The long and winding road to Deep Autonomous Networks: Lessons from real WLAN deployment

Ovidiu Iacoboaiea, Jonatan Krolikowski, Zied Ben Houidi, Dario Rossi
Huawei Technologies France SASU
{ovidiu.iacoboaiea, jonatan.krolikowski, zied.ben.houidi, dario.rossi}@huawei.com

***Abstract*—Deep Reinforcement Learning (DRL) techniques have recently gathered much attention for their ability to learn taking complex decisions in different fields: as such, they are an appealing candidate for network Operation and Management (O&M). In particular, DRL can become a fundamental item in the toolbox of the so called "self-driving networks", especially for tasks such as dynamic resource allocation, that is generally formulated and solved as complex optimization problems. Yet, training and deployment of DRL agents in real-world scenarios face important challenges, that we illustrate in this article using Wireless LANs as a relevant deployment example.**

I. Introduction

Recently, artificial intelligence (AI) techniques that train AI agents to interact with an environment to complete complex tasks with unprecedented skills have gained considerable press attention. Top stories include Google's AlphaGo [1] beating the world champion Lee Sedol in 2016, or OpenAI Five [2] winning online computer-game DOTA2 tournament in 2017. While this is not the first time that AI accomplishes such prowess – it is sufficient to recall that IBM DeepBlue defeated Chess world champion Garry Kasparov in 1997 [3] – there has been important progress in the AI algorithms under the hood.

DeepBlue attained super-human playing capabilities essentially via unprecedented search capabilities [3], resulting from a single-chip chess search engine with massive parallelism (capable of scanning 700,000 positions per second), a complex evaluation function (relying on 8000+ features to assess the move quality), and effective use of a Grandmaster game database (essentially, a book of 4000 openings hand-coded by chess Grandmaster Joel Benjamin that acted as consultant to the DeepBlue team).

Instead, tools such as AlphaGo and OpenAI Five use a radically different technology, based on a specific branch of Reinforcement Learning (RL), namely Deep Reinforcement Learning (DRL). However, while RL theory dates back to the early 70s (see [4] for a historical perspective), the recent astonishing advances of DRL have been made possible by unprecedented computing capabilities. Given the blending between the network and the cloud – which make such computing capabilities *available* – as well as the emergence of low-power deep-learning accelerators known as Tensor Process Units (TPUs) – which make such computing capabilities also *cost-effective* –, the field of communication networks is now actively seeking to exploit techniques such as DRL to automate the complex task of network Operation & Management (O&M).

Ultimately, while DRL technologies are a promising candidate to become a cornerstone for optimal decision making in network O&M, it is not without challenges. For instance, it is worth stressing that OpenAI Five [2] learned by playing over *10,000 years worth of games* against itself and that AlphaGo-Zero [1] was trained with *29 million games* of self-play during 40 days using 4 TPUs, each of which is capable of 90 Tera operations per-second (TOPS). The sheer volume of interactions is key to learning taking successful actions, through a broad and thorough exploration process: while this poses only a computational challenge in the virtual-world of computer games, successfully deploying DRL in communication networks poses additional challenges that are rooted in complexity of learning in the real world.

This article provides a background on DRL (§II-A), illustrating the challenges of DRL deployment in real-world scenarios (§II-B). Next, it focuses on a relevant example use case in the field of networking, namely resource management in Wireless LANs (§III-A), describing the necessary steps to deploy DRL in operational networks (§III-B) before briefly summarizing real-network results (§III-C). It finally summarizes the lessons learned, whose usefulness applies beyond the scope of the illustrated use-case (§IV).

II. Challenges of DRL Deployment

A. RL Background

We provide bacgkround on Reinforcement Learning (RL) and Deep RL (DRL) with the help of Fig. 1. In RL [4], an *agent* learns from interacting with an *environment*: the agent obtains a perception of the environment through a measurable *state* (e.g., parameter configuration, environment properties, etc.) and selects an *action* (e.g., changing some configuration) based on a policy that it is learning. After enforcing the action, the agent observes an updated state and receives feedback about its action, in the form of a *reward* (or a *regret*). Unlike in supervised learning (where the feedback reflects some distance from the optimum), the feedback in RL can be rather seen as praise (or critique) of the action (without any explicit information about the optimal action). Based on this feedback, the agent updates its policy, observes the new environment state, enforces a new action and so on.

