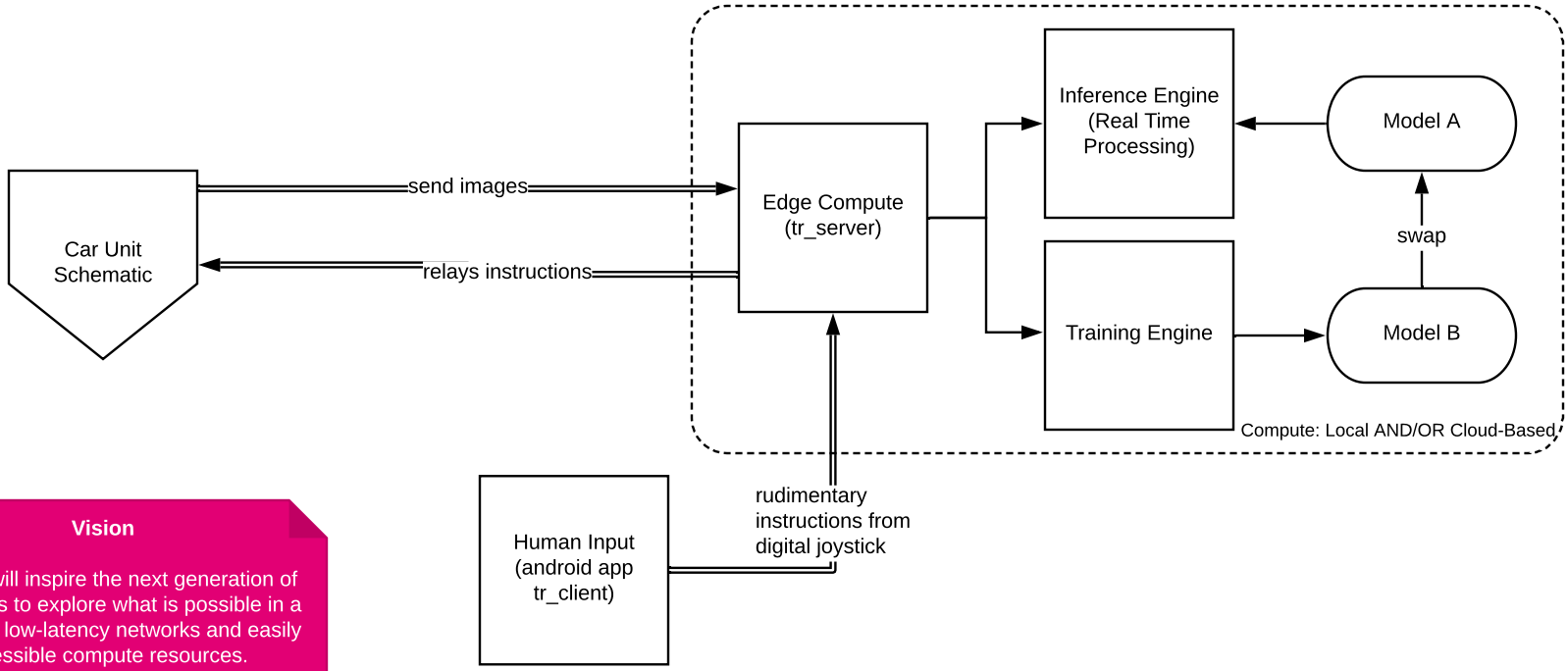
Compute/Network Interaction Overview



Vision

T-Racer will inspire the next generation of developers to explore what is possible in a world with low-latency networks and easily accessible compute resources.

Future State

T-Racer will receive instructions from the inference engine while simultaneously sending images to the training engine, allowing the system to continually refinine and update the model being used by the inference engine and improve performance on the course.

TR Specs

Edge Compute is an i7 laptop (6 cores, hyper-threaded) with a 2080 nvidia gpu from Systems76.

Device running tr_controller attached to car body is Samsung S10.

Device running tr_client for human input is any stock android phone.

