# Multibit Tries Packet Classification with Deep Reinforcement Learning

Hasibul Jamil, Ning Weng

Electrical and Computer Engineering Department Southern Illinois University, Carbondale, USA Email: {mdhasibul.jamil,nweng}@siu.edu

Abstract—High performance packet classification is a key component to support scalable network applications like firewalls, intrusion detection, and differentiated services. With ever increasing in the line-rate in core networks, it becomes a great challenge to design a scalable and high performance packet classification solution using hand-tuned heuristics approaches. In this paper, we present a scalable learning-based packet classification engine and its performance evaluation. By exploiting the sparsity of ruleset, our algorithm uses a few effective bits (EBs) to extract a large number of candidate rules with just a few of memory access. These effective bits are learned with deep reinforcement learning and they are used to create a bitmap to filter out the majority of rules which do not need to be full-matched to improve the online system performance. Moreover, our EBs learning-based selection method is independent of the ruleset, which can be applied to varying rulesets. Our multibit tries classification engine outperforms lookup time both in worst and average case by 55% and reduce memory footprint, compared to traditional decision tree without EBs.

Index Terms packet classification, machine learning, optimization

#### I. INTRODUCTION

Packet classification is a key function to support network applications like firewalls, intrusion detection, and differentiated services and OpenFlow switch. The problem of packet classification is similar to point location problem in a multidimensional geometric space. The meta data in a packet, (i.e., packet headers) contains different fields, representing different dimensions in space. Given a packet, finding out where exactly that packet is located in that space is essentially the packet classification problem. A classifier is a set of rules, each rule specifies a pattern (i.e. values or range of values) on different fields of a packet header. In this way, all these rules could be represented as hypercubes in that same space.

Existing algorithmic solutions to packet classification include decision-tree-based techniques [1] and decomposition-based techniques [2] [3]. Meanwhile, several orthogonal solutions for packet classification have explored, including [4] using the prefix probability, [5] using entropy for a compact data structure, [6] using compressing tables.

However most of these solutions are built on heuristics (e.g., increasing split entropy [1], balancing splits with custom space measures [1], special handling for wildcard rules [7]) that fails to generalize the process of building a decision tree for different set of rules. On the other hand if these solutions are specifically tuned to exploit certain characteristics present in a given ruleset, those characteristics may not be present in another ruleset. As a result, this environment (i.e., ruleset) specific heuristics typically suffers with sub-optimal performance. Another drawback of this hand-tuned heuristics is the absent of a global objective (e.g., tree depth or the number of nodes in the tree). Their decision making is often based on local information (difference between the number of rules in the current node [8], the number of different ranges in different dimensions [7]). This local information is loosely related to the global objectives and that leads to their performance to be sub-optimal.

To address above mentioned limits of heuristics based solutions, we will build a decision tree using deep learning approach. The promising aspect of deep learning in systems and networking problems [9], [10], inspires us to use deep learning in packet classification. Essentially we aim to use learning-based approach to generate an high performance and low memory packet classification engine for any ruleset without relying on heuristics.

In this paper, we present multibit-tries packet classification engine, which is generated by deep reinforcement learning. For a given set of rules, our solution employs a learningbased approach to find out the effective bits. Effective bits (EBs) are essentially the selected bit positions from a 5 field, 104-bit tuple structure, that divides the original rulesets into multiple groups. We use these EBs to retrieve a large number of possible candidate rules in just one memory access. Therefore, deep reinforcement model generates an optimized decision tree for a given ruleset and using that decision tree we calculate required effective bits. The selected effective bits are then used to traverse the original decision tree in multibit tries approach that yields up to 55% performance improvement for different set of ClassBench [11] rules.

In summary, there are three main contributions of this work:

- A deep learning based multi effective bit selection method is proposed by leveraging the statistical characteristics in a ruleset to conduct multi-dimensional lookups.
- Based on effective bit, a multibit tries packet classification engine is designed for the system scalability in both processing throughput and storage requirement
- Our multibit-tire packet classification can achieve classification time improvement up to 55% compared to unibit decision tree with small memory footprint improvement for varying ruleset.

The remainder of this work is organized as follows. Section II gives background for related research on packet classification. Section III presents the proposed method. Experimental setup and results are shown in Section IV. Finally, Section V summarizes and conclude the paper.

