# Computation offloading of a vehicle's continuous intrusion detection workload for energy efficiency and performance


- George Loukas
- Yongpil Yoon
- Georgia Sakellari
- Tuan Vuong
- Ryan Heartfield

- Computing and Information Systems, Greenwich UK



## Abstract

Computation offloading has been used and studied extensively in relation to mobile devices. That is because their relatively limited processing power and reliance on a battery render the concept of offloading any processing/energy-hungry tasks to a remote server, cloudlet or cloud infrastructure particularly attractive. However, the mobile device's tasks that are typically offloaded are not time-critical and tend to be one-off. We argue that the concept can be practical also for continuous tasks run on more powerful cyber-physical systems where timeliness is a priority. As case study, we use the process of real-time intrusion detection on a robotic vehicle. Typically, such detection would employ lightweight statistical learning techniques that can run onboard the vehicle without severely affecting its energy consumption. We show that by offloading this task to a remote server, we can utilse approaches of much greater complexity and detection strength based on deep learning. We show both mathematically and experimentally that this allows not only greater detection accuracy, but also significant energy savings, which improve the operational autonomy of the vehicle. In addition, the overall detection latency is reduced in most of our experiments. This can be very important for vehicles and other cyber-physical systems where cyber attacks can directly affect physical safety. In fact, in some cases, the reduction in detection latency thanks to offloading is not only beneficial but necessary. An example is when detection latency onboard the vehicle would be higher than the detection period, and as a result a detection run cannot complete before the next one is scheduled, increasingly delaying consecutive detection decisions. Offloading to a remote server is an effective and energy-efficient solution to this problem too.



## Keywords

- Computation offloading;
- Intrusion detection;
- Energy efficiency;
- Detection latency;
- Cyber-physical systems;
- Vehicular security


---

# 1. Introduction

Offloading computation tasks from a user's device to a remote server, cloudlet or cloud can have multiple benefits. It can allow the utilisation of more powerful and more flexible computing resources and can provide an on-demand service, while also dramatically reducing the energy cost on the user's device. For these reasons, it has evolved into common practice for mobile devices. We argue that for largely the same reasons, the concept of computational offloading can be extremely useful for demanding, real-time and continuous tasks required by more powerful yet still resource-constrained and time-critical cyber-physical systems, such as vehicles. Yet, the approach remains largely unexplored. In the context of smart cities, it is anticipated that in the near future, the majority of vehicles on the road will benefit from various forms of constant connectivity with smart city infrastructures, as well as with each other. Already a wide range of vehicle-to-infrastructure (V2I) and vehicle-to-vehicle (V2V) technologies for safety and comfort have been developed, with some already deployed operationally. However, the dependence on V2I and V2V communications generates several new cyber threats to vehicles operating in smart cities. Protection against a wide range of evolving smart city cyber threats may be impractical if relying only on the vehicle's onboard processing systems, because of the adverse impact it would have on energy efficiency and latency. This makes computation offloading highly relevant in this context.

As proof of concept, we focus on the provision of high-performance intrusion detection for a robotic vehicle with limited processing resources and a requirement for low detection latency. A key challenge here is that to be meaningful the task needs to be performed continuously, which can be highly impractical for a battery-powered robotic vehicle. At the same time, it needs to provide a detection decision in time before it receives the next data to analyse. This is also a major challenge for advanced detection mechanisms that exhibit high detection latency and become impractical if they detect an attack after it has physically damaged the vehicle, e.g. through a command injection attack that disables or selectively engages the brakes on one side. Here, we develop a prototype system to demonstrate that it is possible to employ offloading in this case to reduce not only the energy cost but also the overall time taken to complete the task, even when the additional networking and processing overheads are taken into account. In several cases experimented with, reducing detection latency is not only useful, but necessary to ensure that detection can still be technically practical for a real-time system.

The practicality of computation offloading relies generally on two factors: performance and energy cost. For the particular task, performance relates to overall detection latency. In the onboard detection case, this effectively corresponds to the time it takes to complete the computation for detection (Fig. 1: top). In the offloading case, it includes the time it takes to send the data to the server over the network, the server to complete the computation and the time to receive the result back from the server (Fig. 1: bottom). However, the usual assumption in modelling offloading is that the latter delay is insignificant in comparison to the first two due to the difference in size between the data transmitted for offloading and the response. In our case, the response is simply a binary value (1 being an attack and 0 a non-attack).

