EnergyShield (ICCPS '23, Submission 251)

EnergyShield is an Autonomous Driving System framework designed to save energy on-vehicle by wirelessly offloading NN computations to edge computers, while at the same time maintaining a formal guarantee of safety. In particular, EnergyShield uses a barrier function/controller shield to provide provably safe edge response times for NN offloading requests; on vehicle evaluation provides a safety fallback.

EnergyShield is described in the paper:

> *EnergyShield: Provably-Safe Offloading of Neural Network Controllers for Energy Efficiency.*
> Mohanad Odema, James Ferlez, Goli Vaisi, Yasser Shoukry and Mohammad Abdullah Al Faruque.
> ICCPS 2023: 14th ACM/IEEE International Conference on Cyber-Physical Systems.

which is appended to this document, and hereafter referred to as [ES23].

[ES23] contains a number of experiments showing the efficacy of EnergyShield in the Carla simulation environment. This README describes how to replicate those results using code packaged in a Docker image. In particular, this artifact reruns from scratch the following experiments from [ES23]:

Experiment 1 ***[ES23, Section 5.2]***: *Energy Efficiency and Safety Evaluation of EnergyShield through Carla Simulation runs*

Experiment 2 ***[ES23, Section 5.3]*** : *Performance Gains from EnergyShield given wireless channel variation*

Experiment 3 ***[ES23, Section 5.4]***: *EnergyShield generality to other DRL agents*

For each of these experiments, this artifact re-generates both new raw data and the analogous plots shown in [ES23].

Contents

0. Terminology

This repeatability artifact uses Docker. We will use the following terminology throughout:

- The **HOST** will refer to the system running Docker (e.g. your laptop).
- The **CONTAINER** will refer to the "virtualized" system created by Docker (this is where the code from the artifact is run).

Commands meant to be executed from a shell on the host or the container will be prefixed with one of the following comments, respectively:

```
# <<< HOST COMMANDS >>>
# <<< CONTAINER COMMANDS >>>
```

1. System Requirements

HARDWARE (HOST):

1. An x86-64 CPU
2. 32GB of RAM
3. An NVIDIA GPU with 8GB of VRAM **(Geforce 20xx series or later; RTX 2070, GTX 2080Ti and V100 cards were tested)**; headless GPUs will work (e.g. servers and Amazon EC2/Microsoft Azure instances)
4. At least 80GB of free disk space on the filesystem where Docker stores images (the filesystem containing `/var/lib/docker` by default)
5. An internet connection that can download ~30GB of data

SOFTWARE (HOST):

1. Un-virtualized Linux operating system; headless and cloud (e.g. Amazon EC2/Microsoft Azure) installs will work (tested on Ubuntu 20.04, but any distribution that meets the remaining requirements should work)
2. Official Linux NVIDIA drivers **(version >= 515.76 is ***REQUIRED***)**
3. A recent version of Docker Engine **(version >= 19.03)**; also known as Docker Server but **not** Docker Desktop
4. A recent version of `git` on the path
5. The `bash` shell installed in `/bin/bash`
6. A user account that can run Docker containers in privileged mode (i.e. with the `--privileged` switch)

> **WARNING:** NVIDIA driver version >=515.76 is a **STRICT REQUIREMENT**. This repeatability artifact **WILL NOT WORK** unless the host has official NVIDIA drivers version 515.76 or higher installed.

2. Setup

(i) Host Paths

Choose an install location on the host:

```
# <<< HOST COMMANDS >>>
INSTALL_LOCATION=/path/to/some/place/convenient
cd "$INSTALL_LOCATION"
```

EnergyShield will be installed on the **host** at:

```
ES_PATH="$INSTALL_LOCATION/EnergyShield"
```

> **NOTE:** All subsequent **HOST** paths in this readme are assumed to be relative to `$ES_PATH` unless otherwise specified.

(ii) Installing EnergyShield/Building the Docker Image

To install EnergyShield and build the EnergyShield Docker image, execute the following in a Bash shell on the host (from `$INSTALL_LOCATION`):

```
# <<< HOST COMMANDS >>>
cd "$INSTALL_LOCATION"
git clone --recursive https://github.com/MohanadOdema/EnergyShield
cd EnergyShield
./dockerbuild.sh # WARNING: downloads ~30GB of data, and may take > 1 hour
even after download!
```

> **NOTE:** `dockerbuild.sh` only ever needs to be run once per host user account, unless an updated version is released.

(iii) Starting a Docker Container

First, make sure you have successfully built an EnergyShield Docker image. See Section 2 *(ii)* above.

Then to start the EnergyShield Docker container, execute the following in a Bash shell on the host (from `$ES_PATH`):

```
# <<< HOST COMMANDS >>>
./dockerrun.sh --interactive --start-carla
```

This should place you at a Bash shell inside a container with EnergyShield installed. The container's Bash shell will have a prompt that looks like:

```
carla@ece2ade62bc5:~$
```

where `ece2ade62bc5` is a unique container id (i.e. yours will be different).

> **WARNING:** if you exit the container's Bash shell, then the **container and all experiments will stop**. You may restart the container with the **host** command:
>
> ```
> # <<< HOST COMMANDS >>>
> ./dockerrun.sh --interactive --start-carla
> ```

(iv) Testing the Container

First, make sure you have an EnergyShield container running. See Section 2 *(iii)* above.

Then from the **container**'s Bash shell execute:

```
# <<< CONTAINER COMMANDS >>>
ps | grep Carla
```

You should see output listing two processes related to Carla:

```
    45 pts/0    00:00:00 CarlaUE4.sh
    52 pts/0    00:01:00 CarlaUE4-Linux-
```

> **WARNING:** if the above command produces no output or output unlike the above, then Carla is not running, and this repeatability artifact **WILL NOT WORK**.
>
> **FIX:** Double check that your host system meets the requirements in Section 1, especially the NVIDIA driver requirements (incorrect NVIDIA drivers are usually what prevents Carla from starting). If your system meets these requirements, then try restarting the container using the following sequence of commands:
>
> ```
> # <<< CONTAINER COMMANDS >>>
> # Exit from the container if it's still running:
> exit
> ```
>
> And then in `$ES_PATH`:
>
> ```
> # <<< HOST COMMANDS >>>
> ./dockerrun.sh --remove # Remove any existsing container
> ./dockerrun.sh --interactive --start-carla
> ```

3. Experiment 1 - Energy Efficiency and Safety Evaluation

In this experiment, we compared EnergyShield with purely on-vehicle NN controller evaluation, both in terms of energy consumption and safety; see [ES23], Section 5.2. This comparison was made using a single RL-trained NN controller driving a fixed track; safety entails avoiding randomly spawned stationary obstacles along this track. This artifact reuses the same track and NN controller from our experiment, but the obstacle locations and instantaneous wireless-link performance are randomized.

To rerun this experiment, ensure that you have an EnergyShield container running (see Section 2 *(iii)*), and execute the following commands in the **container**'s bash shell:

```
# <<< CONTAINER COMMANDS >>>
cd /home/carla/EnergyShield
# Run Experiment 1 Carla simulation
./scripts/run_exp1.sh NUM_EPS
```

`NUM_EPS` is an optional argument specifying the number of episodes to run (default: `NUM_EPS=3`); an episode is defined as one run of the fixed track until either the vehicle completes the track or hits an obstacle. For [ES23], we ran this experiment for 35 episodes.

> **NOTE:** Running this script with the default `NUM_EPS=3` takes around 1 hour on a workstation with 32 GB RAM and NVIDIA GPU 2070 RTX super.

> **NOTE:** When the script finishes, you will be returned to a shell prompt in the container (see Section 2 *(iii)*). If the script is successfully running, status information will be output to the console regularly.

> **NOTE:** Experiments can be run consecutively in the same container without interruption, or in between container restarts.

> **WARNING:** Occasionally, the experiment script may fail to connect to Carla even if Carla is running (see Section 2 *(iv)*). This is a known issue in Carla on slow host machines; if it occurs, simply re-run the script above. If this fails, try restarting the container according to the directions in Section 2 *(iv)*.

The primary outputs of this script are figures that summarize the energy savings and safety properties of EnergyShield (across the number of episodes specified by `NUM_EPS`). These figures can be found on the **HOST** at the following paths:

```
$ES_PATH/container_results/Fig5_Energy.pdf
$ES_PATH/container_results/Fig5_Safety.pdf
$ES_PATH/container_results/Fig6_traj_noise_False.pdf
$ES_PATH/container_results/Fig6_traj_noise_True.pdf
$ES_PATH/container_results/Fig7_Ergy_v_dist.pdf
```

Their filenames match them to the figures that appear in [ES23]. For reference, the figures from [ES23] are also available with the same filenames in `$ES_PATH/paper_results`.

This script also outputs the raw Carla simulation data of each episode (i.e. the simulation time-stamped positions, control actions, etc. of the vehicle); this data is placed in `$ES_PATH/container_results/raw_data`; the analogous raw data from our simulations can be found in `$ES_PATH/paper_results/raw_data` for comparison. The format and structure of this data is described in the subsequent Appendix.

Finally, Figure 7 (Energy vs. distance) is derived from some summary statistics of raw data noted above; these summaries are produced as several `.CSV` files output to `$ES_PATH/container_results/distance/*.csv`.

4. Experiment 2 - Performance under wireless channel variation

In this experiment, we evaluated the energy savings provided by EnergyShield as a function of different wireless-link conditions to the edge; see [ES23], Section 5.3. This experiment uses mostly the same setup as Experiment 1, including the same NN controller and track. However, in this experiment, batches of episodes are run under five different simulated wireless-link conditions. As before, this artifact reuses the same track and NN controller from our experiment, but the obstacle locations and instantaneous wireless-link performance are randomized.

To rerun this experiment, ensure that you have an EnergyShield container running (see Section 2 *(iii)*), and execute the following commands in the **container**'s bash shell

```
# <<< CONTAINER COMMANDS >>>
cd /home/carla/EnergyShield
# Run Experiment 2 Carla simulations:
./scripts/run_exp2.sh NUM_EPS
```

The optional parameter `NUM_EPS` has a similar interpretation to Experiment 1 (default `NUM_EPS=3`), but here it represents the number of episodes to run under each wireless-link condition (of which this experiment contains five).

> **NOTE:** Running this script with the default `NUM_EPS=3` takes around 1 hour on a workstation with 32 GB RAM and NVIDIA GPU 2070 RTX super.

> **NOTE:** When the script finishes, you will be returned to a shell prompt in the container (see Section 2 *(iii)*). If the script is successfully running, status information will be output to the console regularly.

> **NOTE:** Experiments can be run consecutively in the same container without interruption, or in between container restarts.

> **WARNING:** Occasionally, the experiment script may fail to connect to Carla even if Carla is running (see Section 2 *(iv)*). This is a known issue in Carla on slow host machines; if it occurs, simply re-run the script above. If this fails, try restarting the container according to the directions in Section 2 *(iv)*.

The primary outputs of this script are figures that summarize the energy performance of EnergyShield under these different wireless conditions, including the number of local-execution fallbacks required. These figures can be found on the **HOST** at the following paths:

```
$ES_PATH/container_results/Fig8_windows.pdf
$ES_PATH/container_results/Fig9_energy.pdf
```

As before, their filenames match them to the figures that appear in [ES23]. Likewise, the figures from [ES23] are also available with the same filenames in `$ES_PATH/paper_results`.

The raw data from this experiment is similarly placed in `$ES_PATH/container_results/raw_data`, and the analogous raw data from our simulations is in `$ES_PATH/paper_results/raw_data` for comparison. See the the subsequent Appendix for a description of the format and structure of this raw data.

5. Experiment 3 - Comparison Between Multiple Controllers

In this experiment, we again compared EnergyShield with purely on-vehicle NN controller evaluation, both in terms of energy consumption and safety; however, in this experiment, we compared this performance between different NN controllers (four total, including the one used in Experiment 1). Hence, this experiment effectively amounts to re-running Experiment 1 on three additional RL-trained NN controllers; see [ES23], Section 5.4. As before, this artifact reuses the same track and NN controllers from our experiment, but the obstacle locations and instantaneous wireless-link performance are randomized.

> **NOTE:** To get complete results you should run Experiment 1 first.