## II. BACKGROUND AND RELATED WORK

Packet classification is an important and hard problem [12]. Existing solutions can be classified into three categories: hardware solution, hand-turned heuristic algorithmic solutions and machine-learning based solutions. In this section, we will first review several heuristic algorithmic solutions, finally a brief review of machine learning based solutions.

Algorithmic solutions for packet classification include decomposition and decision tree. Decomposition-based solutions work on each field in a ruleset independently using crossproducts [13] [14] or header chucks [15] [2] for the intermediate results. These solutions merge the results from different fields to produce the final match results. As discussed in [16], the time complexity is O(dW) for cross-product solutions and is O(d) for header chuck solutions. The storage complexity is  $O(N^d)$  for [13] [14] [16] and  $O(dN^2)$  for [2]. Decision-treebased approaches [17] [18] [19] analyze all fields in a ruleset to construct tree data structures for an efficient packet header lookup. Tree depth and rule duplication in a decision tree affect the searching efficiency and memory requirement of the implementation. For the matching process, decision-tree-based solutions traverse the tree using field values to make branching decisions at each node until a leaf is reached. According to the discussion in [16], the growth of field numbers in a ruleset results in a linear increase of processing latency and the time complexity is O(d). Based on the nature of decision trees, the rule duplication is carried over to the next layer. The growth of field numbers in a ruleset results in an exponential increase of memory requirement and the storage is  $O(N^d)$ .

Learning based approach could be divided into two categories. One learning-based approach could be getting rid of the decision tree itself, a neural network will output the matching rule for a packet, given the packet's header fields. It has shown that a deep neural network (DNN) could be used to replace B-trees for indexing []. This approach has some serious drawbacks. First, it doesn't assure 100% accuracy which is an absolute requirement for most crucial packet classification services (e.g. firewalls, access controls). The reason for not having a 100% accuracy is due to the fact that training a neural network is essentially a stochastic process. If a DNN replaces the tree, we still need another system to verify if the DNN result is correct or not. Secondly, a packet could match multiple rules in a ruleset so, given a packet, after the first match, system still needs to go through the other rules to see if the first matching rule has the highest priority among all the matching rules or not. In addition, for a large ruleset, the required DNN models will be very large in size and very difficult to train to obtain high accuracy. The other category is rather unorthodox, involves building decision trees utilizing Deep learning (RL) which has been introduced in [20]. There, authors shows how deep

reinfocement learning could be used to generate optimized decision tree for any given ruleset. Reinforcement Learning can learn the most efficient heuristic for a given environment.

Our proposed solution falls into learning based category and we introduces an effective bit selection scheme from a decision tree and using those effective bits in mutibit tries we improve the classification performance and memory footprint per rule.

## III. METHODOLOGY

An learning based system is required to tackle the problem of generating optimal decision trees for different given environment (i.e., rulesets). Once the optimal decision tree is built, we introduce a multibit lookup scheme that generates significantly lower memory footprint and classification time. This multibit lookup scheme finds some number of bit positions in the packet header (i.e., ith & jth bit of SrcIP and/or mth & nth bit of DstIP) to generate a bitmap and essentially enables to retrieve a large number of classifier rules in just one memory access. We called these bits as effective bits (EB). For a decision tree, these processes stops when all decision tree leaves have no more than *binth* (bin threshold) rules, *binth* controls the amount of linear searching at the end of the tree search.

For a decision tree, we reduce the size of the tree by truncating a non-leaf node only if its total number of rules exceeded above a given group-threshold. We named this threshold as group-threshold because it groups a number of internal and leaf nodes together.

In our scheme all the found effective bits are concatenated to form index of a lookup table. This lookup table is pre-computed and stored in memory, so for an incoming packet the value from the effective bit positions is calculated and use that value as index to search in the lookup table. Doing this, our scheme filters out the majority of non-related rules and only conduct costly matching against highly related matching rules.

