Tackling Deployability Challenges in ML-Powered Networks

Noga H. Rotman
The Hebrew University of Jerusalem

1 Introduction

Following the success of Machine Learning (ML) in various fields such as natural language processing, computer vision and computational biology, there has been a growing interest in incorporating ML into the networking domain [5, 6, 14, 4, 9]. Today, ML-based algorithms for prominent networking problems such as congestion control, resource management and routing, perform very well when their training environment is faithful to the operational environment, achieving state-of-the-art results when compared to traditional algorithms. However, the adaptation of these algorithms to function in production environments has not been straightforward, as real-world networks may differ greatly from the data used for training, leading to a drop in performance when unleashed into the wild.

This paper provides an overview of the problems impeding the successful deployment of ML-powered networks. It categorizes proposed solutions into three types, based on the main concern they address. Notably, each category takes place at different stage of the lifecycle of an ML-powered network: in-training, pre-deployment, and online, allowing to employ all three in tandem. We propose a holistic approach to tackling the challenges facing a successful deployment, by intervening at *all* three stages, and integrating the observations obtained at each stage to improve the others.

1.1 Example: ML-based algorithm in the wild

ML-based networking protocols have been very successful when their training environment and test environment match. What happens when this is not the case?

To demonstrate the impact to performance that may occur when the training environment of an ML-powered algorithm differs from the deployed environment, we consider a deep learning solution to the timely problem of adaptive video streaming (ABR) in HTTP-based video streaming.

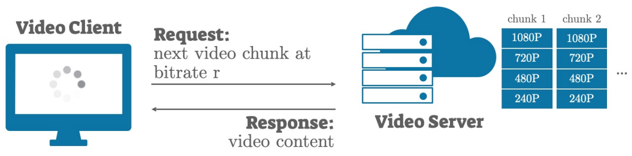

Figure 1: ABR problem overview

In ABR (see Figure 1), a client interacts with a video server. Videos are stored on the server as chunks of (roughly) the same length, each chunk available in different bitrates, corresponding to its quality. When the client is watching a video, they need to request the next chunk at a specific bitrate r. The ABR algorithm, running at the client, is responsible for choosing which bitrate r should be requested from the server. The algorithm must walk a thin line between selecting a lower resolution that may not be satisfactory to the client, and choosing a higher one that may force the client to wait for content while streaming (an event also known as rebuffering), as both effects are known to play an instrumental role in user engagement [1].

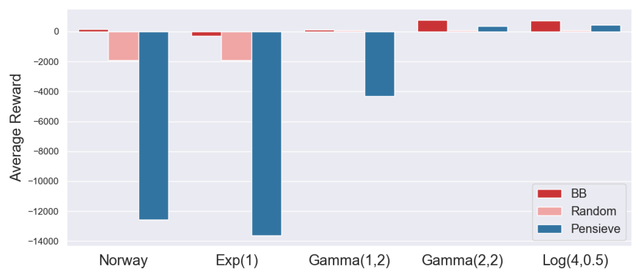

Figure 2: Average reward of an ML-based algorithm, Pensieve, trained on a dataset collected in Belgium, compared to that of BB, a non-learning algorithm, and a random algorithm. When tested on a dataset collected in Norway and two synthetic datasets, the Pensieve agent performs worse than the random algorithm. Figure 2a from [10].

Pensieve [6] applies Reinforcement Learning (RL) [11] to ABR. In [10], we trained Pensieve agents on different datasets, both real-world and synthetic. These agents performed well when the test and training environments were drawn from the same distribution. We then tested their performance when the training and test environments differs. Figure 2 contrasts the performance of a Pensieve agent trained on a dataset collected in Belgium, against two algorithms: Buffer-Based (BB) [3], a manually-crafted and widely deployed protocol, and an algorithm which chooses the next bitrate randomly. In all cases, the performance of BB is better than the one of exhibited by the Pensieve agent. Furthermore, in some cases, such as a test set collected in Norway, the Pensieve agent performed *worse than the random algorithm.* These results raise concerns when considering deploying ML-powered networks.

1.2 The causes for the impact to performance in the real world

There are multiple factors contributing to the drop in performance of an ML-based algorithm in a networking production environment.

First, bad generalization is a known trait of several ML methodologies prevalent in networking, such as RL.

Second, one must consider the usage and evaluation of these algorithms. Typically, when a non-learning algorithm fails, the cause is traceable, as the algorithm's logic is clear. It is then possible to adjust it to better handle the networking conditions causing the failure. On the other hand, when an ML-based algorithm breaks down, the reason is obscured, as neural networks are painfully difficult for humans to understand. Because of this, not only are we unable to effectively investigate the cause of failures, but we are also unable to modify the algorithm to address them.