While the idea behind RL has been reported since the 1950s [4], the classic RL and DRL literature is significantly more recent. Fig. 1 illustrates the most popular learning strategies that have been devised over time. Broadly speaking,

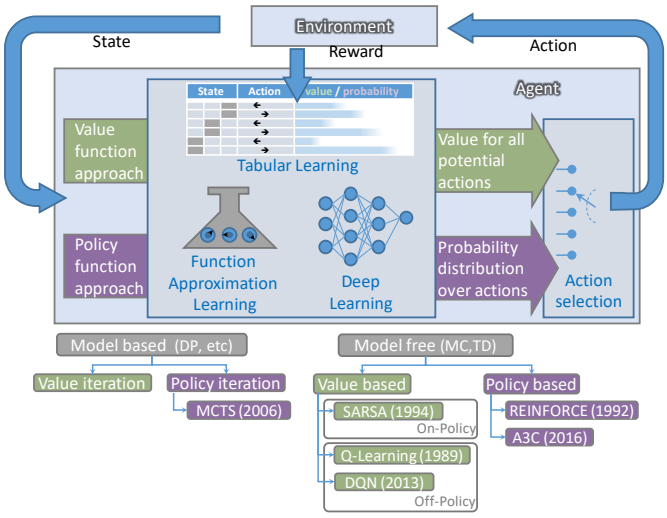

Fig. 1. Reinforcement Learning overview and taxonomy.

when RL techniques learn actions with rewards but without state (e.g. slot machines used in casinos), then the problem can be formulated as a *Multi-Armed Bandit (MAB)*, and as a *Markov Decision Process* (MDP) otherwise. The MDP formulation is based on transition probabilities from current state and action to future states. These probabilities, which model how the environment behaves, can be either known (e.g. follow a given model) or unknown: we limit our attention to the latter case, known as *model-free* RL, as it is also the most common in practice.

In model-free learning, the agent needs to learn not only the policy, but also a model of the environment. In its Monte (MC) Carlo variation, the agent runs its policy for a given period of time (called episode), collecting the raw rewards: at the end of each episode, the policy is updated. Model-free RL can be further sub-divided into *value-based* and *policy-based* methods.

1) Value-based: Roughly speaking, value-based approaches pick actions based on learned estimates of the *values* of their returns. Such a value associated to a state-action pair is often based on the estimated sum of future rewards (or regrets), with possibly a discount factor that is used to regulate the importance of instantaneous vs long-term rewards. Different approaches in this space use (a) different value functions and (b) different ways to estimate them.

a) On-policy vs Off-policy: We distinguish first between on-policy and off-policy learning. The most popular on-policy technique is SARSA (1994), whose name reminds of the State–action–reward–state–action quintuple used to learn its value function, namely: the current state and action as well as the next reward, state and action. In doing so, SARSA couples policy evaluation with the policy decision: the SARSA agent learns indeed the value function associated to the policy it is itself following (which is different from the optimal policy).

Conversely, off-policy mechanisms decouple the learned policy from the policy followed during learning. In Q-learning (1998), the most popular of such techniques, the agent follows an exploration/exploitation policy while learning an estimate of the value function (aka Q-function) that corresponds to the *optimal* policy: after convergence of the training phase, the learned policy consists in picking the actions maximizing the expected value function.

b) Tabular methods vs. approximations: RL methods also differ in the way value functions are estimated. Classic *tabular* Q-learning iteratively updates a Q-table with Q-values for each state-action pair in the entire space. It faces clear scalability problems as soon as the state-action space is large. One way to counter this problem is to learn an *approximated* Q-function. In this case, theoretical convergence guarantees do not hold anymore, but results are acceptable in practice. One recent approach is Deep Q-learning (DQN, 2013), that uses deep neural networks to learn the Q-function. DQN successfully learned to play simple Atari games (2014) and later more complex first person shooters such as Doom (2016). In this case, the neural network architecture must be adapted to the problem and nature of state at hand. For example, in the above video games, the use of Convolutional Neural Networks (CNNs) is particularly helpful since the state is represented by frame images, an input that CNNs excel at processing.