## A. Packet Classification Engine Overview

As shown in Figure 1, a RL system consist of an agent and an environment, where agent repeatedly interact with the environment. The environment consists of a set of rules and a decision tree. Environment provides the current state  $St \in S$ which correspond to the current status of the decision tree. The agent receives this state information and uses a DNN model to choose an action  $At \epsilon A$ , i.e. cut or partition based on a policy. The state and action space are defined in the environment itself. A cut action divides a node along a chosen dimension (i.e., one of SrcIP, DstIP, SrcPort, DstPort, and Protocol) into a number of sub-ranges (i.e., 2, 4, 8, 16, or 32 ranges), and creates that many child nodes in the tree. A partition action divides the rules of a node into disjoint subsets (e.g., based on the coverage fraction of a dimension), and creates a new child node for each subset. Depending on the action taken by the agent, the environment also provides a reward signal  $R_t$ . Here, the goal of the model is to learn an optimized single policy  $\pi(a \mid s)$ , where a is the action and s is the given state so that the cumulative reward after building the tree is maximized. These steps repeated for next time step (t+1) and the tree



Fig. 1: Packet classification engine.

build up incrementally. In summary, this decision tree building process could be casted as RL problem: the environments state is the current decision tree, an action is either cutting a node or partitioning a set of rules, and the reward is the classification time, memory footprint, or a combination of these two. Agent starts with an initial random policy, evaluates this policy with several roll-outs and then update the policy from the rewards of the roll-outs. A roll-out is a sequence of actions that builds a decision tree. All these actions are driven by a policy and a reward is received after completion of the tree. This process continues until the reward matches with the objective value.

## B. Decision Tree via Deep Reinforcement Learning

One interesting fact that could be leveraged on is that the action on a node is entirely depends on the node state itself not the state of the tree. If the sub-tree rooted at a node could be optimized, recursively the tree rooted from root node could be optimized (e.g., the memory access time and memory footprint of the tree could be optimized). The worst condition classification time is essentially the height of the tree considering the matching rule is in the farthest leaf node. And the memory footprint is directly related to the number of nodes in the tree.

The reward signal (Rt) accommodate these two requirements for an action taken to optimize the global objective function of building a performance and memory optimized tree. In this problem formulation, the environment is considered as a series of 1-step decision problems, each step yielding a reward. We call this secondary award and the actual or primary reward for these 1-step decisions is calculated upon completion of relevant sub-tree. Calculation of rewards are done not by summing over time but aggregating across tree branches. This is shown in Figure 2.

Considering a root node S0 in Figure 2, based on current policy, the agent decides to take action a1 to split S0 into S1, S2. Of these child nodes, only S1 needs to be further split (via a2), into S3, S4 and S5. S2, S3, S4 and S5 are leaf nodes. The experiences collected from this roll-out consist of two independent 1-step roll-outs: (S0, a1) and (S1, a2). The total reward R for each roll-out would be -3 and -2 respectively. It is important to mention that there is O(log(n)) delay between action and reward signal in this approach (where n is the total number of nodes of the tree).



Fig. 2: Illustration of reward of machine learning.

## C. Effective Bit Derivation

Once the optimized decision tree for a given ruleset is built, we could further improve its performance by incorporating an idea called concatenated EBs. The process of find the concatenated EBs is shown in Algorithm 1 where a decision tree is given as the input. Each node of the tree has its attribute objects (i.e., number of child nodes, number of total rules, a special variable called *EBSetvalue* storing bit positions pointing that node and its parent node.) All the nodes pointers are arranged in such a way that a depth-first-search (DFS) could be performed (line 1-2). Every node of the tree is then traversed and checked for the selected attributes (line 3-8). Each node contains ranges of five tuples (i.e. min and max value of each dimension) covering all its rules. Whenever a cut action is done to a non-leaf node, the spawned nodes EBSetvalue variable is updates with corresponding bit positions (line 9). To illustrate this operations., let's consider an example. Given a node n, when we cut it to spawn other nodes, if n's state are SrcIPMin, SrcIPMax, DstIPMin, DstIPMax, SrcPortMin, SrcPortMax, Dst-PortMin, DstPortMax, ProtocolMin, ProtocolMax (e.g., all the rules n contain, SrcIPMin is the lowest IP among all the source IP in the ruleset and SrcIPMax is the maximum), spawning to child nodes could be represented by bit positions as shown in Figure 3. Source IP is a 32 bit number and following example illustrate the idea of finding effective bits in more detail. If a node (S1) with following bound spawned to 4 new nodes (S2, S3, S4, S5) and all the numbers are 32 bit in length.

