Invited Paper: Smart Autonomous Cyber-Physical Systems

Elisa Bertino
Department of Computer Science
Purdue University
West Lafayette, Indiana, USA
bertino@purdue.edu

Simin Nadjm-Tehrani
Department of Computer and Information Science
Linköping University
Linkoping, Sweden
simin.nadjm-tehrani@liu.se

Abstract

Cyber-Physical Systems (CPS) integrate computing, networking, and physical processes, making them critical in applications such as smart homes, industrial control systems, autonomous vehicles, smart grids, and medical devices. Ensuring CPS security is essential, as vulnerabilities can have serious consequences. CPS share key security requirements with traditional IT systems—confidentiality, integrity, and availability—but also introduce additional challenges due to real-time constraints, interactions with physical processes, and safety considerations. Standard security practices include secure design principles, redundancy, continuous monitoring, resilient control algorithms, and rigorous verification and validation procedures. However, security techniques must be tailored to specific CPS domains. Some of the requirements may interact with each other, e.g., adding security mechanisms violating timely responses, or lack of security measures impacting safety. The complexity of securing CPS is further heightened by the integration of artificial intelligence (AI), which enables greater system autonomy in tasks like energy optimization and security monitoring. In this paper, we present results from two previous projects that focused on smart IoT systems and avionic systems, respectively. In both cases, arriving at solutions that combine many requirements is at the heart of the methodology. Based on this past work, we discuss open research directions.

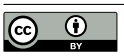
CCS Concepts

• **Security and privacy** → **Systems security**; **Distributed systems security**.

Keywords

Safety, Fault-Tolerance, Reinforcement Learning, IoT Systems, Avionic Systems

ACM Reference Format:
Elisa Bertino and Simin Nadjm-Tehrani. 2025. Invited Paper: Smart Autonomous Cyber-Physical Systems. In *Proceedings of the 2025 ACM Workshop on Secure and Trustworthy Cyber-physical Systems (SaT-CPS '25), June 6, 2025, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3716816.3727974



SaT-CPS '25, Pittsburgh, PA, USA

ACM ISBN 979-8-4007-1502-0/2025/06
https://doi.org/10.1145/3716816.3727974

1 Introduction

Cyber-physical systems (CPS) integrate computing, networking, and physical processes. Examples include smart houses, industrial control systems, autonomous vehicles, smart grids, and medical devices. The security of CPS is crucial because vulnerabilities can lead not only to data breaches but also to physical damage, operational disruptions, or threats to human safety.

CPS share key security requirements with conventional IT systems, namely: (i) Confidentiality: protecting sensitive data and sensitive operations from unauthorized access. (ii) Integrity: preventing malicious alterations to system operations or data. (iii) Availability: protecting systems function from disruptions, especially in critical infrastructure. However, unlike traditional IT systems, CPS security must account for real-time constraints, physical processes, and potential safety risks. Attacks can range from malware, sensor spoofing, and sensor data alterations [5, 15] to denial-of-service (DoS) attacks, requiring a combination of cybersecurity, control systems, and safety engineering approaches to defend against threats.

Typical guidelines for building secure CPS include:

- Secure design: incorporating security and safety considerations from the design phase (e.g., secure-by-design principles).
- Redundancy: introducing backup systems and fail-safe mechanisms to maintain functionality in case of failure.
- Monitoring and anomaly detection: monitor the state of the system continuously to detect faults or malicious activities.
- Resilient control: designing control algorithms that can adapt to disturbances and degraded conditions.
- Verification & validation: Rigorously testing the behavior of the system in various scenarios to ensure that the safety and security requirements are met.

However, because of the diversity of CPS application domains, specific security techniques must be deployed for different domains. For example, consider industrial control systems (ICS) – e.g., power grids and manufacturing. Key challenges for ICS include the reliance on legacy systems not designed with cybersecurity in mind, and real-time requirements, as downtime can cause widespread disruptions. Therefore, relevant security approaches include: (i) Defense-in-depth: layered security combining network segmentation, intrusion detection systems (IDS), and access controls. (ii) Anomaly detection: using machine learning or model-based approaches to identify deviations in sensor data or control commands that may indicate cyberattacks or equipment faults. (iii) Resilient control systems: algorithms designed to maintain stability and safe operation even when parts of the system are compromised.

On the other hand, for autonomous vehicles, key challenges include sensor spoofing (e.g., GPS jamming, LiDAR blinding), which

can mislead navigation systems, and split-second decision-making required for safety. Therefore, relevant security approaches include: (i) Sensor fusion: combining data from multiple sensors (e.g., cameras, LiDAR, radar) to detect and correct for discrepancies. (ii) Redundancy & fail-safes: using backup systems for critical functions such as braking and steering. (iii) Secure communication: using encryption and authentication protocols to prevent message spoofing or hacking of vehicle-to-vehicle (V2V) or vehicle-to-infrastructure (V2I) communication.

The complexity of securing CPS is today complicated by the use of artificial intelligence (AI) techniques, as these techniques allow these systems to increase their autonomy in various tasks, such as energy optimization. As autonomous decisions by a CPS may result in unsafe physical actions, it is important to introduce constraints on these actions. On the other hand, AI techniques can also enhance security. Examples include: (i) AI-driven anomaly detection and predictive analytics to identify potential system failures before they occur, reducing downtime and maintenance costs; (ii) AI-driven analysis of vast amounts of sensor and network data to detect and respond to security threats, cyberattacks, or system anomalies; (iii) AI-driven adaptation of CPS to changing conditions, such as handling concept drift in security monitoring or environmental variations in industrial settings. In addition, AI techniques can be used to find architectural solutions that can remain secure, safe, and provide timeliness in the long run, even though detailed design decisions change some functionality over time.

In this paper, we review some of our previous work on securing CPS in two domains, namely:

- *Smart IoT Systems.* These are autonomous systems consisting of several IoT devices that use reinforcement learning to automatically optimize metrics of interest to end-users/applications. A typical example of such a metric is energy consumption. However, a critical problem is represented by safety and security. Therefore, these systems represent a useful example of an AI-driven autonomous CPS, whose actions have to be constrained to ensure safety and security.
- *Avionics concept design optimisation.* Here we consider the interrelation of the extra functional properties of a CPS in the early concept design stage and use search-based heuristics and AI techniques to navigate within the complex design space. This is demonstrated to considerably speed up the concept design time while at the same time provide example solutions that respect the stipulated redundancy, isolation, and timeliness properties, within given cost constraints.

