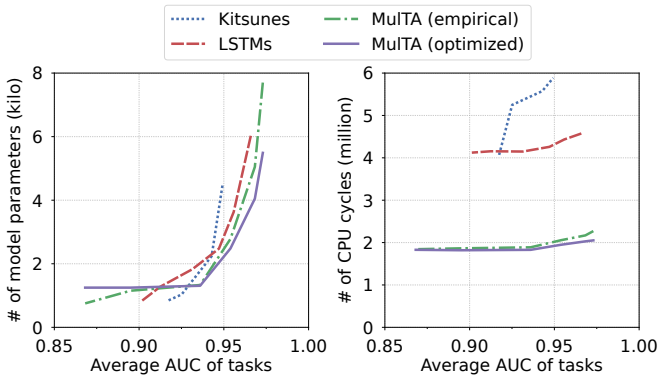

Fig. 4. The AUC-cost balance curves of different model combinations (lower right is better in both subfigures).

and the model learns from all the tasks simultaneously. The losses of tasks are weighted to form a global loss to drive the optimization step.

In *Rainbow*, however, the origination of the multi-task dataset can be different. One may acquire single-task datasets from various sources for each task, and then merge them to build a huge striped dataset to perform multi-task training. For example, network operators may utilize some open-source datasets – which can be more representative and up-to-date – to equip their model with stronger defense capabilities. And these datasets are usually customized to cover a specific task. When a multi-task dataset is merged with several single-task datasets, each task can only have available labels on part of the data items, so we choose another training paradigm *Rainbow* to update the model. In the training phase of *Rainbow*, we split the multi-task dataset into several minisets and ensure the labels within one miniset all come from the same task. Then in a single epoch, we train the model on each miniset in a round-robin manner, while freezing the individual header layers of the tasks that the current miniset does not involve. Given that the latecomer tasks may pull down the detection accuracy of former tasks, one can increase the number of epochs to let the model better recognize the task commonalities to reduce the jitter.

IV. Evaluation

We implement MulTA with Python. The prototype consists of a feature extractor and an LMMoE model. The feature extractor can extract feature vectors from raw network traffic. We build the LMMoE model with Keras (Tensorflow backend). The server we use to train and deploy MulTA is equipped with Intel(R) Core(TM) i7-10700F CPU and NVIDIA GeForce GTX 1650 GPU.

The datasets we use are from a state-of-the-art intrusion detection application, Kitsune [5]. Kitsune performs a series of attacks on its testbed which consists of an IoT network and a surveillance network, and then records the attack traffic. We choose three detection tasks from all of the Kitsune attack behaviors to form our target task set, i.e., *SSDP_Flood*, *SSL_Renegotiation* and *SYN_DoS*.

TABLE II
The detection AUCs of two paradigms during the training process.

Epochs	*2*	*4*	*6*	*8*	*10*
Aurora	0.9358	0.9429	0.9633	0.9694	0.9681
Rainbow	0.9407	0.9573	0.9763	0.9710	0.9736

A. Detection Efficacy & Cost-efficiency

To illustrate that MulTA gets a better trade-off between detection efficacy and cost-efficiency under multi-task scenarios, we measure the AUC-cost balance curves of various methods. We deploy three Kitsune tasks in parallel with the following model combinations: ① three single-task Kitsune models, ② three single-task LSTMs, ③ MulTA whose model size is empirically selected and ④ MulTA whose model size is optimized with our tuning algorithm. We assume an edge scenario where only weak CPUs are available to deploy the models. Two performance metrics are used to quantify the space and time cost-efficiency: the number of model parameters and the elapsed CPU cycles of a single inference action. For model combinations ①②③, we adjust the size of models to achieve different trade-off decisions between detection efficacy (AUC) and cost-efficiency. For ④, we pick the AUC points in ③ as the target accuracy of the size tuning algorithm and also measure the final cost. The results are displayed in Fig. 4. The left subfigure shows that since MulTA enables information sharing among the three tasks, it needs fewer amount of model parameters to achieve identical detection AUC, which indicates better space efficiency. In the right subfigure, we can see that MulTA consumes fewer CPU cycles to complete an inference action, which means its time efficiency is also competitive. Such an improvement is not only brought by the reduction of model size, but also benefits substantially from the better cache affinity of our single-model multi-task architecture. Besides, it is shown that our tuning algorithm is able to further optimize the AUC-cost trade-off on the basis of empirically determined sizes of models, especially under high accuracy requirements.