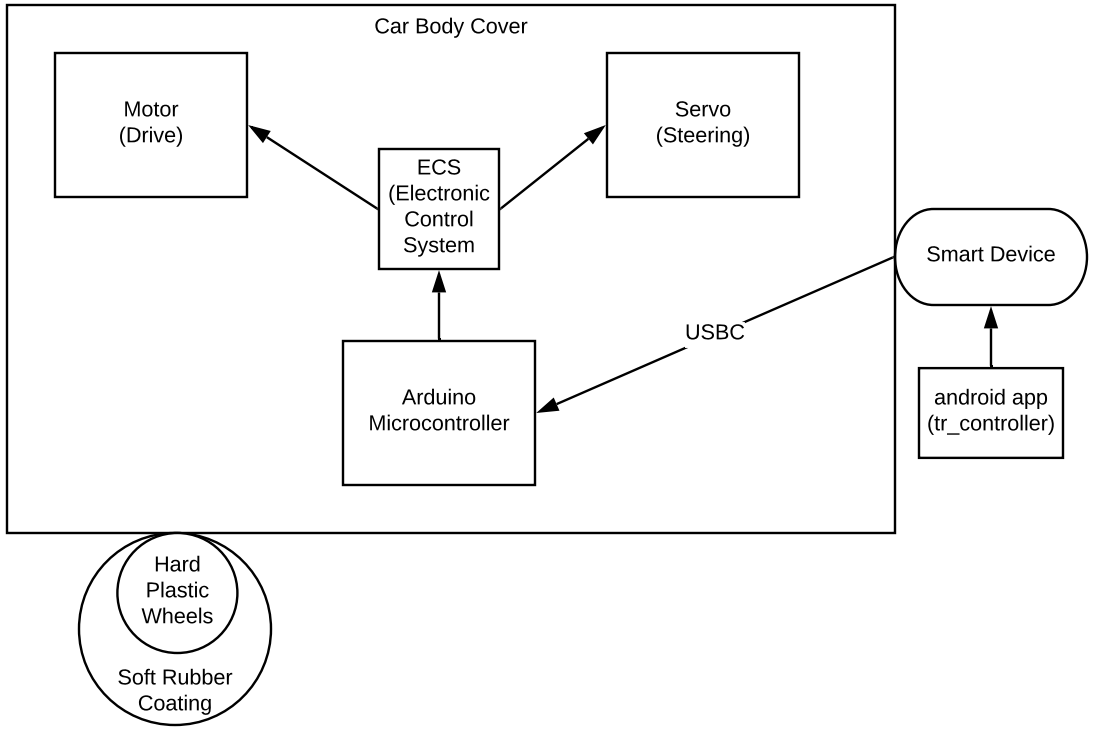
How it Works

Human input is required as part of the initial the training cycle, with a person providing basic "left / right / forward" directions to T-Racer from the digital joystick provided by the tr_client app. This means driving the car a LOT of time and checking periodically to see if it's "enough".

As the car is driven manually, a smart device attached to the car's body captures screen grabs with the tr_controller app. These grabs are transmitted over the network (WiFi, 4G, 5g, etc) to the compute resources located either locally or in the cloud. On the compute, the tr_server app converts the screen grabs to an image format and feeds those images into the training engine. After sufficient training loops (trial and error), a model is generated with the potential to "drive" the car without any human input.

To test the model, it is loaded into the inference engine on the compute (local or cloud). As the car startsup, it transmits screen grabs to the inference engine via the compute, which analyzes the images using the pre-loaded model and calculates numbers for drive and steering. Those numbers are sent back through the compute to the car via the tr_controller app and the car adjusts according to the numbers from the model..