To rerun this experiment, ensure that you have an EnergyShield container running (see Section 2 *(iii)*), and execute the following commands in the **container**'s bash shell:

```
# <<< CONTAINER COMMANDS >>>
cd /home/carla/EnergyShield
# Run Experiment 3 generate model statistics
./scripts/run_exp3.sh NUM_EPS
```

`NUM_EPS` is an optional argument that has a similar interpretation to Experiment 1 (default `NUM_EPS=3`), but here it represents the (common) number of episodes to be run on *each* of the other three agents.

> **NOTE:** Running this script should take about three times as long as running Experiment 1 for the same number of episodes.

> **NOTE:** When the script finishes, you will be returned to a shell prompt in the container (see Section 2 *(iii)*). If the script is successfully running, status information will be output to the console regularly.

> **NOTE:** Experiments can be run consecutively in the same container without interruption, or in between container restarts. However, to incorporate the results of Experiment 1, do not *remove* the container between runs of `run_exp1.sh` and `run_exp3.sh`.

> **WARNING:** Occasionally, the experiment script may fail to connect to Carla even if Carla is running (see Section 2 *(iv)*). This is a known issue in Carla on slow host machines; if it occurs, simply re-run the script above. If this fails, try restarting the container according to the directions in Section 2 *(iv)*.

The primary outputs of this script are a collection of `*.CSV` files containing summary data for each agent, as reported in Table 1 of [ES23]. This data includes average center deviance (CD), Track Completion Rate (TCR) and average energy consumption (E). These files are output as:

```
$ES_PATH/container_results/stats_for_casc_agent_1_new.csv
$ES_PATH/container_results/stats_for_casc_agent_2_new.csv
$ES_PATH/container_results/stats_for_casc_agent_3_new.csv
$ES_PATH/container_results/stats_for_casc_agent_4_new.csv
```

The number in each file name matches the data to the corresponding Controller 1 through 4 as indicated in Table 1. Since Controller 1 was used in Experiment 1, the summary data in `stats_for_casc_agent_1_new.csv` is derived from the output of `run_exp1.sh`.

The raw data for each additional agent is likewise output to `$ES_PATH/container_results/raw_data`, with one folder for each of these additional agents. See the Appendix for a description of the structure and format of this data.

6. Appendix

Raw Data

Under `$ES_PATH/container_results/raw_data/`, every directory describes the evaluation results for a set of episodes. The name suffix of each directory describes the experimental configuration setting these evaluations belong to. For instance:

- Directory `Town04_OPT_ResNet152_Shield2_early`: experiment1 evaluations for EnergyShield early mode at default wireless settings
- Directory `Town04_OPT_ResNet152_Shield2_belay_10Mbps`: experiment2 evaluations for EnergyShield uniform mode when varying wireless channel throughput to 10Mbps

Within each of these directories, four subdirectories exist to describe whether the experiments were conducted with/without gaussian noise and with/without safety filter activated. In each one of these subdirectories, we can find the experimental data in `*.csv` format as follows:

- `plots/*.csv` files containing the individual episode's data incurred by the carla simulator when running each episode for the corresponding experimental setting.
- `valid_data.csv` is the file describing the final statistics for every episode within the corresponding experimental configuration

Information in the former include simulation time, position, steering angle, relative obstacle position, wireless conditions, safety time window, offloading actions, and performance evaluations in terms of latency and energy - all calculated instantaneously for every simulation tick.

Information in the latter include statistics for each episode about total number of ticks, reward, distance traveled, speed, average latency, average energy per inference, average center deviance, and whether the agent has hit an obstacle/curb.

Distance data

For figure 7 (normalized energy variation vs distance (meters)), raw data is used to construct the tables in `$ES_PATH/container_results/distance/*.csv` which aggregate average normalized energy consumption and #occurrences across 1 m increments of 'distance from obstacle' parameter from all episodes within the subdirectory. Each of these tables is then used to construct a plot from the figure describing how this relation varies under different experimental settings.

EnergyShield: Provably-Safe Offloading of Neural Network Controllers for Energy Efficiency

Mohanad Odema
University of California, Irvine
Dept. of Electrical Engineering and Computer Science
Irvine, CA, USA
modema@uci.edu

James Ferlez
University of California, Irvine
Dept. of Electrical Engineering and Computer Science
Irvine, CA, USA
jferlez@uci.edu

Goli Vaisi
University of California, Irvine
Dept. of Electrical Engineering and Computer Science
Irvine, CA, USA
gvaisi@uci.edu

Yasser Shoukry
University of California, Irvine
Dept. of Electrical Engineering and Computer Science
Irvine, CA, USA
yshoukry@uci.edu

Mohammad Abdullah Al Faruque
University of California, Irvine
Dept. of Electrical Engineering and Computer Science
Irvine, CA, USA
alfaruqu@uci.edu

ABSTRACT

To mitigate the high energy demand of Neural Network (NN) based Autonomous Driving Systems (ADSs), we consider the problem of offloading NN controllers from the ADS to nearby edge-computing infrastructure, but in such a way that formal *vehicle* safety properties are guaranteed. In particular, we propose the EnergyShield framework, which repurposes a controller "shield" as a low-power runtime safety monitor for the ADS vehicle. Specifically, the shield in EnergyShield provides not only safety interventions but also a formal, state-based quantification of the tolerable edge response time before vehicle safety is compromised. Using EnergyShield, an ADS can then save energy by wirelessly offloading NN computations to edge computers, while still maintaining a formal guarantee of safety until it receives a response (on-vehicle hardware provides a just-in-time fail safe). To validate the benefits of EnergyShield, we implemented and tested it in the Carla simulation environment. Our results show that EnergyShield maintains safe vehicle operation while providing significant energy savings compared to on-vehicle NN evaluation: from 24% to 54% less energy across a range of wireless conditions and edge delays.

KEYWORDS

Formal Verification, Vehicular, Edge Computing, Autonomous Vehicles, Provable safety, Offloading, Autonomous Driving Systems

1 INTRODUCTION

Advances in the theory and application of Neural Networks (NNs), particularly Deep NNs (DNNs), have spurred revolutionary progress on a number of AI tasks, including perception, motion planning and control. As as result, DNNs have provided a feasible engineering solution to supplant formerly human-only tasks, most ambitiously in Autonomous Driving Systems (ADSs). However, state-of-the-art ADSs require the use of very large DNN architectures to solve essential perception and control tasks, which generally involve processing the output of tens of cameras, LiDARs and other sensors. As a result, contemporary ADSs are only possible with significant computational resources deployed on the vehicle itself, since their DNNs must process such high-bandwidth sensors in closed-loop, in real time. The practical energy impact of high-capacity on-vehicle compute is understudied, but current research suggests that it is profound: e.g., up to a 15% reduction in a vehicle's range [26**?**].

At the same time, advances in semiconductor design and packaging have made possible cheap, low-power silicon; and advances in wireless networking have made high-bandwidth, low-latency radio links possible even in challenging multi-user environments. Together, these advances have led to increasingly ubiquitous, cheap, wirelessly accessible computational resources near the *edge* of conventional hard-wired infrastructure. In particular, it is now possible to achieve reliable, millisecond-latency wireless connections between connected ADSs and nearby edge computing [3, 23, 25].

The ubiquity of edge compute thus suggests a natural way to reduce the *local* energy consumption on ADS vehicles: viz., by wirelessly *offloading* onerous perception and control DNN computations to abundant nearby edge compute infrastructure. However, even modern wireless networks and offloading-friendly DNN architectures (e.g. encoder/decoders) cannot provide *formal guarantees* that bringing edge computing "into the loop" will have equivalent (or even acceptable) performance compared to on-vehicle hardware. This is an unacceptable situation when human lives are at stake: even relatively rare and short delays in obtaining a control action or perception classification can have fatal consequences.

In this paper, we propose the EnergyShield framework as a mechanism to perform DNN-to-edge offloading of ADS controllers but *in a formal, provably safe way*. Thus, EnergyShield is, to the best of our

ICCPS '23, May 09–12, 2023, San Antonio, TX

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

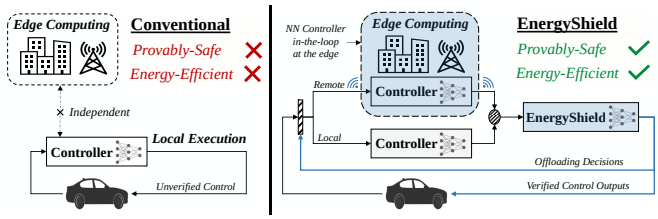

Figure 1: Illustration of Provably-Safe Offloading of Neural Network Controllers for Energy Efficiency.

knowledge, the first framework that enables significantly lower on-vehicle energy usage when evaluating large DNNs by intelligently offloading those calculations to edge compute in a provably safe way; see Figure 1. The primary idea of EnergyShield is to perform safety-aware (state-)contextual offloading of DNN calculations to the edge, under the assumption that adequate on-vehicle computation is always available as a safety fallback. This is accomplished using a controller "shield" as both a mechanism to enforce safety *and* as a novel, low-power runtime safety monitor. In particular, this shield-based safety monitor provides provably safe edge-compute response times: i.e., at each instance, EnergyShield provides a time interval within which the edge-compute must respond to an offloading request in order to *guarantee safety of the vehicle* in the interim. In the event that the edge resources don't provide a response within this time, on-board local compute proceeds to evaluate the relevant DNNs before vehicle safety is no longer assured. Further energy savings are obtained by incorporating an estimator to anticipate edge-compute load and wireless network throughput; a more intelligent offloading decision is made by comparing this estimate against the tolerable edge-compute delay provided by the runtime safety monitor – thus preventing offloads that are likely to fail.

The main technical novelty of EnergyShield is its shield-based runtime safety monitor mentioned above. Although controller "shielding" is a well-known methodology to render generic controllers safe, the shielding aspect of EnergyShield contains two important novel contributions of its own: first, in the use of a shield not only to enforce safety but also as a runtime safety monitor to quantify the *time until the system is unsafe*; and second, in the specific design of that runtime monitor with regard to implementation complexity and energy considerations. In the first case, EnergyShield extends existing notions wherein the *current value* of a (Zeroing-)Barrier Function (ZBF) is used as a runtime monitor to quantify the safety of an agent: in particular, it is novel in EnergyShield to instead use the current value of the ZBF to derive a *sound* quantification of the *time* until the agent becomes unsafe. Moreover, EnergyShield implements this quantification in an extremely energy efficient way: i.e., via a small lookup table that requires only a small number of FLOPS to obtain a guaranteed time-until-unsafe. This particular aspect of the runtime safety monitor is also facilitated by a using a particular, but known, ZBF and shield [14] in EnergyShield: these components are both extremely simple, and so implementable using small, energy efficient NNs [14]. Together, these design choices ensures that any energy saved by offloading is not subsequently expended in the implementation of EnergyShield itself.

We conclude the paper with a significant sequence of experiments to validate both the safety and energy savings provided by the EnergyShield framework. In particular, we tested EnergyShield in the Carla simulation environment [10] with several Reinforcement Learning (RL)-trained agents. Our experiments showed that EnergyShield entirely eliminated obstacle collisions for the RL agents we considered – i.e. made them safe – while simultaneously reducing NN energy consumption by as much as 54%. Additionally, we showed that EnergyShield has intuitive, safety-conscious offloading behavior: when the ADS is near an obstacle – and hence less safe – EnergyShield's runtime safety monitor effectively forced exclusively on-vehicle NN evaluation; when the ADS was further from an obstacle – and hence more safe – EnergyShield's runtime safety monitor allowed more offloading, and hence more energy savings.

Related Work: *Formal Methods for Data-Trained Controllers.* A number of approaches exist to assert the safety of data-trained controllers with formal guarantees; in most, ideas from control theory are used in some way to augment the trained controllers to this end. A good survey of these methods is [9]. Examples of this approach include the use of Lyapunov methods [4, 6, 7], safe model predictive control [19], reachability analysis [1, 15, 16], barrier certificates [24, 28, 31, 34, 36, 38], and online learning of uncertainties [30]. Controller "shielding" [2] is another technique that often falls in the barrier function category [5]. Another approach tries to verify the formal safety properties of learned controllers using formal verification techniques (e.g., model checking): e.g., through the use of SMT-like solvers [11, 22, 32] or hybrid-system verification [12, 18, 37]. However, these techniques only assess the safety of a given controller rather than design or train a safe agent.

