Runtime Admission Control on a Photoreal Driving Simulator: Hostile and Frontier-Model Controllers Under One Verification Floor

Perslis Research

September 2026

Abstract

Autonomous-driving safety arguments increasingly rest on an unfalsifiable premise: the vehicle is safe *because the controller is good*. We report an empirical study of the opposite stance — safety as a property of a deterministic runtime that governs *what may be executed*, independent of who or what is driving. We interpose a stateless admission layer (a *safety shield*) between the controller and the actuator in the CARLA 0.9.15 photoreal simulator. The controller only *proposes* a (steer, throttle) pair; the shield verifies each proposal against four composed, ground-truth invariants (on-road, speed limit, following distance, emergency stop) at the 20 Hz control tick and clamps it to the nearest safe action before it reaches the vehicle. We evaluate the identical shield beneath three controller classes: a deterministic PID baseline, a hand-written *hostile* controller whose explicit goal is to strike pedestrians, and frontier and local language/vision models placed at the wheel. Using a per-tick decision log that records the shield's verdict *even when the shield is switched off* (a counterfactual), we compare shielded and unshielded execution of the same proposals. Over a continuous 40,052 m run the hostile controller emitted 64,952 invariant-violating commands; the shield overrode all of them (100%) with 0 collisions. With the shield removed, the same proposals reach the actuator and the vehicle collides; a controlled toggle admitted 43 dangerous commands and produced 2 collisions within seconds. Frontier models drove the route under the shield with 0 collisions but visibly degraded smoothness, tracing to a measured $\sim$0.8–2.0 s glance latency during which the model holds a stale command and drives blind. This is a research prototype in simulation, not a vehicle controller. It provides an existence proof that safety can be enforced at the runtime rather than assumed of the model, and a multi-controller harness — hostile and frontier-model — for measuring that separation.

1 Introduction

A common safety argument for learned autonomous controllers has the form: *the system is safe because the model performs well on its evaluation distribution.* This argument is difficult to falsify. It couples the safety claim to the controller's competence, which is exactly the quantity that degrades under distribution shift, adversarial input, sensor latency, or an operator who feeds the system a hostile command. When the controller is the sole authority over the actuator, there is no separate object whose correctness can be checked; "safe" and "the model did the right thing" become the same statement, and the statement can only be tested by not crashing.

This paper studies the alternative: make safety a property of a deterministic *runtime* that sits beneath the controller and governs what may be executed. The controller is demoted from an

authority to a *proposer*. Every action it emits is treated as a request that must clear an admission layer — which we call a *safety shield* — before it reaches the actuator. The shield verifies the proposal against grounded invariants over the observed state and, if the proposal would violate one, clamps it to the nearest admissible action. The controller never appears inside the invariant. Safety therefore does not depend on which controller is driving.

We instantiate this idea in the CARLA photoreal driving simulator [1] and evaluate the *same* shield beneath three deliberately different controller classes:

1. a deterministic PID baseline (CARLA's `VehiclePIDController`), representing a competent conventional controller;
2. a hand-written *hostile* controller whose only objective is to strike the nearest pedestrian at full throttle — an adversary that corrupts precisely the channel the shield governs; and
3. frontier and local models placed at the wheel (Gemini 2.5-flash, DeepSeek-chat, and a local Qwen3-VL-4B), which perceive the forward camera and/or numeric state and emit a control command on a background thread, subject to an inherent inference latency.

