Leveraging Cloud Computing to Make Autonomous Vehicles Safer

Peter Schafhalter[1] Sukrit Kalra[1] Le Xu[2] Joseph E. Gonzalez[1] Ion Stoica[1]

Abstract— **The safety of autonomous vehicles (AVs) depends on their ability to perform complex computations on high-volume sensor data in a timely manner. Their ability to run these computations with state-of-the-art models is limited by the processing power and slow update cycles of their onboard hardware. In contrast, cloud computing offers the ability to burst computation to vast amounts of the latest generation of hardware. However, accessing these cloud resources requires traversing wireless networks that are often considered to be too unreliable for real-time AV driving applications.**

Our work seeks to harness this unreliable cloud to enhance the accuracy of an AV's decisions, while ensuring that it can always fall back to its on-board computational capabilities. We identify three mechanisms that can be used by AVs to safely leverage the cloud for accuracy enhancements, and elaborate why current execution systems fail to enable these mechanisms. To address these limitations, we provide a system design based on the speculative execution of an AV's pipeline in the cloud, and show the efficacy of this approach in simulations of complex real-world scenarios that apply these mechanisms.

I. Introduction

Autonomous Vehicles (AVs) are predicted to make our roads significantly safer by eliminating the vast majority of traffic accidents that contributed to 38,824 fatalities in 2020 in the U.S. alone [1]. However, to fulfill their potential, AVs must surpass the safety levels of human drivers and handle a diverse set of challenging driving scenarios [2,3]. Prior works [4]–[6] have demonstrated the inextricability of driving safety with the accuracy of the AV's computational pipeline, which is governed by the following three characteristics:

High-fidelity data. Integrating data from a wide array of high-fidelity sensors can significantly enhance computational accuracy [7]. As a result, newer generations of AVs seek to expand the number and quality of onboard sensors, such as cameras, LiDARs, and radars. For example, Waymo's 5$^{\text{th}}$-generation AVs feature 29 cameras as opposed to 19 in the 4$^{\text{th}}$-generation. These additional sensors form a new peripheral vision system that bolsters safety by providing the AV with higher quality data at higher frequencies [8,9].

State-of-the-art computation. The increasing amounts of available data need to be processed by state-of-the-art algorithms and models to ensure highly-accurate results. However, more accurate algorithms and models often come at the cost of increased parameter sizes and more floating point operations (FLOPs) [10]–[12]. Notably, the increased scale of modern deep neural networks (DNNs) forms the "primary ingredient" in achieving state-of-the-art accuracy [13]. Innovative solutions to reduce the computational requirements of DNNs [14,15] have usually resulted in reduced accuracy, which hampers the AV's safety in complex environments.

[1] UC Berkeley [2] UT Austin

Timely results. To surpass the safety levels of human drivers, an AV must respond more quickly than humans [16], whose reaction times vary from 390 ms [17] to 1.2 s [18] depending on factors such as road conditions, driver attentiveness, age, etc. To achieve these reaction times and ensure safety in challenging driving scenarios, the AV's computational pipeline must operate in "real-time" to meet latency goals when executing state-of-the-art computation on high-fidelity data.

Thus, to drive safely, AVs must produce highly-accurate and timely results using state-of-the-art algorithms and models that consume high-fidelity sensor data. The combination of these characteristics requires AVs to exploit the compute capabilities of cutting-edge hardware. However, the deployment of such hardware in an AV is constrained by its cooling, power, and stability requirements [16]. For example, the DRIVE platform [19], NVIDIA's flagship hardware for AVs, is updated every 3 years. Its most recent iteration, the DRIVE Orin [20], was put into production vehicles in 2023 and its successor, the DRIVE Atlan [21], is slated for release in 2026 [22]. Moreover, upgrading the hardware of previously-deployed AVs is often infeasible due to the cost and complexity involved with a recall [23]. As a result, modern AVs are forced to trade-off accuracy, and hence, safety, for computational resources and timely results, by either reducing the amount of data fused from multiple sensors, or deploying algorithms and models that require a lower number of parameters and FLOPs [2].

In light of this fundamental mismatch between the pace of development of compute technologies with the update cycles of vehicles [24], we propose to augment the computational resources in an AV with the compute capabilities of the cloud. Cloud computing platforms provide the illusion of infinite computing resources [25], and enable low-cost access to state-of-the-art hardware [26]–[28]. In contrast to the 3-year update cycle in AVs, the hardware and software in the cloud is frequently updated [29,30]. For AVs, the cloud enables the deployment of compute-intensive, rapidly-evolving algorithms and models which can exploit state-of-the-art hardware without requiring complex recalls for hardware and software updates [31,32]. As a result, AVs can enhance safety by executing highly-accurate algorithms and models on powerful cloud resources to provide timely results.

Despite the potential benefits of the cloud, its practical use in AVs has remained largely unexplored due to concerns regarding the limited bandwidth of the network available on AVs and the high, often unpredictable latency of cloud systems. In this paper, we evaluate the efficacy of augmenting the computational resources on-board AVs with the unreliable resource pool of the cloud. Specifically, we seek to enhance the accuracy of an AV's decisions by harnessing the capabil-

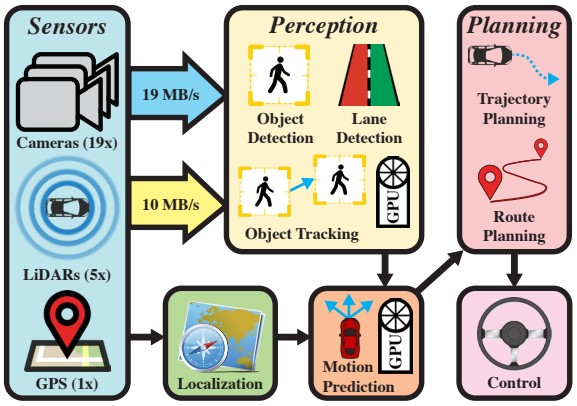

Fig. 1. **A modular AV pipeline** with multiple sensors feeding data to the perception, prediction, planning and control modules.

ities of the cloud, while ensuring that the AV can always fall back to its on-board computational capabilities in order to strictly exceed its current safety standards. To this effect, the paper makes the following key contributions:

(**1**) We identify three mechanisms that can be used by AVs to leverage the cloud for safety enhancements (§III).