Edge Computing for Autonomous Systems. A number of different edge/cloud offloading schemes have been proposed for ADSs, however none to date has provided formal guarantees. Some have focused on scheduling techniques and network topology to achieve effective offloading [8, 13, 29, 33, 35, 40]. Others focused on split and other NN architectures to make offloading more efficient [25, 27].

2 PRELIMINARIES

2.1 Notation

Let $\mathbb{R}$ denote the real numbers; $\mathbb{R}^+$ the non-negative real numbers; $\mathbb{N}$ the natural numbers; and $\mathbb{Z}$ the integers. For a continuous-time signal, $x(t), t \geq 0$, denote its discrete-time sampled version as $x[n]$ for some fixed sample period T (in seconds); i.e. let $x[n] \triangleq x(n \cdot T)$ for $n \in \mathbb{Z}$. Let $\mathbf{1}_a : \mathbb{R} \to \{a\}$ be the constant function with value a; i.e., $\mathbf{1}_a(x) = a$ for all $x \in \mathbb{R}$ (interpreted as a sequence where necessary). Finally, let $\dot{x} = f(x, u)$ be a generic control system with $x \in \mathbb{R}^n$ and $u \in \mathbb{R}^m$, and let $\pi : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^m$ be a (possibly) time-varying controller. For this system and controller, consider a time $\tau \geq 0$ and state x_0, and denote by $\zeta_\pi^{t_0,x_0} : \mathbb{R}^+ \to \mathbb{R}^n$ the t_0-shifted state evolution of this system under controller π assuming $x(t_0) = x_0$. Let $\zeta_\pi^{n_0,x_0}[n]$ indicate the same, except in discrete-time and with zero-order hold application of π. $\|\cdot\|$ and $\|\cdot\|_2$ will denote the max and two-norms on $\mathbb{R}^n$, respectively.

2.2 Kinematic Bicycle Model

Throughout this paper, we will use the kinematic bicycle model (KBM) as the formal dynamical model for our autonomous vehicle [20]. However, we consider the KBM model in terms of state variables relative to a fixed point in the plane, rather than absolute

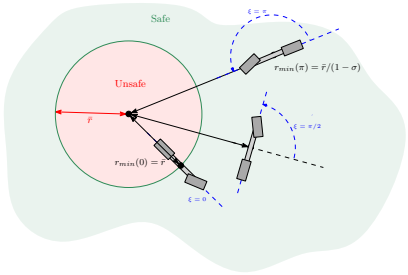

Figure 2: Obstacle specification and minimum barrier distance as a function of relative vehicle orientation, ξ.

Cartesian coordinates. That is, the states are the distance to a fixed point, $\|\vec{r}\|$, and orientation angle, ξ, of the vehicle with respect to the same. These quantities describe a state-space model:

$$\begin{pmatrix} \dot{r} \\ \dot{\xi} \\ \dot{v} \end{pmatrix} = \begin{pmatrix} v\cos(\xi-\beta) \\ -\frac{1}{r}v\sin(\xi-\beta) - \frac{v}{\ell_r}\sin(\beta) \\ a \end{pmatrix}; \quad \beta \triangleq \tan^{-1}\left(\frac{\ell_r}{\ell_f+\ell_r}\tan(\delta_f)\right) \quad (1)$$

where $r(t)$ is distance to the origin; a is the linear acceleration input; δ_f is the front-wheel steering angle input[1]; and $\psi + \xi = \tan^{-1}(y/x)$. We note a few special states for convenience: at $\xi = \pm\pi/2$, the vehicle is oriented tangentially to the obstacle, and at $\xi = \pi$ or 0, the vehicle is pointing directly at or away from the origin, respectively (Figure 2). β is an *invertible function* of the steering control, δ_f.

We also assume that the KBM has a steering constraint so that $\delta_f \in [-\delta_{f_{\max}}, \delta_{f_{\max}}]$. For simplicity, we will consider β directly as a control variable; this is without loss of generality, for the reason noted above. Thus, β is also constrained: $\beta \in [-\beta_{\max}, \beta_{\max}]$. Finally, we define the state and control vectors for the KBM as: $\chi \triangleq (\xi, r, v)$ and $\omega \triangleq (a, \beta)$, where $\omega \in \Omega_{\text{admis.}} \triangleq \mathbb{R} \times [-\beta_{\max}, \beta_{\max}]$, the set of admissible controls. Thus, the dynamics of the KBM are given by $\dot{\chi} = f_{\text{KBM}}(\chi, \omega)$ with f_{KBM} defined according to (1).

2.3 Barrier Functions and Shielding

In the sequel, we will use a controller "shield", which is a methodology for instantaneously correcting the outputs produced by a controller in closed loop; the objective is to make corrections such that the original controller, however it was designed or implemented, becomes safe – hence the "shield" moniker. Specifically, a controller shield is designed around a real-valued function over the state space of interest, called a (Zeroing-) Barrier Function (ZBF). The ZBF directly specifies a set safe states by its sign: states for which the ZBF is nonnegative are taken to be safe. The ZBF in turn indirectly specifies safe controls (as a function of state) in such a way that the sign of the ZBF is invariant along trajectories of the dynamics.

Formally, consider a control system $\dot{x} = f(x, u)$ in closed loop with a state-feedback controller $\pi : x \mapsto u$. In this scenario, a feedback controller in closed loop converts the control system into an autonomous one – the autonomous vector field $f(\cdot, \pi(\cdot))$. In the context of an autonomous system, then, we recall the following definition for a Zeroing-Barrier Function (ZBF):

Definition 1 (Zeroing Barrier Function (ZBF) [39, Definition 2]). *Let $\dot{x} = f(x, \pi(x))$ be the aforementioned closed-loop, autonomous system with $x(t) \in \mathbb{R}^n$. Also, let $h : \mathbb{R}^n \to \mathbb{R}$, and define $C \triangleq \{x \in \mathbb{R}^n : h(x) \geq 0\}$. If there exists a locally Lipschitz, extended-class-K function, α such that:*

$$\nabla_x h(x) \cdot f(x, \pi(x)) \geq -\alpha(h(x)) \text{ for all } x \in C \quad (2)$$

*then h is said to be a **zeroing barrier function (ZBF)**.*

Moreover, the conditions for a barrier function above can be translated into a set membership problem for the outputs of such a feedback controller. This is explained in the following corollary.

Proposition 1. *Let $\dot{x} = f(x, u)$ be a control system that is Lipschitz continuous in both of its arguments on a set $\mathcal{D} \times \Omega_{admis.}$; furthermore, let $h : \mathbb{R}^n \to \mathbb{R}$ with $C_h \triangleq \{x \in \mathbb{R}^n | h(x) \geq 0\} \subseteq \mathcal{D}$, and let α be a class $\mathcal{K}$ function. If the set*

$$R_{h,\alpha}(x) \triangleq \{u \in \Omega_{admis.} | \nabla_x^T h(x) \cdot f(x, u) + \alpha(h(x)) \geq 0\} \quad (3)$$

is non-empty for each $x \in \mathcal{D}$, and a Lipschitz continuous feedback controller $\pi : x \mapsto u$ satisfies

$$\pi(x) \in R_{h,\alpha}(x) \quad \forall x \in \mathcal{D} \quad (4)$$

then C_h is forward invariant for the closed-loop dynamics $f(\cdot, \pi(\cdot))$.

In particular, if π satisfies (4) and $x(t)$ is a trajectory of $\dot{x} = f(x, \pi(x))$ with $h(x(0)) \geq 0$, then $h(x(t)) \geq 0$ for all $t \geq 0$.

Proof. This follows directly from an application of zeroing barrier functions [39, Theorem 1]. □

Proposition 1 is the foundation for controller shielding: (3) and (4) establish that h (and associated α) forms a ZBF for the closed-loop, autonomous dynamics $f(\cdot, \pi(\cdot))$. Note also that there is no need to distinguish between a closed-loop feedback controller π, and a composite of π with a function that *shields* (or filters) its output based on the current state. Hence, the following definition:

Definition 2 (Controller Shield). *Let $\dot{x} = f(x, u)$, h, $\mathfrak{C}_h$, α and $\mathcal{D} \times \Omega_{admis.}$ be as in Proposition 1. Then a **controller shield** is a Lipschitz continuous function $\mathfrak{S} : \mathcal{D} \times \Omega_{admis.} \to \Omega_{admis.}$ such that*

$$\forall (x, u) \in \mathcal{D} \times \Omega_{admis.} . \mathfrak{S}(x, u) \in R_{h,\alpha}(x). \quad (5)$$

2.4 A Controller Shield for the KBM

In this paper, we will make use of an existing ZBF and controller shield designed for the KBM [14]. These function in concert to provide controller shielding for the safety property also illustrated in Figure 2: i.e., to prevent the KBM from entering a disk of radius $\bar{r}$ centered at the origin.

In particular, [14] proposes the following class of *candidate* barrier functions for the KBM:

$$h_{\bar{r},\sigma}(\chi) = h_{\bar{r},\sigma}(\xi, r, v) = \frac{\sigma\cos(\xi/2)+1-\sigma}{\bar{r}} - \frac{1}{r} \quad (6)$$

where $\sigma \in (0, 1)$ is the parameter that characterizes the class; this choice is combined with a class $\mathcal{K}$ function:

$$\alpha_{v_{\max}}(x) = K \cdot v_{\max} \cdot x \quad (7)$$

to form a parametric class of ZBFs. Note also that this class of ZBFs ignores the state variable, v; it is a result in [14] that this class is useful as a barrier function provided the vehicle velocity remains (is controlled) within the range $(0, v_{\max}]$. Note also that the equation

[1] That is the steering angle can be set instantaneously, and the dynamics of the steering rack can be ignored.

has $h_{\bar{r},\sigma}(\chi) = 0$ has a convenient solution, which we denote by $r_{\min}$ for future reference:

$$r_{\min}(\xi) = \bar{r}/(\sigma\cos(\xi/2) + 1 - \sigma). \tag{8}$$

One main result in [14] is a mechanism for choosing the parameter σ as a function of KBM parameters (e.g. ℓ_r) and safety parameter, $\bar{r}$ so that the resulting specific function is indeed a ZBF as required.

Finally, we note that [14] also suggests an extremely lightweight implementation of barrier based on (6). That is, it contains a "Shield Synthesizer" that implements a controller shield by approximating a simple single-input/single-output concave function with a ReLU NN [14, pp 6]. This construction will also prove advantageous later. We denote by $\mathfrak{S}_{\text{KBM}}$ the resulting controller shield, with associated barrier, KBM and safety parameters inferred from the context.

3 FRAMEWORK

NOTE: For the duration of this section, we will denote by x, y and u the state, sensor and control variables of an ADS, respectively; this abstract notation will illustrate the EnergyShield framework free from specific modelling details. A detailed consideration of EnergyShield, with formal proofs and assumptions, appears in the Section 4.

3.1 EnergyShield Motivation and Context

The basic motivation for the EnergyShield framework is the following. Suppose that an ADS contains a large NN, $\mathcal{NN}_c$, that is responsible for producing a control action, u, in response to a sensor signal, y. Further assume that, by virtue of its size, computing an output of $\mathcal{NN}_c$ with on-vehicle hardware consumes significant energy. Thus, it would be advantageous, energy-wise, to *offload* evaluations of $\mathcal{NN}_c$ to edge computing infrastructure: in other words, wirelessly transmit a particular sensor measurement, y, to off-vehicle edge computers, where the output $u = \mathcal{NN}_c(y)$ is computed and returned to the requesting ADS.