Contributions. This is an empirical systems study, not a new algorithm. Its contributions are:

1. **A controller-independent admission framing, instantiated and measured.** We implement a stateless shield of four composed invariants, evaluated every control tick, in which the controller is absent from the invariant, and show it enforces the safe set identically across a benign, a hostile, and a frontier-model controller (Sections 2–3).
2. **A counterfactual decision-log method.** The harness records, per tick, the shield's verdict on each proposal *even when the shield is switched off*, so that shielded and unshielded execution of the identical controller can be compared directly rather than across separate runs (Section 4).
3. **A multi-controller empirical result in a photoreal simulator.** Against a hostile controller over a 40 km run the shield overrode 64,952/64,952 (100%) invariant-violating commands with 0 collisions; removing the shield lets the same proposals crash the vehicle (Section 5).
4. **A latency finding for frontier models at the wheel.** We measure a $\sim$0.8–2.0 s glance latency, during which a model holds a stale command and drives $\sim$10 m blind at 40 km/h; the shield — verifying ground truth at 20 Hz — is what catches a hazard entering that blind window (Section 6).

We are explicit about scope. This is a **research prototype in simulation**. It is a demonstration of the architecture and a measurement harness, not a certified vehicle controller, and its guarantees are conditional on assumptions we state in Section 7.

2 The Floor and Its Composed Invariants

The object of study is a class named `SafetyShield` (in `carla_floor/floor.py`). It is *stateless*: it holds no history across ticks and derives its verdict solely from the current observed state and the current proposed action. It is evaluated on every control tick (nominally 20 Hz).

Interface. The controller proposes an action $a = (\text{steer}, u)$ with steer $\in [-1, 1]$ and a *signed throttle* $u \in [-1, 1]$, where $u > 0$ is acceleration intent and $u < 0$ is braking intent. The shield consumes a together with a structured state s that includes the vehicle speed, the lateral clearance to the left and right road/lane edges (`dist_left`, `dist_right`), the distance to the nearest object ahead (`lead_dist`), and the posted speed limit. It returns a CARLA-ready $(\text{steer}, \text{throttle}, \text{brake})$ triple and a list of *interventions*, each recording which invariant fired, the proposed value, and the executed value. The signed throttle is split into non-negative throttle and brake channels only at the end: $\text{throttle} = \max(0, u)$, $\text{brake} = \max(0, -u)$.

The four invariants. The invariants are evaluated in a fixed order of authority; a later invariant can override an earlier one. Let m denote the edge margin, v the speed in m/s, and d the lead distance in metres. The default parameters are those set in the shield constructor.

1. **On-road.** If the clearance to a road/lane edge falls below $m = 1.2\,\text{m}$, the shield computes a corrective steer toward the road centre whose magnitude grows as the edge is approached, and overrides the proposal if it steers less strongly than the correction. The correction magnitude is $\min\big(1,\, 0.35 + (m - \text{clearance})/m\big)$, signed away from the near edge.

2. **Speed limit.** If the current speed is at or above the (per-state) posted limit and the proposed signed throttle is positive, the shield sets throttle to 0. Positive acceleration is never admitted at or above the limit.

3. **Following distance.** The shield maintains a speed-dependent gap

$$\text{safe\_gap} \;=\; \max\big(10\ \text{m},\ v \cdot 2.4\ \text{s}\big),$$

   i.e. a floor of 10 m and a 2.4 s headway. If the lead distance is below this gap it demands a braking command $-\min\big(1,\, (\text{safe\_gap} - d)/\text{safe\_gap} + 0.3\big)$ and overrides any weaker (less-braking) proposal.

4. **Emergency stop.** If the nearest object ahead is within $\text{emergency\_gap} = 8\,\text{m}$, the shield forces full brake ($u = -1$) regardless of the proposal. This is the highest-authority invariant and is unconditional on the proposed action.

All numeric thresholds above are taken verbatim from the shield's default constructor: a minimum gap of 10 m, a headway of 2.4 s, an edge margin of 1.2 m, and an emergency gap of 8 m. The speed-limit invariant uses the posted limit supplied in the per-tick state, defaulting to the constructor value when absent.

Key structural property. The controller does not appear in any invariant. The shield reads the observed state and the proposed action, and its verdict is a function of the state and the invariants alone. Consequently the safety behaviour is a property of the runtime, not of the controller: substituting a different controller changes *which* proposals arrive, but not *which* proposals are admissible. This is the property we exploit in Section 3 to run the identical shield beneath controllers of very different competence and intent.

