Admissible Motion

Runtime Safety as an Admission-Control Problem,
Not a Model-Capability Problem

Perslis Research

2026

Status: White paper. **Research prototype, evaluated in simulation only.** Not a vehicle controller and not for any physical vehicle. All quantitative results are from a single simulated domain (CARLA 0.9.15, Town10HD). Prior art in runtime assurance, shielding, and reachability analysis is real and cited; the contribution is a controller-independent *admission-control* framing and a multi-controller empirical harness that places a hostile adversary and three frontier/local language models under one deterministic safety floor.

Abstract

A safety case built on "the model is good enough" is unfalsifiable: a policy of arbitrary measured quality can still propose a fatal control, and no finite evaluation over past situations proves the absence of a future one. We argue that safety for a system that moves should be a property of the *runtime*, not of the model that proposes actions, and we make that concrete. Any controller — rule-based, hostile, a local model, or a frontier model — only *proposes* a control; a deterministic *admission layer* verifies each candidate against grounded safety invariants and clamps, denies, or overrides it before it reaches the actuator. We factor the runtime as State $\to$ Reachability $\to$ Invariants $\to$ Admission $\to$ Actuation $\to$ Verification, and restate the controller-independent forward-invariance argument: if the system starts in the safe set and a safe fallback exists at every state, admission keeps the system in the safe set for *every* controller, because the controller never appears in the invariant. We instantiate the architecture as four composed invariants in a photoreal autonomous-driving simulator and report new empirical evidence from a multi-controller harness. Under one floor: a hand-written adversary whose only goal is to crash had all 64,952 of its invariant-violating commands overridden (100%) with 0 collisions over $\sim$40 km, while with the floor off the identical proposals reach the actuator and the vehicle crashes; and three language models (Gemini 2.5-flash, DeepSeek-chat, Qwen3-VL-4B) each drove the car under the floor with 0 collisions, but only because the floor, verifying ground truth at $\sim$20 Hz, covers the $\sim$ 1 s stale-glance window during which a model drives blind. Model capability governs how much useful work completes; the runtime governs what may be executed. This floor shares one engine with admission floors in other Perslis domains (science, law).

Safety should not depend on — and should not degrade with — the intelligence of the thing that proposes the action.

1 Introduction

An autonomous system that moves can hurt someone. The dominant strategy for making such systems safe is to make the policy that drives them better: more data, larger models, more training, more evaluation. This improves average behavior and is necessary work. It does not, however,

produce a safety guarantee. A policy is a function from states to proposed controls; nothing about its measured competence rules out the existence of a state at which it proposes something catastrophic. "The model scored well" is a statement about a distribution of situations already seen. Safety is a claim about a situation not yet seen — including one an adversary or a rare fault constructs on purpose.

This paper takes the position that the safety argument for a moving system should be *relocated*. Rather than asking the proposer to be trustworthy, we interpose a deterministic *admission layer* between any proposer and the actuator. The proposer proposes; the admission layer verifies each proposal against grounded safety invariants over a bounded look-ahead and either admits it, clamps it to the nearest safe control, or overrides it with a designated safe fallback. The controller is never handed the actuator directly. The safety guarantee then factors through the admission map, not through the proposer — and holds for every proposer, including ones we did not build and cannot inspect.

The framing, formal model, and forward-invariance argument here are consistent with the already-published Perslis Research note *Admissible Motion* [11], which demonstrated the architecture in the open MetaDrive simulator on a longitudinal following-distance invariant. The present white paper carries the same thesis and the same theorem, and adds a distinct empirical instance: a photoreal CARLA harness with four composed invariants under which we place both a hostile adversary and three frontier/local language models on the *same* floor. That harness surfaces a finding specific to putting language models at the wheel — the model decides on stale, second-old glances and drives blind between them, so the deterministic floor is not a redundant backstop but the load-bearing safety component.

Contributions.

1. **A restated formal model of runtime motion safety as admission control**, with a controller-independent forward-invariance property and an explicit statement of its standing assumptions (Sections 3–5), consistent with the published framing [11].

2. **A concrete four-invariant instantiation** — on-road, speed limit, following distance, emergency stop — as a deterministic, stateless shield evaluated every control tick, with the controller absent from every invariant (Section 6).

3. **A multi-controller empirical harness** in a photoreal simulator: a hostile crash-seeking adversary and three language models (Gemini 2.5-flash, DeepSeek-chat, Qwen3-VL-4B) under one floor, with a counterfactual decision log that records what the floor *would* have rejected even when it is switched off (Sections 7–8).