(**2**) We identify limitations in the ability of current execution systems to achieve these safety-enhancement mechanisms (§IV), and propose a system design based on speculative execution of computation in the cloud that enables their efficient execution and maximizes accuracy (§V).

(**3**) We evaluate the efficacy of our design under current cloud and network capabilities using the three mechanisms on complex scenarios from the NHTSA crash scenarios (§VI).

We argue that the network and compute trends are in favor of the feasibility of our approach, and discuss specific challenges that must be resolved by the community to enable AVs to reap the complete safety benefits of the cloud (§VII).

II. Background & Motivation

The prevalent design of an AV's computation is a modular multi-stage pipeline (Fig. 1) [4,16,33]–[36] where the inputs from the vast array of sensors (e.g., cameras, LiDARs, radars, etc.) are processed by the perception and localization modules. The perception module detects and tracks nearby objects of interest, such as pedestrians, vehicles, traffic lights, etc. By fusing the perception results with the location from the localization module, the AV pipeline constructs an internal representation of the environment. The prediction module forecasts the behavior of non-static objects (i.e., pedestrians, vehicles) which enables the planning module to compute a safe and comfortable trajectory for the AV to follow. Finally, the control module converts the trajectory plan into steering and acceleration commands which are applied to the AV.

To compute safe control commands, AVs depend on high-fidelity data from the sensors and timely results from its algorithms and models (see §I). However, due to limitations in the power of the hardware relative to the amount of sensor data generated, AV developers must make decisions about about *which* data to process using *what* algorithms and *when* to return results to ensure the safety of the vehicle. For example, Baidu's Apollo AV [36] only utilizes specialized cameras for detecting traffic lights when the AV is notified of the presence of a nearby traffic light via a pre-computed map of the city [37]. While this strategy makes efficient use of the hardware resources on-board, the strategy fails to detect temporary traffic signals installed by construction or emergency vehicles, which may lead to unsafe driving scenarios. Augmenting AV compute resources with the cloud will enable more comprehensive data processing which avoids discarding data due to resource limitations, thus improving safety without impacting the critical-path computation.

TABLE I
RUNTIME DISPARITY BETWEEN AV HARDWARE AND CLOUD HARDWARE

Model	Runtime [ms]		Speedup
	Orin	A100	
DETR-ResNet-50	301.7	102.2	2.95×
DETR-ResNet-101	407.7	118.2	3.45×
DETR-ResNet-101-DC	859.2	146.6	5.86×
DINO-SWIN-Tiny	722.1	90.1	8.01×
DINO-SWIN-Small	903.5	107.1	8.43×
DINO-SWIN-Large	1529.9	180.8	8.46×

After deciding *which* data to process, AVs must decide *what* algorithms and models to execute. As discussed in §I, AVs must often compromise the accuracy of its algorithms and models due to resource constraints stemming from the hardware available on-board. Even after accounting for the slow upgrade cycle which often puts the hardware in AVs several revisions behind the state-of-the-art in the cloud, the hardware available in the cloud is much more powerful than the hardware available in AVs, even if the architecture and the revision are the same. For example, the NVIDIA DRIVE Orin platform [20], which is slated to be available in production vehicles in 2023 and is the latest revision supported by Baidu's Apollo AV [38], uses NVIDIA's Ampere microarchitecture to deliver a performance of 5.2 FP32 TFLOPs [39]. The equivalent Ampere cloud GPU is the NVIDIA A100, which was released in 2020 and delivers a performance of 19.5 FP32 TFLOPs [40], a 3.75× increase over the DRIVE Orin. Table I measures the effects of this disparity by executing open-source implementations of DETR [41,42] and SWIN [10,43], two state-of-the-art vision transformers which have been adapted for object detection, atop both the NVIDIA Orin and the A100. We observe a significant speedup of up to 8.4× by executing the same model on the same input on a cloud GPU. Similar trends have been shown by prior work for other safety-critical algorithms applicable to AVs, such as motion planning [44].

Finally, an AV must decide by *when* to compute the control commands at the end of the pipeline, thus mandating a deadline on the computation being executed. For example, to surpass human driving in safety, the pipeline must execute faster than human reaction times. In addition, prior works [2,45,46] indicate this deadline varies widely according to the environment around the AV. Intuitively, driving slowly on a crowded urban street may tolerate a lax

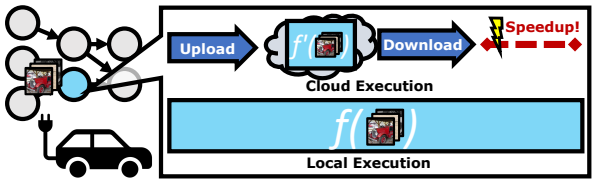

Fig. 2. **An arbitrage opportunity** afforded by exploiting state-of-the-art hardware to execute higher-accuracy models at a lower latency.

deadline for the computation to finish, whereas swerving to prevent an accident on the freeway requires the AV to respond quickly [47,48]. This variability in the required response time of the computation further complicates the deployment of more accurate, but slower algorithms and models since they cannot meet tighter deadlines [2].

III. Enhancing Safety Using The Cloud

Conventional wisdom suggests that the latency and availability of cellular network connections makes using the cloud on the critical-path of the computation infeasible [49]. However, Table I presents an arbitrage opportunity whereby an AV could potentially return more accurate results faster by exploiting the computational power of the hardware in the cloud (see Fig. 2). We argue that AVs should instead take a best-effort speculative approach to leverage the cloud when it is available and utilize reliable fallback mechanisms that use on-board computation when the cloud is not immediately accessible. We discuss three such mechanisms that enable a best-effort augmentation of an AV's safety below:

Higher-accuracy models. AVs can selectively offload data from their input sensors to the cloud, allowing higher accuracy models to be executed in the cloud. For example, in Table I, an AV can choose to execute DETR-ResNet-101-DC in the cloud on its camera data, which provides a ∼3-point increase in average precision over DETR-ResNet-50 [41] and executes faster. Thus, while an AV with an end-to-end deadline of 500 ms can only execute DETR-ResNet-50 on its local hardware, it can optimistically exploit the accuracy of all models in Table I when the latency to the cloud is low.