On the basis of this past work, we also discuss novel research directions.

2 Smart IoT Systems

The advent of advanced communication protocols for devices of the Internet of Things (IoT), such as 6LoWPAN, CoAP, and Zigbee, combined with significant advancements in artificial intelligence (AI), has enabled seamless interconnectivity between IoT devices, leading to the creation of intelligent and autonomous IoT systems. In the consumer market, IoT technology is often associated with the concept of 'smart home'. This encompasses a wide range of devices and appliances, including lighting fixtures, thermostats, home security systems, and cameras, all of which are integrated into common ecosystems. These ecosystems also incorporate various sensors, such as motion, sound, light, heat, and touch sensors, that can be controlled through smartphones or other ecosystem-connected devices.

A popular approach to deliver intelligent IoT services involves the use of trigger action applications (T/A) or applets. These apps enable functionalities ranging from simple tasks, such as "unlocking a door as an authorized user approaches", to complex operations, such as "autonomous vehicle navigation through predefined areas." These applications facilitate communication between the management device and the IoT devices, including sensors and actuators, through edge or cloud computing through API calls. Platforms like IFTTT [8], Zapier [20], and Apiant [1] allow third-party developers to create and deploy custom T/A apps. This approach fosters vibrant developer communities that design applications tailored to various environments, devices, and protocols.

However, the development of smart IoT-based systems faces significant challenges due to the need for seamless interconnection and interoperability among devices, apps, and users. The complex and dynamic nature of IoT ecosystems, where devices and applications interact continuously, introduces potential safety and security risks. Furthermore, T/A apps are often designed with narrow, device-specific goals (e.g., 'turn on the heater if the temperature drops below a set threshold') and lack a holistic understanding of the broader IoT environment. This limited perspective can lead to suboptimal decisions or actions that do not align with global user requirements or overarching system goals.

Existing research has explored the application of reinforcement learning (RL) to optimize IoT systems for specific objectives, such as energy management, efficient resource allocation, and cost minimization. Although these RL-based frameworks excel in achieving their target goals, they often neglect critical aspects of safety and security, which are essential in IoT ecosystems.

This highlights the urgent need for intelligent monitoring systems that not only ensure safety and security but also optimize functionality with a global view of all interconnected devices and their interactions. Such systems should integrate security, functionality, and user-centric goals into a cohesive framework, addressing the shortcomings of current approaches and paving the way for safer, more efficient IoT ecosystems. The design of such systems is challenging because it requires novel constrained RL-based frameworks that can autonomously predict 'optimal' and 'safe' decisions in an IoT system.

In what follows, we first introduce the relevant background, followed by an overview of Jarvis, the first autonomous system based on RL that supports safety and security policies [12], and then discuss open research directions.

2.1 Background

2.1.1 IoT Architecture. At a high level, an IoT architecture can be conceptualized as comprising four key components: devices, edge, cloud, and control devices.

- Devices include sensors, actuators, and appliances that interact with their environment. Examples include smart locks,

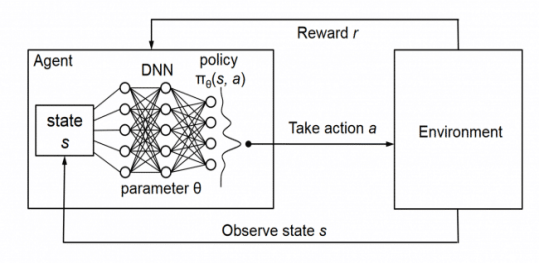


Figure 1: Deep Q Learning Environment (figure from [12])

lights, thermostats, and motion sensors. These devices are designed with standardized "capabilities," enabling them to modify specific "attributes" in response to "commands." For instance, actuators like smart locks or lights can perform actions such as locking/unlocking, powering on/off, or adjusting temperature, while sensors can detect states like motion/no motion, specific sound levels, or temperature variations.

- Edge components act as intermediaries, comprising hubs, routers, and other connection-enabling devices. They facilitate communication between devices and higher-level systems by translating device-specific messages (e.g., "lock/unlock" or "heat/cool") into normalized, edge-readable events (e.g., "door locked" or "temperature reached X degrees").
- Cloud systems encompass services such as databases, analytics servers, and other cloud-based platforms that support data processing, storage, and advanced analysis.
- Control devices include interfaces such as smartphones, tablets, and desktops with which users interact to monitor and control IoT systems.

The operation of these components is organized around an event-driven, publish-subscribe architecture. Devices generate and publish events (e.g., "door opened" or "device turned on"), which are processed by edge handlers and made available to subscribing apps. These apps, having subscribed to relevant events, can respond appropriately, ensuring seamless interoperation and coordination within the IoT ecosystem.

2.1.2 Deep Q Learning. A reinforcement learning (RL) framework [18] (illustrated in Figure 1 [12]) operates within an environment where each state-action pair (s, a) or state transition is associated with a reward function $R(s, a)$, which assigns a reward value r. An RL agent navigates this environment following a policy $\pi_\theta(s, a)$ for a time period θ, accruing a cumulative reward based on the state transitions encountered and the underlying model of state transition probabilities. The objective of an RL system, such as a Q-learning framework, is to discover the optimal policy that maximizes the total accumulated reward. This is achieved through the exploration and evaluation of a Q function, which represents the expected cumulative reward for every state-action pair.

In a deep Q learning system, a Deep Neural Network (DNN) is used to determine the optimal Q function using a temporal difference equation defined as follows:

$$Q_t(s,a) = Q_{t-1}(s,a) + \alpha[R(s,a) + \gamma Max_{a'}\{Q_t(s',a')\} - Q_{t-1}(s,a)]$$

where $Q_t(s, a)$ is the current Q function and $Q_{t-1}(s, a)$ is the previous Q function for the environment. The estimated next state and action are denoted by s' and a', respectively. The learning rate (α) determines to what extent the newly acquired information overrides the old information. The discount factor (γ) determines the importance of future rewards.