Fig. 1.
To ensure continuous coverage, the periodic detection interval needs to be roughly equal to the periodic data collection interval. In the offloading case (bottom), the delay of transmitting the dataset over the network needs to be taken into account.

Figure options

As energy is power times time, the energy cost due to detection depends on the additional power consumed on the vehicle when running the detection computation onboard, as well as the time taken to complete it. In the offloading case, it also includes the energy cost of transmitting the data to the remote server, as transmission power consumption tends to be higher than idle operation power consumption. As with time, the energy cost of receiving the response can also be disregarded.

The key contributions of this work are: (i) A proof of concept prototype for computation offloading to provide a vehicle with access to high-end machine learning algorithms, with a case-study in deep learning for intrusion detection, (ii) a mathematical model of the difference in energy costs between onboard and offloaded computation for continuous periodic tasks, which is validated experimentally, and (iii) an experimental evaluation of the energy consumption and detection latency for a robotic vehicle by offloading deep learning-based detection mechanisms of low, moderate and high complexity.

## 2. Related work

Computation offloading has been thoroughly studied for mobile devices, in terms of both performance and energy efficiency, but to a very limited extent for IoT and cyber-physical systems, especially with regards to energy efficiency. The following is a brief overview of related work in these three areas.

### 2.1. Computation offloading for mobile devices

For mobile devices, offloading decisions are typically based on server speed, network bandwidth, available memory, server loads, and the amount of data needed to exchange between servers and mobile systems. Offloading for mobile devices may be performed with respect to methods, tasks, virtual machines (VMs), and applications. Most implementations, such as CloneCloud, Phone2Cloud, Cuckoo, COMET and MAUI, focus on identifying tasks that can be offloaded at runtime to improve performance and on how this offloading can be achieved.

From the perspective of energy efficiency, offloading is beneficial when the required energy to run an application exclusively on a mobile device is greater than the sum of the energy consumed to transmit the data to the cloud and the energy required by the mobile device whilst it is waiting for the cloud server to finish the computation. Of course, the quality of the network connection of the mobile device can significantly impact the efficiency of computation offloading. Therefore, for meaningful energy savings, the input data transmission needs to be low and the cloud servers need to have sufficiently greater computation power than the mobile device. Energy-aware scheduling of the executions of

offloaded computation into the cloud has also been studied in Zhang et al., where a mobile application is represented by a sequence of tasks that formulate a linear topology and they solve a minimum-energy task scheduling problem as a constrained shortest path problem on a directed acyclic graph. The energy cost of additional communication for offloading has been addressed in Geng et al. in order to make more energy-efficient offloading decisions in cellular networks. Also, computation offloading as a service for mobile devices has been suggested by Shi et al. to bridge the gaps between the offloading demands of mobile devices and the general computing resources, such as VMs, provided by commercial cloud providers.

Offloading the computation to more powerful machines has been proven to be beneficial for various real world mobile applications. For example, CloneCloud has demonstrated savings in energy consumption for virus scanning, image search, and behaviour profiling, where naturally the savings were greater with better network bandwidth (faster data transmission) and larger input sizes (more complex computation). Cuckoo has exhibited improved energy consumption for an object recognition application (from photos taken using the phone's camera) and COMET for various benchmarks including a chess game, an image editor and tools for mathematics.

### 2.2. Computation offloading for IoT systems

The concept of offloading for IoT systems, has been limited to IoT mobile applications, in the context of smart-cities applications, which effectively is the same research related to mobile offloading as in the previous section. For example, Mazza et al. propose a mechanism of offloading parts of smart-city applications to the best access point rather than the nearest one, based on a utility function which describes the Quality of Service in terms of energy cost, computational time and throughput. Rachuri also proposes an offloading mechanism for mobiles based on a multi-criteria (energy, latency and data rate) decision theory, which classifies tasks and dynamically adapts to changes such as mobile battery status and users data plan allowance. The mechanism is tested in the context of smart-building mobile applications. Interestingly, this work also utilises the concept of sensing offloading, where phone sensing tasks are offloaded to an infrastructure of sensors in smart-buildings to further save energy on the devices. However, this may reduce the practicality of using mobile devices in the first place.