 $\begin{array}{l} (S1)1073741824 (SrcIPMin), 2147483648 (SrcIPMax)\\ (S2)1073741824 (SrcIPMin), 1342177280 (SrcIPMax)\\ (S3)1342177280 (SrcIPMin), 1610612736 (SrcIPMax)\\ (S4)1610612736 (SrcIPMin), 1879048192 (SrcIPMax)\\ (S5)1879048192 (SrcIPMin), 2147483648 (SrcIPMax)\\ \end{array}$ 



Fig. 3: Illustration of effective bit selection. This action of spawning 4 different nodes could be rep-

resented by bit positions i and j, where  $\forall i, j$  the following condition must be true  $0 \le i, j \le 103$  as shown in Figure 3. So *EBSetvalue* variable of *S*2, *S*3, *S*4, and *S*5 will be updated with bit position i, j.

Algorithm 1: The pseudo code to extract each nodes representational bit positions

input : decision tree							
output: decision tree with every non-root node							
represented by a set of bit positions s, where							
$0 \le$ every member of $s \le 103$ or updated							
EBSetvalue variable							
// start from root node $S_{ m 0}$ of the							

- given decision tree
- 1 Fetch pointers of all the nodes in the tree ;
- 2 Arrange the node pointers to do depth first search;
  - // EBSetvalue is a object for every
    nodes containing that nodes
    representing bit
- 3 for all the nodes  $S_i$  do
- 4 **if**  $S_i$  is non-leaf **then**
- 5 Find number of children of  $S_i$ ;
- 6 Find dimensions to cut;
- 7 Find range of each dimensions;
- 8 Find bit positions required to represent current node's child nodes;
- 9 Update each child nodes *EBSetvalue* with the bit positions
- 10 end
- 11 end

Algorithm 2: The pseudo code to truncate selected nodesinput: decision tree, group-binth( $th_G$ )

output: truncated decision tree

- // start from root node  $S_0 \ {\rm of}$  the given decision tree
- 1 Fetch pointers of all the nodes in the tree ;
- 2 Arrange the node pointers to do depth first search;
- 3 for all the nodes  $S_i$  except root node do

10 end

## D. Multibit Trie

After building the memory, time or both optimized decision tree, we could further reduce the memory and classification time by introducing multibit tries scheme. In order to give a clear explanation of this scheme, a ruleset as shown in Table I with 3 field which is extracted from 5-field ruleset could be used. The multibit tries scheme is illustrated in Figure 4 and Figure 5 and subsequent lookup table construction is shown in Figure 6.

Rules	Field 1	Field 2	Field 3
$r_1$	0010	1101	1001
$r_2$	1001	000*	100*
$r_{16}$	1011	010*	101*

TABLE I: An example ruleset of 16 rules with 3 fields

As shown in Figure 4, the decision tree consists of a root, terminal or leaf and non-terminal nodes. Algorithm 2 describe the actions required to do the proposed multibit tries. It takes decision-tree and group-binth $(th_G)$  as input and output a truncated version of the decision tree. All the tree nodes pointers are stored so that they could be traversed in DFS manner(line 1-2). Every non-leaf binth except root is then compared with given  $th_G$  for the number of rules attribute (line 4-5). In line 6, if the condition is true, the selected node's pointer from it's parent is reconfigured with selected node's parent's to it's children pointers. After that we delete that node from the tree (line 7).

Following example illustrates this process. S1 is the root and S2, S3, S4, S5 are spawned by cutting S1 in any 5 of the dimensions/ fields or combination of them. For this specific scenario, this cut could be represented by bit i, j. In Figure 6 it is shown that i is field 2 bit1, and j is field 2 bit2.