Car Unit Schematic



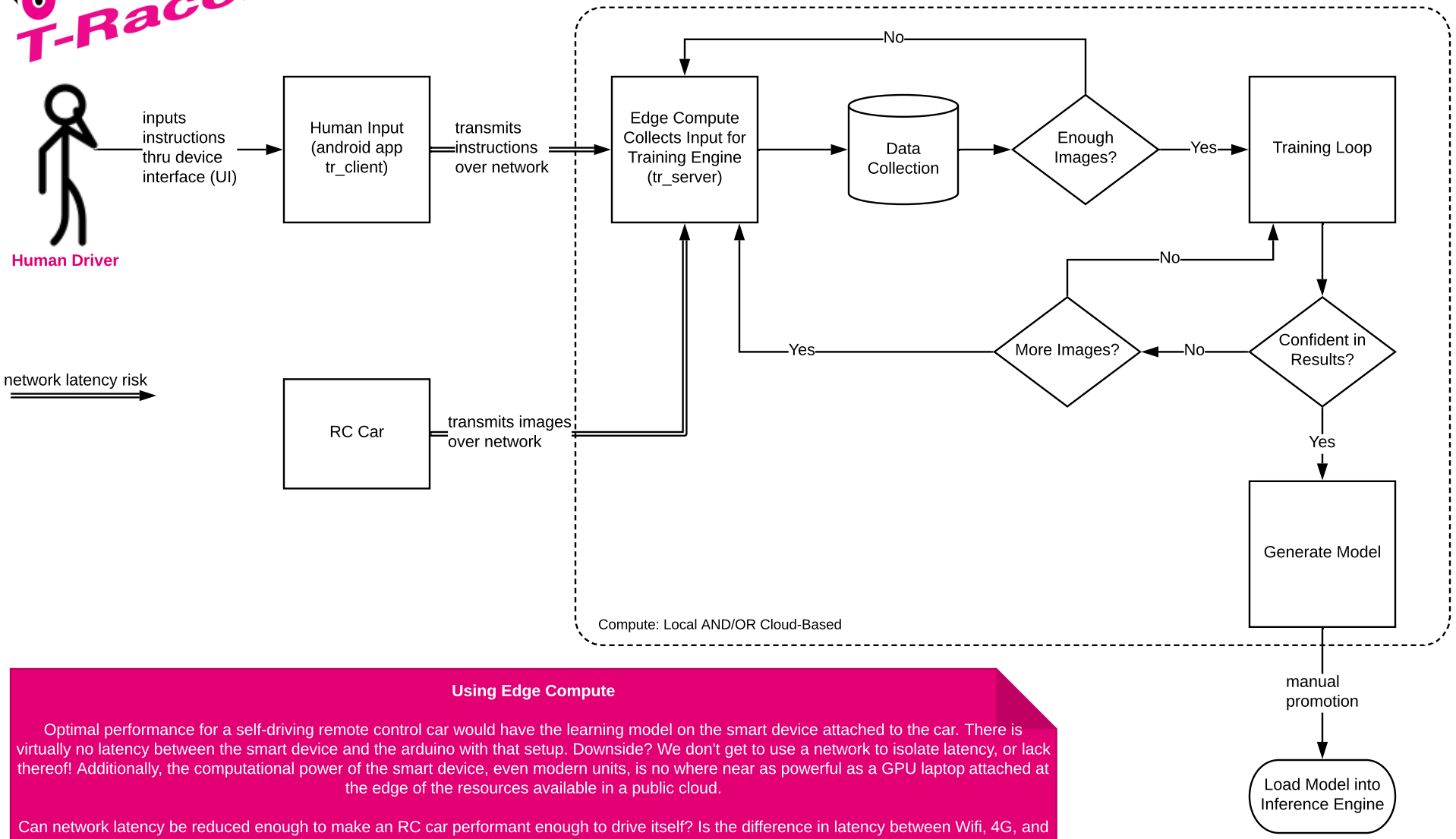
Adruino Microcontroller

The MC does not have the overhead of an OS and is able to run C code natively without any performance issues. This allows the car to have a auto-pause function without any need for network connectivity should receipt of instructions or transmission of screen grabs be interrupted.

Remote Control (RC) Kit

RC kits, including the one used for T-Racer, generally come with three parts: a motor, a servo, and an ECS. By using a stock drift car kit with an inexpensive arduino as the digital controller, combined with sofwtare decisions that optimize for older-model android devices,T-Racer becomes accessible for a relatively low cost way to experiment with autonomous vehicles.

Process Flow - Training Mode



Using Edge Compute

Optimal performance for a self-driving remote control car would have the learning model on the smart device attached to the car. There is virtually no latency between the smart device and the arduino with that setup. Downside? We don't get to use a network to isolate latency, or lack thereof! Additionally, the computational power of the smart device, even modern units, is no where near as powerful as a GPU laptop attached at the edge of the resources available in a public cloud.