2) Policy-based: Policy-based model-free RL approaches aim at learning directly the policy function, as opposed to learning the expected reward: in simple terms, the policy maps a state directly to a given action, without the need to go via an estimation of the state-action value. Generally the policy is stochastic: i.e., for each state, a probability distribution over actions is preferred over a single deterministic action. In the learning process, the stochastic policy is often parameterized and gradient-based methods are used to approximate it. The REINFORCE (1992) algorithm is one of the earliest and most famous incarnations of such methods: subsequent improvements in convergence speed and stability were recently obtained with A3C (2016) by using an Actor-Critic architecture employing two entities, where the output of the Critic (learning a state return estimate) guides the Actor (essentially learning and producing the action).

3) The D in DRL: Deep learning offers new perspectives to leverage the power of reinforcement learning. In practice, beyond the specific reinforcement learning strategy, the decisive factor is the neural network architecture that learns the value (i.e. DQN, A3C critic network) or the policy (e.g. A3C actor network). Such neural network architectures must be adapted to learn the right representations depending on the nature of the input state: as seen earlier, a CNN is suitable when the input state consists of images, whereas a DRL-based chatbot would better use a transformer network. Similarly, various DRL based approaches to solve combinatorial graph problems used suitable neural network architectures such as Graph Neural Networks, pointer networks and graph attention networks [5]. We get inspiration from the latter architectures to design our own deep learning solution.

B. DRL deployment challenges

Most highly publicised successes of DRL do not illustrate well the complexity and challenges of applying such technology in real world systems. Unlike algorithms such as AlphaGo [1] or OpenAI Five [2] which evolve in the safe virtual environment of computer games, many DRL-based candidate systems need to interact with the physical world. A self-driving vehicle, or a self-driving network, cannot employ a trial-and-error learning strategy on the field: if things go wrong during exploration, it may have serious business (network) and potentially fatal (vehicle) consequences. The same problem applies to other domains, including industry 4.0 control systems, financial trading systems and in general any system whose actions can lead to harmful consequences. Seen from this angle, it becomes evident that applying DRL in the physical world comes with a set of additional challenges which we briefly overview in the following and that need to be accounted for in real-world usecases.

1) Training safely: The first and likely biggest challenge to overcome is the curse of failure during state exploration: DRL agents must explore the space and in doing so, it is inevitable that they sometimes fail in order to learn from the resulting regrets; however, in reality, one does not need to literally fall off a cliff to learn that it is not a good thing to do. A commonly employed safe training strategy is to learn by interacting with a digital twin, i.e., a computer-*simulated* model of the real environment. Simulator realism thus becomes a key aspect for deployment.

2) Data availability: A complementary approach is *imitation learning*, where instead of interacting with a real or simulated environment, a database of state-action traces is used for offline batch-based agent training. For instance, thanks to its fleet of several hundred thousand self-driving-ready cars, Tesla now has a huge amount of state-action pairs (over 1 billion miles with auto-pilot on alone [6]), which might be used to learn to mimic human driver behavior. However, data-driven approaches in general are vulnerable to data scarcity and quality issues – as not all actors can access such volumes of data. This fact limits the potential of training DRL agents in a data-driven way in real-world usecases tremendously.

3) Training duration: Training duration is a clear bottleneck: as earlier exemplified for games, agents need several "lifetimes" of interaction with the environment, e.g. the 10,000 years equivalent for OpenAI Five [2], which clearly is unrealistic for training on real systems. Even training offline with real data or digital twins can take a significant amount of time: for instance, it takes 70,000 GPU hours to train the full self-driving Tesla pilot prototype [6], which is around one year for a single node with 8 GPUs. In practice, training needs to be offloaded to a large fleet of data center servers equipped with GPUs and TPUs, which may be only affordable for a few big players.