Offloading for IoT systems is also used for facilitating authentication and authorisation. For example, Hummen et al. have utilised it to improve the practicality and applicability of the Datagram TLS protocol in IoT applications by offloading to a delegation server its most demanding task, which is the public-key cryptography required for the connection establishment. Compared to the standard public-key-based handshake, they have shown that offloading can reduce memory overhead by 64%, computations by 97% and network transmissions by 68%, but there was no evaluation of the reductions in energy costs.

### 2.3. Computation offloading for cyber-physical systems

While both IoT and cyber-physical systems are paradigms based on networked embedded systems as the core technology, and the concepts are largely overlapping, the former focuses more on sensing and the related data analytics, while the latter focuses more on time-critical interaction with the physical environment. Given the importance that physical impact has in cyber-physical intrusion detection, we look at the related section of the literature for

offloading for cyber-physical systems separately here. First steps have been taken in introducing offloading to cyber-physical systems, usually in relation to vehicular systems and external servers. An example is the work by Nimmagadda et al., which describes the real-time task of a robot's object recognition and tracking offloaded to a server if the offloading response time (server execution time plus data transfer time) is less than the local execution time. More recently, Toma and Chen proposed a decision mechanism for hard real-time embedded systems that selects what and when to offload to a server, by taking into consideration also timing constraints such as local execution time and offloading response time. They further adopt their scheme to reserve resources in the server to ensure the offloading latency. To validate their mechanism, they used an image-processing application in a robot system.

Wang et al. addressed the issue of reducing mobile data traffic while maintaining good QoS by offloading mobile data traffic of vehicles with the use of Wi-Fi and VANET technologies rather than using cellular networks. Liu et al. investigated the impact of components that are unreliable, in terms of their timing, and suggest a method for utilising them, by estimating their worst case response time. Finally, in Wan et al., VMCIA is an architecture for integrating vehicular systems and mobile cloud computing, which can also be used for computation offloading.

Contrary to mobile device offloading, the related work for cyber-physical systems places emphasis only on the time-sensitivity of offloading for vehicles, focusing on latency benefits and not in energy efficiency. This is a significant omission because vehicles are fundamentally constrained by their energy source and rate of energy consumption not only for their mobility and autonomy, but also for their use of advanced computation algorithms with high processing demands. We address this omission here with a mathematical model, a prototype implementation and thorough experimental evaluation. We start in Section 3 with a description of the testbed and the prototype implementation for studying the feasibility of offloading a vehicle's high-end intrusion detection. The corresponding mathematical model is detailed in Section 4, followed by the experimental evaluation in Section 5.

## 3. Offloaded intrusion detection prototype

In this section, we detail the experimental setup, including the robotic vehicle that requires real-time intrusion detection, the server to which intrusion detection will be offloaded, the details of the intrusion detection process and the networking configuration allowing the exchange of the offloaded data between the vehicle and the server. To demonstrate the feasibility of offloading high-end intrusion detection that is not normally available to cyber-physical systems due to its heavy processing, we employ deep learning of low, moderate and high complexity.

### 3.1. The vehicle

The vehicle that we have used for development and experimentation is a four-wheel-drive robotic vehicle controlled via an on-board Intel Atom computer running the Linux Fedora 32-bit operating system. In terms of specifications, the processor is a dual-core Intel Atom D525 at 1.8 GHz with 1 MB L2 Cache and 2 GB DDR3 memory at 1066 MHz. An Arduino micro-controller is responsible for driving the motors. The vehicle also carries a USB pan and tilt camera for situational awareness and remote navigation. Standard magnetic encoders are

fitted to the two rear wheel motors providing the angular position and speed of the vehicle. The vehicle is controlled remotely by a client-server application that functions by relaying commands in real-time over a TCP socket to the vehicle control board via wired Ethernet or 802.11g/n Wi-Fi interface. The vehicle TCP server listens on port 7000 for incoming client connections and once a connection is established, the operator is then able to issue commands to the vehicle's respective controllers. The client TCP connection persists for the duration of vehicle operation and control.