B. Comparison between two Training Paradigms

We evaluate the detection performance of MulTA under the two training paradigms, *Aurora* and *Rainbow*. We organize the three single-task datasets of Kitsune in two different ways to form two multi-task datasets. For *Aurora*, we merge the three datasets into one and supplement the missing labels in other task dimensions with negative labels (i.e., mark them as "normal" traffic). For *Rainbow*, we simply stack the three datasets together. Then we train MulTA on the two multi-task datasets respectively. We record the average AUCs of three tasks during the training processes. The results are displayed in Table II. MulTA achieves pretty good detection accuracy under both *Aurora* and *Rainbow*, while the convergence speeds also show no obvious distinction. We attribute the lower AUCs of *Aurora* to the error-prone label filling procedure during the dataset merging, and this will not occur in realistic scenarios of *Aurora* since the multi-labeled datasets are collected in an identical position.

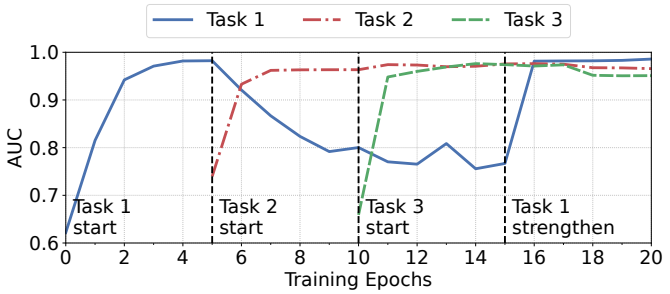

Fig. 5. Detection AUCs of original and latecomer tasks during the incremental training process.

C. Incremental Deployment

In a real deployment scenario, new security challenges constantly arise, and network operators have to append latecomer tasks to the running framework. Leveraging the three tasks of Kitsune, we evaluate MULTA on its stability upon incremental deployment. We add new tasks to an already trained model and observe the disturbance it causes to the original tasks. First, we train the model on a single task, and after it is finished we keep on training the same model on the second task, and then the third one. After these steps, we expect the model to be able to support all three tasks. We measure the AUCs of each task during the training procedure. The results are shown in Fig. 5. We can see that different tasks react differently when a new task joins. The detection efficacy of Task 2 is maintained during the training of latecomer Task 3, which is the most ideal case in our expectation. However, Task 1 shows stronger sensitivity upon the arrival of latecomers, since its detection efficacy is drastically pulled down by Task 2 and 3. Nevertheless, we use the same dataset to strengthen the training results of Task 1 (the rightmost part of Fig. 5). It is effortless for us to bring the accuracy back in merely a single epoch. This is because the learned knowledge of Task 1 is not destroyed but hidden inside the multi-task model.

V. CONCLUSION

In this paper, we identify the deployability predicament of ML-based network traffic analysis applications, which has become increasingly important due to the growing number of diverse network security threats. To address this, we propose MULTA, a multi-task traffic analysis framework to exploit the underlying relevance among different traffic analysis tasks and integrate them within a single ML model. To generalize the feature extraction process, we devise a sequence-based feature representation to meet the requirements of various tasks. We leverage the LMMoE model to learn from the input features and preserve common and task-specific knowledge, and then use a tuning algorithm to automatically find the suitable size of the model. We also employ two training paradigms – *Aurora* and *Rainbow* – to train MULTA. The evaluation shows that MULTA promises close detection accuracy and better cost-efficiency in comparison with other baseline methods when serving multiple tasks, and can also comply with the requirements of practical scenarios. We hope MULTA can help network operators to consolidate their security systems easily.