4) Generalization: Another real-world challenge that DRL systems face is generalization to unseen conditions. For instance, when digital twins are used for training, realism of the simulation is of primary concern – the less faithful the simulator, the lower the quality of the agent. To overcome this problem, several techniques can be used that mitigate simulator shortcomings. For example, one way to enhance generalization is to share experience across multiple learners, as e.g., used when training multiple robots in parallel to perform tasks such as grasping[1]. Another approach is to complement simulation with emulated realistic environments. For example, AWS deepracer[2] provides an environment to train DRL driving models on simulator and, to lower the transfer gap, it also relies on 1/18 scale prototype cars with near-real conditions. A further path is to enhance model-driven simulation with data-driven complements – for example in [7], the physics-based simulator of Light Detection and Ranging (LiDaR) point clouds used for the training of self-driving vehicles is augmented with real data – a relatively low hanging fruit with potentially significant payoff.

5) Explainability and Trust: Finally, human operators need to be convinced of trusting and using DRL systems to allow their deployment. Explainability of the algorithm output helps lowering the adoption barrier. Trust may then be gained step-by-step: for instance, instead of acting directly on the real system, DRL can provide interpretable suggestions together with explanations for their triggers for the operator to validate. Google DeepMind adopted this strategy for data center cooling, where the first RL algorithm version[3] acted as a recommendation engine for the operator. Only later they moved to a fully autonomous version[4], maintaining failsafe option to revert back to human control at any time, in addition to rule-based heuristics as backup.

III. Deploying DRL for WLAN control

A. WLAN Background

We apply DRL to the real-time configuration of Wireless Local Area Networks (WLANs), aiming at automating the configuration of the network to maximize the performance under changing traffic conditions. WLANs are defined in the IEEE 802.11 standard and its amendments. The most popular setup is infrastructure-based, where user devices (called stations or STAs) such as smartphones or laptops connect to a fixed Access Point (AP) that typically acts as a gateway to the Internet. In office buildings or on university campuses, STAs can access a fleet of APs distributed over a large area. The APs are governed by a central Access Controller (AC) that coordinates AP configurations. The left portion of Fig. 2 shows a schematic representation of such a campus WLAN. Learning optimal AC decisions is the DRL usecase we focus on in this paper. In the following, we describe the most important actions in WLAN management, with examples of classic WLAN management algorithms (generally based on heuristics) that exploit these actions.

[1]https://ai.googleblog.com/2016/10/how-robots-can-acquire-new-skills-from.html accessed on 13.04.2021

[2]https://aws.amazon.com/deepracer accessed on 13.04.2021

[3]https://deepmind.com/blog/article/deepmind-ai-reduces-google-data-centre-cooling-bill-40 accessed on 13.04.2021

[4]https://deepmind.com/blog/article/safety-first-ai-autonomous-data-centre-cooling-and-industrial-control accessed on 13.04.2021