Different attacks have different impacts on the computation, communication and physical operations of the robot. In particular, the physical impact is not only an adverse effect of a cyber-physical attack, but also an indication and potentially a signature of the attack, hence an opportunity for detection. Monitoring and analysing these physical features has been shown to improve the performance of a system designed to detect cyber attacks against a vehicle.

### 3.2. The server for task offloading

Offloading is carried out on a mid-range Dell PowerEdge R430 1U rack server equipped with Intel Xeon E5-2640 v3 processor (2.6 GHz, 20 M Cache), 16 GB RDIMM 2133 MT/s Dual Rank memory and a Broadcom 5720 Quad Port 1 GBE network card.

### 3.3. The continuous task to be offloaded

The deep learning based detection employed here is a binary classification mechanism aiming to determine whether the host system is under attack or not. By studying a supervised set of collected sensor and actuator data, a deep neural network model can see patterns of high-level abstractions in input data through a complex architecture of non-linear transformations of the data. As the data is collected in time sequence, we are applying a Recurrent Neural Network, which is highly suitable for capturing temporal behaviour. To deal with the problem of vanishing gradient, we have applied the Long Short Term Memory technique to remember significant features for an arbitrary length of time. For the Deep Learning prototyping, we used the Python neural network library Keras to run on top of the TensorFlow/Theano library. After a training phase that takes several hours, we produce two output files: the deep learning model architecture and weight values, which can then be used to run the binary classification process of the intrusion detection in real-time either onboard the vehicle or offloaded to a server.

To be meaningful, the task of intrusion detection needs to run continuously. For the particular type of classification, continuous operation means running at periodic intervals roughly equal to the intervals used for collection of the data for the features used for the classification (Fig. 1). This ensures that there are no "blind spots" in the classification process, and if it misses a cyber attack it will be because of the classification method's inaccuracy, not because the corresponding data were not used in any detection interval.

### 3.4. The networking configuration of offloading

The network testbed consists of three discrete modules, an 802.11n wireless local area network (WLAN), a point to point wide area network (WAN) and a remote server. The WLAN provides the vehicle with mobile connectivity to a local network gateway conducting

port forwarding between the vehicle and the deep learning server for offloading, through an SSH tunnel over the WAN. Using the client-side URL transfer library *libcurl* and *PyCURL* (Python Interface to libcurl), the vehicle offloads detection tasks by uploading sensor data samples, at interval period $T$ (for which we try five different values: $T=0.5s, T=0.5s, T=1s, T=1s, T=2s, T=2s, T=5s$ $T=5s$ and $T=10s$ $T=10s$) to an Apache HTTPS service on the deep learning server. The HTTPS protocol was selected to perform client-server data transfer to provide a secure, reliable and energy efficient communication channel. Specifically, HTTP over SSLv3 provides data confidentiality and integrity of HTTP payloads containing vehicle sensor data, whilst HTTPs native reuse of existing persistent connections, data transfer pipelining and automatic data compression optimises TCP performance and packet transfer speed.

All data communication between the vehicle and remote server is vehicle initiated, through use of python scripts making calls to the libcurl library. The scripts operate as a set of continual loops. On initiation, a sensor sample is retrieved and transferred via an HTTP POST to the deep learning server. If the POST is successful, another loop is then spawned, continually polling the server with HTTP GET requests until a detection result is successfully retrieved. On receipt of the detection result, the next sensor sample is then collected and the HTTP data transfer process is repeated (Fig. 2).

Fig. 2.
The network architecture used for offloading.