The problem with this approach is largely from a safety point of view. In particular, the controller $\mathcal{NN}_c$ was designed to operate *in real time and in closed-loop*: i.e. the control action at discrete-time sample n is intended to be computed from the sensor measurement at the *same* time sample[2]. In the notation of discrete-time signals (see Section 2.1), this means: $u[n] = \mathcal{NN}_c(y[n])$. However, offloading a sensor measurement, $y[n]$, to the edge entails that the correct output of $\mathcal{NN}_c(y[n])$ will not be back on-vehicle before some non-zero number of samples, say Δ. Thus, $\mathcal{NN}_c(y[n])$ will not be available at time n to set $u[n] = \mathcal{NN}_c(y[n])$ as intended; rather, the soonest possible use of the output $\mathcal{NN}_c(y[n])$ will be at time $n + \Delta$, or $u[n + \Delta] = \mathcal{NN}_c(y[n])$. This delay creates obvious safety issues, since the state of the vehicle – and hence the correct control to apply – will have changed in the intervening Δ time samples. More importantly, even the "correct" control action applied at $n + \Delta$ may be insufficient to ensure safety: e.g., after Δ samples have elapsed, it may be too late to apply adequate evasive steering.

3.2 EnergyShield Structure

If we assume that the ADS itself has enough on-vehicle compute to obtain an output $\mathcal{NN}_c(y[n])$ *in real time*, then this safety problem above becomes one of making an *offloading decision*: ideally one that saves energy without compromising safety. That is, should a particular computation of $\mathcal{NN}_c$ be offloaded to the edge? And how long should the ADS wait for a response so as to ensure the situation is correctable?

The nature of the offloading decision means that EnergyShield must address two essential and intertwined issues in order to ensure safety of the ADS vehicle during offloading. On the one hand, EnergyShield must be able to *correct* the control actions provided by $\mathcal{NN}_c$ after an offload decision (see explanation above). On the other hand, EnergyShield must limit the duration it waits for each offloading request, so that actions provided by $\mathcal{NN}_c$ *can* be corrected in the first place; i.e., among all possible offloading delays, Δ, it is not immediate which may be corrected and which may not (loosely speaking, $\Delta = \infty$ is one delay that likely cannot be corrected). In this sense, knowing that a particular response-delay, Δ' is correctable essentially characterizes how to take an offloading decision, since it provides an ***expiration on safety***: i.e., proceed to offload, and wait for a response until Δ' samples have elapsed – at which point resume local evaluation of $\mathcal{NN}_c$.

In particular, EnergyShield has two central components to achieve these ends:

- **C1: Controller Shield.** EnergyShield contains a controller shield (see Section 2.3), which ensures that safety is maintained irrespective of offloading-delayed controller outputs; in other words, it corrects unsafe behavior of $\mathcal{NN}_c$ that results from unanticipated (to $\mathcal{NN}_c$) changes in vehicle state during offloading delays.
- **C2: Runtime Safety Monitor.** EnergyShield contains a runtime safety monitor that provides the ADS an upper bound, $\Delta_{\max}$ (in samples), on how long it should wait for a response to one of its offloading requests to maintain safety, *assuming no updates to the control action in the meantime*; i.e., provided the offload delay is $\Delta \leq \Delta_{\max}$, then **C1**, the controller shield, can guarantee safe recovery after holding the last control signal update through offload delay period (**C1** may use on-vehicle computation if necessary). ***In other words,*** $\Delta_{\max}$ ***specifies and expiration for the safety guarantee provided by* C1 *using on-vehicle computation.***

Naturally, **C1** and **C2** need to be designed together, since their objectives are mutually informed. Indeed, in the specific design of EnergyShield, these components are designed from the same ZBF (defined in Section 2.3): see Section 4 for formal details.

Unfortunately, neither of the components **C1** nor **C2** can operate effectively on the same raw sensor measurements, $y[n]$, used by the controller; this is especially true given our intention to implement them via ZBFs and controller shields. In particular, both require some (limited) state information about the ADS in order to perform their tasks. Thus, EnergyShield requires a perception/estimator component to provide state information to **C1** and **C2**. Note that we deliberately exclude the design of such an estimator from the EnergyShield framework in order to provide additional flexibility: in particular, since the controller $\mathcal{NN}_c$ may effectively contain an estimator, we wish to allow for estimation to be offloaded, too – provided it is executed locally just-in-time before informing **C1** and **C2** (see Section 3.3). Nevertheless such an estimator is necessary for EnergyShield, so we include it as component:

[2] In our formal consideration, we will model a one-sample computation delay.

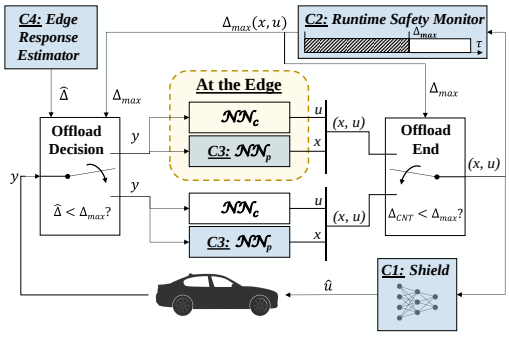

Figure 3: EnergyShield Framework

C3: State Estimator. EnergyShield requires (minimal) state estimates be provided as input to **C1** and **C2**. By convention, this estimator will be a NN denoted by $\mathcal{NN}_p : y \mapsto x$. *We assume that $\mathcal{NN}_p$ can be computed by on-vehicle hardware in one sample period.*

The interface of **C3** with both **C1** and **C2** makes the latter two components (state-)context aware. *That is, EnergyShield makes **context-aware offloading decisions** based on the current vehicle state.* Moreover, it is important to note that since the prior control action will be held during offload $\Delta_{\max}$, the output of **C2**, control in addition to state dependent: that is, **C2** actually produces an output $\Delta_{\max}(\hat{x}, u)$ for (arbitrary) state $\hat{x}$ and the control u applied just before offload.

EnergyShield has one further important component, but one that is motivated purely by energy savings with no effect on safety. Crucially the known expiration of safety provided by **C2**, i.e. $\Delta_{\max}(x)$, affords the opportunity to use additional information in making an offload decision. In particular, an estimate of the anticipated edge response time, $\hat{\Delta}$, can be used to *forego* offloads that unlikely to complete before the expiration of the safety deadline, $\Delta_{\max}(x)$. For this reason, EnergyShield contains an estimator of edge response time to preemptively skip offloads that are likely to fail:

C4: Edge-Response Estimator. EnergyShield specifies that an estimate of the current edge response time, $\hat{\Delta}$, is provided to inform offloading decisions.

We note, however, that *EnergyShield doesn't specify a particular estimator to be used in this component*: any number of different estimators may be appropriate, and each estimator may lead to different energy profiles depending on its efficacy. Moreover, since $\hat{\Delta}$ is never used to override $\Delta_{\max}(x)$, safety is preserved irrespective of the specific estimator used.

The interconnection of the components **C1** through **C4** in EnergyShield is illustrated in Figure 3. Note that component **C3**, the state estimator, is connected to components **C1** and **C2**, the controller shield and safety runtime monitor, respectively. Also note that the output of **C2** provides a signal $\Delta_{\max}(x)$ to the offloading decision switch; also informing that decision is the estimate of immediate edge-response times provided by component **C4**.

3.3 Semantics of an EnergyShield Offloading Decision

In this subsection, we consider the timeline of a single, hypothetical EnergyShield offloading decision to illustrate the details of the interplay between the components described in Section 3.2. In particular, suppose that an offloading interval has just been completed, and at time index n_0 a new offloading decision is to be taken.

We call the time between the initialization of an offloading decision and the time that offloading decision has been resolved an ***offloading period*** (the resolution is either by a response from the edge or a fail-over to on-vehicle compute).

Timeline:

[$n_0 - 1$] *Last time index of previous offload period.*

- Assumption: $\hat{x}[n_0] \triangleq x[n_0 - 1] = \mathcal{NN}_p(y[n_0 - 1])$ is always computed locally in the last sample of the previous offloading period.

[n_0] *Initial time index of new offload period.*

- The first sample of the new offload period inherits a locally computed $\hat{x}[n_0]$ from the previous offloading period.
- $\hat{x}[n_0]$ is provided to **C1** to correct $u[n_0]$ as calculated by the previous offloading. Let this correct control action be $\hat{u}[n_0]$
- Fix control action applied at n_0: i.e. $u_{0\text{-h}} = \hat{u}[n_0]$, with $\hat{u}[n_0]$ as calculated above.
- $\hat{x}[n_0]$ and $u_{0\text{-h}}$ are provided to **C2**, the runtime safety monitor, to generate $\Delta_{\max}(\hat{x}[n_0], u_{0\text{-h}})$
- **C4** generates an estimate for the edge response time, $\hat{\Delta}$ based on all packets exchanged so far.
- ***Offloading Decision:***
  - *Proceed with offload, if:* $\hat{\Delta} \leq \Delta_{\max}(\hat{x}[n_0], u_{0\text{-h}})$ **AND** $\Delta_{\max}(\hat{x}[n_0], u_{0\text{-h}}) \geq 1$; i.e., proceed to transmit $y[n_0]$ to the edge. Initialize offload duration counter: $\Delta_{\text{cnt}} = 1$
  - *Otherwise, terminate offload period, and use local fail-safe.* Skip to ***Unsuccessful Offload*** with $\Delta_{\max}(\hat{x}[n_0], u_{0\text{-h}}) = 1$.

[⋮]

[$n_0 + \Delta_{\text{cnt}}$] *Offload in progress; no edge response and* $\Delta_{cnt} < \Delta_{max}(\hat{x}[n_0])$

- Maintain zero-order hold of $u[n_0 + \Delta_{\text{cnt}}] = u_{0\text{-h}}$.
- Increment Δ_{cnt}: $\Delta_{\text{cnt}} \leftarrow \Delta_{\text{cnt}} + 1$.

[⋮]

Now, the current offload period ends in one of two ways:
I) Successful Offload: *(resume timeline from $n_0 + \Delta_{cnt}$)*

[⋮]

[$n_0 + \Delta$] *Edge response received;* $\Delta = \Delta_{cnt} \leq \Delta_{max}(\hat{x}[n_0], u_{0\text{-}h})$

- Apply received control action $u[n_0 + \Delta] = \mathcal{NN}_c(y[n_0])$.
- Initiate local evaluation of $\mathcal{NN}_c$ for next time interval.
- Initiate local evaluation of $\mathcal{NN}_p$ for next time interval.

- $n_0 - \Delta$ becomes time $n_1 - 1$ for the starting index of the next offload period. (See $n_0 - 1$ time index above.)

II) Unsuccessful Offload: *(resume timeline from $n_0 + \Delta_{cnt}$)*

$[\quad \vdots \quad]$

$[n_0 + \Delta_{\max}]$ *No edge response received, and safety expired;* $\Delta_{cnt} = \Delta_{max}(\hat{x}[n_0], u_{0\text{-}h})$

- Maintain original zero-order hold control $u[n_0 + \Delta_{\max}(\hat{x}[n_0], u_{0\text{-h}})] = u_{0\text{-h}}$ (there have been no updates.)
- Initiate local evaluation of $\mathcal{NN}_c$ for next time interval.
- Initiate local evaluation of $\mathcal{NN}_p$ for next time interval.
- $n_0 - \Delta_{\max}(\hat{x}[n_0])$ becomes time $n_1 - 1$ for the starting index of the next offload period. (See $n_0 - 1$ time index above.)

In particular, note two crucial facts. First, if **C2** returns $\Delta_{\max}(\hat{x}, u) = 0$, then it effectively forces pure on-vehicle evaluation of $\mathcal{NN}_c$ and $\mathcal{NN}_p$. Second, we ensured that an up-to-date estimate of the state is always available for both **C1** and **C2** before they have to act.

4 ENERGYSHIELD: PROVABLY SAFE OFFLOADING

4.1 Main Formal Result

4.1.1 Formal Assumptions. We begin this section with a list of formal assumptions about the ADS. These are largely based around the structure of EnergyShield, as described in Section 3.