Third, modern communication systems are architecturally complex and extremely dynamic. Encompassing all possible scenarios in a training set is simply not possible, as deployment environments are too diverse.

2 Tackling deployability challenges

In this section we classify proposed solutions for mitigating the performance degradation caused by bad generalization into three categories, based on the main concern they target (see Section 1.2), and provides a short overview of each. Further discussion of these categories can be found in Section 3.

In-training enhancements target the components responsible for the initial creation of an ML-based algorithm: the data used for training, the actual training process, or both. Successfully adapting ML methodologies to the networking domain is an arduous task, as the problems these techniques were originally created for and tested on differ greatly from networking problems.

Pre-deployment analysis aims to eliminate potential problems prior to deployment by providing insights as to the actions taken by a trained ML-based networking protocol, and the reasons leading to them.

Online assurances attempts to intervene during deployment, in order to avoid a sudden drop in performance when network conditions change.

2.1 In-training enhancements

Puffer [13] is an online service streaming live U.S. TV stations. Born as a research project, it serves as a playground for evaluating and comparing various ABR protocols. The authors proposed a new ML-based algorithm to the ABR problem, where the learning agent is trained daily *in-situ*, using data obtained from its deployment environment during the last fourteen days. Puffer won the NSDI'20 community award, as the data collected is made available online.

Another notable example is Genet [12], which targets the generalization problem in *RL-based* networking protocols by focusing the training procedure on the most challenging environments, instead of choosing them uniformly at random. To do so, the authors use Curriculum Learning [8]. While this methodology has proved useful in other domains, applying it in a networking context is nontrivial, as it is unclear how to measure the "difficulty" of a network environment.

2.2 Pre-deployment analysis

In [7], the authors introduce two categories of ML-powered networking systems: *local* and *global*. Their framework, Metis, translates the trained neural network employed by a local or global system into either a decision tree or hypergraphs. Both of these representations are much easier for humans to understand and evaluate. Interestingly, this approach advocates for the deployment of the *representation* in place of the original ML-based algorithm, thus allowing to modify the algorithm running in production directly. It is worth noting that there are ML-powered networking systems that cannot be addressed using this formalism.

Formal Verification is a mathematical approach for reasoning about a neural network's behavior. It provides provable guarantees of specified requirements; for example, for ABR, one can ascertain that when the conditions of the network do not allow for high average quality, the algorithm opts for a lower resolution over constantly rebuffering. This approach is used in [2] to evaluate three proposed ML-based networking protocols. A known disadvantage of this approach is that it is hard to scale to larger neural networks. In networking, however, most ML-based algorithms involve relatively small neural networks, making this approach feasible for various ML-based networking algorithms.

2.3 Online assurances

The last technique aims to rein in the possible costs of bad generalization in a production environment by replacing the ML-based algorithm with a "safer" option, when the decisions of the former are incoherent/uncertain. The motivation for this methodology is simple: in communication networks, there are many hand-crafted protocols that have been deployed in various networks for years, some for decades. While these algorithms may not enjoy the high performance achieved by ML-based algorithms, they were designed to withstand disastrous circumstances, and are thoroughly tested "in battle". Therefore, if we were able to successfully identify *online* when an ML-based algorithm is making incoherent decisions, we would be able to provide a kind of a "safety net", by enabling to switch to a "safer" option once such a need arises. Further incentive can be found in Figure 2, demonstrating that on test sets in which the Pensieve agent fails to generalize, BB could potentially be used to improve the overall performance.

Realizing this technique requires addressing several major challenges. First, although the detection of uncertainty in the behavior of an ML-based algorithm has been explored in different contexts, no standard method has yet to emerge. Second, translating an uncertainty signal into a measurable amount that can be calculated *online*. Third, forming heuristics by which to set thresholds on the calculated value, in order to determine whether the system should switch to a "safe" algorithm.

We presented this concept in [10]. We investigated three possible signals: uncertainty in the algorithm's input, uncertainty in the actions selected, and uncertainty in the algorithm's evaluation of its future benefit from the chosen actions. We tested these signals and compared their impact on Pensieve agents. We have found that two of the tested signals show promise.

3 Discussion and conclusions

Despite encouraging advances, realizing the promise of ML-powered networks is still elusive. This paper discussed the challenges encountered during deployment of these systems, summarized current techniques, and offered a fresh perspective of these practices.