## 4. Energy saving model for offloaded intrusion detection

We denote $n$ as the number of data points collected in real-time for the intrusion detection process, and $\tau$ as the interval at which each data point is collected. As the process of detection is continuous rather than one-off, we estimate energy costs for a period $T$, which corresponds to a window of $n$ data points, as $T=n\tau$ $T=n\tau$. We denote as $t_c$ and $P_v$ the average time to complete the detection and power consumption of the vehicle respectively when it carries out the detection onboard. $t_c$ is not a constant but a function, which may be modelled differently for different tasks or systems. Here, it depends primarily on the complexity of the detection approach and the data size $dn$ of the dataset of the most recent $n$ data points collected, where $d$ is the data size for one data point. Also, $P_i$ is the power consumption of the vehicle when idle, and $P_x$ when transmitting data to the server. We can consider $P_v$, $P_i$ and $P_x$ as relatively static values for a vehicle like ours. We have confirmed this experimentally. As an example, see Fig. 3 showing $P_v$ against time in representative runs of the deep learning algorithm on the vehicle at a configuration of low complexity and for different values of n. Between the five different configurations, the average value ranges between 61.15 W and 62.58 W. For simplicity, for our mathematical model's application, we use the average values for the duration of all five runs, as measured experimentally. The assumption that the power consumption is relatively static across time and across different values of n may not hold in systems that experience significant variability. For example, $P_v$ may vary noticeably over time if computation involves multiple phases of continuously varying processing needs; $P_i$ may vary a lot while the vehicle operates on uneven terrain, and $P_x$ could vary due to network variability (e.g., due to increased radio channel contention). Our setup is, on purpose,

relatively simple, to facilitate the evaluation of the proof of concept before expanding into factors of potential variability in future work.

Fig. 3.
Power consumption on vehicle against time for representative runs of the deep learning algorithm on the vehicle at a configuration of low complexity and for different values of $n$.

The energy saving $\Delta E$ for a detection period $T$ is the energy $E_v$ consumed on the vehicle if the detection is run onboard minus the energy $E_o$ consumed on the vehicle if the detection is offloaded. If the complexity of the task is sufficiently low so that the time $t_c$ it takes to complete is lower than the period ($T \geq t_c$), then the energy cost $E_v$ in a period $T$ includes the energy cost of running the detection onboard for time $t_c$ and the energy cost of idle operation for the remaining time of $T-t_c$ $T-tc$. Otherwise, $E_v$ is the energy cost of running the detection onboard for time $T$. In the offloading case, in accordance with the usual representation in the task offloading literature, the length of time for the wireless transmission can be modelled simply as $\frac{dn}{r_x}$, where $dn$ is the size of the data transmitted and $r_x$ the wireless transmission rate. $E_o$ includes the energy cost of wirelessly transmitting the data to the server for time $\frac{dn}{r_x}$ and the energy cost of idle operation for the remaining time of $T - \frac{dn}{r_x}$ (Table 1).

$$E_v = \mathbb{1}[T \geq t_c](P_v t_c(\gamma, d, n) + P_i(T - t_c)) + \mathbb{1}[T < t_c]P_v T$$

$$E_o = P_i\left(T - \frac{dn}{r_x}\right) + P_x\frac{dn}{r_x}$$

$$\Delta E = E_v - E_o = \mathbb{1}[T \geq t_c](P_v t_c + P_i(T - t_c)) + \mathbb{1}[T < t_c]P_v T - P_i\left(T - \frac{dn}{r_x}\right) - P_x\frac{dn}{r_x}$$

Table 1.
List of symbols.

| Symbol | Description |
|---|---|
| $T$ | Detection period |
| $n$ | Number of data points collected in period $T$ |
| $d$ | Data size for one data point |
| $\tau$ | Interval at which each data point is collected |
| $P_v$ | Average power consumption of the vehicle when detection is run onboard |
| $P_i$ | Average power consumption of the vehicle when idle |
| $P_x$ | Average power consumption of the vehicle when transmitting data to server |
| $t_c$ | Time to complete detection when it is run onboard |
| $r_x$ | Wireless transmission rate |
| $E_v$ | Energy consumed on the vehicle when detection is run onboard |
| $E_o$ | Energy consumed on the vehicle when detection is offloaded |

| Symbol | Description |
|---|---|
| $\Delta E$ | Energy saving in a detection period $T$ |
| $\gamma, \beta$ | Linear regression coefficients for $t_c$ following simple $t_c = \gamma dn + \beta$ format |
| **1[.]1[.]** | 1[X]=1 1[X]=1 if $X$ is true and 0 otherwise |

Where 1[X]=1 1[X]=1 if $X$ is true and 0 otherwise.