Assumption 1 (ADS Safety). *Consider the a fixed point in the plane as a stationary obstacle to be avoided by the ADS, and a disk or radius $\bar{r}$ around the origin to be a set of unsafe states; see Figure 2.*

Assumption 2 (ADS Model). *Let Assumption 1 hold. Thus, suppose that the ADS vehicle is modeled by the KBM dynamics in* (1)*. Suppose further that interactions with this model happen in discrete time with zero-order hold. Let T be the sampling period.*

Assumption 3 (ADS Sensors). *Let Assumptions 1 and 2 hold. Suppose the KBM-modeled ADS has access to samples of a sensor signal, $s[n] \in \mathbb{R}^N$, at each discrete time step, and there is a perception NN, $\mathcal{NN}_p : s[n] \mapsto \chi[n]$, which maps the sensor signal at each discrete time to the (exact) KBM state at the same time instant, $\chi[n]$.*

Assumption 4 (ADS Control). *Let Assumptions 1 - 3 hold. Suppose this KBM-modeled ADS vehicle has a NN controller, $\mathcal{NN}_c : s \mapsto \omega$, which at each sample has access to the sensor measurement s.*

Assumption 5 (ADS Local Computation). *Let Assumptions 1 - 4 hold. Suppose that the output of $\mathcal{NN}_p$ and $\mathcal{NN}_c$ can be computed by ADS on-vehicle hardware in less than T seconds – i.e., not instantaneously. Thus, suppose that the control action is obtained with a one-sample computation delay when using on-vehicle hardware: i.e., the control action applied at sample $n+1$ is $\omega[n+1] = \mathcal{NN}_c(s[n])$.*

4.1.2 Component Design Problems. There are two central problems that need to be solved: i.e., corresponding to the design of the two main components of EnergyShield, **C1** and **C2** (see Section 3).

The solutions to these problems are deferred to Sections 4.2 and Section 4.3, respectively. We state them here in order to facilitate the statement of our main result in the next subsection.

Problem 1 (Controller Shield Design **(C1)**). *Let Assumptions 1 - 5 hold. Then the problem is to design: first, design functions h and α such that they constitute a ZBF for the KBM (see); and then using this ZBF, design a controller shield, $\mathfrak{S}$ for the KBM model. The resulting controller shield must have the following additional property for* ***a discrete-time version of the KBM*** *with zero-order-hold inputs:*

- *Let $\chi[n_0 - 1]$ and $\chi[n_0]$ be KBM states such that $h(\chi[n_0 - 1]), h(\chi[n_0]) > 0$, and let $\chi[n_0]$ result from a feasible input $\hat{\omega}[n_0 - 1]$ applied in state $\chi[n_0 - 1]$. Then the control action*

$$\hat{\omega}[n_0] = \mathfrak{S}(\chi[n_0 - 1], \omega[n_0]) \tag{9}$$

must yield a state $\chi[n_0 + 1]$ such that $h(\chi[n_0 + 1]) > 0$; i.e., the controller shield preserves safety under discretization of the KBM and one-step estimation delay (associated with $\mathcal{NN}_p$), as in the case of no computations being offloaded.

Problem 2 (Runtime Safety Monitor Design **(C2)**). *Let Assumptions 1 - 5 hold, and assume that h, α and $\mathfrak{S}$ solve Problem 1. Then the problem is to design a runtime safety monitor:*

$$\Delta_{max} : \mathbb{R}^3 \times \Omega_{admis.} \to \mathbb{N} \tag{10}$$

with the following property:

- *Let $\chi[n_0 - 1]$ be such that $h(\chi[n_0 - 1]) > 0$. Then for constant control, $\omega = \omega[n_0]$, applied to the discretized KBM starting from $\chi[n_0 - 1]$ the following is true:*

$$\forall n = 0, \ldots, \Delta_{max}(\chi[n_0 - 1], \omega[n_0]) \,.\, h(\chi[n_0 - 1 + n]) > 0 \tag{11}$$

i.e. the constant control $\omega = \omega[n_0]$ preserves safety for at least $\Delta_{max}(\chi[n_0 - 1], \omega[n_0])$ samples from state $\chi[n_0 - 1]$.

(The delay in $\chi[n_0 - 1]$ accounts for the computation time of $\mathcal{NN}_p$.)

4.1.3 Main Result. Given solutions to these problems, it is straightforward to conclude our main formal result.

Theorem 1. *Let Assumptions 1 - 5 hold, and assume a ZBF for the KBM dynamics, using which Problem 1 and Problem 2 can be solved. Then the offloading policy described in Section 3.3 preserves safety of the KBM-modeled ADS (Assumptions 1 and 2).*

Proof. The proof follows largely by construction. Each offload period is limited in duration by the runtime safety monitor; thus, a safety monitor that solves Problem 2 will ensure safety under the specified constant control action during the offload period. Then by the additional property of the controller shield in Problem 1, safety can be maintained after the offloading period ends: i.e., either by a deciding to perform a new offload if there remains significant safety margin, or by executing the shield in closed loop because there is no offload safety margin. □

Corollary 1. *Let Assumptions 1 - 5 hold, and consider the ZBF for the KBM dynamics specified in Section 2.4. Then the controller shield in Section 4.2 uses this ZBF and solves Problem 1; likewise, the runtime monitor in Section 4.3 uses this ZBF and solves Problem 2. Hence, our implementation of EnergyShield is safe.*

4.2 KBM Controller Shield

Fortunately, we have access to a preexisting ZBF and controller shield designed for the KBM: see Section 2.4 [14]. That is, the ZBF is available after using the design methodology in [14] to choose the parameter σ (see Section 2.4); for simplicity, we will omit further discussion this design process. Thus for this section, we refer to a fully implemented controller shield as $\mathfrak{S}_{\text{KBM}}$, with the understanding that it has been designed for the relevant KBM model and safety parameter $\bar{r}$ (see Figure 2); viz. Assumptions 1 and 2.

Thus, the main requirement of this section is to alter $\mathfrak{S}_{\text{KBM}}$ as necessary so that it satisfies the additional property required in Problem 1. This is the subject of the following Lemma, which also defines the alterations in question.

Lemma 1. *Let Assumptions 1 - 5 hold as usual, and let $\mathfrak{S}_{KBM}$ be a controller shield designed under these assumptions as per Section 2.4. Then there exists a $\rho > 0$ such that the following controller shield:*

$$\mathfrak{S}^{\rho}_{KBM} : ((r,\xi,v),\omega) \mapsto \begin{cases} \mathfrak{S}_{KBM}((r-\rho,\xi,v),\omega) & r-\rho \geq r_{min}(\xi) \\ \beta_{max} & r-\rho \geq r_{min}(\xi) \wedge \xi \geq 0 \\ -\beta_{max} & r-\rho \geq r_{min}(\xi) \wedge \xi < 0 \end{cases} \tag{12}$$

solves Problem 1.

The proof of this Lemma is deferred to Appendix A.

A further remark is in order about Lemma 1. Note that the altered controller shield $\mathfrak{S}^{\rho}_{\text{KBM}}$ maintains the energy efficient implementation of the controller shield $\mathfrak{S}_{\text{KBM}}$ as designed in [14]; indeed only two additional ReLU units are required, which amounts to a trivial per-evaluation increase in energy consumption.

4.3 KBM Runtime Safety Monitor: Generating Safety Expiration Times

Recall that the runtime safety monitor of EnergyShield must provide an *expiration* on the safety of the vehicle during an offload period, throughout which only a single, fixed control input is applied. This expiration must come with a *provable guarantee* that the vehicle safety is not compromised in the interim. In the formulation of EnergyShield and Problem 2, this means only that $h_{\bar{r},\sigma}$ must remain non-negative until the expiration of the deadline provided by the runtime safety monitor: see the condition (11) of Problem 2.

This formulation is convenient because it means that the problem can again be analyzed in continuous time, unlike our consideration of Problem 1 above: the conversion back to discrete time involves a floor operation; and compensating for the one-sample state delay induced by computing $\mathcal{NN}_p$ involves subtracting one sample from the result. That is, to solve Problem 2 and design an EnergyShield-safety monitor, it is sufficient provide a (real) time, ν, s.t.:

$$\forall t \in [0,\nu] \;.\; h\big(\zeta^{0,\chi[n_0-1]}_{1_{\omega[n_0]}}(t)\big) > 0. \tag{13}$$

In other words, the flow of f_{KBM} started from $\chi[n_0-1]$ and using constant control $\omega[n_0]$ maintains $h > 0$ for the duration $[0,\nu]$. We emphasize again that such a ν can be converted into the correct sample units expected for the output $\Delta_{\max}(\chi[n_0-1],\omega[n_0])$ by using a floor operation and subtracting one. With this context, we have the following Lemma, which solves Problem 2.

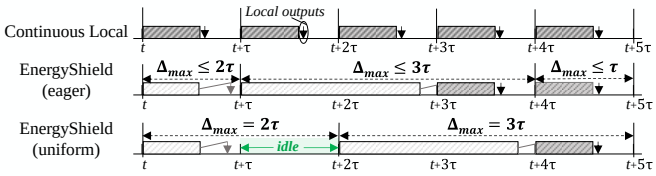

Figure 4: The continuous local and EnergyShield operational policies for a time axis t and base time windows of length τ. Darker instances indicate local execution.

Lemma 2. *Let Assumptions 1 - 5 hold as usual. Let*

$$\Delta_{max}(\chi[n_0-1,\omega[n_0]) \triangleq \max(\lfloor \nu(\chi[n_0-1],\omega[n_0])\rfloor - 1, 0) \tag{14}$$

where $\nu = \nu(\chi[n_0-1],\omega[n_0])$ solves the equation:

$$\sqrt{2}\cdot L_{h_{\bar{r},\sigma}} \cdot \|f_{KBM}(\chi[n_0-1],\omega[n_0])\|_2 \cdot \nu \cdot e^{L_{f_{KBM}}\cdot\nu} = h(\chi[n_0-1]) \tag{15}$$

for $L_{h_{\bar{r},\sigma}}$ and $L_{f_{KBM}}$ upper bounds on the Lipschitz constants of $h_{\bar{r},\sigma}$ and f_{KBM}, respectively.

Then $\Delta_{max}(\chi[n_0-1,\omega[n_0])$ so defined solves Problem 2.

The proof of Lemma 2 is deferred to Appendix B.

Lemma 2 specifies a complete solution to Problem 2, as claimed. However in its immediate form, it requires numerically solving (15) with each evaluation of $\Delta_{\max}(\chi[n_0-1],\omega[n_0])$; i.e., each time a safety expiration time is requested from the runtime safety monitor (every sample in the case where the offloading period is terminated before offload). The nature of (15) is such that solving it numerically is not especially burdensome – especially compared to the NN evaluations it replaces; however, as a practical matter it is also possible to implement it as a LUT to achieve greater energy efficiency.

5 EXPERIMENTS AND FINDINGS

The purpose of this section is to assess the following key aspects of EnergyShield: (*i*) the extent of energy savings achievable compared to the conventional continuous local execution mode given a wide variety of network conditions and server delays, (*ii*) its ability to enforce the safety through obstacle collision avoidance, (*iii*) how representative the upper bounds of the edge response time (Δ_{max}) are of the inherent risks existing in the corresponding driving context, and (*iv*) its generality when integrated with controllers of distinctive behavioral characteristics, in particular when it comes to different learnt driving policies.

5.1 Experimental Setup

5.1.1 Operational Policies: In addition to the baseline continuous local execution, we designate two EnergyShield offloading modes:

- ***Eager:*** in which a new offloading interval is immediately started if either edge responses have been received at the ADS or Δ_{max} expired.
- ***Uniform:*** in which the start of a new offloading interval is always delayed until Δ_{max} expires, regardless of whether edge responses have been received or not.

We define both these modes to reflect the attainable behavioral trade-offs of EnergyShield with regards to realizing an ideal control behavior or maximizing energy efficiency. This distinction is illustrated through the first offloading interval in Figure 4 in which the

uniform EnergyShield idles upon its retrieval of the edge responses until Δ_{max} expires unlike the eager EnergyShield mode.

5.1.2 Experimental Scenario: We perform our experiments using the CARLA open-source simulator for autonomous driving research [10]. We follow the setup proposed in [14], and implement a similar experimental scenario[3]. Basically, the scenario involves a four-wheeled vehicle travelling from a starting position A to destination B along a 100m motorway track with 4 pedestrian obstacles in its path. The first obstacle spawns after 40m of the track, while the remaining spawning positions are uniformly spaced between the first obstacle's position and that of the final destination – with a potential $\pm 10m$ variation along the longitudinal axis.

5.1.3 Experimental Settings: Throughout this section, *all* of our experiments are conducted under different combinations of the following two binary configuration parameters:

- **S**: this binary variable indicates whether the Controller Shield component is *active* (see Section 3.2).
- **N**: this binary variable indicates whether this is a more challenging, "noisy" version of the experimental test case.

