CARMA: Towards Personalized Automotive Tuning

Tobias Flach  
University of Southern California  
Los Angeles, CA 90089, USA  
flach@usc.edu

Nilesh Mishra  
University of Southern California  
Los Angeles, CA 90089, USA  
nmishra@usc.edu

Luis Pedrosa  
University of Southern California  
Los Angeles, CA 90089, USA  
luis.pedrosa@usc.edu

Christopher Riesz  
Rutgers University, Camden  
Camden, NJ 08102, USA  
criesz@camden.rutgers.edu

Ramesh Govindan  
University of Southern California  
Los Angeles, CA 90089, USA  
ramesh@usc.edu

Abstract

Wireless sensing and actuation have been explored in many contexts, but the automotive setting has received relatively little attention. Automobiles have tens of onboard sensors and expose several hundred engine parameters which can be tuned (a form of actuation). The optimal tuning for a vehicle can depend upon terrain, traffic, and road conditions, but the ability to tune a vehicle has only been available to mechanics and enthusiasts. In this paper, we describe the design and implementation of CARMA (Car Mobile Assistant), a system that provides high-level abstractions for sensing automobile parameters and tuning them. Using these abstractions, developers can easily write smartphone “apps” to achieve fuel efficiency, responsiveness, or safety goals. Users of CARMA can tune their vehicles at the granularity of individual trips, a capability we call personalized tuning. We demonstrate through a variety of applications written on top of CARMA that personalized tuning can result in over 10% gains in fuel efficiency. We achieve this through route-specific or driver-specific customizations. Furthermore, CARMA is capable of improving user satisfaction by increasing responsiveness when necessary, and promoting vehicular safety by appropriately limiting the range of performance available to novice or unsafe drivers.

Categories and Subject Descriptors

D.2.13 [Software Engineering]: Reusable Software—Domain engineering; I.7 [Computers in Other Systems]: Consumer Products

∗This material is based upon work supported by the National Science Foundation under Grants No. CNS-021778, CCF-0830569 and CCF-1048606 and CCF-820230. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation (NSF).

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SenSys ’11, November 1–4, 2011, Seattle, WA, USA.  
Copyright 2011 ACM 978-1-4503-0718-5/11/11 ...$10.00

General Terms

Design, Experimentation, Performance

Keywords

Automobile, Engine Control Unit, Scanning, Tuning

1 Introduction

Networked sensing and actuation have been explored in many natural and man-made environments, most recently in smart-grid and smart-building technologies [42, 28]. The emergence of smartphones has added a new dimension to networked sensing, enabling context awareness [35] and participatory sensing [37]. However, one ubiquitous man-made artifact, the automobile, has received relatively less attention.

Automobiles have tens of onboard sensors which can continuously monitor engine parameters such as the air flow, the current gear setting, the engine RPM and speed. More importantly, automobiles can be tuned by modifying engine parameters, such as RPM limits, transmission shift points, or air fuel mixtures. In general, automotive tuning is a complex and dangerous task, so it is performed today by trained mechanics, enthusiasts, or fleet operators.

There are, however, benefits to being able to tune cars on a larger scale. Specifically, personalized tuning, in which a user tunes a car at a granularity of a trip, can potentially have significant fuel efficiency benefits. Most cars come with factory default parameters and are rarely modified, but as with any parameterizable system, a single set of parameters is unlikely to be optimal across all settings. Indeed, the fuel efficiency of a car can vary widely with the route, the congestion along the route, terrain, and other factors (Section 2).

Personalized tuning can determine route-specific engine parameters to squeeze efficiency gains.

Fuel efficiency is not the only motivation for personalized tuning. This capability can be used to improve safety or to increase the responsiveness1 of the car when necessary. For example, it is possible to set speed and RPM limits on cars

1Responsiveness is the measure of how quickly and forcefully a vehicle reacts to changes in throttle position. This includes engine braking while off the throttle, and maintaining a lower gear for quicker acceleration during throttle increases.
in order to prevent misuse or rash driving, a capability that is useful when handing off a car to a teen-aged driver.