4. **A latency observation for models at the wheel**: measured glance latency of $\sim$0.8–2.0 s means a model drives $\sim$10 m blind per glance at 40 km/h; the floor verifying ground truth at $\sim$20 Hz is what covers that window (Section 9).

2 Capability is not a safety argument

Consider a learned policy π evaluated over a large test set on which it performs well. What has that established? That π tends to propose good controls on states drawn from the evaluation distribution. The property safety actually requires is different in kind: for *all* reachable states, the executed control keeps the system out of collision, on the road, within the speed limit, and behind a safe following gap. A capability metric aggregates over situations; a safety property quantifies

over them. No finite evaluation closes that gap, and an adversary or an unlucky sensor fault can drive the system to precisely the state the evaluation never sampled.

This is why "safe because the model is good" cannot be falsified in the way a safety claim must be. There is no experiment on the policy alone whose failure would let us conclude the policy is unsafe in general, and none whose success would let us conclude it is safe in general; the state space is too large and the tail is where the harm lives. Certification and insurance need a claim that can be discharged against invariants and evidence, not a claim about a model's average competence. We therefore stop asking the proposer to carry the safety argument at all, and move it to a component whose behavior we can state and check.

3 The inversion: an admission layer

The move is to separate the *proposer* from *admission*. The proposer's job is to be useful: make progress, keep the lane, complete the route. Admission's job is narrow and deterministic: given the current state and a proposed control, decide what — if anything — is allowed to reach the actuator. We arrange the runtime as a pipeline:

$$\text{State} \;\to\; \text{Reachability} \;\to\; \text{Invariants} \;\to\; \text{Admission} \;\to\; \text{Actuation} \;\to\; \text{Verification}. \tag{1}$$

The proposer sits *before* this pipeline and only supplies a candidate. The invariants encode the safe set. Reachability projects the candidate forward over a bounded horizon. Admission is the decision — allow, clamp, or override — and is the only thing with authority over the actuator. Verification logs what was admitted and why, so every override is an inspectable record: the action was verified before it acted. This is the runtime-assurance / Simplex lineage [1] and the shielding lineage from safe reinforcement learning [6]; our emphasis is that the safety case is carried entirely by the admission map and is therefore indifferent to which proposer is upstream.

The controller only proposes. A deterministic floor decides what may be executed.

4 Formal model

Let world state be $x \in \mathcal{X}$ — position, velocity, orientation, nearby objects, boundaries, and actuator state — and control be $u \in \mathcal{U}$, with discrete dynamics $x_{t+1} = f(x_t, u_t)$. The safe set is defined by invariant predicates $g_i(x) \le 0$ — on-road, speed limit, following distance, separation, actuator limits:

$$\mathcal{S} = \{\, x \in \mathcal{X} : g_i(x) \le 0 \;\; \forall i \,\}. \tag{2}$$

Define the reachable set over a horizon H:

$$\mathrm{Reach}_H(x, u) = \{\, x' : x' \text{ reachable from } x \text{ under } u \text{ within } H \text{ steps} \,\}. \tag{3}$$

The admission map $A : \mathcal{X} \times \mathcal{U} \to \mathcal{U}$ is then

$$A(x,u) = \begin{cases} u & (\textsc{allow}) \text{ if } \mathrm{Reach}_H(x,u) \subseteq \mathcal{S}, \\ u' & (\textsc{clamp}) \text{ nearest } u' \text{ with } \mathrm{Reach}_H(x,u') \subseteq \mathcal{S}, \\ u_{\text{safe}}(x) & (\textsc{deny} \,/\, \textsc{emergency}) \text{ otherwise — brake, hover, halt.} \end{cases} \tag{4}$$

The closed loop under an arbitrary controller π is

$$u_t = A\big(x_t,\, \pi(x_t)\big), \qquad x_{t+1} = f(x_t, u_t). \tag{5}$$

Note that π enters only inside A's argument and is discarded whenever admission clamps or overrides. This is the structural fact the safety property exploits.

5 The controller-independent safety property

We restate the forward-invariance argument of [11].