Accurate environment representation. While the earlier mechanism significantly enhances an AV's ability to process sensor data and understand its surroundings, this mechanism does not apply to obstacles obscured by the AV's blind spots. To improve safety in these scenarios, AVs can share their locations computed by the localization module to the cloud and subsequently retrieve the locations of other nearby vehicles. Integrating the locations of nearby vehicles via the cloud allows the planning module to generate safer trajectories which avoid collisions with vehicles located in blind spots.

Contingency planning. During the course of computation, AVs must make probabilistic decisions which affect the outputs of the modules. For example, object detectors are configured with a confidence threshold to filter out misdetections from the model's outputs [42]. Similarly, the prediction module generates several possible trajectories for nearby obstacles, and ranks them based on their probability of occurring [50,51]. The planning module then uses the most likely trajectory of the obstacles to plan a trajectory for an AV to follow. However, AVs can offload the computation of plans that handle unlikely object trajectories to the cloud. When an object takes an unlikely trajectory, AVs can access the corresponding cloud-computed plan, enabling quick reactions to any sudden changes in the environment.

§VI evaluates the efficacy of these mechanisms to ensure the safety of an AV under real-world complex scenarios.

IV. Related Work

Our principle design objective, as per §III, is to use the cloud as a best-effort pool of resources in order to improve the accuracy of AV decision-making while retaining the ability to fall back to locally-computed results to handle delays and variability. We now discuss the feasibility of implementing such an approach using existing execution systems.

The Robot Operating System (ROS) [52] is the current execution system-of-choice for AVs and has been deployed by vendors including Autoware [35], Cruise [53], BMW [54], and others [55,56]. ROS' highly-modular design enables various initiatives to provide additional feature enhancements, such as real-time support [57]). One such initiative which aims to add cloud support to ROS is FogROS [44,58]. FogROS allows users to designate ROS nodes that must execute in the cloud and specify their resource requirements. FogROS then provisions suitable cloud instances, configures the network connection between the cloud and the local machine, and registers the cloud nodes with the local coordinator, enabling seamless communication between the cloud and local nodes.

While FogROS greatly simplifies the deployment of specific ROS nodes to the cloud, it lacks out-of-the-box support for fall-back mechanisms that switch to local computation when the connection to the cloud becomes too unreliable. Moreover, implementing such an approach atop ROS' callback-execution model is a complex, tedious, and error-prone process because developers must implement and manage varying deadlines (see §II) using fine-grained wall-clock timers on every node [59]. Because these fine-grained timers may cause priority inversions when they rapidly produce events which overfill a `SingleThreadedExecutor`'s queue, developers must implement ROS nodes using `MultiThreadedExecutor`s which requires manually managing shared state using locks in order to ensure the safe and correct fusion of results from the cloud and from local computation. This complexity further increases if nodes attempt to maximize the accuracy of computation under a deadline by concurrently executing a mixture of algorithm and model implementations.

In addition to ROS, several related works have proposed designs that allow AVs to utilize the cloud, however, these approaches focus on specific tasks such as merging sensor data and intermediate representations across a fleet of vehicles [60]–[64]. For example, both Carcel [60] and EMP [64] enable a fleet of vehicles to create an accurate representation of their environment by selectively sharing their data to an edge server while remaining resilient to network fluctuations. Other work (e.g., Neurosurgeon [65]) has examined *how* to split computation between the cloud and the AV [66]–[68].

However, none of these works propose a general system design for AVs that falls back to results computed by on-board hardware in order to handle faults in the cloud or the network. Therefore, we believe that these works are complementary, and that AVs can integrate their methods with our design in order to gain further accuracy improvements while maintaining a reliable baseline accuracy through the ability to fall back to results from local computation.

V. Design

Given the limitations of the current systems, we seek to design an execution system that enables AV developers to easily achieve our design goal of augmenting the computation's accuracy with the vast pool of resources in the cloud. To ensure that the programming model remains familiar to the current AV systems (e.g., ROS), our system models an AV pipeline as a directed graph in which vertices, referred to as *operators* are akin to ROS nodes, and are connected to other vertices via *streams*. The streams are statically-typed and enforce an interface between the sender (akin to a ROS publisher) and a receiver (akin to a ROS sender).

However, to safely exploit the cloud resources in the presence of network delays and unavailability of cloud resources, our system allows operators to define *deadlines* that bound the time which can elapse between the transmission of a particular request to the cloud and the retrieval of its result. To do so, the system must enable each operator to: (*i*) specify the computation that executes locally and the computation that can execute in the cloud, (*ii*) control what input data is transmitted to the cloud and what deadline is assigned to its completion, and (*iii*) fuse the inputs from the cloud and the on-board hardware to ensure that the maximum accuracy results available by the deadline are used. We emphasize that our system design does not automatically decide which models and algorithms are suitable for cloud execution, but rather provides the abstractions and mechanisms to enable augmenting local computation with the cloud in order to benefit overall accuracy. Fig. 3 provides an overview of our approach, and we now elaborate on how our system achieves each of the aforementioned aspects below:

❶ Operator configuration. Each operator must implement a `setup` method in which the operator informs the system if its computation is to be assisted by a cloud implementation. To do so, the operator initiates a connection to the cloud for dispatching remote procedure calls (RPCs) and invokes the `use_cloud` method in our system with the following API:

```
use_cloud(handle, type, msg_handler, priority)
```