2.2 Overview of Jarvis

2.2.1 IoT Environment. The design of Jarvis assumes an environment represented by a finite state machine (FSM), consisting of k devices $\{D_1, .., D_k\}$, η users $\{U_0, ..., U_\eta\}$, and m apps $\{ap_0, , ..., ap_m\}$. Manual operations in the model are represented by a pseudo app ap_0. Each device in the environment is modeled by a set of device-states and a set of device-actions. At any point of time, device D_i can be in one of a set with a number i_{ss} of device-states: $\{p_{i_0}, p_{i_1}, ...p_{i_{ss}}\}$. At a *time instance* t, a device-action can be executed on device D_i from a set with a number i_{as} of device-actions: $a_i^t \in \{a_{i_0}, a_{i_1}, \ldots, a_{i_{as}}\}$. Specifically for IoT platforms, device capabilities and device attributes can be translated to device-actions and device-states respectively. Each device D_i has a transition function δ_i which is the link between a device-action and a device-state. For a device D_i in state p_{i_x} and having device action a_{i_y} take place on it, $\delta_i(p_{i_x}, a_{i_y}) = p_{i_{x'}}$ gives the new state of the device. Along with this, each device D_i has a dis-utility function $\omega_i(p_{i_x}, a_{i_y})$ which represents the dis-utility per *time instance* that results if the execution of device-action a_{i_y} is delayed in state p_{i_x}. A device can exist in different locations and thus have varying contexts in terms of accessibility, user permissions etc. Jarvis thus follows the container based approach commonly used by IoT platforms. Each container acts as a boundary between the devices; the containers are organized hierarchically according to user accounts, locations, and groups. Therefore, device D_i can only be accessed by a set of authorized users u_i, $u_i \subseteq \{U_0, U_1, ..., U_\eta\}$, depending on its location l_i and its group g_i, and corresponding device and app subscription policies.

2.2.2 State Transition Model. For the overall environment state S_t, at the next *time instance* $t + 1$, a set of authorized users $U^t \subseteq \{U_0, U_1, ..., U_\eta\}$ can use a set of apps $AP^t \subseteq \{ap_0, ap_1, ..., ap_m\}$ to perform an *action* A_t on a set of devices $D \subseteq \{D_1, D_2, .., D_k\}$, to get the new state of the environment S_{t+1}. So the state transition of the environment is represented as the current state $S_t = (s_0, s_1, ., s_i, ., s_k)$ plus a set of at most k (one per each device) device-actions. $A_t = \{a_0^t, a_1^t, ..., a_k^t\}$ is the set of actions taken at *time instance* t by a set of users U^t through a set of apps AP^t resulting in the next state S_{t+1} at *time instance* $t+1$. The next state is computed using the transition function for each device and corresponding action on the device such that $S_{t+1} = (\delta_0(s_0, a_0^t), \delta_1(s_1, a_1^t), ..., \delta_k(s_k, a_k^t) = \Delta(S_t, A_t)$ where Δ is the overall transition function of the environment.

Definition 2.1. [12] A *FSM* consists of tuple (SS, AS, Δ) where: $SS = \bigcup_{i=0}^{v} S_i$ is the state space with $v = \prod_{i=0}^{k} i_{ss}$; $AS = \bigcup_{i=0}^{v} A_i$ is the action space with $v = \prod_{i=0}^{k} i_{as}$; and Δ is the overall state transition function. The overall state of the *FSM* at time instance t is defined as $S_t = (s_0, s_1, ., s_i, ., s_k)$, where s_i is the state of the i-th device such that $s_i \in \{p_{i_0}, p_{i_1}, ...p_{i_{ss}}\}$.

State transitions are monitored in terms of "episodes", where each episode is characterized by two parameter: time period T and interval I. The state transitions occur every I time units, such as seconds, minutes, and hours, until the timestamp reaches T time units, after which the state is reset to the initial state and marks the end of an episode. An episode basically consists of T/I *time instances* at which the state transitions of the environment are recorded. For example, for $\{T, I\} = \{60, 1\}$ minutes, the episodes are an hour long with state transitions every minute.

The following definition defines the Jarvis model of the IoT environment state transitions in terms of episodes.

Definition 2.2. [12] An episode is a tuple (N, S_0, T, I). $N = \{S_0, S_1, ., S_t, ..S_n\}$ is an ordered list of states reached in the episode where each next state $S_{t+1} = \Delta(S_t, A_t)$ for an action A_t and $0 < t \leq n$; $n = \lceil T/I \rceil$; S_0 is the initial state of the episode; T is the time period; I is the interval.

2.2.3 Jarvis Optimization Problem. The functionality optimization goal is modeled as an Markovian decision problem (MDP), a sequential decision-making problem where the results are under the control of an agent. The agent's goal is to maximize functionality as specified by the user by choosing a sequence of actions for the upcoming episode of the environment. In the Jarvis model, the functionality requirements defined by the user are measured through a reward function. The specified functionality requirements determine the utility ($F()$) that the user gains, which is one part of the reward function in the environment. The other part derives from the di-sutility ($D()$) caused to the user in terms of delays, waiting time, and discomfort. The general structure of the reward function is defined as follows: $R(S, S', t) = F(S, S', t) - D(S, S', t)$ where S is the current state, S' is the next state of the environment, and t is the current *time instance* of the episode. The goal of Jarvis is thus to maximize the cumulative reward at the end of the episode, which is an MDP defined formally as follows.

Definition 2.3. [12] A MDP is a tuple (F, R, P, T, I, S_0). F is the FSM of the environment; R is the reward function; P is the state transition probability table; T is the time period; I is the interval; and S_0 is the initial state of the environment. The agent's goal is to find a strategy of actions according to P, maximizing the total value of R of the next episode, for the environment in state S_i in F where $0 \leq i \leq \lceil T/I \rceil$.

The application of the RL approach to IoT systems requires addressing two major challenges, which we discuss in what follows together with the approach adopted in the design of Jarvis.

Safety/Security of State Transitions: State transitions in IoT systems can pose safety or security risks to users or the environment. Uniform state transition probabilities are unsuitable; instead, transitions must depend on context and environment. Unsafe or insecure transitions should have probabilities set to zero, requiring identification of environment-specific safety and security policies before defining the state transition probability table. Examples of safety policies in a smart house environment are "the front door must be locked when nobody is in the house or when everyone in the house is sleeping," and "the security system and fire alarm must always be on."Examples of security policies are "only authorized users should enable/disable devices" or "installation of new software on a device can only be authorized by a user with the admin authorization for the device or a user with delegation from an admin."