This paper explores the design and implementation of CARMA, a system for smartphones that is designed to democratize personalized tuning (Section 3). Because it runs on smartphones, CARMA is highly portable and makes trip-granularity tuning convenient. CARMA is also programmable, enabling the development of smartphone apps that permit personalized tuning for flexibly achieving fuel efficiency, responsiveness, and safety goals.

CARMA is designed to enable the following personalized tuning scenario (Figure 1). Before embarking on a trip to a mountain resort, Alice pulls out her smartphone and launches a CARMA app (Section 4) designed to tune her car for fuel efficiency. She indicates the target destination and selects the route she prefers. The app accesses an online database of routes and route characteristics to determine speed limits, the expected congestion, the density of stoplights, and the terrain along the route. It then analyzes, based on car sensor readings obtained from previous trips, her driving habits on hilly roads, and sets engine parameters to improve fuel efficiency on her trip without sacriﬁcing responsiveness. When she reaches her destination, she invokes another app that installs a speed limiter on her car, before handing it off to a valet.

This paper takes a ﬁrst step towards this goal by instantiation many of the elements of the scenario. We have designed and implemented the CARMA system, which provides high-level abstractions for automotive sensing and control. We have also implemented several apps that illustrate many of the beneﬁts that come from programmability, portability, and Internet-connectedness. To our knowledge, CARMA is the only system to have these capabilities.

Using our implementation of CARMA on the Android operating system, we demonstrate the beneﬁts of personalized tuning. In experiments (Section 5) conducted over 1100 miles and spanning a wide variety of trafﬁc conditions, we ﬁnd that CARMA’s personalized tuning obtains 10% or more fuel economy gains on most trips. This may seem modest, but, if applied to the 250 million cars on US roads today, over $19 billion in savings could be generated annually. We also demonstrate CARMA’s ﬂexibility, by implementing and evaluating apps that promote safety and responsiveness, and attempt to customize cars based on driver behavior.

2 Motivation for Personalized Tuning

Modern car engines have several parameters that determine their performance. Examples of such parameters include air ﬂow, transmission shift points, and spark timings. Settings for these parameters can determine the car’s fuel efﬁciency and its responsiveness. Cars come with factory defaults for these settings, and tuning modiﬁes these parameters. By personalized tuning, we mean the ability to modify a car’s parameters at the granularity of a single trip: for example, to optimize engine performance for the particular driver, or the route, or trafﬁc conditions.

A motivating scenario for personalized tuning is the teen-mode or valet-mode. Before handing off a car to a teen or a valet, an owner could limit engine performance to prevent misuse or rash driving. When the car is returned, the owner would restore the original settings before his or her next trip.

Beyond ensuring public safety, the premise underlying personalized tuning is that a single set of parameters cannot possibly be appropriate for different conditions (drivers, routes, and trafﬁc). To assess this premise, we conduct the following series of experiments on two different cars: Car A, owned by Driver A is a 2011 Ford Fiesta (manual transmission), and Car B, owned by Driver B is a 1998 Toyota Corolla (manual transmission). For each car, we have the owner drive the car under the following scenarios: Int, a drive along a local section of an Interstate highway during an off-peak time; Int-T, a drive along an interstate highway during rush hour; Hill, a segment of an undulating, but lightly traveled, road; and Street, a ﬂat section of a moderately busy thoroughfare with several trafﬁc lights in a large metropolitan area. In addition, we conduct one experiment, Switch, in which we switch drivers for the two cars, while following the same route as Int. For each experiment, we measure the fuel economy (in miles per US gallon) using the methodology described in Section 5. Each experiment was conducted in parallel to ensure that both cars experienced roughly the same traffic conditions.

Figure 2 depicts the results of these experiments for the two cars. Since the cars are from different manufacturers, we cannot perform a meaningful comparison across cars. However, for a given car, there are signiﬁcant performance differences across the experiments. For example, the Hill experiment has almost 30% lower fuel economy than the Int route, for both cars, while the Street experiment has a 50% lower fuel economy than Int for Car A, and a 60% lower fuel economy for Car B. This is not surprising, and is also well known, but motivates our work: there is signiﬁcant variability across different conditions, so it must be possible to extract better performance by tuning the car separately for each condition.