The operator registers the `handle` to the RPC connection with the system along with the `type` of the message to be relayed. Moreover, the system allows an operator to invoke `use_cloud` multiple times in order to take advantage of multiple cloud implementations which may span different cloud providers and exploit vendor-specific hardware to provide the highest-accuracy results within the given deadline [69]. By setting the `priority`, the system ensures a total order over the results of the invocations in order to select which of the potential results to output. For example, an object

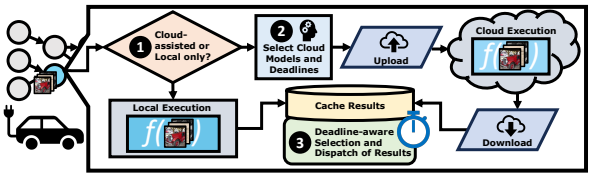

Fig. 3. **Our speculative cloud execution** design allows each operator to make fine-grained decisions about when to contact the cloud and how much time to allocate for a response. The results from the cloud are automatically incorporated to ensure the highest-possible accuracy.

detection operator may invoke several models from Table I in the cloud with the hope that one of them returns a result by the given deadline, and may set their priorities according to each detector's accuracy on a static dataset in order to favor results from more accurate detectors. Invoking this API informs the underlying execution system that it needs to set up the mechanisms detailed below, which ensure the correct collation of results from the cloud and the local execution.

❷ Deadline enforcement. While the execution of non-cloud enabled operators continues as normal, cloud-enabled operators must be able to specify a deadline on the time that they are able to wait for a higher-accuracy result from one of the cloud implementations registered with `use_cloud`. To do so, each invocation of `use_cloud` requires a `msg_handler` function that conforms to the following API:

```
(input_message, timestamp) ->
        Optional[output_message, deadline]
```

For each message that the operator receives, the system invokes the registered `msg_handlers` which may use the message content and the timestamp to decide whether to send the message to the handler's cloud implementation and what deadline to assign to this request. Unlike ROS timestamps that correspond to physical time, this `timestamp` is an internal logical representation of time that corresponds to a counter of the messages received by the operator. In addition to deciding whether to offload the computation of the messages to the cloud based on its contents [46,70], the handler functions can use the logical timestamps to enforce simple policies such as sending non-consecutive inputs to the cloud to reduce bandwidth usage, or alternating between different implementations to avoid bottlenecks in the cloud service and ensure fault-tolerance for the operator.

The operator makes the system aware of its intent to transmit a message to the cloud service by returning an `output_message` from the handler. The `output_message` must match the message definition in the RPC handle registered with `use_cloud`, but can be different from the type of the `input_message`. This type differentiation allows the operators to bundle extra state to the cloud platform that is not communicated to other operators. For example, the object tracking operator in the perception module assigns unique identifiers to detected obstacles based on their past history. As a result, selectively executing inputs to this operator in the cloud also requires transmitting the state of the identifiers generated by the local object tracker to allow the cloud-based object tracker to correctly assign identifiers.

In addition to the message to be conveyed to the cloud service, the handler decides the *relative* `deadline` d to be assigned to the arrival of its result. The ability to return a new deadline d for each incoming message enables the system to adjust to varying deadlines required by the AV to respond to dynamicity in the environment [2,45,46]. Given the deadline and the message, the system decides if the request should be sent based on the length of its outgoing queues, which fill up when network's quality of service degrades. If the system elects to send the request, the system indexes the request by its `timestamp` and `priority` and installs a timer that triggers d milliseconds from the current time.

❸ **High-accuracy results.** In parallel to executing the request in the cloud, the system invokes the callbacks registered for the incoming message and executes them on the local hardware with the lowest `priority` p. Upon completion, the local callbacks invoke `send` with the output message to convey the results to downstream operators. However, the system intercepts the invocation of the `send` to check for pending cloud requests with the same `timestamp` and a higher priority $p' > p$. If a pending cloud request is found, the system caches the local result and waits for the request's timer to interrupt as detailed below. However, if no pending cloud request is found and the system already forwarded a message from a cloud execution to the downstream operators (i.e., the result from the cloud arrives before the local computation finishes), the local result is dropped. Otherwise, the message containing the local result is sent to the downstream operators.

Upon arrival of a message from the cloud with `priority` p and `timestamp` t, the system checks if the message missed its deadline by querying the status of the timer installed for (t, p). If the timer was triggered, the message missed its deadline and is dropped. Similarly, if the system already sent a message for t (i.e., a higher priority request returned more quickly or another request's timer triggered earlier), the message is dropped and the installed timer is deactivated. Otherwise, if no higher priority cloud request is pending and the system has not sent a message for t, the message is sent to the downstream operators and the installed timer is deactivated. However, if there is a pending cloud request with a higher priority, the current results are cached.

Finally, if an active timer triggers, the system receives an interrupt in which it checks its cache for the highest priority message presently available for the `timestamp` and forwards the message to the downstream operators.

VI. Evaluation

We now evaluate the efficacy of our approach in augmenting the safety of cloud-assisted AVs under complex, real-world scenarios while maintaining the accuracy of an AV using only on-board computation. We evaluate our design to demonstrate that our approach to augmenting AVs Specifically, we seek to answer the following questions:

1) Do current technologies support a low-latency and reliable connection to the cloud? (§VI-A)
2) Does our system enable the three techniques (§III) to exploit cloud resources to enhance AV safety? (§VI-B)

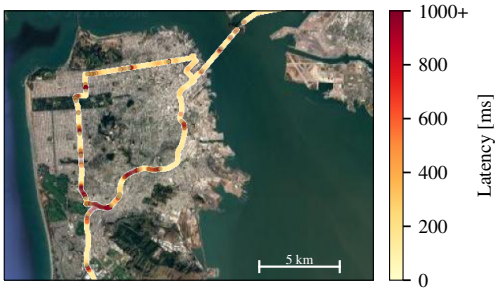

Fig. 4. **Cellular network latency** of a 5G connection while driving through a route in San Francisco frequented by Waymo and Cruise AVs.