Addressing this challenge requires defining safe state transitions. In Jarvis, state transitions that occur naturally during a defined learning phase are considered safe. This phase, set up by the user, involves approving or performing actions manually to ensure that the transitions are safe and secure, that is, they verify all the specified safety and security policies. These transitions are recorded as trigger-action (T/A) behaviors: **T: Current State** $S_t \rightarrow$ **A: Next Action** A_{t+1}, forming the training dataset TD. To prevent learning benign device malfunctions or user errors as unsafe, the dataset is filtered using a feedforward artificial neural network (ANN) trained via backpropagation with user-labeled benign anomalies. Labeling can be done offline or in real time based on user preferences. Filtered transitions and their instance counts are stored, and only transitions exceeding a specific threshold $Thresh_{Env}$ are assigned uniform probabilities. All others are marked as null to prevent unsafe transitions.

Unknown Reward Function: Quantifying the exact "utility" or "reward" for actions is challenging due to non-Markovian factors, such as environmental variables (e.g., electricity prices, temperature) and user behavior. As a result, the reward function $R(s, a)$ is not strictly Markovian and depends on the state, action, and specific time instance, making its precise value uncertain for the agent.

To address the challenge of unknown reward functions, the approach is to estimate the reward function using user input and previous experiences. The estimated (smart) reward function for environment state S_t, action A, and *time instance* t is defined as:

$$R_{smart}(S, A, t) = \sum_{j=0}^{\kappa} (f_j) F_j(s, a, t) - \frac{I}{kT} \sum_{i=0}^{k} \omega_i(s_i, a)(t - t')$$

User inputs define κ functionality requirements through normalized reward functions F_j, such as energy consumption, electricity costs, temperature difference, or network usage. Each reward is assigned a weight f_j, reflecting user priorities. These weights enable the system to learn strategies aligned with user goals while considering the overall environment. Users can adjust weights to prioritize specific goals, but the system autonomously selects strategies that balance all requirements. The user can alter weights to give more preference to one goal over the others but the essential strategy choices are made by the system keeping in mind the entire environment.

The second part of the expression represents the estimated dis-utility caused by each device based on its state and predefined dis-utility values over episodes of duration T with intervals of size I. Dis-utility refers to user discomfort or waiting time, estimated from past user behavior. Higher dis-utility indicates a significant deviation from typical user preferences, influenced by the time difference $t - t'$ between the current and preferred state-action instances. The normalized ω_i function captures the dis-utility cost for each device. Incorporating dis-utility into the reward function ensures the agent balances functionality optimization with user convenience. For example, in a smart home, the agent might avoid actions that conserve maximum energy (e.g., shutting off all appliances) if it causes high user discomfort. Reward function parameters

(f_j and ω_i) are tuned to balance utility and dis-utility using a utility-dis-utility ratio X. The ratio X reflects user preferences and the environment set-up.

2.2.4 Q Learning Algorithm. The FSM of the IoT environment is used to build a simulated environment where an agent can run multiple episodes to find the optimal and safe device actions for the upcoming episodes. The agent balances exploration and exploitation according to the exploration rate ϵ. The exploration of the agent is constrained by security and safety policies at each step by using the safe state transition table (P_{safe}) learned based on the approach described above. Random batches of previous agent experiences are selected and replayed to learn cumulative rewards according to a discount factor γ and a batch size B_{size}. Finally, the random batch with cumulative rewards is used to further train the RL framework DNN in order to learn optimal Q table values for each state action pair and timestamp of the episode.

2.2.5 Experimental Results. Several experiments to evaluate Jarvis were carried out in a smart home environment and its performance was evaluated using simulated and real-world data. In the experiments Jarvis was able to detect 100% of the 214 manually crafted safey and security violations collected from previous work and was able to correctly filter 99. 2% of the user-generated benign anomalies and malfunctions from safety violations. With respect to functionality benefits, Jarvis was evaluated using real-world smart home datasets on energy use minimization, energy cost minimization, and temperature optimization. The experimental evaluation showed that Jarvis has significant advantages over normal device behavior in terms of functionality and over general unconstrained RL frameworks in terms of safety and security.

2.3 Future Research Directions

2.3.1 Complex IoT Systems. It is important to note that in the formulation of the MDP for Jarvis, optimal actions are chosen with respect to the current state and timestamp of the episode. It is possible that in complex IoT systems, a more sophisticated policy identification is required in terms of higher-order temporal trajectories. In this case, an MDP model that relies on the immediate previous state is a limitation. A research direction is to address this limitation by incorporating the temporal parameters of the episode in the state definition of the model. Such an approach would result in more fine-grained optimization policies, but at the cost of a higher number of state spaces and computation cost. It is also important to analyze the optimal temporal parameters and specifics of the DNN, like number of hidden layers, activation functions, optimizers, network organization (feed forward/recurrent/convolutional), based on different device configurations and user requirements, for different IoT systems.

2.3.2 Adapting Security and Safety Policies. The Jarvis framework requires a learning phase by which the framework can learn safe state transitions by "observing" the user or explicitly asking the user. To minimize such training activity, an approach is to use some existing policies and then adapt them to the IoT system of interest. An approach is to generalize existing policies using symbolic learning. An example of such a specific form of symbolic learning is inductive logic programming. An inductive logic programming system aims to find a set of logical rules, called a hypothesis, that together with some background knowledge explain a set of positive and negative examples. FastLAS [9] is a well-known symbolic learner based on inductive logic programming. Systems like FastLAS are usually able to generalize quite well from specific examples. Informally, the generalization capabilities of a learner like FastLAS are due to the fact that, when evaluating multiple hypotheses, the hypothesis with the shorter rules is preferred (based on the Occam's razor principle). The selected hypothesis is thus the most general.

In order to apply such an approach to IoT systems, one would need to not only observe the user behavior, but also collect context information about the actions executed by the user. For example, time of the day when a certain action is executed and events occurrences in response to which actions are executed. In addition, knowledge about the functions of IoT devices is critical. For example, which devices can execute actions that allow external access to a given house.

Therefore, given a set of policies learned in a given IoT system (referred to as the source IoT system) and context information, one can identify which of these policies are relevant for the IoT system of interest (referred to as target IoT system) based on the devices in the latter. One can then execute a short training phase to verify which policies are confirmed by the observed user actions, which ones are not confirmed, and thus must be discarded, and which user actions are not covered by the policies. In the latter case, new policies would need to be generated and added to the initial set of policies. The sharing of learned policies and ontologies describing IoT devices can be supported by community-based mechanisms (see the notion of WikiHow for IoT devices [3]).