Some of this variability is intrinsic. Notice that the performance of the two cars under different conditions is quali-
tatively similar. This is because the work done to overcome friction and wind resistance on a high-speed interstate without traffic is different from the work done to climb hills or to start and stop the car in traffic. This intrinsic variability cannot be exploited by tuning.

There is also extrinsic variability that comes from having non-optimal parameter settings. For example, every car has a “sweet spot” engine RPM setting at which fuel consumption is lowest for a required power output; lower or higher RPM values can increase fuel consumption [32]. It may be possible to tune the car’s transmission shift points so that the engine RPM is maintained in a narrow band around this sweet spot. Default settings usually permit a larger RPM band to increase vehicle responsiveness.

Another example of extrinsic variability is illustrated by comparing Int and Int-T for the two cars. Driver A is actually more fuel-efficient on the interstate in rush-hour, than in the absence of traffic; the opposite is the case for Driver B, whose Int-T fuel economy is 12% lower than Int. After these experiments, we interviewed both drivers and found that Driver A drives aggressively on an empty interstate, but tries to maintain a steady speed in rush-hour traffic. Driver B, on the other hand, drives steadily on an empty interstate, but aggressively during rush-hour traffic (filling in the space between his car and the one ahead to prevent lane changes). Engines can be tuned to dampen these forms of aggression, trading off some responsiveness for increased fuel efficiency.

A concrete example of extrinsic variability is illustrated by our Switch experiment. Figure 3 shows the CDF of the throttle position\(^2\) for Drivers A and B driving Car B on Int. The throttle position is measured as a percentage, and corresponds to the degree to which the accelerator is depressed. As Figure 3 shows, Driver B has almost 9% higher fuel economy compared to Driver A because his throttle position is lower more often; for 90% of the experiment, his throttle position is lower than 25%, while for driver A the corresponding number is almost 40%. This form of extrinsic variability can be exploited by appropriately tuning the engine.

In this paper, we explore the potential of personalized tuning by:
- designing a system called CARMA for conveniently and flexibly tuning cars (Section 3), and
- demonstrating that personalized tuning can exploit extrinsic variability to obtain significant fuel efficiency or responsiveness gains under different conditions (Section 5).

### 3 CARMA Design and Implementation

In this section, we begin with an introduction of automotive sensing and control, then provide an overview of CARMA, a system for personalized tuning. We follow this with a description of its components, and a discussion of the implications of our work. The next section evaluates CARMA in detail.

#### 3.1 Automotive Sensing and Control: Background

CARMA provides two capabilities, sensing and control of automobiles. Before we describe its design in detail, we introduce the kinds of sensing and control possible in modern automobiles and describe some terminology. CARMA provides higher-level abstractions for these capabilities, thereby enabling flexible automotive personalization.

##### 3.1.1 Sensing or Scanning

Modern cars are equipped with an Engine Control Unit (ECU), an electronic device which controls engine and transmission operations. The ECU can be queried for several “sensor” values using a standard called On-Board Diagnostics II (OBD-II). This standard was developed to provide vehicles with self-diagnostics and reporting capabilities and is mandatory in all cars sold in the United States after 1996.\(^3\)

The ECU is generally accessed using an OBD-II port. A few standardized messaging formats (all of them based on the Society of Automotive Engineers’ J1979 [40]) define a uniform method for communicating with the ECU. Over the years, multiple signaling protocols have been used over the OBD-II interface, but all cars sold since 2008 in the United States are now required to use the ISO 15765-4 (CAN) [34] standard signaling protocol. To interface a computer with the OBD-II port, an OBD-II-to-serial adapter is used for our setup. The port can also be accessed wirelessly by using an additional Serial-to-Bluetooth adapter.

Using OBD-II, it is possible to continuously sense (or scan), in near-real-time and while the car is operational, instantaneous values of several parameters: vehicle speed, engine RPM, throttle position, air flow, and so forth. While there are different ways to obtain these values, the most general approach is request-response: sending a request for a parameter value to the ECU elicits a response from the ECU containing the corresponding value. Using this, it is possible to write software that can scan a set of parameters periodically. Each parameter is associated with a parameter ID.

---

\(^2\)By throttle position we mean the extent to which the gas pedal is pushed down (measured in percentage). This is measured by an ECU sensor and logged during experiments.