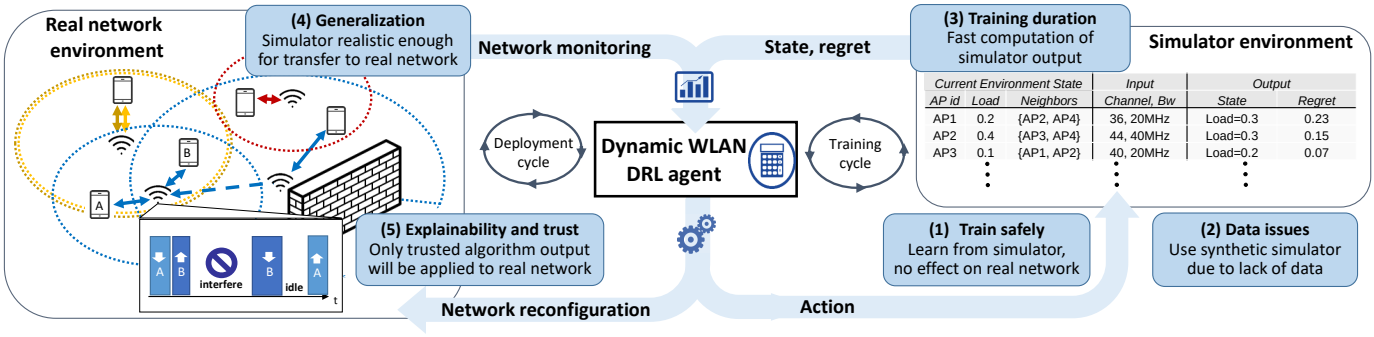

Fig. 2. Background on WLAN environment, and overall schema for the DRL training and deployment cycles.

1) Primary Channel Selection and Bonding: Each AP is configured to use a specific *primary channel* (within the 2.4GHZ or the 5GHz band), performing downlink and uplink transmissions in a half-duplex manner. In Fig. 2, the channels are represented by the colors around the respective coverage areas. Ideally, only one device within receiver vicinity transmits on one channel at the same time, avoiding collisions and data loss. This is achieved through the listen-before-talk mechanism of Carrier-sense multiple access with collision avoidance (CSMA/CA). As a consequence, APs and STAs on the same channel share airtime. The time that a transmitter waits while the channel is busy is called interference (time). Unfortunately, interference cannot be avoided in highly dense areas since channels are scarce: depending on the regulatory region, only 3–4 channels are non-overlapping in the 2.4 GHz band, and around 20 channels on the 5 GHz band. It is thus not always possible to put neighboring APs on different channels. In Fig. 2 for instance, the two lower APs share the *blue* channel, whose usage time is split between (uplink and downlink) communications of STAs A and B, idle time and interference. Interference is particularly detrimental in busy parts of the network, such as conference rooms. As the network conditions evolve over time, it is beneficial to dynamically adjust the channel allocation.

In modern WLANs, an AP may be configured to allow the aggregation (aka *bonding*) of several channels, to increase bandwidth and consequently throughput: in Fig. 2, the top-left AP uses two channels (dark and light yellow). While there are some obvious benefits, bonding increases the number of neighbors competing for the same channels, complicating the channel allocation. Real-time channel management in WLANs is typically done using heuristics acting on channel selection, as for instance in [8].

2) Power Management: STAs attach to a serving AP based on criteria that differ by vendor and device generation. The most important aspect, however, is the received signal strength indicator (RSSI) that needs at least to exceed a device-specific threshold. A higher RSSI is generally beneficial for the user, since it allows for higher data rate and thus better throughput. Besides environmental factors such as shadowing through walls and other obstacles, the downlink RSSI is mainly determined by the distance between AP and STA and the *AP transmit power*: in Fig. 2, different transmit powers are reflected in different coverage ranges. While increasing AP transmit power would benefit the data rates of closer STAs, it would also increase the number of attached STAs, and hence competitors on the radio resource. Additionally, surrounding low-power APs may be underused and more distant STAs and APs may see the high-power AP as an interferer, and network performance may degrade. Hence, an optimal power allocation needs to provide good coverage in the entire network area while limiting interference and balancing load among APs. Power management actions are used for instance in [9], [10].

3) Backoff Timers: When a transmitter (AP or STA) wants to transmit but finds the channel busy, it backs off and waits for a certain time (several microseconds) before the next attempt. The *backoff timer*, defined through Enhanced Distributed Channel Access (EDCA), depends on the access type to which the transmission belongs. It is usually shortest for time-sensitive traffic such as voice calls, and longer for e.g. email. Configuring these backoff timers allows to target traffic-specific quality of service. The EDCA contention mechanism is for instance exploited in [11].

B. DRL WLAN challenges

We design [12], implement and deploy a DRL-based WLAN channel management solution (agent), that autonomously reconfigures (action) a real operational network (environment) every 10 minutes, based on telemetry data (state) received at sub-minute timescale. While training and deploying this system, we faced the same challenges introduced in §II-B that we tackled as described in the following. Fig. 2 shows the relation between training, deployment and the various challenges schematically.