With regards to the particular application of real-time detection, the $T < t_c$ case is highly impractical, because it means that detection runs are effectively queued, introducing to subsequent runs an additional queuing delay, which is continuously increasing towards infinity. So, we can consider the $T \geq t_c$ case as the only practical one and simplify Eq. (3) as:

$$\Delta E = P_v t_c + P_i(n\tau - t_c) - P_i\left(n\tau - \frac{dn}{r_x}\right) - P_x\frac{dn}{r_x}$$

$$= (P_v - P_i)t_c + (P_i - P_x)\frac{dn}{r_x}$$

Note that the vehicle is technically never idle, because it is moving and receiving commands from its operator continuously. However, for terminology purposes and in conformance with the computation offloading literature, by "idle" here we refer to the state where the vehicle is not carrying out specifically detection-related processing or detection-related transmission. Also, for simplicity, we take into account only the power to transmit and not to receive the detection response, as the latter is practically insignificant and can be considered as part of the idle state for a robotic vehicle like ours which is continuously receiving data from its operator, but is transmitting only when necessary.

The type of model that is appropriate for estimating time $t_c$ depends on the algorithm (or set of algorithms) utilised by the intrusion detection mechanism, as well as the architecture of the processing system. For the vehicle and intrusion detection mechanism at hand, we have observed experimentally that $t_c$ is highly linear against the size of the input data $dn$ utilised (Fig. 4), following a simple $t_c = \gamma dn + \beta$ format, where $\{\gamma, \beta\} \in R$ depend on the complexity of the deep learning configuration. Simple linear regression on experimental measurements yields $\{\gamma, \beta\}_{low} = \{0.0003, -0.0982\}$ for the lower complexity case we experimented with, $\{\gamma, \beta\}_{moderate} = \{0.0004, -0.1665\}$ for the moderate case, and $\{\gamma, \beta\}_{high} = \{0.0007, -0.3134\}$ for the higher complexity one. The particular three cases are explained in Sections 5.1–5.3.

Fig. 4.
Onboard computation time *vs.* input data size for deep learning models of different complexity, illustrating the linearity of $t_c$.

As a result, for the particular periodic task of real-time deep learning detection of cyber attacks based on the $n$ data points in a period $T$, Eq. 5 becomes:
<img height="42" border="0"

As a result, for the particular periodic task of real-time deep learning detection of cyber attacks based on the *n* data points in a period *T*, Eq. 5 becomes:
Equation.

We validate this model by comparing with experimental results in Section 5.4.

## 5. Experimental evaluation

Here, we evaluate experimentally the practicality of offloading for intrusion detection based on deep learning models of low, moderate and high complexity. Their complexity loosely depends on the number of features measured in the vehicle, as well as the number of neurons in the deep learning model. The precise parameters of these three deep learning model configurations (number of features and number of neurons) were chosen experimentally so as to be meaningful in terms of detection accuracy, as well as relatively practical in terms of the time taken to train each model and the time needed to produce a detection result when tested. We have experimented with configurations of much higher complexity too, but they required several days of training, yet did not produce any noticeable improvement in detection accuracy in our case. Although the details of the detection approaches are beyond the scope of this paper, for context, we provide a brief summary of the attack and the detection.

We employ a command injection attack, which in previous research has been shown to be particularly difficult to detect with lightweight detection mechanisms. For consistency, we employ the same settings and scenario as in Vuong et al., where the vehicle receives commands from its legitimate operator to move forward, and at the same time receives rogue "stop" or "turn left" commands from an attacker. As a result, the vehicle exhibits both inconsistent cyber behaviour and intermittent physical jittering during the attack. The detection provided in Vuong et al. is based on a decision tree-based approach, which runs comfortably on the vehicle but achieves an accuracy rate of only 72.8%. Although this is not within the scope of this paper, to illustrate the motivation for employing a higher complexity approach, we report that with deep learning, our detection method exceeds 85% accuracy. Fig. 5 shows the setup of the command injection attack scenario and offloaded detection.

In our experiments, we have used $\tau=0.02\,s, \tau=0.02s, d=60 d=60$ bytes and *n* varying between 25 and 500 data points in a period $T$ , ranging correspondingly between 0.5 s and 10 s. The transmission rate is $r_x=125000 rx=125000$ bytes/s. See Table 2 for quick reference.