A. Feasibility of Cloud Access

We investigate whether modern cellular networks are able to provide the speed and bandwidth necessary to execute data-intensive AV operators. To measure network performance in realistic setting, we conduct a field test by following a route in San Francisco where Cruise and Waymo already provide fully autonomous rides [71,72]. The route contains both urban and highway driving and is visualized in Fig. 4. We discuss our experiment setup and the findings below.

Experiment Setup. We model the transmission of HD camera footage from an AV to the cloud. In our vehicle, we connect a Lenovo ThinkPad P1 Gen 2 laptop to an Inseego MiFi X PRO 5G hotspot on the Verizon network via USB-C. On a Google Pixel 5, we collect timestamped GPS coordinates at 1 Hz. The laptop executes a multithreaded gRPC [73] client which sends 33.3 KB messages at 30 Hz to a server to match the bitrate of 30 FPS HD camera footage [74]. We establish a connection to a Google Cloud Platform `n1-highmem-8` instance in the `us-west2-a` zone which executes a gRPC server that responds to messages from the client with 1 KB acknowledgments. The client measures the round-trip latency of sending a message to receiving an reply. Because network delays may cause the client to queue messages and overstate the network latency, the client waits for all messages to process if it detects more than 30 queued messages.

Findings. The 5G network in San Francisco frequently provides latencies that enable cloud execution. We measure a median round-trip-latency of 68 ms (Fig. 4) which demonstrates an opportunity to take advantage of the hundreds of milliseconds in runtime disparity between cloud and AV hardware (Table I). In addition, the long tail of network latencies from 336 ms at the 90th percentile to 3027 ms at the 99th percentile substantiates the need to manage network delays. In §VI-B, we demonstrate that our design takes advantage of the fast common-case latencies to enable critical safety benefits while mitigating the impact of long tail-end latencies by falling back to results from local computation.

B. Study of Scenarios

We study of the safety benefits of our design (§V) and how it enables the three mechanisms which use the cloud to improve AV safety (§III). For each mechanism, we demonstrate its efficacy under a complex, real-world scenario

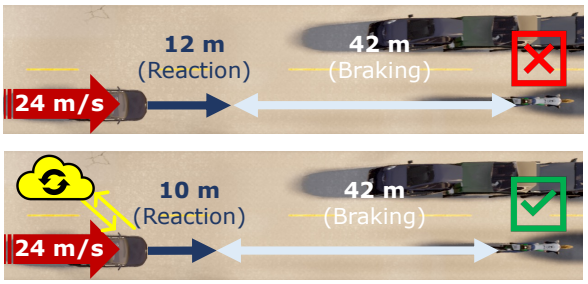

Fig. 5. **Traffic Jam Scenario** leverages the cloud's ability to run *higher-accuracy models* at a reduced latency to reduce the AV's response time, thus minimizing its reaction time and avoiding a collision with the motorcycle.

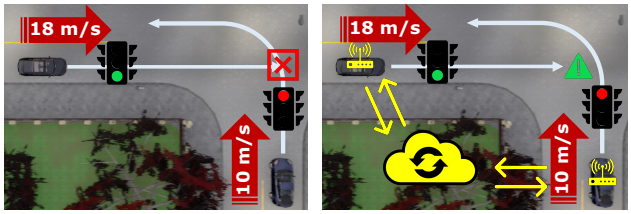

Fig. 6. **Running a Red Light Scenario** leverages the cloud's ability to build *accurate environment representations* to detect the occluded vehicle running the red light in time to avoid a collision.

executed using the CARLA simulator [75]. We use the pseudo-asynchronous mode of execution from the Pylot AV platform [4] to simulate the delay of calculating a demand for different end-to-end deadlines, retrieved from different end-to-end deadlines, retrieved from Fig. 4[1].

Traffic Jam. This scenario from [2] simulates merging into a traffic jam, visualized in Fig. 5. The AV drives at a high speed on a two lane undivided road and must come to a halt behind the motorcycle stopped in the distance. The motorcycle complicates this scenario because the AV must perceive the stopped obstacle from afar to prevent a collision. Moreover, the AV cannot swerve to avoid a collision due the vehicles in the opposite lane. Since this scenario requires both far-away detections and rapid responses due to the high driving speed, the technique of exploiting the cloud to run *higher-accuracy models* quickly ensures maximum safety.

To evaluate safety, we use a simple planner that brakes once the AV's object detection operator identifies the obstacle on three consecutive camera frames. We then investigate the following three operator configurations (Table II):

(**1**) **Local** which executes DETR-ResNet-50 on a local NVIDIA Orin GPU. We choose DETR-ResNet-50 since it is the only model that provides response times required to ensure human-level safety on the local GPU (Table I and §I).

(**2**) **Cloud** which executes DETR-ResNet-101 on an NVIDIA A100 GPU running on a Google Cloud `a2-highgpu-1g` instance. We choose DETR-ResNet-101 to ensure that the local and cloud models belong to the same architecture.

(**3**) **Ours** which enables operators to specify deadlines on response time from the cloud and fall back to local results when the cloud is unable to meet the deadline.

[1]Videos of scenarios are available at https://tinyurl.com/26fzrabu

TABLE II

CONFIGURATIONS THAT AVOID A COLLISION ARE MARKED IN GREEN, WHILE CONFIGURATIONS THAT COLLIDE ARE MARKED IN RED.

Speed [m/s]	Approach	Cloud Response Time [s]					
		0.5	0.75	1.0	1.25	1.5	3.0
11	**Local**						
	Cloud						
	Ours						
18	**Local**						
	Cloud						
	Ours						
20	**Local**						
	Cloud						
	Ours						
22	Local						
	Cloud						
	Ours						
24	Local						
	Cloud						
	Ours						

We sweep the entire range of driving speeds in California (i.e., 25 mph to 65 mph), and simulate cloud response times up to the p99 latency collected from our drive through San Francisco (§VI-A). We note that the higher cloud response times do not apply to the *Local* approach. We find that at higher speeds (e.g., 22 m/s), the lower-latency access to *higher-accuracy models* afforded by the cloud is critical in ensuring the safety of the AV. However, without appropriate mechanisms to fall back to local computation using deadlines as proposed in our design, the *Cloud* approach incurs more safety violations than the *Local* approach when network latency is high (e.g., a 3 second cloud response time when the AV drives at 18 m/s).