In particular, the noisy version entails perturbing the obstacles' spawning positions by additional values sampled from a normal distribution $\mathcal{N}(0, 1.5m)$ along both the longitudinal and latitudinal axis. As an example, the configuration settings (S = 1, N = 0) indicate that the experiment was performed with Controller Shield active and with no perturbations in the obstacles' spawning positions.

5.1.4 Simulation Setup: For the controller model, its first stage entails two concurrent modules: an object detector as the large NN model of the ADS and a β Variational Autoencoder (β-VAE) providing additional latent feature representations of the driving scene. Both components operate on 160×80 RGB images from the vehicle's attached front-facing camera. In the subsequent stage, a Reinforcement Learning (RL) agent aggregates the detector's bounding box predictions, latent features, and the inertial measurements (δ_f^c, v, and a) to predict vehicle control actions (steering angle and throttle). The inertial measurements can be fetched directly from CARLA, whose positional and orientation measurements are also used directly to calculate r and ξ relative the vehicle's current nearest obstacle for obstacle state estimation. We trained the RL controller agents using a reward function, $\mathcal{R}$, that aims to maximize track completion rates through collision avoidance while minimizing the vehicle's center deviance from the primary track. For the definition of $\mathcal{R}$ as well as details of the RL agents see Appendix C.

5.1.5 Performance Evaluations: We use a pretrained ResNet-152 for our object detector and benchmark its performance in terms of latency and energy consumption when deployed on the industry-grade Nvidia Drive PX2 Autochauffer ADS. We found that a single inference pass on the ResNet-152 took ≈ 16 ms, and accordingly, we fixed the time-step in CARLA at 20 ms since the detector-in-the-loop was the simulation's computational bottleneck. To evaluate the wireless transmission power, we use the data transfer power models in [17] and assume a Wi-Fi communication link.

[3]https://github.com/shieldNN/shieldNN2020

5.1.6 Wireless Channel Model: We model the communication overheads between the ego vehicle and edge server as:

$$L_{comm} = L_{Tx} + L_{que} \quad s.t., \; L_{Tx} = \frac{a}{\phi}, \tag{16}$$

where L_{que} represents potential queuing delays at the server whereas L_{Tx} is the transmission latency defined by the size of the transmission data, a, over the experienced channel throughput, ϕ. Here, we assume ϕ as the "effective" channel throughput experienced at the ego vehicle, which takes into consideration potential packet drops, retransmissions, etc. We leverage a Rayleigh distribution model as in [27] to sample throughput values $\phi \sim Rayleigh(0, \sigma_\phi)$ with zero mean and σ_ϕ scale (default σ_ϕ=20 Mbps). Details on the the modelling of the queuing delays as well as the server response time estimation (i.e., $\hat{\Lambda}$) are also provided in Appendix C.

5.2 EnergyShield Evaluations

The purpose of this experiment is to assess the controller's performance when supplemented with EnergyShield in terms of energy efficiency and safety. For every configuration of S and N, we run the test scenario for 35 episodes and aggregate their combined results.

Energy Efficiency: We first assess the energy efficiency gains offered by EnergyShield compared to the baseline continuous local execution. As illustrated Figure 5, the left barplot demonstrates that both modes of EnergyShield substantially reduce the energy consumption footprint of the NN compared to local execution across all S and N configurations. For instance, under the default configuration (S = 0, N = 0), EnergyShield energy reductions reach 20% and 40.4% for the eager and uniform modes, respectively. These numbers further improve for the subsequent configurations in which N = 1 or S = 1. Upon inspection, we find that this is the result of the ego vehicle encountering more instances in which obstacles are not in the direct line-of-sight of its heading. The reasons being that at N = 1, some obstacles can be displaced out of the primary lane that the ego vehicle follows to complete the track, whereas at S = 1, such instances result from the Controller Shield applying corrective behaviors on the NN's predicted steering outputs, resulting in more tangential orientations of the vehicle with respect to the obstacles (i.e., $\xi \rightarrow \pm\pi/2$). Accordingly, large values of Δ_{max} – about 4-5 time samples (equivalent to 80-100 ms) – are increasingly sampled, and that automatically translates into more offloading decisions. For instance at (S = 1, N = 0), the energy efficiency gains reach 24.3% and 54.6% for the respective eager and uniform modes.

Safety Evaluation: To assess the EnergyShield's ability to enforce safety, we designate track completion rates (TCR %) as a comparison metric to signify the proportion of times the vehicle was able to complete the track without collisions. Taking the local execution mode as the test scenario, the right barplot of Figure 5 shows that without an active Controller Shield (S = 0), collisions with the pedestrian obstacles cause the TCR% to be 65.7% at N = 0, and even less at 22.9% for the noisy test case (N = 1). However, when the Controller Shield is active (S = 1), collisions are completely avoided and the TCR (%) values jump to 100% for both cases. This is also visible through the respective improvements in $\mathcal{R}$ which reached 13.3% and 61.1%. To further demonstrate such occurrences, we analyze in Figure 6 the ego vehicle's chosen trajectories across 3 episodes of dissimilar (S, N) configurations. As shown, the (S =

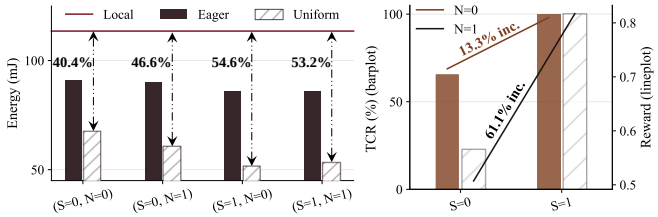

Figure 5: EnergyShield's energy efficiency gains with respect to continuous local execution (*left*) and safety analysis in terms of the $\mathcal{R}$ evaluation and % TCR (*right*).

0, N = 0) instance incurs a collision with the pedestrian object and does not complete the track. An active Controller Shield (S = 1), however, enforces a left or right corrective maneuvering action for obstacle avoidance and maintaining safety. Pictorial illustrations of these actions are also provided in the Figure.

Energy vs. Distance: To assess how representative the Δ_{max} upper bounds provided by the Runtime Safety Monitor are of the corresponding driving scene context, we examine EnergyShield's energy consumption at different distances from the nearest obstacle (r). The hypothesis is that larger r values imply relatively "safer" driving situations, which would result in larger values of Δ_{max} to be sampled, and accordingly more offloading instances enhancing the NN's energy efficiency. As shown in Figure 7, we plot the average experienced normalized energy ratings of the two modes of EnergyShield with respect to local execution against r across every configuration's set of 35 episodes. Each tick on the horizontal axis accounts for an entire range of 1m distances rather than a single value – e.g., a value of 2 on the horizontal axis encompasses all distances in the range [2 - 3). At close distances (r < 4m), we find that EnergyShield modes incur almost the same energy consumption overhead as that from the default local execution. This is mainly accredited to the Runtime Safety Monitor recognizing the higher risks associated with the close proximity from the objects, and accordingly outputting smaller values of Δ_{max} that can only be satisfied by local execution. As the distance from obstacles increases, so do the values of Δ_{max}, causing a gradual increase in the number of offloading instances, followed by a progressive reduction in energy consumption. For instance, the eager and uniform modes achieve 32% and 66% respective reductions in energy consumption at r = 13 m for the (S = 1, N = 1) configuration. Even more so, all configurations of the respective eager and uniform modes at the (r > 20m) bracket realize 33% and 67% respective energy gains.

5.3 Wireless Channel Variations

In this experiment, we assess how the performance gains of EnergyShield are affected given variations of the wireless channel conditions. Specifically, given potential changes in the channel throughput, ϕ, or the queuing delays, q, we investigate to what extent do the energy savings offered by EnergyShield vary. Additionally, we examine for every set of experimental runs what percentage of their total elapsed time windows were extra transition windows needed to complete a single offloading instance, which we denote by the % *Extra Transit Windows* metric. From here, we first analyze such effects when varying $\sigma_\phi \in \{20, 10, 5\}$ Mbps

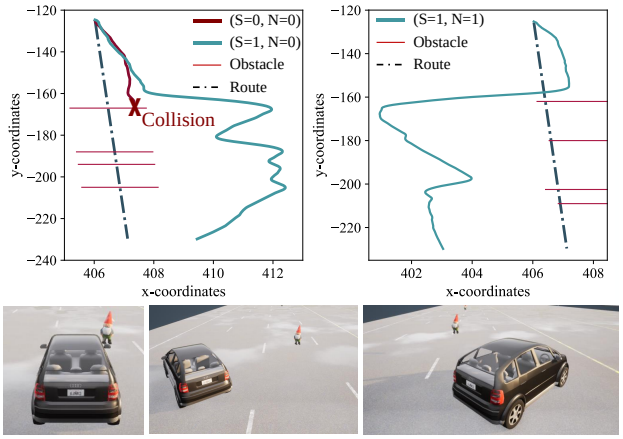

Figure 6: Top: Example trajectories followed by the ego vehicle with the start point at the top. Bottom: illustration of how the ego vehicle under the aforementioned operational modes behaved in reaction to the first encountered obstacle.

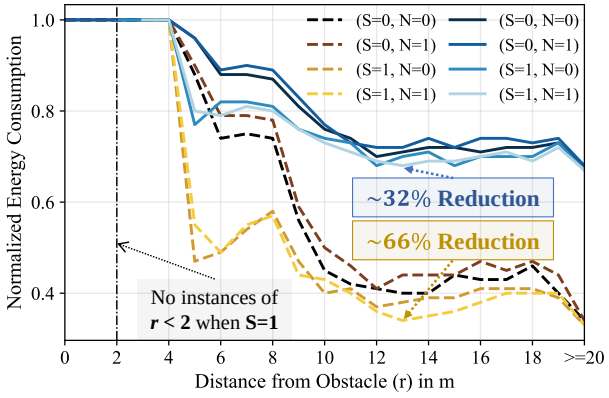

Figure 7: Normalized Energy Gains for the eager (*solid*) and uniform (*dashed*) EnergyShield modes with respect to the distance from obstacle (r) in m.

given a fixed q = 1 ms, and then when varying $q \in \{10, 20, 50\}$ ms given a fixed σ_ϕ = 10 Mbps. For the uniform EnergyShield, we notice in Figure 8 that the % *Extra Transit Windows* drops for the contrasting conditions of high throughput (σ_{phi} = 20 Mbps) and high queuing delays (q = 50 ms), reaching medians of 7% and 8%, respectively. This can be justified in light of how the benign channel conditions (σ_ϕ = 20 Mbps) indicate that the majority of offloading instances are concluded in a single time window with no considerable need for extra transmission windows. Whereas at unfavorable wireless conditions (q = 50 ms), $\hat{\Delta}$ values often exceed Δ_{max}, leading EnergyShield to opt for local execution more often so as to avoid wireless uncertainty, lowering the total number of transmission windows alltogether. Such effects are also visible in the twin Figure 9 as EnergyShield's energy consumption varies across these contrasting conditions, reaching respective medians of 45% and 93% of the local execution energy at σ_ϕ=20 and q=50.

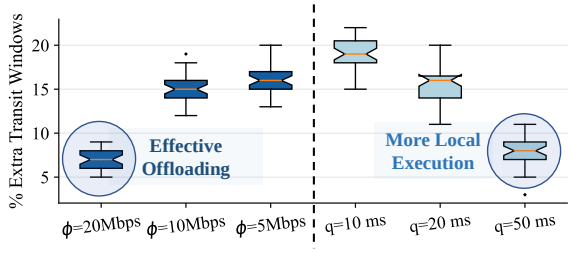

Figure 8: Analyzing the % extra transit windows over 35 episodes of uniform EnergyShield given various σ_ϕ and q.

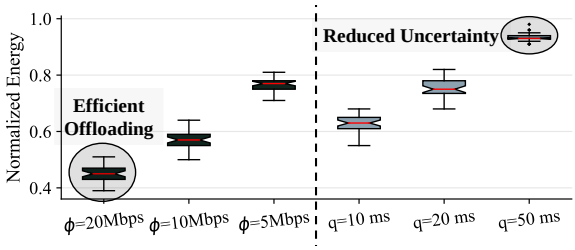

Figure 9: Analyzing the Normalized Energy cons. over 35 episodes of uniform EnergyShield given various σ_ϕ and q.