\(^3\)The requirements are specified by the 1990 Amendments to the Clean Air Act [43]. In Europe the related EOBD standard was made mandatory in 2001 via the European Emission Standards Directive 98/69/EC [11].

---

![Figure 3. Throttle position CDF in the Switch experiment](image-url)
because most tuning systems provide very low levels of abstraction, allowing users to modify arbitrary combinations of parameters. Changing a parameter can have significant side effects. For example, igniting spark plugs earlier can improve performance, but may induce knock retard which generates undesired vibrations and can result in catastrophic engine failure. Some software gives the user extensive instructions on safe modifications, yet most commercial software struggles with the flexibility-safety trade-off. Existing tuning systems have erred towards giving users greater flexibility, and for this reason their use has been limited to experts who deeply understand the operation of their car. Typically, these experts make modifications by trial-and-error: they modify the car, scan all relevant parameters during a test drive, and iterate on this procedure until they are satisfied with the result. They also exchange their tuning experiences in online forums, promoting shared learning [7].

CARMA seeks a middle ground in this spectrum, and is designed to provide more tuning choices than the limited number of built-in modes, but in a manner that is more accessible to the average consumer.

### 3.2 Overview and System Architecture

CARMA is a smartphone-based system that provides make and model-independent programming abstractions for scanning and tuning automobiles. This capability, motivated by our findings in Section 2, permits the development of apps\(^4\) that take advantage of Internet connectivity and allow non-expert users to customize their vehicles for routes, terrain, and traffic conditions, and can result in improved performance, fuel efficiency, and safety. To our knowledge, no such capability exists today.

Existing tools have several shortcomings that motivate the need for CARMA. Because the OBD-II signaling protocols are standardized, many scanning tools are publicly available, some even for smartphones [22, 18]. These tools cannot be used for tuning. As discussed in Section 3.1.2, the tuning tools are often not extensible or modular and have been developed for less portable computer platforms (laptops or desktops). Moreover, they often only provide a graphical user interface for setting specific engine parameters. These tools are not programmable and cannot be invoked from another application. Thus, these tools have been designed assuming infrequent customization and expert knowledge.

In contrast, CARMA’s programmability ensures the development of applications which can be used by non-experts.

Our current CARMA prototype can be used to develop apps that perform many kinds of useful personalizations. For example, one app could automatically increase the responsiveness of a vehicle on a hilly route, by querying a geospatial database over the Internet to obtain the elevation gain on the route, then calculating the appropriate gear shift points to give the driver a greater sense of control. Another app could exploit the diversity in fuel efficiency (Section 2) under different route and traffic conditions and tunes the engine to the specific route or the expected traffic conditions. Yet another app could personalize the car’s parameters for a specific driver, by analyzing a GPS-tagged history of engine

---

\(^4\)We use this term to denote applications for smartphones.
parameters, thereby exploiting user diversity. Finally, other apps could promote safety, by analyzing an online database of routes to set vehicle speed and RPM limits commensurate with the route under consideration.

Moreover, these customizations can be applied at the granularity of a single trip, and can be changed for different drivers. This capability arises from CARMA’s portability: CARMA is implemented on smartphones for this reason. As mentioned above, the use of smartphones has another benefit: apps can consult online databases to make intelligent tuning decisions. That said, the use of smartphones is not a fundamental requirement of CARMA (any programmable, Internet-connected embedded computer with some capabilities for user interaction would suffice), but the ubiquity of smartphones makes them an attractive choice for CARMA.

The eventual aim of CARMA is to enable an ecosystem for the development of third-party tuning apps, and to make custom vehicle tuning as common, simple, and safe as mapping. Many other challenges must be overcome to achieve this vision, as discussed in Section 3.6.

The CARMA software architecture is illustrated in Figure 4. At a high level, CARMA is sub-divided into two major subsystems: the Scanning Subsystem that reads current sensory data from the phone (e.g., GPS) and from the car (e.g., RPM) and the Tuning Subsystem that allows engine parameter modifications. Each of these sub-systems exposes a procedural API and each communicates with the vehicle’s ECU using the OBD-II port, connected via Bluetooth. In addition, the Scanning subsystem accesses the smartphone’s sensors in order to tag sensor readings with time and location.