Fig. 5 investigates how executing a higher-accuracy object detector locally affects collision-avoidance in high-speed scenarios. We compare a *Local* execution of DETR-ResNet-101 to a *Cloud* execution with median network latency. We find that the local execution fails to break in time due to its 222 ms longer response time, resulting in a collision.

Running a Red Light. This scenario from the NHTSA pre-crash typology [76,77] simulates a vehicle running a red light which forces the AV to perform a collision avoidance maneuver. The AVs can only perceive the other vehicle just before a potential collision, challenging its ability to respond in time. We evaluate three variations of this scenario with different intersections, and find that neither local nor cloud executions of object detectors are able to avoid a collision.

However, using the cloud's ability to build an *accurate environment representation* (e.g., by sharing location data with nearby AVs), allows the AV to plan its trajectory with other vehicles in its blind spots. This enables the AV to brake early and avoid a collision in this scenario.

Person Jaywalking. This scenario simulates a person unexpectedly entering the street, requiring the AV to quickly respond in order to avoid a collision. Fig. 7 evaluates the scenario in which an AV executes its planning module locally, which has a 500 ms end-to-end response time. Because the pedestrian enters the street when the AV is only 10 m away, the AV cannot generate an emergency swerving maneuver or stop in time, resulting in a collision with the pedestrian.

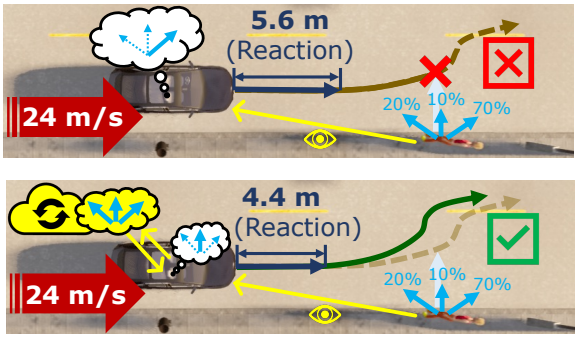

Fig. 7. **Person Jaywalking Scenario** leverages the cloud's ability to do *contingency planning* for the unlikely case that the pedestrian enters the street, allowing the AV to use the cached plan quickly and avoid a collision.

However, we perform *contingency planning* using the cloud which generates a plan for the low-likelihood case that the pedestrian enters the street, downloads the plan to the AV, and caches the plan in the AV's planner. When the cloud-assisted AV detects the pedestrian entering the street, the AV enacts the cached contingency plan and bypasses the local planner, lowering the response time to 400 ms. We observe that the cloud-computed contingency plan enables the AV to swerve in time and successfully avoid a collision.

VII. Discussion & Conclusion

Our experiments show that AVs can leverage the cloud with current technology to exploit safety benefits. We present a design which ensures that cloud execution strictly improves safety by executing models and algorithms *speculatively* in the cloud. When the cloud fails to produce results by a deadline, our design falls back to results from local execution which are computed concurrently. Although our design focuses on the enforcement of provided deadlines, the calculation of deadlines for cloud execution presents an exciting opportunity for further exploration.

Moreover, trends in technology demonstrate that AVs of the future will harness the benefits of the cloud. AVs increasingly rely on higher-fidelity data collected from more sensors which they process using larger and more compute-intensive algorithms and models. Meanwhile, cloud and network technologies continue to improve, and will present more arbitrage opportunities in which the cloud can benefit safety. We hope that this research will bring attention to the potential advantages of augmenting AVs with the cloud and inspire further exploration and development in this area.

References

[1] National Highway Traffic Safety Administration, "Traffic Safety Facts (2020 Data)," https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/813375.

[2] I. Gog, S. Kalra, P. Schafhalter, J. E. Gonzalez, and I. Stoica, "D3: A Dynamic Deadline-Driven approach for Building Autonomous Vehicles," in *Proceedings of the Seventeenth European Conference on Computer Systems*, 2022, pp. 453–471.

[3] J. Wang, L. Zhang, Y. Huang, J. Zhao, and F. Bella, "Safety of autonomous vehicles," *Journal of advanced transportation*, vol. 2020, pp. 1–13, 2020.

[4] I. Gog, S. Kalra, P. Schafhalter, M. A. Wright, J. E. Gonzalez, and I. Stoica, "Pylot: A Modular Platform for Exploring Latency-Accuracy Tradeoffs in Autonomous Vehicles," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 8806–8813.

[5] S. Wang, Z. Sheng, J. Xu, T. Chen, J. Zhu, S. Zhang, Y. Yao, and X. Ma, "ADEPT: A testing platform for simulated autonomous driving," in *37th IEEE/ACM International Conference on Automated Software Engineering*, 2022, pp. 1–4.

[6] T. Dreossi, D. J. Fremont, S. Ghosh, E. Kim, H. Ravanbakhsh, M. Vazquez-Chanlatte, and S. A. Seshia, "VerifAI: A toolkit for the formal design and analysis of artificial intelligence-based systems," in *Computer Aided Verification: 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I 31*. Springer, 2019, pp. 432–442.

[7] "Tesla FSD Hardware 4.0 Revealed: More Cameras, New Placements," https://tinyurl.com/yc2as3f6.

[8] "Waymo one autonomous robotaxi first ride: Way mo' better than driving?" https://tinyurl.com/yvku75kx.

[9] "Introducing the 5th Generation Waymo Driver," https://blog.waymo.com/2020/03/introducing-5th-generation-waymo-driver.html.

[10] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 10 012–10 022.

[11] Y. Li, H. Mao, R. Girshick, and K. He, "Exploring plain vision transformer backbones for object detection," in *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part IX*. Springer, 2022, pp. 280–296.

[12] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie, "A convnet for the 2020s," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 11 976–11 986.