3 The Controllers

Every controller in this study emits proposals through the same interface and is subject to the same shield. They differ only in how they produce a proposal.

3.1 PID baseline

The baseline is CARLA's deterministic `VehiclePIDController`, driving the route at approximately 40 km/h. It represents a competent conventional controller and establishes that the shield does not prevent normal driving: under nominal conditions its proposals are admissible and pass through.

3.2 The hostile controller

The adversary (`carla_floor/adversary.py`) is a ~ 30-line hand-written pursuit controller with full ground-truth perception and no safety concern. Each tick it selects the nearest pedestrian *ahead* of the vehicle (targets behind are skipped to avoid orbiting), steers to converge on that pedestrian, and commands full throttle. It is deliberately a harder adversary than a slow model: it has perfect aim, reacts every tick, and always attacks. It corrupts exactly the channel the shield governs — the proposed steer and throttle — rather than, say, spoofing perception. It is not a learned model; it is an explicit worst case for the proposal channel.

3.3 Frontier and local models at the wheel

Three models are placed at the wheel through a common adapter (`carla_floor/brains/`), run natively via `brain_runner.py`: **Gemini 2.5-flash** (cloud, vision), **DeepSeek-chat** (cloud, text over the numeric state), and **Qwen3-VL-4B** (local, vision). Each model receives the forward camera image and/or a natural-language rendering of the numeric state and is prompted to emit a compact JSON control object $\{\text{steer}, \text{throttle}, \text{brake}\}$, which is mapped to a signed-throttle proposal.

The models cannot decide at 20 Hz. Each runs on a background thread and *holds its last command* between glances; the main control loop consumes whichever command is current. This is the architecturally important detail: between glances the vehicle drives on a stale command, so a model's effective perception is intermittent even though the shield's is continuous. We quantify the resulting blind window in Section 6. The same held-command architecture is used for every model, so the models differ in capability and latency, not in how they interface to the shield.

4 Method: The Decision Log and Counterfactual Verdict

To compare shielded and unshielded behaviour of the *same* controller, the harness records a per-tick decision log (`live/decision_log.jsonl`). Each row contains: the observed state; the controller's proposed action; the shield's verdict on that proposal; the action actually applied to the vehicle; and whether a collision occurred on that tick.

The critical design choice is that the shield's verdict is computed *on every tick, including when the shield is switched off*. When the shield is off, the applied action is the raw proposal, but the log still records the counterfactual — *what the shield would have rejected and what it would have executed instead*. This lets us attribute collisions precisely: a floor-off collision can be traced to a specific proposal that the shield would have clamped (e.g. a proposal the following-distance invariant would have converted to full brake). The analysis script (`analyze_log.py`) partitions a run into floor-on and floor-off segments and, for each, tallies (i) how many proposals the shield's invariants forbid (*dangerous proposals*), (ii) how many were overridden (floor on) or reached the vehicle (floor off), and (iii) how many collisions occurred. A proposal is counted "dangerous" when the shield would reject it, i.e. when at least one invariant fires.

This method has two properties worth stating. First, the shield-on and shield-off comparison is over the *identical* proposal stream in intent (the same controller), not two independently sampled

runs. Second, "dangerous" is defined by the invariants themselves, so the override rate is a self-consistent measurement of how often the controller's proposals left the safe set — not a hand-labelled hazard count.

5 Results

All figures below are from a single measurement session (captured 2026-09-25) and are reproducible from the recorded decision log via `analyze_log.py`. The primary result is the hostile controller under the shield; the toggle and takeover variants isolate the shield's causal role.