The Tuning subsystem is internally divided into three conceptually distinct parts. The ModAPI exports car-independent abstractions for modifying engine parameters. Apps access only this API and do not have access to other components of the Tuner subsystem. The ECU Binary Reader/Writer component performs low-level communication with the electronic control unit, and provides facilities to read and write the system parameters. The Binary Modification Subsystem updates the system’s firmware in response to calls made to the ModAPI. These components are model-dependent; our current implementation instantiates these components for a specific car model and year (Section 3.4).

CARMA is implemented on the Android 2.2 operating system, and is presented to apps as an Android Background Service. Apps communicate with the CARMA API using Android’s native RPC system. CARMA mediates concurrent access to the ECU, and ensures reliable reads and writes.

3.3 Scanner

Like many sensor APIs, the Scanner component provides API calls to select the “sensors” to retrieve data from, and the frequency with which to access them. In CARMA, each engine parameter is modeled as a distinct sensor, and apps are provided with an abstraction of a sensing session. Apps can call start_session(pid_set, frequency) to begin a sensing session; pid_set is a list of symbolic values for engine parameters to be retrieved. All PIDs in the pid_set are retrieved at the specified frequency: this ensures that the set of specified PIDs is sampled at approximately the same time. If the frequency is not specified, the Scanner attempts to retrieve pid_set as quickly as possible (see below).

Once this function is invoked, the Scanner repeatedly retrieves pid_set until a stop_session() is called. After retrieving this set, the Scanner returns it asynchronously to the application. Internally, the scanner performs format conversions (the “Formula” column in Table 1). In this manner, the Scanner abstracts both the details of the OBD-II signaling protocol and the messaging formats for applications.

Since not all standard PIDs are supported by all manufacturers, the Scanner exports a get_supported_PIDs() call which returns the PIDs supported on the currently connected vehicle. Furthermore, for convenience, other sensors on the phone are also modeled as special PIDs. For example, the GPS sensor is modeled using three distinct PIDs, one each for latitude, longitude, and altitude. Similarly, the clock and the sample interval (the time required to acquire the specified set of PIDs once) are also modeled as PIDs.

The Scanner is implemented as a service in Android and apps create Android listeners to receive sensor readings asynchronously. Multiple apps can concurrently access the Scanner component. Our default ScanApp, which uses the Scanner API, provides a user interface that lets the user select which PIDs to scan, and invokes the Scanner component. It also stores session identification information and all scan results in a database so that other applications can analyze this historical data to perform customizations or to later upload this data to a cloud service for trend analysis.

Our Scanner implementation supports two different commercially available OBD-II port adapters. The ELM327 [10] adapter includes built-in Bluetooth support, and abstracts some of the details of the OBD-II signaling. A more expensive alternative, the AVT adapter [3] requires a Bluetooth-to-serial dongle, but essentially provides a serial link abstraction to the ECU, so the scanner must implement the OBD-II signaling and message formats. The latter also supports tuning (as described below), while the former does not. We used the ELM327 adapter for the experiments in Section 2, and the AVT adapter for the experiments in Section 5.

The frequency at which engine parameters can be
scanned is limited by two factors. Recall that each PID must be individually scanned using a request-response protocol. First, the round-trip time of each request-response can vary significantly across manufacturers, depending on the capabilities of the ECU. Second, the time to retrieve one pid_set increases linearly with the number of PIDs, since each PID requires a separate request and response. For example, scanning 10 PIDs on one of our vehicles takes 850 ms. A small subset of manufacturers also provide a stream interface, in which one can specify the pid_set once, and the ECU returns the requested set periodically. For the same 10 PIDs, latency of retrieval is 250 ms using the stream interface.

3.4 Tuner: The Binary Reader/Writer

To our knowledge, most ECUs run a bare-minimum operating system for real-time control of engine operations, not a general purpose operating system. As such, they do not provide advanced facilities for updating the operating system in-place. Instead, modifications to the ECU generally require retrieving the entire system image including the code and data segments, modifying the image appropriately, and rewriting the modified system image. On our test car these modifications can only be performed when the car is stationary, since the ECU is disabled while being programmed and reboots afterwards. However, in some cases, aftermarket ECUs [4] permit parameter tuning while the engine is operational, a capability that can increase CARMA’s potential.