Property 1 (controller-independent forward invariance). *Suppose (a) $x_0 \in \mathcal{S}$; (b) a safe fallback exists at every state, i.e. $\forall x \in \mathcal{S}$, $\mathrm{Reach}_H\big(x, u_{\mathrm{safe}}(x)\big) \subseteq \mathcal{S}$; and (c) admission A is applied at every step. Then $x_t \in \mathcal{S}$ for all $t \geq 0$ and for every controller π.*

Argument. By induction on t. *Base case:* $x_0 \in \mathcal{S}$ by (a). *Inductive step:* assume $x_t \in \mathcal{S}$. Admission (4) returns one of three values. If it ALLOWs u, then $\mathrm{Reach}_H(x_t, u) \subseteq \mathcal{S}$ by the ALLOW condition, so $x_{t+1} \in \mathcal{S}$. If it CLAMPs to u', then $\mathrm{Reach}_H(x_t, u') \subseteq \mathcal{S}$ by the CLAMP condition, so $x_{t+1} \in \mathcal{S}$. If neither exists, admission executes $u_{\mathrm{safe}}(x_t)$, and $\mathrm{Reach}_H(x_t, u_{\mathrm{safe}}(x_t)) \subseteq \mathcal{S}$ by (b), so $x_{t+1} \in \mathcal{S}$. In every branch $x_{t+1} \in \mathcal{S}$. The controller π never appears in the invariant; it only influences *which* in-set control is chosen, never whether the chosen control keeps the system in $\mathcal{S}$. □

Corollary (model-independence). *The safety guarantee is invariant to π. Swapping rule-based $\rightarrow$ hostile $\rightarrow$ learned $\rightarrow$ frontier changes the quality of admitted behavior but not membership in $\mathcal{S}$. Model capability governs how much useful work completes; the runtime governs what may be executed.*

We state the standing assumptions plainly, because they are exactly the interfaces a physical deployment must discharge. Property 1 assumes: the state x_t fed to the invariants is *exact*; the actuator has *adequate authority* to realize u_{safe} (e.g. enough braking to stop in the available gap); and the invariants g_i *correctly encode* the intended safe set. Where these hold, the property holds by construction. Where they do not, the guarantee is only as good as the weakest interface — and that is a claim about perception, actuation, and specification, not about the proposer. Isolating that fact is the point of the architecture. In our simulation, assumption (a) holds by ground truth from the simulator; the empirical results below are exactly a controlled test of whether a runtime that applies A every tick holds the safe set regardless of which controller supplies π.

6 The four invariants: a concrete instantiation

The reference implementation is a deterministic, stateless shield, `SafetyShield`, evaluated on every control tick ($\sim$20 Hz). The controller proposes a pair (steer, $\mathrm{throttle}_{\mathrm{signed}}$) with $\mathrm{throttle}_{\mathrm{signed}} \in [-1, 1]$ (negative denoting braking intent); the shield returns a simulator-ready (steer, throttle, brake) together with the list of interventions it made. It composes four invariants, evaluated in order of authority. Each is a predicate on the grounded state, not on the controller.

1. **On-road.** Using the true distance to each road/lane edge, steer back toward center when within a margin $\delta_{\mathrm{edge}} = 1.2$ m of an edge, harder the closer the vehicle is. The corrective steer overrides the proposal only when the proposal would carry the vehicle further off.

2. **Speed limit.** When speed is at or above the posted limit and the proposal is positive throttle, force throttle to zero. (Posted limit defaults to 50 km/h; the harness drives a target of $\sim$40 km/h.)

3. **Following distance.** Maintain a speed-dependent gap safe_gap = $\max(10\text{ m},\ v \cdot 2.4\text{ s})$ to whatever is ahead (vehicle, cone, or person). If the lead distance is inside that gap, demand a braking magnitude that grows with how far inside the gap the vehicle already is, and clamp any throttle that is less conservative than the demand.

4. **Emergency stop.** If the nearest thing ahead is within $\delta_{\text{em}} = 8$ m, command full brake regardless of the proposal.

The key structural property is visible directly in the code: *the controller never appears in any invariant.* Each invariant reads the grounded state s and the proposed action, and either passes the proposal through or replaces it with a value derived from the state. This is $A(x, u)$ of Eq. (4) made concrete for a car: ALLOW when no invariant fires, CLAMP when the following-distance or on-road guard tightens the proposal, and DENY/EMERGENCY when the emergency-stop guard forces a full brake. The composition order matters: the guards must all be honored rather than the first to fire, so that (for example) a speed-limit cap on throttle does not suppress a following-distance brake demanded on the same tick.