Table 2.
Evaluation parameters.

| Parameter | Value |
|---|---|
| $\tau$ | 0.02 s |
| $T$ | 0.5 s, 1 s, 2 s, 5 s, 10 s |
| $n$ | 25, 50, 100, 250, 500 |
| $d$ | 60 bytes |

| Parameter | Value |
|---|---|
| $P_v$ | 61.99 W |
| $P_i$ | 55.93 W |
| $P_x$ | 59.34 W |
| $r_x$ | 125,000 bytes/s |
| $\gamma, \beta$ | low: {0.0003,−0.0982}, {0.0003,−0.0982}, moderate: {0.0004,−0.1665}, {0.0004,−0.1665}, high: {0.0007,−0.3134 |

## 5.1. Low complexity detection

Here, the deep learning-based detection takes into account only nine features collected on the vehicle and a relatively simple neural network design with 600 hidden neurons. In Fig. 6, the break-even line corresponds to the points where the detection latency would be equal to the detection period. Any value higher than this line renders detection technically impractical, because if it is higher than the detection period, it introduces an infinitely increasing queuing delay to subsequent detection runs.[2] From the particular perspective, we see in Fig. 6 that detection of low complexity is practical in all cases when run onboard. However, with the exception of the n=25n=25 case, detection latency is reduced dramatically when offloaded. In fact, the greater the number of data points, the greater the benefit of offloading in terms of detection latency reduction. This was expected as the shorter the time it takes to complete a task, the lower the benefit of offloading it to a server, because of the impact of the network delay.

Fig. 6.
Detection latency in the case of low complexity detection. The break-even line corresponds to the points where the detection latency would be equal to the detection period.

## 5.2. Moderate complexity detection

Here, the deep learning-based detection takes into account 9 features and a more complex neural network design with 800 hidden neurons. In Fig. 7, we observe that with the exception of the first case (n=25n=25), where detection latency is lower than the period *T*, in all other cases it is marginally impractical when run onboard, but practical in all cases when offloaded. Again, the greater the number of data points, the greater the benefit of offloading.

Fig. 7.
Detection latency in the case of moderate complexity detection. The break-even line corresponds to the points where the detection latency would be equal to the detection period.
Fig. 8.
Detection latency in the case of high complexity detection. The break-even line corresponds to the points where the detection latency would be equal to the detection period.

### 5.3. High complexity detection

Here, the deep learning-based detection takes into account 45 features and a more complex neural network design with 1000 hidden neurons. In Fig. 8, we observe that detection of high complexity is impractical in all cases when run onboard, but only in the first case (n=25n=25) when offloaded. Again, the greater the number of data points, the greater the benefit of offloading.

### 5.4. Energy saving

Here, we test the validity of the model presented in Section 4 by comparing its prediction with our energy measurements from actual experimental runs (Fig. 9). For the energy costs between onboard and offloaded detection to be comparable (and thus the saving measurements of offloading to be meaningful), both onboard and offloaded detection cases need to be technically practical. For this reason, we perform our comparison on the low complexity detection case, which meets this requirement. Fig. 9 shows that the prediction based on the model is very close to the real measurements on the testbed. The left vertical axis corresponds to the energy saving over a period T=nτ,T=nτ, while the right vertical axis corresponds to the vehicle's total energy saving over 24 h (86,400 s) of continuous operation, estimated as $\frac{86400}{n\tau}\Delta E$. While $\Delta E$ (which corresponds to one period) is linear against $n$, note that the total energy saving over 24 h is not, because it involves a division by $n$.

Fig. 9.
Comparison of mathematical model and experimental measurements for low complexity detection (left vertical axis: duration $T$; right vertical axis: duration 24 h).