While this model holds across many manufacturers, the size and layout of the system image can vary across manufacturers, and even across different model years. This is because the system image depends upon the type of processor used in the ECU and on the specific version of the operating system, which may evolve from year to year as new engine features are introduced.

For CARMA, it is unsafe to allow apps to modify the entire system image directly since that can lead to unpredictable vehicle behavior. Moreover, it is important to abstract the process of retrieving and rewriting the system image so that applications need not deal with variability across manufacturers or model years.

At a high level, CARMA provides a transactional abstraction for modifying engine parameters. An app may call prepare() to begin the process of modification. CARMA’s ModAPI, discussed in Section 3.5, defines several API function calls to set or modify parameters. A commit() call commits all of the modifications made after the prepare() atomically; applications may also abort() a set of modifications. CARMA mediates access to these calls so that concurrent modifications are serialized.

A central challenge in the design of CARMA is the implementation of this API; in CARMA, the API is implemented by the Binary Reader/Writer components. The methods to update the binary images on vehicles are proprietary. As discussed before, the format of the binary image itself can vary across manufacturers and models. A comprehensive Binary Reader/Writer implementation must, therefore, contain model-specific implementations of these API calls.

Since our goal was to demonstrate the flexibility and benefits of CARMA, we have implemented a Binary Reader/Writer only for a small subset of vehicles. Specifically, CARMA currently supports about 24 different 1998/1999 General Motors models. All of these vehicles use the same ECU\(^5\). Our Binary Reader/Writer implementation is reverse-engineered from a commercial tuner implementation [7] for these models. In general, our reverse-engineering methods employ code disassembly and packet capture.

The implementation of prepare() loads the binary image from the ECU onto memory. There are two steps in the process: an authentication step, followed by a download step. Before executing these steps, prepare() attempts to acquire a lock, thereby ensuring exclusive access to the binary for modification.

The authentication step uses challenge-response: it involves receiving a 2-byte seed challenge from the ECU for which the correct 2-byte key response has to be sent back within a bounded time. The key generation algorithm depends upon the seed and an identifier for the operating system. Although we were able to reverse-engineer this algorithm, in many cases it is also possible to identify the correct key by using brute force since the seed request always triggers the same seed response in many ECU models [26].

In the download step, the Binary Reader/Writer first uploads a small bootstrap routine to the ECU. Subsequently, it sends requests to this bootstrap routine to successively download 1 Kbyte chunks of the binary image. For the ECU model we support, the image is 512 Kbytes in size.

The implementation of commit() is similar, but with one performance optimization: when writing the updated image, we only write the 128 Kbyte data segment since the code segment is never altered. Finally, abort() simply releases the lock and flushes the in-memory copy of the binary image.

The Binary Reader/Writer communicates with the ECU over Bluetooth. A Bluetooth-to-serial adapter is connected to the AVT OBD-II adapter [3]; AVT supports ECU writes, in addition to scanning.

3.5 ModAPI and Binary Modification

At the core of CARMA lies its ModAPI, a collection of predefined modifications (or mods). Our prototype currently supports eleven such mods, listed in Table 2. Examples of mods supported by ModAPI include: setting an RPM limit, setting a speed limit, changing the RPM values at which gear shifts occur, enabling exhaust gas re-circulation (if equipped), changing the spark timings, setting the speed at which fuel injectors are disabled, and so on. These mods were designed in consultation with a domain expert who believes that many responsiveness, fuel efficiency, or safety improvements can be accomplished with these modifications.

Intuitively, the ModAPI follows a layered approach that raises the level of abstraction relative to existing commercial tuning software (Section 3.1.2): rather than exporting low-level engine parameter settings, it encapsulates common tuning primitives into abstract mods. The rationale for this is similar to that for other high-level programming systems: by capturing many common tuning primitives, CARMA aims to support the development of rich apps that enable non-expert

\(^5\)GM Serv. No. 16236757
users to achieve performance and fuel efficiency goals without needing to understand the details of engine mechanics.