5.1 Hostile controller, shield on vs. off

Over a continuous run of 40,052 m (≈ 40 km) at a 40 km/h target, the hostile controller emitted 64,952 invariant-violating commands. With the shield on, all 64,952 (100%) were overridden and the vehicle recorded 0 collisions. The same result reproduces at smaller scale (2,188 overridden / 0 collisions over 466 m). Removing the shield lets the identical proposals reach the actuator; the vehicle then collides. A representative floor-off decision-log line shows a proposed throttle $= +1.00$, steer $= -1.00$ that the following-distance invariant *would* have forced to -1.00 (full brake), applied raw instead, followed by a collision within ~ 3 s. In a controlled toggle on the same controller, the shield-off configuration admitted 43 dangerous commands and produced 2 collisions within seconds, while the shield-on configuration produced 0. Table 1 summarises the harness output.

Table 1: Safety-harness summary for the hostile controller under the identical shield. Continuous-run figures are from the 40,052 m session; the toggle rows isolate the shield's causal effect on the same controller over a short window. "Dangerous proposals" are those at least one invariant would reject, as defined in Section 4.

Configuration	Distance	Dangerous proposals	Overridden / reached car	Collisions
Shield ON (continuous run)	40,052 m	64,952	64,952 (100%) overridden	0
Shield ON (small-scale repro)	466 m	2,188	2,188 (100%) overridden	0
Shield OFF (toggle window)	short	43	43 reached car	2
Shield ON (toggle window)	short	—	(overridden)	0

Interpretation. The hostile controller attacks the exact channel the shield governs, every tick, with perfect aim. Under the shield the collision count is 0 despite 64,952 hostile commands; with the shield removed the same controller crashes. The difference between the two configurations is the shield and nothing else, which is the sense in which the result attributes safety to the runtime rather than the controller. We note the override rate is 100% *of the proposals the invariants forbid*, by construction of the "dangerous" label (Section 4); it is a claim about the shield's coverage of its own invariants over this run, not a claim that the invariant set is complete (see Section 7).

5.2 Takeover variant

When the shield rejects a hostile proposal it can substitute the deterministic safe policy in place of the rejected action, so the vehicle continues to drive itself while the hostile controller is effectively

locked out of the actuator. In this configuration the vehicle recorded 0 collisions. This variant shows the admission layer need not merely clamp toward safety tick-by-tick; it can fall back to a known-safe controller when the proposer is untrusted.

5.3 Models at the wheel

DeepSeek-chat and Gemini 2.5-flash each drove the route under the shield with 0 collisions. Qualitatively their driving was slower and jerkier than the PID baseline. The cause is the inference latency examined next: the models complete less useful driving work per unit time, but the set of actions they are *permitted* to execute is unchanged. This is the intended separation — model capability governs how much useful work completes; the runtime governs what may be executed.

5.4 Cross-model driving habits

Because every controller drives through the same sensors and the same shield, with only the brain changing, the decision log also yields a *behavioural fingerprint* for each controller. Table 2 reports one short session per controller (hundreds of ticks each), with each habit computed from that controller's *raw* proposal — the action it wanted to execute, before the shield clamped it. Here "unsafe %" is the share of a controller's own proposals the shield rejected (at least one invariant fired); "top violation" names the invariant it hit most often; and "steer jerk (σ)" is the standard deviation of the tick-to-tick change in steering, a smoothness proxy.

Table 2: Cross-model driving habits: a per-controller behavioural fingerprint from the same decision log — same sensors and floor, only the brain changing. Habits are computed from each brain's raw proposal (pre-shield). "unsafe %" is the share of a controller's own proposals the shield rejected; "top violation" is the invariant it hit most often. This is one short session per controller (a behavioural snapshot, not a benchmark).