Can network latency be reduced enough to make an RC car performant enough to drive itself? Is the difference in latency between Wifi, 4G, and eventually 5g as great as we're hoping? Using edge compute with T-Racer, rather than on-device modeling, allows the network componnet to be further isolated in an attempt to answer this question and bring the network to life in a tangible way.

By using a local (GPU) laptop for the compute during automomous driving sessions, the latency from the round trips to public clouds are eliminated, allowing the focus to rest squarely on the netowrk transmissions that remain in the process.

Process Flow - Autonomous Mode

Spectators!

transmits images over network

Edge Compute Accepts Image (tr_server)

Inference Engine Analyzes the Image with the Model (from Training)

interprets

RC Car (Smart Device tr_controller)

Model Generates Instructions (Drive & Steering)

Compute Sends Instructions to Car Smart Device

transmitted over network

network latency risk

compute delay risk

Smart Device Receives Instructions

Arduino Continually Checks for Recent Instructions from the Smart Device

New Instructions Received?

Yes

No

Instruction: Motor & Servo Numbers from Model

Instruction: Pause Car

Arduino Forwards Instructions to ECS

ECS Controls Motor & Servo

Car Executes Instruction

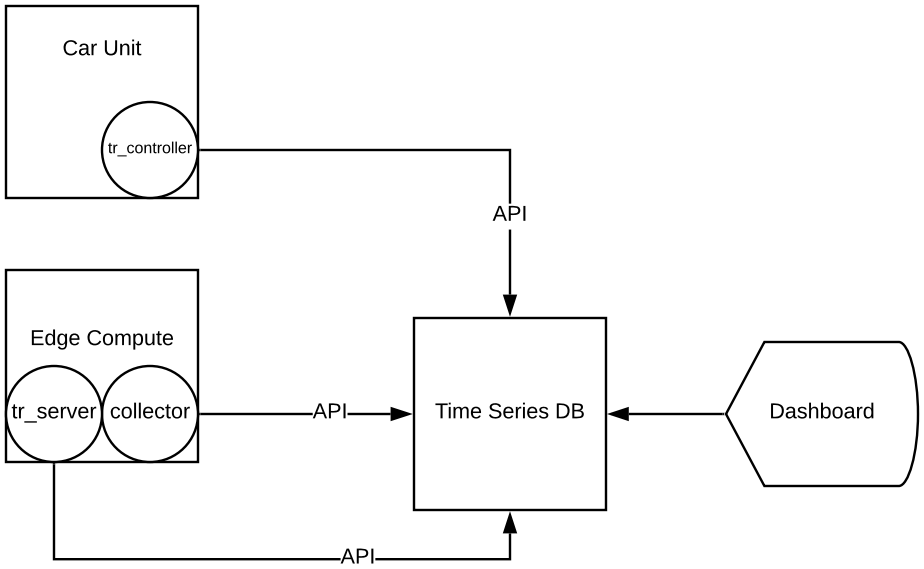
Delay During Interpretation

Once an image is set to the inteference engine for analysis, a delay is expected as calculations are performed. Using a local GPU laptop for this step is expected to isolate the network latency from any latency imposed by round trips to public cloud provides.

Garbage Collection

As a java device, the android-based smart device will occasionally need to perform garbage collection, interrupting the systems ability to receive or transmit. During this time, the Arduino MC will be able to take over and transmit new instructins to "pause" T-Racer until normal function can resume.

Telemetry & Dashboard Design



InfluxData: Telegraf & InfluxDB

InfluxData is an open-source time series data base platform. InfluxDB is a time series database. Telegraf is a collector agent that runs on the server, collecting metrics such as system stats, networking, and data from 40+ plugins.

Elastic DB & Kibana + Canvas
Elastic is an open storage solution

Dashboard & Visualization

Once data is collected, there are near limitless options for visualization. The InfluxData platform includes Chronograf for visualization. Grafana is another popular open source choice. Exploring options available in Azure may also be useful (depedening on latency concerns).