REFERENCES

[1] V. Rimmer, D. Preuveneers, M. Juarez, T. Van Goethem, and W. Joosen, "Automated website fingerprinting through deep learning," in *NDSS*, 2018.
[2] P. Sirinam, N. Mathews, M. S. Rahman, and M. Wright, "Triplet fingerprinting: More practical and portable website fingerprinting with n-shot learning," in *CCS*, 2019, pp. 1131–1148.
[3] D. Barradas, N. Santos, and L. Rodrigues, "Effective detection of multimedia protocol tunneling using machine learning," in {*USENIX*} *Security*, 2018, pp. 169–185.
[4] J. Han, C. Huang, F. Shi, and J. Liu, "Covert timing channel detection method based on time interval and payload length analysis," *Computers & Security*, vol. 97, p. 101952, 2020.
[5] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: an ensemble of autoencoders for online network intrusion detection," in *NDSS*, 2018.
[6] N. Msadek, R. Soua, and T. Engel, "Iot device fingerprinting: Machine learning based encrypted traffic analysis," in *IEEE WCNC*. IEEE, 2019, pp. 1–8.
[7] M. Zhang, J. Bi, K. Gao, Y. Qiao, G. Li, X. Kong, Z. Li, and H. Hu, "Tripod: Towards a scalable, efficient and resilient cloud gateway," *IEEE JSAC*, vol. 37, no. 3, pp. 570–585, 2019.
[8] G. Siracusano, S. Galea, D. Sanvito, M. Malekzadeh, G. Antichi, P. Costa, H. Haddadi, and R. Bifulco, "Re-architecting traffic analysis with neural network interface cards," in *USENIX NSDI*, 2022, pp. 513–533.
[9] T. Swamy, A. Zulfiqar, L. Nardi, M. Shahbaz, and K. Olukotun, "Homunculus: Auto-generating efficient data-plane ml pipelines for datacenter networks," in *ACM ASPLOS*, 2023, pp. 329–342.
[10] R. Caruana, "Multitask learning: A knowledge-based source of inductive bias," in *ICML*, ser. ICML'93. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993, p. 41–48.
[11] J. Baxter, "A bayesian/information theoretic model of learning to learn via multiple task sampling," *Machine learning*, vol. 28, pp. 7–39, 1997.
[12] Y. Lu, A. Kumar, S. Zhai, Y. Cheng, T. Javidi, and R. Feris, "Fully-adaptive feature sharing in multi-task networks with applications in person attribute classification," in *IEEE CVPR*, 2017, pp. 5334–5343.
[13] I. Misra, A. Shrivastava, A. Gupta, and M. Hebert, "Cross-stitch networks for multi-task learning," in *IEEE CVPR*, 2016, pp. 3994–4003.
[14] L. Duong, T. Cohn, S. Bird, and P. Cook, "Low resource dependency parsing: Cross-lingual parameter sharing in a neural network parser," in *ACL-IJCNLP*, 2015, pp. 845–850.
[15] Y. Yang and T. M. Hospedales, "Trace norm regularised deep multi-task learning," in *ICLR*. OpenReview.net, 2017. [Online]. Available: https://openreview.net/forum?id=rknkNR7Ke
[16] J. Ma, Z. Zhao, X. Yi, J. Chen, L. Hong, and E. H. Chi, "Modeling task relationships in multi-task learning with multi-gate mixture-of-experts," in *ACM KDD*, 2018, pp. 1930–1939.
[17] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton, "Adaptive mixtures of local experts," *Neural computation*, vol. 3, no. 1, pp. 79–87, 1991.
[18] A. Søgaard and Y. Goldberg, "Deep multi-task learning with low level tasks supervised at lower layers," in *ACL*, 2016, pp. 231–235.
[19] P. Narang, S. Ray, C. Hota, and V. Venkatakrishnan, "Peershark: detecting peer-to-peer botnets by tracking conversations," in *2014 IEEE S&P Workshops*. IEEE, 2014, pp. 108–115.
[20] M. S. Pour, A. Mangino, K. Friday, M. Rathbun, E. Bou-Harb, F. Iqbal, S. Samtani, J. Crichigno, and N. Ghani, "On data-driven curation, learning, and analysis for inferring evolving internet-of-things (iot) botnets in the wild," *Computers & Security*, vol. 91, p. 101707, 2020.
[21] L. Wang, K. P. Dyer, A. Akella, T. Ristenpart, and T. Shrimpton, "Seeing through network-protocol obfuscation," in *ACM CCS*, 2015, pp. 57–69.
[22] Y. Zhong, W. Chen, Z. Wang, Y. Chen, K. Wang, Y. Li, X. Yin, X. Shi, J. Yang, and K. Li, "Helad: A novel network anomaly detection model based on heterogeneous ensemble learning," *Computer Networks*, vol. 169, p. 107049, 2020.
[23] H. F. Alan and J. Kaur, "Can android applications be identified using only tcp/ip headers of their launch time traffic?" in *WiSec*, 2016, pp. 61–66.
[24] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic, "Robust smartphone app identification via encrypted network traffic analysis," *IEEE TIFS*, vol. 13, no. 1, pp. 63–78, 2017.
[25] Y.-s. Lim, H.-c. Kim, J. Jeong, C.-k. Kim, T. T. Kwon, and Y. Choi, "Internet traffic classification demystified: on the sources of the discriminative power," in *Co-NEXT*, 2010, pp. 1–12.