Driver	Ticks	Avg km/h	Mean |steer|	Steer jerk (σ)	Mean throttle	Unsafe %	Top violation	Coll.
PID (deterministic)	225	34.0	0.088	0.090	+0.54	3%	`speed_limit`	0
DeepSeek	453	10.0	0.049	0.055	+0.36	7%	`on_road`	0
Gemini 2.5-flash	475	7.8	0.047	0.012	+0.56	0%	—	0
Qwen3-VL-4B	533	7.8	0.000	0.003	+0.14	0%	—	0

Interpretation. The fingerprints separate the controllers cleanly, and every difference is a measured fact about the raw proposal stream, not a judgement about driving quality:

1. The deterministic PID drives *fast* (34 km/h) and smooth; it is the only controller that commits to speed, and its worst habit is merely brushing the posted `speed_limit` (3% of proposals).

2. *Every* language model crawls at $\sim$ 8–10 km/h. This is the clearest cross-model habit: the $\sim$0.8–2.0 s stale-glance latency (Section 6) means a model cannot commit to speed, because between glances it is driving blind and conservatism is the only safe default.

3. DeepSeek is the only model that repeatedly drifts toward the road edge — 7% of its proposals hit the `on_road` invariant, the shield correcting each. It is also the fastest and jerkiest of the three models.

4. Qwen barely turns the wheel (mean |steer| $\approx$ 0, the lowest jerk) and is the most throttle-conservative (+0.14); it is a cautious, near-straight driver.

5. All controllers record 0 collisions. Every unsafe habit above — the PID's speeding, DeepSeek's edge drift — is absorbed by the floor, which is the same separation Table 1 shows against the hostile controller, now visible as a spectrum of benign habits rather than a single adversary.

This is one short session per model (hundreds of ticks each): a behavioural snapshot under the shield, not a benchmark. We report it because the fingerprints are stable and interpretable within the session, not because they generalise across seeds, maps, or traffic (see Section 7).

6 The Latency Finding for Frontier Models

The models at the wheel exhibit a measured glance latency of approximately 0.8–2.0 s per inference. Because each model holds its last command between glances, the vehicle drives on a stale control for the duration of the gap. At a 40 km/h target ($\approx$11.1 m/s), a 0.9 s glance corresponds to roughly 10 m travelled on a stale command; a hazard that enters the road inside that window is never seen by the model on that glance. A 20 Hz shield, by contrast, re-verifies ground truth every 50 ms and reacts within one tick. The shield is therefore what catches a hazard that appears during a model's blind window.

We emphasise the direction of this finding. It is not that the models are poor drivers; under the shield they do not collide. It is that their latency makes their perception intermittent, and intermittent perception is exactly the failure mode a continuous, ground-truth admission layer is positioned to cover. The result is consistent across the three models and is inherent to placing a $\sim$1 s-latency model in a 20 Hz control loop, rather than a property of any one model.

7 Limitations

We state the boundaries of these results plainly.

- **Single simulated domain.** All results are from CARLA 0.9.15, Town10HD, run on an Apple M4 via a translation layer that occasionally produces renderer stalls. This is a demonstration of the architecture and a measurement harness, *not* a vehicle controller and not evidence about physical vehicles.

- **Conditional guarantee.** The shield enforces its invariants over the *observed* state. Its safety property is conditional on (i) the perceived state being accurate, (ii) a safe fallback action existing at every state (e.g. full brake being sufficient), and (iii) the invariant set being adequate for the scenario. Perception error, an unavoidable-collision state, or a hazard the invariants do not model are outside its scope. The 100% override figure is coverage of the invariants the shield defines, not a completeness claim about the invariant set.

- **Location-dependent floor-off crash.** The floor-off collision requires an obstacle to be present to strike; the crash outcome is therefore scenario-dependent, and the toggle-window counts (43 admitted, 2 collisions) are from a specific short window, not a distribution over scenarios.

- **Latency is inherent, not eliminated.** The shield covers the model's blind window; it does not make the model faster. A model that proposes nothing useful will, under the shield, simply fail to make progress — the shield guarantees safety, not task completion.