2.3.3 Real-Time RL Framework. From an architectural point of view, Jarvis is based on a centralized RL system that controls all devices in a given IoT system. The centralized RL system would typically be connected to control devices, such as mobile phones and desktops. Such an approach allows one to run the framework on a machine with adequate resources. However, there are applications with real-time constraints on the actions taken by IoT devices. Relevant examples can be found in mobile devices, computing while communicating through wireless channels [2], which have to perform autonomous data acquisition [4] or some data analysis tasks. For applications with real-time constraints, a centralized RL architecture is not suitable.

To address real-time requirements, a possible approach is to adopt design a split edge-based architecture, such as the DeepWiERL framework [14], by which two RL systems are deployed: (a) an edge RL system, referred to as control RL system - which is the Jarvis framework, and (b) a device RL system, referred to as operative RL system - which simply decides the next action to be executed based on the current device state. The edge RL system continuously learns and adapts its DNN, implementing the Q function, and periodically transmits the parameters of this DNN to the operative RL system. The operative RL system then modifies the parameters of its own DNN accordingly.

For the Jarvis specific approach, an additional element that must be transferred from the edge RL system to the operative RL system is the set of security and safety policies, as DeepWiERL does not address safety and security and only focuses on adapting parameters

for wireless communications of devices. The policies do not need to be periodically transmitted but only when there are changes in these policies. In addition, each device would receive only the policies relevant to it.

The design of the split edge-based RL architecture requires addressing a challenge related to the case when multiple IoT devices have to collaborate to perform actions, as the DeepWiERL only covers the case of a single device. To address such a problem, one needs to include, among the actions that devices can take, the communication actions among IoT devices. The control RL system can then also learn which communication actions are beneficial and need to be executed by the devices, depending also on the communication capabilities of the devices. Such an approach has the advantage of scalability and decentralization as the devices would be autonomous and directly able to initiate communications with other devices without having to rely on some central intermediary.

Another critical challenge related to real-time requirements is that RL has high convergence times [10]. In a split architecture, this issue would affect the bootstrap phase of the control RL system. To address this issue, one possibility is to use transfer learning techniques based on generative adversarial networks (GANs) [17]. This approach allows one to transfer a neural network trained on a dataset in a source domain to a target domain that does not have many training data. Lack of training data is a common occurrence in several domains, including cyber security. Previous work [17] has used GANs to create a domain-invariant mapping of a source dataset (SD) and a target dataset (TD). Experiments show that such an approach is effective for target domains with limited training data.

The application of such a methodology to quickly bootstrap the DNN in the control RL system has two variations:

- *Reward Knowledge Transfer*: Under this variation, some explorations are performed by the control RL system. Then a GAN is used to generate augmented or synthetic data (enhanced exploration data) by minimizing domain loss between SD and TD. The resulting dataset is then used to train the DNN of the RL control system. A key environment-specific parameter in this setting is the bias introduced during the training of the DNN in the control RL system. For optimal convergence, the new exploration samples should be preferred over stale samples from the SD. This approach requires two learning/training steps, one to train the GAN and one to train the DNN at control RL system.
- *Quality Value Knowledge Transfer*: This variation is similar to previous one except that here one would transfer knowledge about the RL model directly by exchanging Q values instead of exploration samples. Here, the TD is generated from the baseline control RL system model (Q values) trained by some new explorations. Here, the GAN has two tasks: (i) minimize domain loss and (ii) optimize the Q function. The key goal here is to balance domain loss and quality loss while training the GAN. Since there is only one learning step here, one would need to introduce a bias (TD over SD) into the GAN itself; this can be done when selecting batches to train the GAN.

Research is required to refine, evaluate, and compare these approaches.

3 Trade-off Analysis at Early Concept Stage

Jarvis operates on actions performed by k autonomous devices within a trade-off space between the optimization of certain goals and safety and security. The set of devices, apps, and potential action space is fixed in that devices provided by vendors have predefined actions that they can make; for example, a smart lock can only open or close the door it is managing.

We now move on to a completely different kind of trade-off analysis in the life cycle of CPS. Here, the focus is on two distinguished characteristics: (1) the expected system life-time is very long, the architectural decisions of which application to run on which processing component are not fixed at an early concept design stage; and (2) the system has critical fault tolerance, security, and timing requirements that are specified early, need to be assured while the design space is explored, as the design progresses, but long before detailed design and implementation is fixed.

An example is an avionic system with airborne application functions where multiple possible concepts are studied in the early conception stage. Many decisions made at this stage impact future adaptations and further development in non-trivial ways. For example, once a network topology is fixed and the placement of functions on computing elements is decided, a further change for addition of redundant network paths for fault tolerance may become costly since it may change other satisfied requirements. Isolation of data exchanges within sub-nets to ensure information security across multiple sub-nets may be possible in a range of concept designs, but impacts allocation of software to potentially different hardware components. The addition of new functionality (realized as new software applications) without jeopardizing the end-to-end timeliness of calculations at a later implementation stage may be hard to consider and expensive to verify. In these systems achieving "smartness" and "autonomy" is much more challenging than in conventional IoT systems (like the ones addressed by Jarvis). Here, smartness amounts to finding the right configurations in the design space in a novel and efficient way. Autonomy translates to finding the trade-offs with a good deal of tool support.

Similar instances of the problem exist when allocating microservices to a set of compute elements in base band units in radio access controllers in future generation networks, where capacity, latency and power consumption requirements co-exist with non-trivial trade-offs [16, 19].

In the rest of this section, we elaborate on the challenges encountered in addressing the high-level problem with avionics as an example. Although studying safety in avionics systems has a long tradition, the impact of security breaches on safety has only recently been modeled, leading to methods to deal with such risks[13]. Still, hazard and risk analyses are typically performed when architectural decisions have been made and the system is hard to change in many dimensions.

3.1 Terminology

To make the problem space clear in the avionics concept design stage, we introduce the following terminology used to characterize various solutions in recent works, starting with the NetGAP methodology[6].

- **Application**: defines functionality and is represented by software **processes** that exchange messages and work together to provide the expected behavior.
- **Platform:** the set of conceivable hardware and system software components on which processes can be hosted or deployed. **Hardware modules** provide **resources** (i.e. communication bandwidth, memory, and computing capacity) used by application processes in their computation or communication.
- **Platform Configuration** or **Topology**: is the arrangement of a selected set of hardware modules according to a certain pattern of interconnection.