Based on Eq. 6, offloading is beneficial in terms of energy efficiency if $\Delta E > 0$, which solving for $n$ corresponds to:

$$n > \frac{\beta(P_i - P_v)}{(P_v - P_i)d\gamma + \frac{d}{r_x}(P_i - P_x)}$$

For the average power consumption values measured in our experiments, this yields $n > 5.54$ for the low complexity case, $n > 7.02$ for medium, and $n > 7.51$ for high. So, the lowest numbers of data points for which offloading would be beneficial for energy efficiency are n=6,n=6,n=8n=8 and n=8n=8 respectively, which are considerably lower than the minimum required by the deep learning approach to provide a meaningful detection result (experimentally observed to be around n=25,n=25, hence the lowest value chosen in the experiments reported here). This explains why offloading reduces the energy consumption on the vehicle in all our experiments.

The main strength of offloading such a task is that, in most cases, the vehicle benefits from both reduced energy costs and reduced detection latency at the same time. This is illustrated in Fig. 10, which displays the experimental results in terms of total energy consumption

against detection latency achieved for onboard and offloaded detection, for the low, moderate and high complexity cases (although the latter two are not practical in terms of latency when run onboard). Each arrow displays the change of energy consumption and latency values when moving from onboard to offloaded detection. Where the arrow's direction points towards the bottom left, offloading has achieved both targets (lower energy consumption and lower latency). We see that, from the perspective of the vehicle, the experiments have shown that energy was saved in all cases, as was also predicted by our mathematical model. Overall detection latency increases for $n=25$ and $n=50$ in low complexity, and for $n=25$ in high complexity detection), but is reduced in all other cases.

Fig. 10.
Energy consumption *vs.* detection latency for onboard and offloaded detection. Each arrow displays the change of energy consumption and latency values from onboard to offloaded detection.

In terms of selecting optimal $n$, the mathematical model shows that energy saving of a continuously operated detection over a period of 24 h is monotonically increasing against $n$. So, the larger the $n$ the better. However, in practice, both mathematical model and energy measurements agree that it levels off between $n=150$ and $n=250$, and from then on, any improvements in long-term energy efficiency are marginal. Also, $n$ is at the same time a key parameter of the accuracy of the deep learning algorithm because it represents the amount of data available to take a detection decision. So, in practice, the identification for optimal value should take into account both energy efficiency and accuracy. This is part of our future work.

## 6. Conclusions

Intrusion detection for time-critical cyber-physical systems, such as vehicles, has been traditionally limited to lightweight techniques because they are constrained by their timeliness and energy efficiency requirements. Detecting that a vehicle is under a cyber attack after the attack has caused physical damage is impractical. Similarly, running a processing-heavy detection algorithm that would reduce considerably the operational autonomy of the vehicle by increasing its energy consumption is again impractical. We have produced a mathematical model which estimates the energy cost of a continuous deep learning based detection and allows identifying the minimum number of data points used by the detection algorithm where offloading saves energy. For the particular robotic vehicle and remote offloading server experimented with, this was always the case, as the minimum required was considerably lower than the minimum needed by the detection algorithm. In terms of overall detection latency, we have also observed significant improvement, but in our analysis and experiments, we have assumed a simple network which introduces network delay that is rather consistent. So, although we took the network delay into account, our focus was on the variations of the latency introduced by the detection itself. This can be a reasonable assumption for networks controlled by the same operators as the vehicle, but would be too optimistic in scenarios where the network is a general-purpose network not under the same users' control (e.g., over the Internet if using a remote cloud). In future work, we will take into account a variety of more diverse network configurations (including multi-hop connections) and unstable or volatile network conditions and study how and to what extent they affect the practicality of offloading.

Performing a highly demanding task continuously (as in this case, intrusion detection based on deep learning) may not always be a wise use of resources even if offloaded to a server. For example, we have shown that to ensure that it is completed in time in every detection period, deep learning based detection may need to be limited to low complexity approaches. One method that may be more practical is to employ a traditional lightweight alternative, such as decision-tree based detection, in continuous mode, and only trigger offloaded deep learning analysis to confirm detection when an attack is suspected or to determine its type or likely impact. Such hybrid approaches are commonly used measures for improving efficiency. The problem remains though that if the initial lightweight detection exhibited false negatives, then it could fail to trigger the deep learning phase in cases of actual attacks. In that case, the benefits of offloading are rendered irrelevant. Minimising false negatives is an open research problem for cyber-physical system security in general, which we are also working towards by developing low-latency deep learning binary classification methods.