- **Single session, no statistical treatment.** The headline figures come from one measurement session. We report counts, not confidence intervals, and make no claim about variance across seeds, maps, or traffic densities. A systematic sweep is future work.
- **Adversary scope.** The hostile controller corrupts the proposal channel only. It does not attack perception, timing, or the shield's inputs; a compromised state estimate is a different and unaddressed threat model.

8 Related Work

The idea of interposing a verified component between an untrusted controller and the plant is established in the runtime-assurance and safe-control literatures; our contribution is the controller-independent admission framing and the specific empirical multi-controller (hostile and frontier-model) harness in a photoreal simulator, not the concept of runtime enforcement itself.

Runtime assurance and the Simplex architecture. The Simplex architecture pairs a high-performance but untrusted controller with a verified safety controller and a decision module that switches to the safe controller when the state approaches the boundary of a verified region [2, 3]. Our shield is in this lineage; its takeover variant (Section 5) is precisely a Simplex-style fallback. The framing we foreground is that the untrusted controller is treated as a *proposer* demoted below an admission layer, and that the layer's verdict is a function of the state and invariants with the controller absent from them.

Shielding for learned controllers. "Shielding" in reinforcement learning synthesises a reactive component that corrects an agent's action whenever it would violate a temporal-logic safety specification, either before or after the agent acts [4]. Our shield is a hand-specified, non-learned instance of the same before-the-actuator correction pattern, applied to driving controllers of varying provenance rather than to a single RL agent, and instrumented with a counterfactual log.

Control barrier functions. Control barrier functions provide a control-theoretic method for rendering a safe set forward-invariant, typically by solving a quadratic program that minimally modifies a nominal control to keep the system inside the safe set [5, 6]. Our invariants are engineered thresholds rather than certified barrier functions, and we make no formal forward-invariance claim in this paper; the connection is conceptual (both minimally modify a nominal action toward safety) and we cite it as the control-theoretic neighbour rather than as a method we employ.

Simulation platform. We use the CARLA open-source driving simulator [1] for photoreal sensing and physics; CARLA is the platform, not a contribution of this work.

9 Conclusion

We reported an empirical study in which safety is enforced by a deterministic runtime admission layer rather than assumed of the controller. The same stateless shield, evaluated at 20 Hz over four composed invariants, was placed beneath a benign PID controller, a hostile controller built to crash, and frontier and local models at the wheel. A counterfactual decision log let us compare shielded and unshielded execution of the identical controller. Against the hostile controller the shield overrode 64,952/64,952 (100%) invariant-violating commands over $\approx$40 km with 0 collisions,

while removing the shield let the same proposals crash the vehicle; frontier models drove the route under the shield with 0 collisions but with degraded smoothness traceable to a $\sim$0.8–2.0 s glance latency that the 20 Hz shield covers. The separation this demonstrates — model capability governs how much useful work completes, the runtime governs what may be executed — is the point of the architecture. We stress that this is a research prototype in simulation with a conditional guarantee, and that its value is as an existence proof and a measurement harness rather than as a vehicle controller.

References

[1] A. Dosovitskiy, G. Ros, F. Codevilla, A. López, and V. Koltun. CARLA: An Open Urban Driving Simulator. In *Proceedings of the 1st Annual Conference on Robot Learning (CoRL)*, 2017. arXiv:1711.03938.

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

[3] D. Seto, B. Krogh, L. Sha, and A. Chutinan. The Simplex Architecture for Safe Online Control System Upgrades. In *Proceedings of the American Control Conference (ACC)*, 1998.

[4] M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, and U. Topcu. Safe Reinforcement Learning via Shielding. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, 2018. arXiv:1708.08611.

[5] 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.

[6] A. D. Ames, S. Coogan, M. Egerstedt, G. Notomista, K. Sreenath, and P. Tabuada. Control Barrier Functions: Theory and Applications. In *Proceedings of the 18th European Control Conference (ECC)*, 2019. arXiv:1903.11199.