In tree's next level, S2, S3 and S4 are again divided into new node pairs of (S6, S7), (S8, S9), (S10, S11) respectively. This individual cuts could be represented by bit k,l & m, where k is field 1 bit 3, l is field 3 bit 0, m is field 3 bit2. Multibit tries scheme enables us to concatenate bit i, j, k, l&m and create a direct relationship between root node S1 and S6, S7, S8, S9, S10, S11 as shown in Figure 5. One important thing to mention here is we select to shrink S8 and S11, but not S12, 13 or S18 because S8 and S11 contains number of rules more than a given threshold. This threshold is called group-binth. On the other hand, by binth, we represent the threshold number of rules that we used to decide a terminal or leaf node is reached or not. So, introducing group binth concept enables us to build a tree with less memory footprint and lower depth. The reason for lower memory footprint is because we can eliminate intermediate non-terminal nodes in final tree. This also enables to lessen the tree depth which essentially enables a smaller number of memory access to reach of leaf node.

## IV. RESULTS

## A. Experiment Setup

We used python to build the tree environment, that is the tree and all the actions. We used Proximal policy optimization (PPO) [21] along with actor-critic algorithm as described in [20] was used to generate the optimized trees for the rulesets presented in the result section.





Fig. 5: Multibit tries derivation from decision tree in Figure 4. Note: the don't care is omitted in the illustrating tree. For example, the edge from State S1 to S5 should be ijxxx.

## B. Memory Footprint Result

Figure 7 illustrates the memory footprint for different leaf-binth values for different rulesets. It clearly shows that the higher binth (leaf-binth) value yields a lower memory footprint. This is because higher binth value implies that during construction of a tree, the leaf nodes could be achieved in earlier state (i.e., with fewer cuts) than for a lower binth value, making the total number of nodes in the tree smaller.

## C. Performance Result

Figure 8 describes the effect of varying group-binth in classification time (tree depth) for different rulesets. If we consider to average all the nodes depth (i.e. average tree depth), we will get results described in figure 9. With the higher groupbinth value, the less number of non-terminal nodes will be



Fig. 6: Methodology of lookup table construction for multibit trie.



Fig. 7: Memory footprint (bytes per rule) for different leaf binth value with group binth equaling to 160.

truncated from the decision tree compared with lower-binth. For this reason, with increase in group-binth, both the worst and average case performance decreases (i.e. the generated tree depth for worst case and average tree depth for average case increases).

## D. Comparisons with Decision Tree

The performance gain and memory footprint with multibit tries scheme is described in Table II. With leaf-binth =16 and group-binth =40, both in worst and average case,the



Fig. 8: Classification time (tree depth) for varying group binth value. The leaf binth value is 16.

	memory footprint (bytes per rule)		# of worst case memory accesses		# of average case memory accesses	
ruleset	decision tree	multibit decision tree	decision tree	multibit decision tree	decision tree	multibit decision tree
acl3_1k	332.55	319.91	9	5	4.52	2.98
acl4_1k	311.93	296.88	11	8	6.81	3.03
acl5_1k	21.13	19.75	9	4	5.43	2.69
acl5_10k	23.204	21.29	18	10	8.03	3.01
ipc1_1k	185.68	178.03	10	7	5.19	3.009
ipc2_1k	182.62	172.008	14	8	6.15	3.34

TABLE II: Performance & memory requirement for Decision Tree and multibit tries decision tree with leaf-binth value=16 & group-binth value=40



Fig. 9: Average classification time (average tree depth) for varying group binth value. The leaf binth value is 16.

classification time, and memory footprint are improved with our multibit tries scheme. From the table II, should note that the memory footprint is not significantly lower in multibit tries scheme. This is due to the fact that in multibit tries scheme, we essentially truncate some selected non-leaf nodes. Compare to the size of a leaf node, a non-leaf node has significantly lower memory requirement as non-leaf node doesn't have any rules stored in them and only contain it's child nodes pointers and some identification and status variables. All these enables multibit tries scheme to achieve up to 55% better performance . This is significant improvement in performance in expense of pre-computing the group lookup table (as shown in Figure 6). V. CONCLUSION

In this paper, we present a learning-based packet classification algorithm and evaluate its performance for varying ruleset. By exploiting the sparsity of rulesets, our algorithm uses a few effective bits to divide a large ruleset into multiple subsets with low rule replication for a lower memory usage at the offline stage. These effective bits are first selected by deep reinforcement learning and then are concatenated based on group binth. Using these effective bits, our algorithm can filter out the majority of rules which do not need to be full-matched to improve the online system performance.