1) Training safely: Obviously, training on the real network would inevitably lead to the exploration of bad network configurations harming user experience. As this option is not viable for business considerations, we are forced to train the DRL agent using a simulator: the tradeoff we face is that the simulator needs to be simple enough to allow for reasonable training time, and at the same time, realistic enough to favor the transfer of learning to the real deployment.

2) Data issues: An alternative option would have been to train offline with real network configurations as ground-

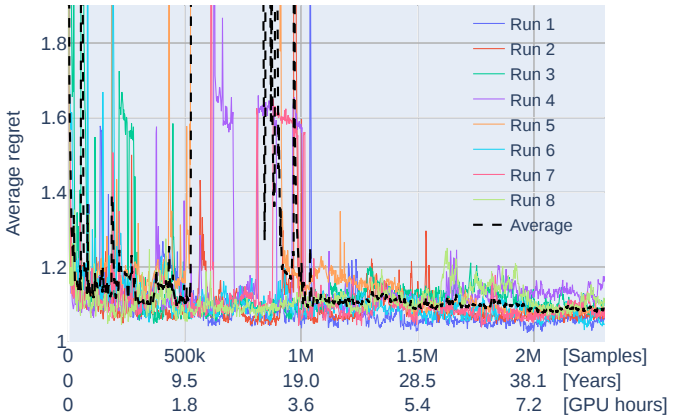

Fig. 3. *Training duration*. Average regret (dashed black line) during training converges after more than 1 million interactions with the environment: this would require several years in a real deployment (at the considered timescale) and already requires about 8 hours worth of GPU time.

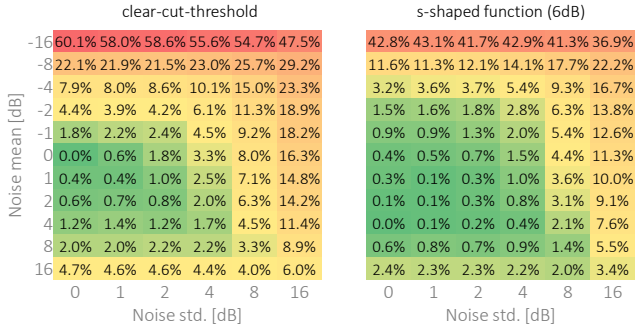

Fig. 4. *Generalization*: Assessment of relative performance degradation for controlled environmental changes

truth. Although we have access to live measurements and long historical data from real networks, since the network is sparsely reconfigured (once per day), our data lake does not contain the several hundred thousand samples per network that DRL would need for training, . We therefore opted for exploiting the data to enhance simulator realism. For example, our simulator uses an RSSI threshold to decide which APs are considered as neighbors: we increase realism by fitting this parameter to maximize similarity between the interference estimations of the simulator and those measured in the real network (which in our case corresponds to an RSSI threshold of -82dBm).

3) Training duration: Two factors impact the training duration: (i) the convergence of DRL weights during the training process which affects the number of interactions, and (ii) the execution time of each simulated interaction, during which the DRL training process remains idle waiting to receive state and regret feedback from the environment. As for (i), we calibrated the training phase carefully to avoid getting stuck in local minima, mainly by adjusting the learning rate during training such that more aggressive updates (e.g. higher steps) are performed at the beginning, followed by smaller steps allowing the system to gradually stabilize. As for (ii), the duration of a simulated interaction can quickly become a bottleneck, for which we rule out the use of packet-level simulations (such as ns2 or ns3) and leverage a fast custom simulator, with negligible computational complexity. Fig. 3 illustrates the regret evolution over multiple independent training runs: the x-axis reports the number of iterations, the GPU training time (including the simulation time, measured in hours) and the equivalent duration of the training process had it been performed at the same timescale in a real environment (measured in years).