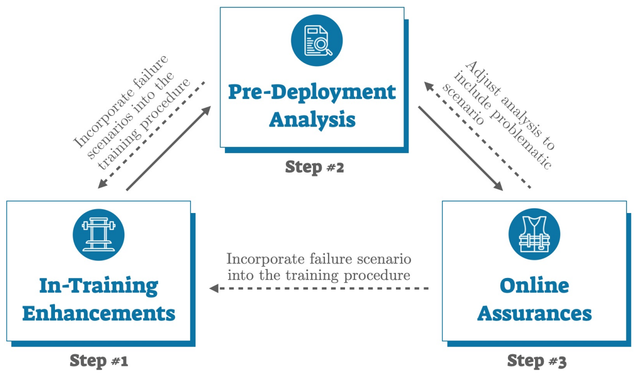
Pre-Deployment Analysis
Step #2
In-Training Enhancements
Step #1
Online Assurances
Step #3
Incorporate failure scenarios into the training procedure
Adjust analysis to include problematic scenario
Incorporate failure scenario into the training procedure

Figure 3: Each category takes place at a different phase, allowing not only to utlize all three, but continuously enhancing the algorithm based on its observable behavior at each step.

A key insight of this paper is that the categories presented in Section 2 are *complimentary*, as each occurs at a different stage of the ML-powered network pipeline: during the initial training of the algorithm, pre-deployment, and online. We claim that in order to enable the successful deployment of ML-powered networks, we must address the problems at *all* three stages (see Figure 3). First, in-training enhancements should be applied, creating a more resilient algorithm. Then, pre-deployment analysis of the resulting algorithm should be performed, which may lead to further enhancements of the training procedure and data. Finally, the system should be adjusted to provide online assurances, so that when the algorithm eventually fails in production, the reasons can be investigated and resolved while avoiding a critical hit to performance.

This approach essentially calls for a paradigm shift when considering these algorithms, from a “one shot“ scenario to an iterative process, in which the algorithm is re-examined, re-adjusted and re-tested prior to and during deployment.

4 Acknowledgments

Many thanks to the organizers of the ACM SIGMETRICS ’23 Workshop on Measurements for Self-Driving Networks at Orlando, Florida, on June of 2023, for their invitation, and to the participants of the workshop, for the engaging discussion. Special thanks to Dr. Yotam Feldman.

5 References

[1] Florin Dobrian, Vyas Sekar, Asad Awan, Ion Stoica, Dilip Joseph, Aditya Ganjam, Jibin Zhan, and Hui Zhang. Understanding the impact of video quality on user engagement. *ACM SIGCOMM computer communication review*, 41(4):362–373, 2011.

[2] Tomer Eliyahu, Yafim Kazak, Guy Katz, and Michael Schapira. Verifying learning-augmented systems. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, pages 305–318, 2021.

[3] Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. In *Proceedings of the 2014 ACM conference on SIGCOMM*, pages 187–198, 2014.

[4] Nathan Jay, Noga H. Rotman, Brighten Godfrey, Michael Schapira, and Aviv Tamar. A deep reinforcement learning perspective on internet congestion control. In *International Conference on Machine Learning*, pages 3050–3059. PMLR, 2019.

[5] Hongzi Mao, Mohammad Alizadeh, Ishai Menache, and Srikanth Kandula. Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM workshop on hot topics in networks*, pages 50–56, 2016.

[6] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. Neural adaptive video streaming with pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 197–210, 2017.

[7] Zili Meng, Minhu Wang, Jiasong Bai, Mingwei Xu, Hongzi Mao, and Hongxin Hu. Interpreting deep learning-based networking systems. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 154–171, 2020.

[8] Sanmit Narvekar, Bei Peng, Matteo Leonetti, Jivko Sinapov, Matthew E Taylor, and Peter Stone. Curriculum learning for reinforcement learning domains: A framework and survey. *The Journal of Machine Learning Research*, 21(1):7382–7431, 2020.

[9] Yarin Perry, Felipe Vieira Frujeri, Chaim Hoch, Srikanth Kandula, Ishai Menache, Michael Schapira, and Aviv Tamar. *DOTE*: Rethinking (predictive) *WAN* traffic engineering. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 1557–1581, 2023.

[10] Noga H Rotman, Michael Schapira, and Aviv Tamar. Online safety assurance for learning-augmented systems. In *Proceedings of the 19th ACM Workshop on Hot Topics in Networks*, pages 88–95, 2020.

[11] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[12] Zhengxu Xia, Yajie Zhou, Francis Y Yan, and Junchen Jiang. Genet: automatic curriculum generation for learning adaptation in networking. In *Proceedings of the ACM SIGCOMM 2022 Conference*, pages 397–413, 2022.

[13] Francis Y Yan, Hudson Ayers, Chenzhi Zhu, Sadjad Fouladi, James Hong, Keyi Zhang, Philip Levis, and Keith Winstein. Learning in situ: a randomized experiment in video streaming. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 495–511, 2020.

[14] Francis Y Yan, Jestin Ma, Greg Hill, Deepti Raghavan, Riad S Wahby, Philip Levis, and Keith Winstein. Pantheon: the training ground for internet congestion-control research. *Measurement at http://pantheon. stanford. edu/result/1622*, 2018.