Our multibit tries classification engine outperforms lookup time both in worst and average case and memory footprint, compared to traditional decision tree without EBs. The performance gain is due to multibit tries enables the classifier to traverse the decision tree several level at a time. The memory reduction is due to multibit tries allows us to truncate some selected non-leaf nodes in the decision tree. Preliminary evaluation of small size of ruleset is one limitation of this work. Nevertheless, we believe that our multibit tries is an important step towards learning based high-performance packet classification solution.

#### REFERENCES

- [1] P. Gupta and N. Mckeown, "Classifying Packets with Hierarchical Intelligent Cuttings," *IEEE Micro*, vol. 20, no. 1, pp. 34–41, 2000.
- [2] T. V. Lakshman and D. Stiliadis, "High-speed Policy-based Packet Forwarding Using Efficient Multi-dimensional Range Matching," in *Proc.* ACM SIGCOMM, 1998, pp. 203–214.
- [3] F. Baboescu and G. Varghese, "Scalable Packet Classification," in Proc. ACM SIGCOMM, 2001, pp. 199–210.
- [4] O. Rottenstreich and J. Tapolcai, "Lossy compression of packet classifiers," in *Proc. ACM/IEEE ANCS*, 2015, pp. 39–50.
- [5] G. Antichi and etc., "Ja-trie: Entropy-based packet classification," in Proc. IEEE HPSR, 2014, pp. 32–37.
- [6] O. Rottenstreich and etc., "Compressing Forwarding Tables," in Proc. IEEE INFOCOM, 2013, pp. 1231–1239.
- [7] S. Singh and etc., "Packet classification using multidimensional cutting," in SIGCOMM, 2003, pp. 213–224.
- [8] C. Hsieh, N. Weng, and W. Wei, "Scalable many-field packet classification for traffic steering in SDN switches," *IEEE Trans. Network and Service Management*, vol. 16, no. 1, pp. 348–361, 2019.
- [9] M. Dong, T. Meng, D. Zarchy, E. Arslan, Y. Gilad, B. Godfrey, and M. Schapira, "PCC vivace: Online-Learning congestion control," in 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18). Renton, WA: USENIX Association, Apr. 2018, pp. 343–356. [Online]. Available: https://www.usenix.org/conference/nsdi18/ presentation/dong
- [10] H. Mao, R. Netravali, and M. Alizadeh, "Neural adaptive video streaming with pensieve," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 197–210. [Online]. Available: https://doi.org/10.1145/3098822.3098843
- [11] D. E. Taylor and J. S. Turner, "ClassBench: A Packet Classification Benchmark," *IEEE/ACM ToN*, vol. 15, no. 3, pp. 499–511, 2007.
- [12] D. E. Taylor, "Survey and Taxonomy of Packet Classification Techniques," ACM Computating Surveys, vol. 37, no. 3, pp. 238–275, 2005.
- [13] D. E. Taylor and J. S. Turner, "Scalable packet classification using distributed crossproducing of field labels," in *Proc. IEEE INFOCOM*, 2005, pp. 269–280.
- [14] V. Srinivasan and etc., "Fast and Scalable Layer Four Switching," in Proc. ACM SIGCOMM, 1998, pp. 191–202.
- [15] P. Gupta and N. McKeown, "Packet Classification on Multiple Fields," ACM SIGCOMM Computer Communication Review, vol. 29, no. 4, pp. 147–160, 1999.
- [16] G. Pankaj and M. Nick, "Algorithms for Packet Classification," *IEEE Network*, vol. 15, no. 2, pp. 24–32, 2001.
- [17] A. Kennedy and X. Wang, "Ultra-High Throughput Low-Power Packet Classification," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 22, no. 2, pp. 286–299, 2014.
- [18] H. Lim and etc., "Boundary Cutting for Packet Classification," *IEEE/ACM Transactions on Networking*, vol. 22, no. 2, pp. 443–456, 2014.
- [19] B. Yang and etc., "Practical Multituple Packet Classification Using Dynamic Discrete Bit Selection," *IEEE Transactions on Computers*, vol. 63, no. 2, pp. 424–434, 2014.
- [20] E. Liang, H. Zhu, X. Jin, and I. Stoica, "Neural packet classification," in *Proceedings of SIGCOMM* '19, 2019, pp. 256–269.
- [21] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.