3.2 NetGAP for State Exploration

Here we give a brief overview of NetGAP and illustrate the overall scheme in Figure 2, reused from [6].

NetGAP uses graphs to represent topologies and employs a graph grammar to describe how platform modules can potentially connect. A candidate topology is formed by applying the grammar rules one by one, starting with an initial graph, which may be empty, until a topology that meets the application requirements is discovered. This involves solving 3 subproblems, denoted SP1 to SP3, as follows.

The process allocation sub-problem (SP1) consists of finding a suitable allocation of software processes (with their associated communication needs) to computing modules to be consistent with the computation capacities of the processing modules, and characterise communication needs for each computation module. This problem can be solved by a genetic algorithm (GA).

The topology generation sub-problem (SP2) consists of building the topology graph by deciding how many and which types of modules (computing modules, communication modules, etc.) should be present in the topology and how they should interconnect to respect the safety (requiring fault tolerance), timeliness, and security related requirements.

The mapping sub-problem (SP3) consists of finding a mapping between the solutions of the first two problems. This means establishing which computing modules in the solution for SP1 (now hosting software processes) correspond to which hardware modules in the topology generated as a solution to SP2.

SP1 thus focuses on meeting the application's requirements for communication and computation resources. Its solution suggests a suitable number of computing modules, but also defines the inter-module communication patterns (how often will messages be exchanged and how big message sizes are envisaged) to observe when tackling the next two subproblems.

Solving SP2 starts from an initial topology graph (which can be either an empty graph or a partially known topology), applying various sequences of graph grammar rules leading to alternative topologies. Each topology connects the computation modules using the available communication modules. Monte Carlo Tree Search (MCTS) is applied to study alternative topologies for consideration. This process involves the repeated resolution of SP2 and SP3 until a termination condition is met.

Different topologies allow different routings of inter-process messages and induce different end-to-end delays. In addition, safety requirements inducing fault tolerance aspects are respected by some topologies and not others. Similarly, security-related requirements disallow or promote certain topologies.

SP3 is a subproblem embedded in SP2, which attempts to see whether the application and topology requirements are satisfied by a given candidate topology. Alternative methods can be used to solve SP3. In the original NetGAP approach, a fast genetic algorithm was used to represent the cross-influence of the two partial solutions, namely the outcome of the process allocation problem and the topology generation problem. The algorithm evaluates the merits, i.e. the *rewards* of each configuration in terms of the envisaged requirements.

In a refinement of the approach, NeuralGAP [7], the topology evaluation problem is formalised and solved using Graph Convolutional Networks (GCN).

3.3 Topology Evaluation by NeuralGAP

Once the state space of possible topologies is known, the candidate topologies are to be evaluated to meet the software application requirements, which may have unknown interactions between functional and extra-functional aspects. Meeting dependability and timeliness requirements has subproblems such as routing data exchanges in the network, finding shortest paths and cliques, and clustering network elements according to their different attributes. To understand the complexities in the solution space, we consider the following formalisation.

Let $T = (V, E, l_E, l_V)$ be the graph representing a candidate topology, where V represents a set of hardware modules, E is a set of directed edges representing the links between the hardware modules, and l_E and l_V are labelling functions that assign labels to edges and vertices, respectively. Let $V_p \subseteq V$ be the subset of vertices of T labeled as processing modules. Let $P = \{p_1, \ldots, p_m\}$ be a finite set of software processes, and $G = \{g_1, \ldots, g_n\}$ be a partition over the set P, i.e. each process p in P is contained in some g_i and $\sum_i^n |g_i| = m$. Finally, let $w_i : V_p \longrightarrow G$ be a bijective mapping from labeled (processing) vertices in V_p to process groups in G and $W = \{w_1, \ldots, w_h\}$ be the set of all such possible mappings. Note that in practice, each w_i represents a possible allocation of software processes to processing modules in V_p.

Now, let $f : (T, \mathbb{A}) \times W \longrightarrow [0, 1]$ be a function that evaluates the extent to which a topology T with the mapping w_i is able to host the envisaged application given the requirements expressed by $\mathbb{A}$.

The reward of a topology T based on a grouping G (representing the mapping of processes onto modules) evaluated against the requirements $\mathbb{A}$, is expressed by the function $r(T, G, A)$.

Then our goal is to find the maximum reward expressed as follows:

$$r(T, G, \mathbb{A}) = f(T, w_{max}, \mathbb{A}),$$
$$w_{max} = \underset{w_i \in W}{argmax} \quad f(T, w_i, \mathbb{A})$$

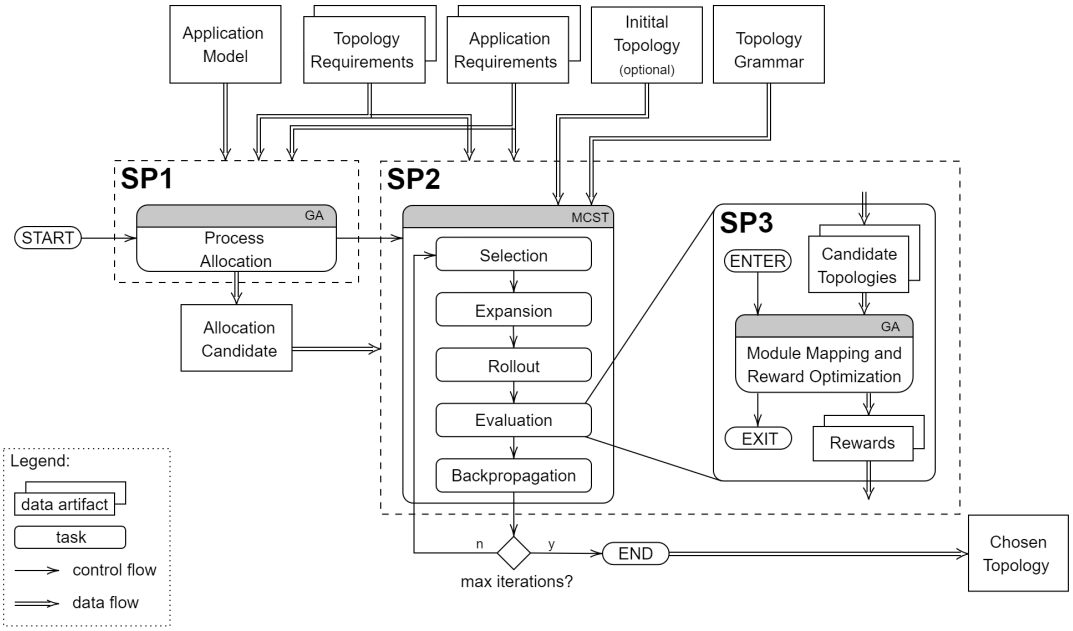


Figure 2: The main loop of NetGAP[6]. Grey shaded areas identify the techniques used to solve each subproblem.

where w_{max} is the maximum reward, calculated across all mappings in W.