5.4 Generality

To assess the generality of EnergyShield, we train 3 additional RL controllers to evaluate how consistent EnergyShield is with regards to maintaining safety guarantees, and how the energy efficiency gains would vary given a distinctive driving behavior for each agent. Hence, we repeat the primary test runs for the additional controllers in which we average the energy consumption and TCR across 35 episodes of each viable (S, N) configuration. We also report the average center deviance (CD) experienced by the ego vehicle from the primary track lane as a metric to characterize the different driving behaviors of each controller. In our experimental test case, larger CD imply larger r values, that is, controllers with low values of CD tend to drive closer to the obstacles at higher risks of collisions (maximizing $\mathcal{R}$ through minimizing CD), whereas larger values of CD indicate the agents have learnt to take the farther lanes of the track in order to maximize track completions through prioritizing collision avoidance (maximizing $\mathcal{R}$ through maximizing TCR). In the following, we highlight the key findings and refer the reader to Table 1 in Appendix C for the detailed results.

Our first key finding is that given S = 0, TCR (%) is dependent on CD ratings as only the controllers with CD > 5m consistently achieve the 100% TCR. At S = 1, however, this dependency no longer holds at S = 1 when the Controller Shield enforces the obstacle avoidance safety guarantees, pushing all controllers to achieve 100% TCR irrespective of CD values. The other interesting finding here is that across both modes of EnergyShield, the average energy consumption is less for controllers with larger CD ratings. For instance at (S = 1, N = 1), when the RL controllers are arranged in an increasing order of their CD values 2.8⟶3.6⟶5.4⟶5.7 m, the average energy consumption per inference decreases in the respective order 53.1⟶45.7⟶42.1⟶39.8 mJ. Indeed, this highlights EnergyShield's capability of conducting *safe* and *effective* context-aware offloading, especially in light of how the Runtime Safety Monitor can provide large Δ_{max} to realize more energy gains in the safer situations (e.g., larger distances from obstacles), and how the Controller Shield always maintains the safety guarantees independent of offloading decisions.

REFERENCES

[1] Anayo K Akametalu, Jaime F Fisac, Jeremy H Gillula, Shahab Kaynama, Melanie N Zeilinger, and Claire J Tomlin. 2014. Reachability-based safe learning with Gaussian processes. In *53rd IEEE Conference on Decision and Control*. IEEE.

[2] Mohammed Alshiekh, Roderick Bloem, Ruediger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. 2017. Safe Reinforcement Learning via Shielding. (2017).

[3] Sabur Baidya, Yu-Jen Ku, Hengyu Zhao, Jishen Zhao, and Sujit Dey. 2020. Vehicular and edge computing for emerging connected and autonomous vehicle applications. In *ACM/IEEE Design Automation Conference (DAC'2020)*.

[4] Felix Berkenkamp, Matteo Turchetta, Angela Schoellig, and Andreas Krause. 2017. Safe model-based reinforcement learning with stability guarantees. In *Advances in neural information processing systems*.

[5] Richard Cheng, Gábor Orosz, Richard M Murray, and Joel W Burdick. 2019. End-to-end safe reinforcement learning through barrier functions for safety-critical continuous control tasks. (2019), 3387–3395 pages.

[6] Yinlam Chow, Ofir Nachum, Edgar Duenez-Guzman, and Mohammad Ghavamzadeh. 2018. A lyapunov-based approach to safe reinforcement learning. In *Advances in neural information processing systems*. 8092–8101.

[7] Yinlam Chow, Ofir Nachum, Aleksandra Faust, Edgar Duenez-Guzman, and Mohammad Ghavamzadeh. 2019. Lyapunov-based safe policy optimization for continuous control. *arXiv preprint arXiv:1901.10031* (2019).

[8] Mingyue Cui, Shipeng Zhong, Boyang Li, Xu Chen, and Kai Huang. 2020. Offloading Autonomous Driving Services via Edge Computing. *IEEE Internet of Things Journal* 7, 10 (2020), 10535–10547.

[9] Charles Dawson, Sicun Gao, and Chuchu Fan. 2022. Safe Control with Learned Certificates: A Survey of Neural Lyapunov, Barrier, and Contraction Methods.

[10] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. 2017. CARLA: An open urban driving simulator. In *Conference on robot learning*. PMLR, 1–16.

[11] Souradeep Dutta, Susmit Jha, Sriram Sankaranarayanan, and Ashish Tiwari. 2018. Output range analysis for deep feedforward neural networks. In *NASA Formal Methods Symposium*. Springer.

[12] Mahyar Fazlyab, Alexander Robey, Hamed Hassani, Manfred Morari, and George Pappas. 2019. Efficient and accurate estimation of lipschitz constants for deep neural networks. In *Advances in Neural Information Processing Systems*.

[13] Jingyun Feng, Zhi Liu, Celimuge Wu, and Yusheng Ji. 2018. Mobile edge computing for the internet of vehicles: Offloading framework and job scheduling. *IEEE vehicular technology magazine* 14, 1 (2018), 28–36.

[14] James Ferlez, Mahmoud Elnaggar, Yasser Shoukry, and Cody Fleming. 2020. Shieldnn: A provably safe nn filter for unsafe nn controllers. *arXiv preprint arXiv:2006.09564* (2020).

[15] Jaime F Fisac, Anayo K Akametalu, Melanie N Zeilinger, Shahab Kaynama, Jeremy Gillula, and Claire J Tomlin. 2018. A general safety framework for learning-based control in uncertain robotic systems. *IEEE Trans. Automat. Control* 64, 7 (2018).

[16] Vijay Govindarajan, Katherine Driggs-Campbell, and Ruzena Bajcsy. 2017. Data-driven reachability analysis for human-in-the-loop systems. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*. IEEE, 2617–2622.

[17] Junxian Huang, Feng Qian, Alexandre Gerber, Z Morley Mao, Subhabrata Sen, and Oliver Spatscheck. 2012. A close examination of performance and power characteristics of 4G LTE networks. In *Proceedings of the 10th international conference on Mobile systems, applications, and services*. 225–238.

[18] Radoslav Ivanov, James Weimer, Rajeev Alur, George J Pappas, and Insup Lee. 2019. Verisig: verifying safety properties of hybrid systems with neural network controllers. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*. 169–178.

[19] Torsten Koller, Felix Berkenkamp, Matteo Turchetta, and Andreas Krause. 2018. Learning-based model predictive control for safe exploration. In *2018 IEEE Conference on Decision and Control (CDC)*. IEEE.

[20] Jason Kong, Mark Pfeiffer, Georg Schildbach, and Francesco Borrelli. 2015. Kinematic and Dynamic Vehicle Models for Autonomous Driving Control Design. In *2015 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 1094–1099.

[21] Jlin2018architectural Shih-Chieh Lin et al. [n. d.]. The architectural implications of autonomous driving: Constraints and acceleration. In *Proc. of Intl. Conf. on Arch. Support for Programming Languages and Operating Systems (ASPLOS'18)*.

[22] Changliu Liu, Tomer Arnon, Christopher Lazarus, Clark Barrett, and Mykel J Kochenderfer. 2019. Algorithms for verifying deep neural networks. *arXiv preprint arXiv:1903.06758* (2019).

[23] Shaoshan Liu, Liangkai Liu, Jie Tang, Bo Yu, Yifan Wang, and Weisong Shi. 2019. Edge computing for autonomous driving: Opportunities and challenges. *Proc.*

IEEE 107, 8 (2019), 1697–1716.

[24] Christian Llanes, Matthew Abate, and Samuel Coogan. 2022. Safety from Fast, In-the-Loop Reachability with Application to UAVs. In *2022 ACM/IEEE 13th International Conference on Cyber-Physical Systems (ICCPS)*. IEEE, 127–136.

[25] Arnav Malawade, Mohanad Odema, Sebastien Lajeunesse-DeGroot, and Mohammad Abdullah Al Faruque. 2021. SAGE: A Split-Architecture Methodology for Efficient End-to-End Autonomous Vehicle Control. *ACM Transactions on Embedded Computing Systems (TECS)* 20, 5s (2021), 1–22.

[26] Aniruddh Mohan, Shashank Sripad, Parth Vaishnav, and Venkatasubramanian Viswanathan. 2020. Trade-offs between automation and light vehicle electrification. *Nature Energy* 5, 7 (2020), 543–549.

[27] Mohanad Odema, Luke Chen, Marco Levorato, and Mohammad Abdullah Al Faruque. 2022. Testudo: Collaborative Intelligence for Latency-Critical Autonomous Systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2022).

[28] Alexander Robey, Haimin Hu, Lars Lindemann, Hanwen Zhang, Dimos V Dimarogonas, Stephen Tu, and Nikolai Matni. 2020. Learning Control Barrier Functions from Expert Demonstrations. *arXiv preprint arXiv:2004.03315* (2020).

[29] Kengo Sasaki, Naoya Suzuki, Satoshi Makido, and Akihiro Nakao. 2016. Vehicle control system coordinated between cloud and mobile edge computing. In *2016 55th Annual Conference of the Society of Instrument and Control Engineers of Japan (SICE)*. IEEE, 1122–1127.

[30] Guanya Shi, Xichen Shi, Michael O'Connell, Rose Yu, Kamyar Azizzadenesheli, Animashree Anandkumar, Yisong Yue, and Soon-Jo Chung. 2019. Neural lander: Stable drone landing control using learned dynamics. In *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 9784–9790.

[31] Mohit Srinivasan, Amogh Dabholkar, Samuel Coogan, and Patricio Vela. 2020. Synthesis of control barrier functions using a supervised machine learning approach. *arXiv preprint arXiv:2003.04950* (2020).

[32] Xiaowu Sun, Haitham Khedr, and Yasser Shoukry. 2019. Formal verification of neural network controlled autonomous systems. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*. 147–156.

[33] Sihai Tang, Bruce Chen, Harold Iwen, Jason Hirsch, Song Fu, Qing Yang, Paparao Palacharla, Nannan Wang, Xi Wang, and Weisong Shi. 2021. VECFrame: A Vehicular Edge Computing Framework for Connected Autonomous Vehicles. In *2021 IEEE International Conference on Edge Computing (EDGE)*. IEEE, 68–77.

[34] Andrew J Taylor, Andrew Singletary, Yisong Yue, and Aaron D Ames. 2020. A control barrier perspective on episodic learning via projection-to-state safety. *arXiv preprint arXiv:2003.08028* (2020).

[35] Haoxin Wang, BaekGyu Kim, Jiang Xie, and Zhu Han. 2019. E-auto: A communication scheme for connected vehicles with edge-assisted autonomous driving. In *ICC 2019-2019 IEEE International Conference on Communications (ICC)*. IEEE.

[36] Li Wang, Evangelos A Theodorou, and Magnus Egerstedt. 2018. Safe learning of quadrotor dynamics using barrier certificates. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2460–2465.

[37] Weiming Xiang, Diego Manzanas Lopez, Patrick Musau, and Taylor T Johnson. 2019. Reachable set estimation and verification for neural network models of nonlinear dynamic systems. In *Safe, Autonomous and Intelligent Vehicles*.

[38] Wei Xiao, Noushin Mehdipour, Anne Collin, Amitai Y Bin-Nun, Emilio Frazzoli, Radboud Duintjer Tebbens, and Calin Belta. 2021. Rule-based optimal control for autonomous driving. In *Proceedings of the ACM/IEEE 12th International Conference on Cyber-Physical Systems*. 143–154.

[39] Xiangru Xu, Paulo Tabuada, Jessy W. Grizzle, and Aaron D. Ames. 2015. Robustness of Control Barrier Functions for Safety Critical Control. *IFAC-PapersOnLine* 48, 27 (2015), 54–61.

[40] Xumiao Zhang, Anlan Zhang, Jiachen Sun, Xiao Zhu, Y Ethan Guo, Feng Qian, and Z Morley Mao. 2021. Emp: Edge-assisted multi-vehicle perception. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*. 545–558.