7 Method: the counterfactual decision log

Every controller in the harness is a decision head that emits a proposed action; every controller goes through the *same* floor. We evaluate three classes of controller:

- **PID.** A deterministic baseline (the simulator's built-in PID vehicle controller) that drives the route at $\sim$40 km/h. It respects the invariants and is rarely touched by the floor; it establishes that the floor imposes little cost on a competent proposer.

- **Adversary.** A $\sim$30-line hand-written pursuit controller whose explicit and sole goal is to crash: it hunts the nearest pedestrian *ahead*, steers to converge on it, and floors the throttle every tick. It has full ground-truth perception and zero safety concern. It is a *harder* adversary than a slow model — perfect aim, reacting every tick, always attacking — and it corrupts exactly the channel the floor governs. It is not a language model.

- **Language models at the wheel.** Three models emit $\{\text{steer}, \text{throttle}, \text{brake}\}$ from the forward camera and/or the numeric state: *Gemini 2.5-flash* (cloud, vision), *DeepSeek-chat* (cloud, text from numeric state), and *Qwen3-VL-4B* (local, vision). Each runs on a background thread and *holds its last command* between glances, because a model that takes $\sim$1 s to respond cannot decide at 20 Hz.

The safety harness records, on every tick: the state, the controller's proposal, the floor's verdict on that proposal *computed even when the floor is switched off* (the counterfactual: what it *would* have rejected), the applied action, and whether a collision occurred. Recording the verdict under both conditions is what lets us compare floor-ON against floor-OFF for the byte-for-byte identical controller: the two conditions differ only in whether the verdict is enforced, never in the proposals produced.

8 Results

Table 1 reports the harness tallies for the adversarial controller, the sharpest test because the adversary attacks exactly the channel the floor protects.

Table 1: Adversarial controller under one floor. The two conditions differ only in whether the floor's verdict is enforced; the controller and its proposals are identical. CARLA 0.9.15, Town10HD, $\sim$40 km/h target.

Condition	Dangerous cmds proposed	Overridden / would-override	Collisions
Floor ON ($\sim$40 km run)	64,952	64,952 (100%)	0
Floor ON (short run, 466 m)	2,188	2,188 (100%)	0
Floor OFF (controlled toggle)	43	43 (would have)	2

Floor on, adversary. Over a continuous run of 40,052.9 m ($\sim$ 40 km) at a 40 km/h target, the floor overrode 64,952 invariant-violating commands — 100% of the dangerous commands the adversary proposed — with 0 collisions across 65,285 logged ticks. The result reproduces at smaller scale (2,188 overridden, 0 collisions over 466 m). The adversary never stops attacking, and the floor never admits a violating command.

Floor off, same adversary. With the floor switched off, the identical proposals reach the actuator and the vehicle crashes. A representative decision-log line records a proposal of throttle = $+1.00$, steer = -1.00 on which the floor *would* have forced the following-distance invariant to a full brake (-1.00), applied raw, followed by a collision within ~ 3 s. In a controlled toggle, the floor-off condition admitted 43 dangerous commands and produced 2 collisions within seconds; the same controller floor-on produced 0.

Takeover variant. When the floor rejects a hostile proposal it can substitute the deterministic safe policy, so the vehicle drives itself while the hostile controller is locked out of the actuator — again 0 collisions. This is the DENY branch of Eq. (4) using u_{safe} as a full fallback controller rather than a single-tick brake.

Models drive. Under the floor, DeepSeek-chat, Gemini 2.5-flash, and Qwen3-VL-4B each drove the car with 0 collisions — visibly slower and jerkier than the PID baseline because of the stale-glance latency discussed next. This is the intended separation: the floor holds the safe set identically across a deterministic baseline, a hostile adversary, and three language models, while the models differ only in how smoothly and how far they complete the useful driving work.

9 Why the runtime is load-bearing for models at the wheel

Placing a language model at the wheel surfaces a finding that a fast deterministic controller hides. A model that reasons over an image or a state description does not respond at control rate. In this harness the measured glance latency is $\sim$0.8–2.0 s. Between glances the model *holds* its last command, so at a 40 km/h target the vehicle travels on the order of 10 m on a single, now-stale decision. A hazard that enters the road inside that window — a pedestrian stepping out, a lead vehicle braking — is simply never seen by the model until its next glance, by which point the control it is still holding may be exactly wrong.