4) Generalization: Generalization to conditions unseen during training means, on the one hand (i) generalizing to networks of arbitrary size and density, and on the other hand (ii) transferring well to the more complex physics of the real network. We tackle (i) by a novel auto regressive sequential decoder whose input features at each step are engineered to reflect the changing internal state of the decoder that is described in [12]. As for (ii), we investigate the robustness of the DRL algorithm by training in an ideal noiseless environment and testing in a noisy one. We systematically apply Gaussian noise with controlled means and standard deviations to the AP neighborhood (i.e., RSSIs each AP sees from all others) and observe the impact on the regret. The left-hand side of Fig. 4 reports the relative percentual increase of the regret with respect to ideal conditions (noiseless training and testing) when neighborhood is defined with a simple threshold. Overall, the picture confirms DRL to be robust for a wide range of additive noise. Additionally, consider that a negative noise causes the corresponding RSSI to fall below the neighborhood threshold, leading to interference underestimation. Unsurprisingly, the figure confirms underestimation to be more harmful than overestimation, which validates our conservative simulator design choice.

Now, in the real network, neighborhood is not exactly a clear-cut threshold: the right-hand side of Fig. 4 further tests the algorithm on the same noisy conditions, but using a more complex neighborhood definition: in particular, neighborhood interference is smoothly taken into account by using an S-shaped sigmoid function with a spread of 6dB centered around the clear-cut threshold. Interestingly, using this "loose" definition does not degrade the results and leads to even better resistance to noise, suggesting good generalization abilities for our algorithms.

5) Explainability and Trust: Last, while the above results show our DRL algorithm to be capable of closed-loop WLAN control, as introduced earlier, it is essential that network engineers develop trust in the algorithm decisions before letting them run unattended on thousands of customer deployments. As DRL decisions are intrinsically less interpretable than heuristics, this can be achieved by human-understandable explanations of the algorithm decisions and expected gains [13], to ensure that algorithm decisions are

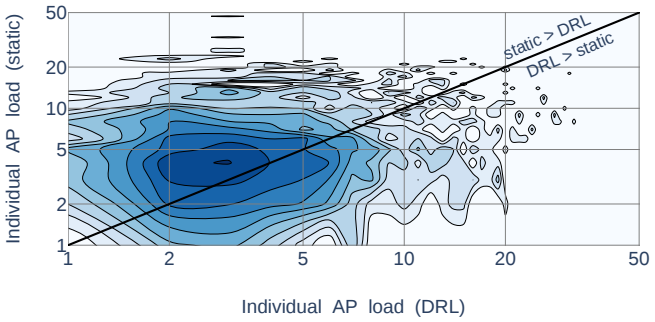

Fig. 5. *Real deployment*. Channel utilization heatmap, comparing load of same AP and same time-of-day slots across different WLAN control algorithms (static and DRL) running on different days.

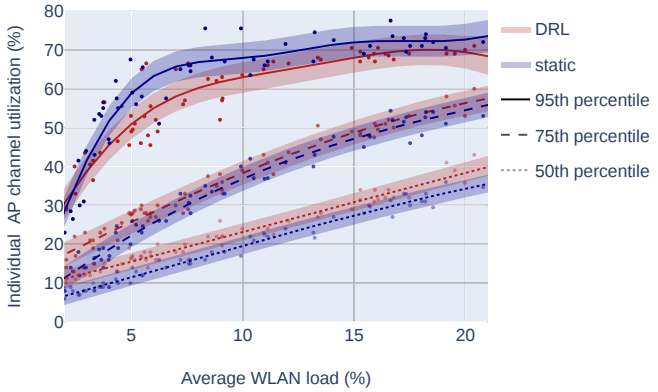

Fig. 6. *Real deployment*. Statistically unbiased comparison of the breakdown of individual AP load (y-axis) for same average network load (x-axis).

perceived as both *beneficial* and *safe* by the operator using it.