A PROOF OF LEMMA 1

Proof. In this proof, let $\chi[n'] = (r[n'], \xi[n'], v[n']), n' \in \mathbb{Z}$.

The proof will be constructive. To this end, recall that we have assumed a capped (controlled) maximum vehicle velocity of $v_{\max}$. Thus, let $\gamma = 2 \cdot v_{\max} \cdot T$, where T is the sample period (see Section 2.1). As a consequence, also note that for $r[n_0-1] > r_{\min}(\xi[n_0-1])$:

$$r[n_0+1] > r[n_0-1] - 2 \cdot v_{\max} \cdot T > \bar{r} - \gamma \tag{17}$$

so that on any two-sample period

$$|\dot{\xi}| \leq v_{\max} \cdot \left(\frac{1}{\bar{r}-\gamma} + \frac{1}{\ell_r}\right) \tag{18}$$

Then observe that the Lipschitz constant of the function $r_{\min}$ is bounded by $L_{r_{\min}} \leq \frac{\bar{r}\cdot\sigma}{2\cdot(1-2\cdot\sigma)^2}$. Finally, conclude that

$$\begin{aligned}|r_{\min}(\xi[n_0-1]) - r_{\min}(\xi[n_0+1])| \\ \leq \tfrac{\bar{r}\cdot\sigma}{2\cdot(1-2\cdot\sigma)^2} \cdot v_{\max} \cdot \left(\tfrac{1}{\bar{r}-\gamma} + \tfrac{1}{\ell_r}\right) \cdot 2 \cdot T \triangleq \eta\end{aligned} \tag{19}$$

Then choose $\rho \triangleq \eta + \gamma$.

Now, given the structure of the amended shield in (12), establishing the conclusion of Problem 1 can be broken into three cases:

(i) $r[n_0] \geq r_{\min}(\xi[n_0]) + \rho$; (irrespective of the position of $r[n_0-1]$)
(ii) $r[n_0] < r_{\min}(\xi[n_0]) + \rho$ and $r[n_0-1] \geq r_{\min}(\xi[n_0-1]) + \rho$;
(iii) $r[n_0] < r_{\min}(\xi[n_0]) + \rho$ and $r[n_0-1] < r_{\min}(\xi[n_0-1]) + \rho$.

In each of the three cases, we need to show that for the next state, $h(\chi[n_0+1]) > 0$, or equivalently $r[n_0+1] > r_{\min}(\xi[n_0+1])$.

Case (i) and Case (ii). The claim follows for these cases for essentially directly by the choice of ρ above. In *Case (ii)*, we have that

$$\begin{aligned}r[n_0+1] - r_{\min}(\xi[n_0+1]) \geq \big(r[n_0-1] - r_{\min}(\xi[n_0-1])\big) \\ - \big(r[n_0+1] - r_{\min}(\xi[n_0+1]) - r[n_0-1] - r_{\min}(\xi[n_0-1])\big).\end{aligned} \tag{20}$$

From the above calculations, we see that the second term on the right-hand side of (20) is bounded below by $-\rho$ (using the triangle inequality). Thus

$$r[n_0+1] - r_{\min}(\xi[n_0+1]) \geq \big(r[n_0-1] - r_{\min}(\xi[n_0-1])\big) - \rho \tag{21}$$

and the desired conclusion follows since $r[n_0-1] - r_{\min}(\xi[n_0-1]) \geq \rho$ by assumption. A similar approach proves *Case (i)*: simply repeat the calculations in the definition of ρ, only over one sample.

Thus, it remains to consider *Case (iii)*. This case is somewhat easier, because the control signal is being overridden, so the state delay presents technical difficulties as above. Thus, it follows almost directly from the properties of the controller shield as designed in [14]. In particular, the ShieldNN verifier establishes that the boundary between "safe" and "unsafe" controls is a concave (resp. convex) function of ξ for $\xi \in [0, \pi]$ (resp. $\xi \in [-\pi, 0]$). Hence, by [14, Theorem 1], the constant control $\beta_{\max}$ (resp. $-\beta_{\max}$) always preserves safety for *any duration of time* starting from a state $\xi \in [0, \pi]$ (resp. $\xi \in [-\pi, 0]$). □

B PROOF OF LEMMA 2

Proof. By the arguments above, the form of $\Delta_{\max}$ in (14) will solve Problem 2, provided (15) implies (13). Thus, we confine the proof to showing this fact.

To begin, observe that:

$$\begin{aligned}h_{\bar{r},\sigma}(\zeta^{0,\chi[n_0-1]}_{1_{\omega[n_0]}}(t)) \geq \Big|h_{\bar{r},\sigma}(\zeta^{0,\chi[n_0-1]}_{1_{\omega[n_0]}}(t)) - h_{\bar{r},\sigma}(\chi[n_0-1])\Big| \\ - \Big|h_{\bar{r},\sigma}(\chi[n_0-1])\Big|\end{aligned} \tag{22}$$

by the triangle inequality. Consequently:

$$\begin{aligned}\Big|h_{\bar{r},\sigma}(\zeta^{0,\chi[n_0-1]}_{1_{\omega[n_0]}}(t)) - h_{\bar{r},\sigma}(\chi[n_0-1])\Big| \leq \Big|h_{\bar{r},\sigma}(\chi[n_0-1])\Big| \\ \implies h_{\bar{r},\sigma}(\zeta^{0,\chi[n_0-1]}_{1_{\omega[n_0]}}(t)) \geq 0\end{aligned} \tag{23}$$

Hence define $z(\zeta^{0,\chi[n_0-1]}_{1_{\omega[n_0]}}(t)) \triangleq \Big|h_{\bar{r},\sigma}(\zeta^{0,\chi[n_0-1]}_{1_{\omega[n_0]}}(t)) - h_{\bar{r},\sigma}(\chi[n_0-1])\Big|$.

By the Grönwall inequality, we have further that:

$$z(\zeta^{0,\chi[n_0-1]}_{1_{\omega[n_0]}}(t)) \leq \sqrt{2} \cdot L_{h_{\hat{F},\sigma}} \cdot \|f_{\text{KBM}}(\chi[n_0-1], \omega[n_0])\|_2 \cdot t \cdot e^{L_{f_{\text{KBM}}} \cdot t} \tag{24}$$

where $L_{h_{\hat{F},\sigma}}$ and $L_{f_{\text{KBM}}}$ are as in the statement of the Lemma. Observe that the function on the right-hand side of (24) is monotonic in t. Thus, we can use (23) to claim that if ν solves (15) (derived immediately from (24) and (23)), then the claim of the Lemma holds. □

C ADDITIONAL EXPERIMENTAL DETAILS

C.1 Training Details

The primary RL agent training was conducted under the (S = 0, N = 0) configuration settings using the Proximal Policy Optimization (PPO) algorithm for a total of 1800 episodes. In the last 400 training episodes, we randomized the ego vehicle's spawning position and orientation along its lateral dimension to aid the agent in learning how to recover from maneuvering moves. For the β-VAE, we used the pretrained model from [14] which was trained to generate a 64-dimensional latent feature vector from Carla driving scenes.

C.2 Reward Function

For the reward function $\mathcal{R}$, we defined:

$$\mathcal{R} = \begin{cases} -P, & \text{collision or } CD > CD_{th} \\ +P, & \text{track completed successfully} \\ f_{\mathcal{R}}(v, CD, \vartheta, r), & \text{otherwise} \end{cases} \tag{25}$$

in which P is large positive number, v is the vehicle's velocity, CD is the vehicle's center deviance from the center of the track, CD_{th} is a predetermined threshold value, ϑ represents the angle between the heading of the vehicle and the tangent to the curvature of the road segment, and r is the distance to the closest obstacle. As shown, $\mathcal{R}$ can evaluate to: (*i*) (+P) if it completes the track successfully (large positive reward), (*ii*) (−P) if it incurs a collision or deviates from the center of the road beyond CD_{th}, or (*iii*) a function $f_{\mathcal{R}}(\cdot)$ of the aforementioned variables given by:

$$f_{\mathcal{R}}(v, CD, \vartheta, r) = \omega_1 \cdot f_1(v) + \omega_2 \cdot f_2(\mathcal{CD}) + \omega_3 \cdot f_3(\vartheta) + \omega_4 \cdot f_4(r)$$

$$\text{s.t.,} \quad f_1(v) = \begin{cases} \frac{v}{v_{min}}, & v < v_{min} \\ 1 - \frac{v - v_{target}}{v_{max} - v_{target}} & v > v_{target} \\ 1, & otherwise \end{cases}$$

$$f_2(CD) = \max(1 - \frac{l_{center}}{l_{max}}, 0)$$

$$f_3(\vartheta) = \max(1 - |\frac{\vartheta}{\pi/9}|, 0)$$

$$f_4(dist) = \max(\min(\frac{||r||}{r_{max}}, 1), 0)$$

in which $v_{min}, v_{max}, v_{target}$ are the minimum, maximum, and target velocities, respectively. l_{center} is the lateral distance from the center of the vehicle to the designated track. ϑ is the angle between the head of the vehicle and the track's tangent. For our experiments, we set v_{min}=35 km/hr, v_{target}=40 km/hr, v_{max}=45 km/hr, r_{max} = 20 m, l_{max} = 10 m, and P= 100.

C.3 Performance Evaluations

We use the standard TensorRT library to compile our models on the Nvidia Drive PX2 ADS platform as an optimized inference engine and measure its execution latency. To evaluate the local execution power, we measure the difference in average power drawn by the Nvidia Drive PX2 when processing and idling.

C.4 Queuing Delays

We leverage the The M/M/1/k model for the queuing delays, L_{que}, which entails $q_c = \frac{(1-\rho)(\rho)^c}{1-\rho^{C+1}}$ representing the probability that an offloaded task will find c tasks stored in the server's buffer of size C upon arrival with ρ being the average server load. We assume each task contributes an extra 1 ms delay, and thus, q_c positions directly translate to L_{que} in ms. The default settings for queuing delays entail C = 4000 and ρ = 0.97 unless otherwise was stated.

C.5 Edge Response Estimation

As offloading decisions are made based on estimates of the prior edge response times, the estimated communication latency, $\hat{L}_{comm}$, at time n can be defined as a function of the k previous values of the effective throughput ϕ and queuing delays q as follows: $\hat{L}_{comm}(n) = \Phi(\phi_{n-k:n-1}, q_{n-k:n-1})$. For our experiments, we set k = 5 and employ a moving average function to evaluate Φ.

Table 1: EnergyShield Performance across 4 different RL controllers. Each RL agent learnt to travel the route through a distinctive policy represented by its center deviance (CD) from the primary track. The RL Controllers are numerically arranged in the increasing order of CD with Controller 1 being the main RL controller used in all evaluations.

Policy	(S, N)	Controller 1			Controller 2			Controller 3			Controller 4		
		CD(m)	TCR(%)	E(mJ)	CD (m)	TCR(%)	E(mJ)	CD(m)	TCR(%)	E(mJ)	CD(m)	TCR(%)	E(mJ)
Local	(0, 0)	0.92	65.7	113.5	2.3	100	113.5	5.5	100	113.5	5.8	100	113.5
	(0, 1)	0.8	22.9		2.3	97.1		5.5			5.8		
	(1, 0)	2.87	100		3.5	100		5.5			5.9		
	(1, 1)	2.91	100		3.6	100		5.4			5.7		
EnergyShield (*eager*)	(0, 0)	0.8	68.6	90.8	2.3	100	90	5.5	100	79.7	5.8	100	77.6
	(0, 1)	0.8	34.3	90	2.3	100	89			80	5.7		78.9
	(1, 0)	2.8	100	85.8	3.5	100	81.6			79.6	5.7		77.5
	(1, 1)	2.8	100	85.9	3.6	100	81.9			78.5	5.8		77.9
EnergyShield (*uniform*)	(0, 0)	0.9	74.3	67.7	2.3	100	63.5	5.5	100	43.7	5.8	100	39.7
	(0, 1)	0.7	22.9	60.6	2.3	97.1	63.1	5.5		44.4	5.7		40.8
	(1, 0)	2.9	100	51.5	3.5	100	44.5	5.5		43.6	5.8		40.1
	(1, 1)	2.8	100	53.1	3.6	100	45.7	5.4		42.1	5.7		39.8