Solving the last equation corresponds to solving SP3 and highlights the first challenge of the topology evaluation problem: as there might exist up to $n!$ (factorial of the number of process groups) possible mappings in W, finding w_{max} is computationally expensive.

The second challenge of the topology evaluation problem is pertinent to the existence of requirements in $\mathbb{A}$ that result in complex morphological analysis of the topology T within f. These requirements are typically use-case specific and hard to predict in advance. They encompass tasks such as finding paths, cliques, and minimal covers in a graph, among others. Although efficient algorithms exist for some tasks, others are difficult to approximate when evaluating thousands of candidate solutions.

NeuralGAP addresses the named complexities by adopting a hybrid approach whereby genetic algorithms and GCNs are combined to make fast evaluations of the possible topologies and mappings.

The hybrid evaluator works in two steps: filtering and refinement. During the filtering step, the neural network swiftly analyzes candidate topologies. If the reward for the neural network exceeds a defined threshold, the topology is passed to the genetic algorithm for further refinement. If the score falls below the threshold, the solution is abandoned.

This mechanism enables the rapid sorting and pinpointing of promising candidates for in-depth analysis, while ignoring those that do not offer viable solutions. The choice of threshold th_{ref} is determined empirically and should be chosen considering the reward function and the accuracy of the GCN module. Figure 3 shows the internal organization of the proposed hybrid evaluator.

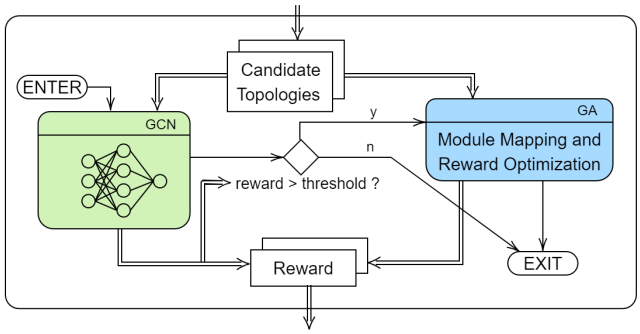


Figure 3: The hybrid evaluation module of NeuralGAP [7]. Green shaded box shows the GCN module (filtering step), blue shaded box represents the GA evaluator (refinement step). The double weight arrows represent the flow of data, and the thin arrows represent the control flow.

3.4 Use Case and Outcomes

The analysis of the state space exploration was evaluated on a realistic avionics use case with two components, a mission-oriented part and a flight-critical part. The mission-oriented part contains 8 periodic processes and 31 periodic messages, and the flight-critical part consists of 91 periodic processes that exchange 629 periodic messages. The application of NetGAP on this usecase identified 60 different solutions in around 7 hours (with a mean time to find a solution of 477 s). These could be charted as clusters of highly resilient ones, low or high degree of link overloads, low or high latency scores, and costs (reflecting the amount of hardware used in the topology). These solutions were also compared with a nearly

optimal solution that was calculated using the mixed integer linear programming (MILP) approach. After 12 hours of running the MILP, a solution with almost 8% distance to the expected optimal was accepted and charted. Several NetGAP solutions were "close" to the MILP solution with various prioritisations of requirements. This justifies applying the mix of heuristics in the NetGAP approach, since multiple good enough solutions were identified in seconds[6].

Moving on to NeuralGAP, the speedup was evaluated on the same use case, but this time focusing on how the MCTS termination condition affects the found solutions. First, in terms of convergence, it was shown that the same set of experiments performed with NetGAP, when repeated with NeuralGAP, led to finding high-reward solutions faster. Several variations were studied: when the time to terminate was specified at different levels(15s, 45s, 90s, and up to 300s). It was first at a higher timeout (300s) that the two methods produced similarly attractive outcomes. NeuralGAP significantly improves search space exploration for the same amount of time spent doing it. In other words, for the same time span, the solver of SP3 in NeuralGAP can consistently explore more of the solution space than the genetic algorithm, helping the main search loop prune out worse regions and focus on looking for solutions in the more promising regions. This was then confirmed by charting the amount of state space covered given a fix number of iterations[7].

The usability of the approach by practitioners has been partially confirmed through discussions with industrial stakeholders. The method is conceived as being more systematic and more efficient compared to concept design in classic avionics systems engineering. To quantify the gains, one would have to use the method in a real case within industry, the data for which would unfortunately not be shareable. Having an open synthetic use case "representing" real cases is in some sense the nearest one can get to quantifying the benefits.

3.5 Future Research Directions

While the above project started with industrial needs in the aerospace sector, we believe that the inclusion of generic methods (graph grammars, MCTS, genetic algorithms and GCNs) makes its adaptation possible in other domains. An example is topology generation for the cloud-RAN use case with latency and energy efficiency requirements [11].

Adding to complexities of such cloud-RAN solutions is the degree of independence of the core network functionality vis-à-vis access network connectivity provision. In the forthcoming evolution of future networks beyond 5G, some scenarios envision open RAN interfaces whereby different providers are able to provide similar connectivity solutions, and the operators can mix and match different solutions based on cost, resilience, timeliness, security, privacy or other requirements. This space makes the analysis of the overall network resilience a difficult task depending on knowledge about the individual sub-components. On the one hand, a multi-provider setup may give the impression of better fallback options upon failure. On the other hand, quantifying the gains compared to costs requires significant analysis that can easily end up in similar search spaces as the one discussed above.

Further instances of these trade-off problems also exist in automotive, industry 4.0, and multi-cloud (or hybrid cloud) solutions for societal critical functions. We believe that variations of the scheme need to be studied in those contexts to identify whether they provide similar gains.