C. WLAN DRL Real Deployment

Tackling the above challenges lead to a successful DRL deployment: we briefly report on months worth of tests on real network deployment, by running the trained DRL algorithm vs daily static optimization every other week. Fig. 5 contrasts the channel utilization of a 34 APs deployment where we observe approximately one thousand STAs on a typical day. We construct a heatmap from the scatter plot where each point represents the average channel utilization for the same AP during 10 minutes at the same time-of-day and day-of-week for the two algorithms, over all APs: this allows to assess the impact of dynamic DRL channel management from a spatial viewpoint, i.e., from the point of view of the same AP. We can note the tendency of improvements as the *center* for the highest density moves *above* the diagonal.

Clearly, while the traffic is similar in every week due to seasonal behavior of the users, the traffic conditions are not identical, which can bias the comparison. We take this confounding factor into account by comparing in Fig. 6 the breakdown of the AP utilization (y-axis) for the same average network load (x-axis): it is easy to see that, as expected, DRL relieves APs with high channel utilization (notice the 95th percentile decrease) by shifting load to lightly loaded APs (notice the median increase), which is desirable from the perspectives of load balancing and fairness.

Overall, these results confirm that by properly taking into account the main challenges tied to real-world deployment, DRL can hold its promise to become a fundamental building block of network O&M.

IV. Conclusion

The DRL paradigm when applied to the control of complex systems such as large networks promises to improve decision making over human intuition and classic heuristics. While most network-related DRL research focuses on ideal scenarios and are evaluated via simulation, we discuss here the challenges that arise when deploying DRL into the real world. In particular, we illustrate how DRL can improve real-time channel management on large-scale operational WLAN.

To successfully extend these promising DRL results to other network problem and use-cases, and especially ensure successful deployment in practice, we can generalize our main lessons as follows. It appears that DRL training requires digital twins, such as simulators. Indeed, solely learning from existing network data may not be feasible (given the sheer number of samples needed for training) nor desirable (as it does not offer enough action diversity, so missing unsafe actions). Conversely, learning from simulation provides the best tradeoff among safety (i.e., to explore also unsafe actions), simplicity (for training duration) and realism (e.g., as simulators can be enhanced with real-world data and be used to assess controlled generalization). Finally, deployment of trained DRL models for real-time inference still requires a pedagogic effort toward the human operators interacting with it (in order to gain their trust), as well as offering failbacks to legacy systems until the trust is gained in a pervasive manner.

References

[1] D. Silver *et al.*, "Mastering the Game of Go without Human Knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[2] https://openai.com/projects/five/ accessed on 13.04.2021.

[3] M. Campbell *et al.*, "Deep Blue," *Artificial intelligence*, vol. 134, no. 1-2, pp. 57–83, 2002.

[4] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. The MIT Press, 2018.

[5] N. Vesselinova *et al.*, "Learning combinatorial optimization on graphs: A survey with applications to networking," *IEEE Access*, 2020.

[6] A. Karpathy, "Pytorch at Tesla," in *Pytorch DevCon'19*, 2019.

[7] S. Manivasagam *et al.*, "Lidarsim: Realistic lidar simulation by leveraging the real world," 2020.

[8] A. Bhartia *et al.*, "Measurement-Based, Practical Techniques to Improve 802.11ac Performance," in *ACM IMC*, 2017.

[9] N. Ahmed and S. Keshav, "A Successive Refinement Approach to Wireless Infrastructure Network Deployment," in *IEEE WCNC*, 2006.

[10] V. Shrivastava *et al.*, "Understanding the Limitations of Transmit Power Control for Indoor WLANs," in *ACM IMC*, 2007.

[11] D. Deng *et al.*, "Contention window optimization for IEEE 802.11 DCF access control," *IEEE Transactions on Wireless Communications*, 2008.

[12] O. Iacoboaiea *et al.*, "Real-Time Channel Management in WLANs: Deep Reinforcement Learning versus Heuristics," in *IFIP Networking*, 2021.

[13] J. Krolikowski *et al.*, "WiFi Dynoscope: Interpretable Real-time WLAN Optimization," in *IEEE INFOCOM, Demo session*, 2021.