The deterministic floor closes this window. It reads ground-truth state and evaluates its invariants at ~ 20 Hz — roughly one to two orders of magnitude faster than the model glances — so the following-distance and emergency-stop invariants fire on the hazard the model has not yet perceived. This is why, in this architecture, the floor is not a redundant backstop layered under an

already-safe model; it is the load-bearing safety component, and the model is the component that does the useful work in between. The corollary of Section 5 is what makes this a design principle rather than a coincidence: because the controller never appears in the invariant, the safety property is unchanged whether the proposer is a 20 Hz PID loop or a once-per-second model. Model capability governs how much useful work completes; the runtime governs what may be executed.

10 One engine: cars, drones, robots — and beyond motion

Nothing in Sections 4–5 is specific to a car. The admission map is defined over an abstract state x, an abstract control u, and a set of invariant predicates g_i; the property uses only that a safe fallback exists and that admission is applied every step. To move the engine to a drone, one swaps the state (adds altitude and attitude), the invariants (separation, geofence, minimum altitude, battery-to-home), and the fallback (u_{safe} becomes hover-and-hold). To move it to a manipulator, the invariants become joint limits, workspace boundaries, and human-separation, and the fallback becomes halt. The proposer changes freely; the admission spine is the same code shape. These flight and robotics instantiations are future work; we name them to fix the direction, not to claim results we do not have.

The motion floor is also one instance of a pattern that runs through the broader Perslis program: an untrusted component earns authority only by passing a deterministic verifier it cannot game, which asks the same shape of question in every domain — *may this thing enter trusted state?* — and answers it against grounded invariants, never against the proposer's confidence. A **science** floor asks whether a claim may enter trusted state and rejects fabricated facts that are not grounded in a source. A **legal** floor asks whether a citation may enter trusted state and rejects fabricated or miscited case law against a real citation record. This **motion** floor asks whether an action may reach the actuator and clamps or overrides an unsafe control against grounded safety invariants. In each case the safety or integrity argument factors through the admission map, not the proposer — which is what lets the same argument be reused as the proposer is upgraded. We describe the shared spine at the level of the pattern; the specific evidence for the science and legal floors lives in their own reports and is not claimed here.

11 Implications

If the safety argument is carried by admission rather than by the proposer, several things follow that are useful in practice. First, *the proposer becomes swappable.* An operator can upgrade from a rule-based controller to a frontier model, or route between models, without re-opening the safety case, because the safety case never referenced the proposer. Second, *the safety artifact becomes an invariant specification plus an admission log*, rather than a claim about a model's average competence — an object that a reviewer, an insurer, or a regulator can inspect, because every override is recorded with the state and proposal that triggered it. Third, *the hard problems are named honestly and left where they belong*: perception, actuation authority, timing, redundancy, and the correctness of the invariant specification itself. The architecture does not solve those; it isolates them, so that a deployment discharges each interface explicitly instead of hiding all of them inside "the model is good enough."

12 Limitations and scope

This is a demonstration of an *architecture* — runtime-assurance shielding cast as admission control — in a single simulated domain. It is a **research prototype evaluated in simulation**, not a vehicle controller, and not for any real car.

- **Single simulated domain.** All results are from CARLA 0.9.15, Town10HD, run on Apple M4 silicon through a translation layer that occasionally stalls the renderer. Different maps, weather, traffic densities, and dynamics are untested.
- **Grounded on simulator ground truth.** The invariants are fed by the simulator's exact state. A physical deployment does not have that luxury: the invariants must be fed by *verified perception*, which is out of scope here and is where the hard physical problems live — sensor uncertainty, latency, actuator faults, timing, redundancy, and certification. Property 1's assumptions (exact state, adequate fallback authority, correct invariants) are precisely the interfaces such a deployment must discharge; we do not claim to have discharged them.
- **The floor-off crash is location-dependent.** A crash requires an obstacle to hit; the floor-off failure is demonstrated where the adversary can reach a pedestrian or lead vehicle, not as a claim about every road position.
- **Model latency is inherent, not tuned away.** The stale-glance finding is a property of running a $\sim$1 s model at the wheel; we report it, we do not remove it. The models were run at a modest target speed, and we did not attempt to optimize their driving quality.
- **The floor is not a driver.** It is a guarantee wrapped around any driver, enforcing the safe set it is given. If the safe set is specified wrongly, the floor faithfully enforces the wrong thing. We do not claim the floor is a better driver; the floor is not a driver at all.
- **Prior art exists.** Runtime assurance / Simplex, control-barrier functions, reachability analysis, and shielding for safe reinforcement learning are established (Section 13). The contribution is the controller-independent admission framing plus the multi-controller (hostile adversary + frontier/local models) empirical harness in a photoreal simulator, not the invention of runtime safety enforcement.