Further research is needed to find out which criteria to formalize for each problem at hand (what should the reward functions look like) and how to ensure that system owners are made aware of the impact of a given choice on the whole solution.

Finally, the generative component in our approach was based on genetic algorithms (NetGAP), and later complemented by GCNs in NeuralGAP. Other generative AI approaches can be studied and compared with the gains achieved in the presented papers.

4 Conclusions

In this paper, we discuss two research projects in different CPS domains and highlight relevant research directions. The potential of AI in the design and run-time management of complex CPS is still in its infancy. Much research is needed to realise the full potential of AI in enhancing security, for adding functionality in a more efficient manner, and for understanding the trade-off between different requirements at the design stage. This needs further exploration in these domains and others. However, using AI to realise a system function may also introduce some risks, in addition to the safety risks mentioned in Section 2, as AI models may not always be consistent. They can also be attacked by smart adversaries aiming to induce model mis-classifications, e.g. evasions in a monitoring context, and for creation of new attack surfaces such as backdoors.

Acknowledgments

The work reported in this paper has been funded by the National Science Foundation under grants 2112471 and 2229876. The authors were also partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation in Sweden.

References

[1] Apiant. 2025. Connect + Automate = More time. https://apiant.com/ [Last Accessed: Jan 20, 2025].
[2] Sergio Barbarossa, Lorenzo Sardellitti, and Paolo Di Lorenzo. 2014. Communicating while Computing. *IEEE Signal Processing Magazine* 31, 6 (2014), 45–55.
[3] Elisa Bertino, Geeth Ashish de Mel, Alessandra Russo, Seraphin Calo, and Dinesh Verma. 2017. Community-based self generation of policies and processes for assets: Concepts and research directions. In *2017 IEEE International Conference on Big Data, BigData 2017, Boston, MA, USA, December 11-14, 2017*. IEEE Computer Society.
[4] Elisa Bertino and Mohammad R. Jahanshahi. 2018. Adaptive and Cost-Effective Collection of High-Quality Data for Critical Infrastructure and Emergency Management in Smart Cities - Framework and Challenges. *ACM Journal on Data and Information Quality (JDIQ)* 10, 1 (2018), 1:1–1:6.
[5] Yushi Cheng, Xiaoyu Ji Ji, Wenjun Zhu, Shibo Zhang, Kevin Fu, and Wenyuan Xu. 2024. Adversarial Computer Vision via Acoustic Manipulation of Camera Sensors. *IEEE Transactions on Dependable and Secure Computing* 21, 4 (2024), 3734–3750.
[6] Rodrigo S. de Moraes and Simin Nadjm-Tehrani. 2024. NetGAP: A graph- grammar approach for concept design of networked platforms with extra- functional requirements. *Engineering Applications of Artificial Intelligence* 133 (July 2024), 108089.
[7] Rodrigo S. de Moraes and Simin Nadjm-Tehrani. 2024. NeuralGAP: Deep Learning Evaluation of Networked Avionic Architectures. In *The 19th European Dependable Computing Conference (EDCC)*. 107–110. https://doi.org/10.1109/EDCC61798.2024.00031
[8] IFTTT. 2025. Welcome to IFTTT! https://ifttt.com/explore/new_to_ifttt [Last Accessed: Jan 20, 2025].
[9] Mark Law, Alessandra Russo, Elisa Bertino, Krysia Broda, and Jorge Lobo. 2020. FastLAS: Scalable Inductive Logic Programming Incorporating Domain-Specific

Optimisation Criteria. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, New York, NY, USA, February 7-12, 2020*.

[10] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. 2013. Playing Atari with Deep Reinforcement Learning.. In *CoRR abs/1312.5602*.

[11] Ligia Maria Moreira Zorello, Marco Sodano, Sebastian Troia, and Guido Maier. 2022. Power-Efficient Baseband-Function Placement in Latency-Constrained 5G Metro Access. *IEEE Transactions on Green Communications and Networking* 6, 3 (September 2022), 1683–1696.

[12] Anand Mudgerikar and Elisa Bertino. 2020. Jarvis: Moving Towards a Smarter Internet of Things. In *40th IEEE International Conference on Distributed Computing Systems (ICDCS 2020)*. IEEE, 122–134.

[13] Daniel Patrick Pereira, Celso Hirata, and Simin Nadjm-Tehrani. 2019. A STAMP-based ontology approach to support safety and security analyses. *Journal of Information Security and Applications* 47 (2019), 302–319. https://doi.org/10.1016/j.jisa.2019.05.014

[14] Francesco Restuccia and Tommaso Melodia. 2020. DeepWiERL: Bringing Deep Reinforcement Learning to the Internet of Selfe-Adaptive Things. In *IEEE International Conference on Computer Communications (Infocom), 6-9 July 2020*. IEEE ComSoC.

[15] Mohsen Rezvani, Aleksandar Ignjatovic, Elisa Bertino, and Sanjay K. Jha. 2015. Secure Data Aggregation Technique for Wireless Sensor Networks in the Presence of Collusion Attacks. *IEEE Transactions on Dependable and Secure Computing* 12, 1 (2015), 98–110.

[16] Mahdi Sharara, Sahar Hoteit, Véronique Vèque, and Francesca Bassi. 2022. Minimizing Power Consumption by Joint Radio and Computing Resource Allocation in Cloud-RAN. In *IEEE Symposium on Computers and Communications (ISCC)*. 1–6. https://doi.org/{10.1109/ISCC55528.2022.9912943}

[17] Ankush Singla, Elisa Bertino, and Dinesh Verma. 2020. Preparing Network Intrusion Detection Deep Learning Models with Minimal Data Using Adversarial Domain Adaptation. In *Proceedings of the 2020 ACM Asia Conference on Computer and Communications Security, AsiaCCS 2020, Taipei, Taiwan, October 05-08, 2020*.

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

[19] Ruikun Wang, Jiawei Zhang, Zhiqun Gu, Shuangyi Yan, Yuming Xiao, and Yuefeng Ji. 2022. Edge-enhanced graph neural network for DU-CU placement and lightpath provision in X-Haul networks. *Journal of Optical Communications and Networking* 14, 10 (2022), 828–839. https://doi.org/10.1364/JOCN.465369

[20] Zapier. 2025. Automate without limits. https://zapier.com/ [Last Accessed: Jan 20, 2025].