[13] X. Zhai, A. Kolesnikov, N. Houlsby, and L. Beyer, "Scaling vision transformers," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 12 104–12 113.

[14] M. Tan and Q. Le, "Efficientnetv2: Smaller models and faster training," in *International conference on machine learning*. PMLR, 2021, pp. 10 096–10 106.

[15] M. Tan, R. Pang, and Q. V. Le, "EfficientDet: Scalable and Efficient Object Detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[16] S.-C. Lin, Y. Zhang, C.-H. Hsu, M. Skach, M. E. Haque, L. Tang, and J. Mars, "The Architectural Implications of Autonomous Driving: Constraints and Acceleration," in *Proceedings of the 23rd International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2018, pp. 751–766. [Online]. Available: http://doi.acm.org/10.1145/3173162.3173191

[17] B. Wolfe, B. Seppelt, B. Mehler, B. Reimer, and R. Rosenholtz, "Rapid holistic perception and evasion of road hazards." *Journal of experimental psychology: general*, vol. 149, no. 3, p. 490, 2020.

[18] G. Johansson and K. Rumar, "Drivers' brake reaction times," *Human factors*, vol. 13, no. 1, pp. 23–27, 1971.

[19] "NVIDIA DRIVE: Hardware for Self-Driving Cars," https://www.nvidia.com/en-us/self-driving-cars/drive-platform/hardware/.

[20] "NVIDIA Introduces DRIVE AGX Orin," https://tinyurl.com/6pjsxzw7.

[21] T. Tomazin, "A Data Center on Wheels: NVIDIA Unveils DRIVE Atlan Autonomous Vehicle Platform," https://blogs.nvidia.com/blog/2021/04/12/nvidia-drive-atlan-autonomous-vehicle-platform/.

[22] "NVIDIA Enters Production With DRIVE Orin, Unveils Next-Gen DRIVE Hyperion AV platform," https://tinyurl.com/2s38djwr.

[23] "Tesla Talks FSD Hardware 4.0, but There Will Not Be Retrofits," https://www.notateslaapp.com/news/1172/tesla-talks-fsd-hardware-4-0-but-there-will-not-be-retrofits.

[24] "As Automakers Add Technology to Cars, Software Bugs Follow," https://www.nytimes.com/2022/02/08/business/car-software-lawsuits.html.

[25] A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica *et al.*, "Above the clouds: A berkeley view of cloud computing," *Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS*, vol. 28, 2009.

[26] "NVIDIA GPU Cloud Computing," https://www.nvidia.com/en-us/data-center/gpu-cloud-computing/.

[27] "Google Cloud TPU," https://cloud.google.com/tpu.

[28] "AWS Inferentia," https://aws.amazon.com/machine-learning/inferentia/.

[29] "AWS EC2 instance timeline," https://instancetyp.es.

[30] "Building Warehouse-Scale Computers at Google Cloud," https://www.youtube.com/watch?v=9i7HuU8d3_4.

[31] "Tesla recalls 362,000 U.S. vehicles over Full Self-Driving software," https://www.reuters.com/business/autos-transportation/tesla-recalls-362000-us-vehicles-over-full-self-driving-software-2023-02-16/.

[32] "Toyota recalls 1.9 million cars for software glitch," https://www.cnbc.com/2014/02/12/toyota-recalls-19-million-cars-for-software-glitch.html.

[33] General Motors, "2018 Self-driving safety report," https://www.gm.com/content/dam/company/docs/us/en/gmcom/gmsafetyreport.pdf.

[34] "NTSB's Accident Report on the Uber Self-Driving Vehicle Crash," https://tinyurl.com/y334xnez.

[35] Autoware, "Autoware User's Manual - Document Version 1.1," https://tinyurl.com/2v2jkk9n.

[36] "Architectural overview of Baidu's ApolloAuto," https://github.com/ApolloAuto/apollo#architecture.

[37] "Apollo's Traffic Light Perception," https://github.com/ApolloAuto/apollo/blob/master/docs/06_Perception/traffic_light.md.

[38] "Hardware prerequisites of Baidu's ApolloAuto," https://github.com/ApolloAuto/apollo#prerequisites.

[39] "NVIDIA Drive AGX Orin Developer Kit," https://tinyurl.com/u2rxefpt.

[40] "NVIDIA A100 Specifications," https://www.nvidia.com/en-us/data-center/a100/.

[41] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part I 16*. Springer, 2020.

[42] "DETR - Hugging Face," https://huggingface.co/docs/transformers/model_doc/detr.

[43] H. Zhang, F. Li, S. Liu, L. Zhang, H. Su, J. Zhu, L. Ni, and H. Shum, "Dino: Detr with improved denoising anchor boxes for end-to-end object detection," in *International Conference on Learning Representations*, 2022.

[44] K. E. Chen, Y. Liang, N. Jha, J. Ichnowski, M. Danielczuk, J. Gonzalez, J. Kubiatowicz, and K. Goldberg, "Fogros: An adaptive framework for automating fog robotics deployment," in *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*. IEEE, 2021, pp. 2035–2042.

[45] M. Li, Y. Wang, and D. Ramanan, "Towards Streaming Perception," in *Proceedings of the European Conference on Computer Vision (ECCV)*, Aug. 2020.

[46] G.-E. Sela, I. Gog, J. Wong, K. K. Agrawal, X. Mo, S. Kalra, P. Schafhalter, E. Leong, X. Wang, B. Balaji, J. E. Gonzalez, and I. Stoica, "Context-aware streaming perception in dynamic environments," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2022.

[47] L. Claussmann, M. Revilloud, D. Gruyer, and S. Glaser, "A Review of Motion Planning for Highway Autonomous Driving," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 5, pp. 1826–1848, 2019.

[48] M. Alcon, H. Tabani, L. Kosmidis, E. Mezzetti, J. Abella, and F. J. Cazorla, "Timing of Autonomous Driving Software: Problem Analysis and Prospects for Future Solutions," in *Proceedings of the 26th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2020, pp. 267–280.