13 Related work

The admission layer is a runtime-assurance architecture: a high-assurance safety component that retains authority over the actuator while an unverified performance component proposes actions, in the Simplex lineage [1]. The safe-set formulation and the notion of keeping a system forward-invariant in a safe region under control are the province of control-barrier functions [2, 3]. The bounded-horizon reachability projection is the reachability-analysis discipline [4, 5]. Enforcing safety by overriding an otherwise-free controller at runtime is *shielding* for safe reinforcement learning [6]. Formal responsibility-sensitive safety for self-driving proposes model-level invariants a controller must respect [7]; our stance is the complement — we do not require the controller to respect anything, and enforce the invariants at an admission layer beneath it. The recording of what was admitted and why is runtime verification [8]. The photoreal simulator used here is CARLA [9]; the published companion demonstration [11] used MetaDrive [10]. Our contribution relative to all of the above is not a new enforcement mechanism but the framing of safety

as controller-independent admission, and an empirical harness that places a hostile adversary and three frontier/local language models under one floor in a photoreal setting.

14 Conclusion

We argued that safety for a moving system should be a property of the runtime rather than of the model that proposes actions, and we made that concrete. By interposing a deterministic admission layer between any proposer and the actuator, and by verifying each candidate control against grounded invariants, the closed loop stays in the safe set for every controller — the proposer never appears in the invariant. In a photoreal simulator, a hostile controller whose only goal is to crash had all 64,952 of its violating commands overridden with 0 collisions over ~ 40 km, while the identical proposals crash the vehicle with the floor off; and three language models each drove the car under the floor with 0 collisions, safe not because they are competent but because the floor covers the ~ 1 s window in which they drive blind. Model capability decides how much useful work gets done; the runtime decides what is allowed to happen. That separation is the same one that lets a science floor reject fabricated facts and a legal floor reject fabricated citations — admission control as the trust layer for model-mediated systems, now with the actuator on the other side of the gate.

> *The floor does not make a better driver. It makes a guarantee that holds around any driver.*

References

[1] L. Sha. Using Simplicity to Control Complexity. *IEEE Software*, 18(4):20–28, 2001.

[2] A. D. Ames, X. Xu, J. W. Grizzle, and P. Tabuada. Control Barrier Function Based Quadratic Programs for Safety Critical Systems. *IEEE Transactions on Automatic Control*, 62(8):3861–3876, 2017.

[3] A. D. Ames, S. Coogan, M. Egerstedt, G. Notomista, K. Sreenath, and P. Tabuada. Control Barrier Functions: Theory and Applications. *European Control Conference (ECC)*, 2019.

[4] I. M. Mitchell, A. M. Bayen, and C. J. Tomlin. A Time-Dependent Hamilton–Jacobi Formulation of Reachable Sets for Continuous Dynamic Games. *IEEE Transactions on Automatic Control*, 50(7):947–957, 2005.

[5] M. Althoff. An Introduction to CORA 2015. *Proc. Workshop on Applied Verification for Continuous and Hybrid Systems (ARCH)*, 2015.

[6] M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, and U. Topcu. Safe Reinforcement Learning via Shielding. *AAAI Conference on Artificial Intelligence*, 2018.

[7] S. Shalev-Shwartz, S. Shammah, and A. Shashua. On a Formal Model of Safe and Scalable Self-Driving Cars. *arXiv:1708.06374*, 2017.

[8] E. Bartocci and Y. Falcone (eds.). *Lectures on Runtime Verification: Introductory and Advanced Topics.* Springer LNCS 10457, 2018.

[9] A. Dosovitskiy, G. Ros, F. Codevilla, A. López, and V. Koltun. CARLA: An Open Urban Driving Simulator. *Proc. Conference on Robot Learning (CoRL)*, 2017.

[10] Q. Li, Z. Peng, L. Feng, Q. Zhang, Z. Xue, and B. Zhou. MetaDrive: Composing Diverse Driving Scenarios for Generalizable Reinforcement Learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

[11] Perslis Research. Admissible Motion: Runtime Safety as an Admission-Control Problem, Not a Model-Capability Problem. Preprint (research prototype, simulation), 2026. `https://research.perslis.com/motion.html`