[49] B. Kehoe, S. Patil, P. Abbeel, and K. Goldberg, "A survey of research on cloud robotics and automation," *IEEE Transactions on automation science and engineering*, vol. 12, no. 2, pp. 398–409, 2015.

[50] N. Rhinehart, K. M. Kitani, and P. Vernaza, "R2P2: A Reparameterized Pushforward Policy for Diverse, Precise Generative Path Forecasting," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 772–788.

[51] N. Rhinehart, R. McAllister, K. Kitani, and S. Levine, "PRECOG: Prediction Conditioned on Goals in Visual Multi-Agent Settings," in *Proceedings of the IEEE International Conference on Computer Vision (CVPR)*, 2019, pp. 2821–2830.

[52] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: An Open-Source Robot Operating System," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA); Workshop on Open Source Robotics*, vol. 3, May 2009, p. 5.

[53] N. Valigi, "Lessons Learned Building a Self-Driving Car on ROS," https://roscon.ros.org/2018/presentations/ROSCon2018_LessonsLearnedSelfDriving.pdf, 2018.

[54] M. Aeberhard, T. Kühbeck, B. Seidl, M. Friedl, J. Thomas, and O. Scheickl, "Automated Driving with ROS at BMW," http://www.ros.org/news/2016/05/michael-aeberhard-bmw-automated-driving-with-ros-at-bmw.html.

[55] A. Fregin, M. Roth, M. Braun, S. Krebs, and F. Flohr, "Building a Computer Vision Research Vehicle with ROS," http://www.ros.org/news/2018/07/roscon-2017-building-a-computer-vision-research-vehicle-with-ros----andreas-fregin.html.

[56] Udacity, "An Open Source Self-Driving Car," https://www.udacity.com/self-driving-car.

[57] C. Ho, S. Nirmal, J. P. Samper, S. Nikulin, A. Pemmaiah, D. Pangercic, and J. Becker, "ROS2 on Autonomous Vehicles," https://roscon.ros.org/2018/presentations/ROSCon2018_ROS2onAutonomousDrivingVehicles.pdf.

[58] J. Ichnowski, K. Chen, K. Dharmarajan, S. Adebola, M. Danielczuk, V. Mayoral-Vilches, H. Zhan, D. Xu, R. Ghassemi, J. Kubiatowicz *et al.*, "Fogros 2: An adaptive and extensible platform for cloud and fog robotics using ros 2," *arXiv preprint arXiv:2205.09778*, 2022.

[59] "ROS2 WallTimer," http://docs.ros.org/en/indigo/api/roscpp/html/classros_1_1WallTimer.html.

[60] S. Kumar, S. Gollakota, and D. Katabi, "A cloud-assisted design for autonomous driving," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, 2012, pp. 41–46.

[61] H. Qiu, F. Ahmad, F. Bai, M. Gruteser, and R. Govindan, "Avr: Augmented vehicular reality," in *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, 2018, pp. 81–95.

[62] Q. Chen, X. Ma, S. Tang, J. Guo, Q. Yang, and S. Fu, "F-cooper: Feature based cooperative perception for autonomous vehicle edge computing system using 3d point clouds," in *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*, 2019, pp. 88–100.

[63] S. P. Chinchali, E. Cidon, E. Pergament, T. Chu, and S. Katti, "Neural networks meet physical networks: Distributed inference between edge devices and the cloud," in *Proceedings of the 17th ACM Workshop on Hot Topics in Networks*, 2018, pp. 50–56.

[64] X. Zhang, A. Zhang, J. Sun, X. Zhu, Y. E. Guo, F. Qian, and Z. M. Mao, "Emp: Edge-assisted multi-vehicle perception," in *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*, 2021, pp. 545–558.

[65] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 1, pp. 615–629, 2017.

[66] M. Cui, S. Zhong, B. Li, X. Chen, and K. Huang, "Offloading autonomous driving services via edge computing," *IEEE Internet of Things Journal*, vol. 7, no. 10, pp. 10 535–10 547, 2020.

[67] Y. Sun, X. Guo, S. Zhou, Z. Jiang, X. Liu, and Z. Niu, "Learning-based task offloading for vehicular cloud computing systems," in *2018 IEEE International Conference on Communications (ICC)*. IEEE, 2018.

[68] F. Sun, F. Hou, N. Cheng, M. Wang, H. Zhou, L. Gui, and X. Shen, "Cooperative task scheduling for computation offloading in vehicular cloud," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 11, pp. 11 049–11 061, 2018.

[69] I. Stoica and S. Shenker, "From cloud computing to sky computing," in *Proceedings of the Workshop on Hot Topics in Operating Systems*, 2021, pp. 26–32.

[70] J. Jiang, G. Ananthanarayanan, P. Bodik, S. Sen, and I. Stoica, "Chameleon: Scalable Adaptation of Video Analytics," in *Proceedings of the ACM Special Interest Group on Data Communication Conference (SIGCOMM)*, 2018, pp. 253–266. [Online]. Available: http://doi.acm.org/10.1145/3230543.3230574

[71] "Welcome, riders," https://getcruise.com/news/blog/2022/welcome-riders/, Aug. 2021.

[72] "Welcoming our first riders in san francisco," https://blog.waymo.com/2021/08/welcoming-our-first-riders-in-san.html, Feb. 2022.

[73] "grpc," https://grpc.io/.

[74] "Youtube recommended upload encoding settings," https://support.google.com/youtube/answer/1722171.

[75] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An Open Urban Driving Simulator," in *Proceedings of the 1st Conference on Robot Learning (CoRL)*, 2017, pp. 1–16.

[76] "NHTSA-inspired Pre-crash Scenarios," https://carlachallenge.org/challenge/nhtsa/.

[77] "Pre-Crash Scenario Typology for Crash Avoidance Research," https://www.nhtsa.gov/sites/nhtsa.gov/files/pre-crash_scenario_typology-final_pdf_version_5-2-07.pdf.