This approach also creates a separation of concerns that increases CARMA’s safety and interoperability. The Tuning subsystem, developed by specialists and car manufacturers, restricts sensitive and car specific functionality to the lower layers of the system, thereby promoting safety. Developers of user applications are restricted to the ModAPI, allowing applications to be developed in a safe and vehicle agnostic fashion. This separation of concerns can prevent poorly-designed or malicious apps from causing damage.

Our current set of mods is by no means complete, and we expect the ModAPI to evolve with experience. At any instant in the evolution of ModAPI, there will always be expert users for whom the supported mods may not suffice: these users will need low-level access to individual engine parameters provided by existing commercial tuning software. However, these deeply specialized modifications, once they have been fully tested and their impacts well understood, can then be encapsulated as mods in the ModAPI. Indeed, this is how the set of mods in Table 2 were obtained.

The challenge in the implementation of these modifications is in determining how to apply them to the binary image. For the ECU that we have chosen, the data segment of the binary image is a collection of over 500 different tables, each one of which governs one low-level engine parameter [21]. The rows and columns of each table specify the values for the parameter under different conditions. For example, the desired RPM in idle state depends on the engine coolant temperature (ECT) since ECT values below a certain threshold have a negative effect on the car’s performance. To capture this dependency, the idle RPM table specifies idle RPM settings as a function of engine coolant temperature; idling is set higher for lower temperatures thus increasing the amount of heat generation and consequently the ECT value.

The complexity of the semantics of the parameters and their dependence on other engine parameters is the reason why it is necessary to develop a higher level of abstraction (the mod) to modify engine parameters. In general, each mod’s implementation may modify several tables, in a manner that is consistent with the dependency between parameters. In practice, this dependency must be learned from engine specifications obtained from the manufacturer, or reverse-engineered by enthusiasts through trial and error. Thus, ModAPI effectively encapsulates domain knowledge in updating parameters, and provides a safe update abstraction which can be used by application developers without complete knowledge of the intricacies of engine operation.

Each mod allows the user to set, increase, or decrease a tuning variable (such as RPM limit, speed limit, or transmission shift points). Its implementation performs bounds checking of tuning variable values, and updates entries in one or more tables in a consistent way. It also performs format conversions and scaling, as necessary. All mods operate on the current system image obtained from a call to prepare() (Section 3.4). Mods invoked before a prepare() or after a commit() fail. We omit a detailed discussion of the implementation of each mod for space reasons.

3.6 Discussion

The current instantiation of CARMA was designed as a proof-of-concept to demonstrate the advantages of personalized tuning. Much needs to happen before personalized tuning becomes a reality. While our current implementation is constrained by the test vehicles we had at our disposal, we believe that CARMA can be extended to work on other vehicles. CARMA has already been designed to be vehicle-agnostic, so it can be extended to support other vehicles without modifying applications.

The scanning subsystem follows the OBD-II standard, but extended and enhanced PIDs are easy to add. A more significant challenge is presented in porting the tuning capabilities to other vehicles. CARMA’s Tuner was reverse-engineered from a proprietary implementation. In general, this careful reverse engineering may not scale unless auto enthusiasts can be incentivized to contribute to an opensourced CARMA. Manufacturers will likely need to actively participate in the development of personalized tuning systems and apps. There are encouraging signs that they may do this: a few manufacturers have started providing limited controls and access to car diagnostics using smartphone apps (e.g., GM’s OnStar MyLink [14], or Fiat’s EcoDrive [12]).

Another challenge is to converge to the right set of mods; this can happen only with iterative experimentation over a wider range of models. Moreover, the safety of each mod is an important issue. In our own development process, we have had to be extremely careful in developing and testing these modifications to ensure the safety of our experimenters. We set up a test bench in our laboratory (Figure 5), which consists of an ECU extracted from a 1998 Pontiac Grand Prix. We used an AVT-838 adapter [3] to interface with the OBD-II port. Although this setup appears to be bulky, we believe it is possible to embed a streamlined Bluetooth-to-OBD II port in future vehicles. We tested each modification with a 1998 Pontiac Grand Prix Automatic belonging to one of the authors. To minimize the risk of damaging the test vehicle, all mods and apps were validated by comparing the system binary generated by CARMA with one produced manually, from the same set of changes, using a commercial tuner [7]. Even so, we irreversibly damaged several ECUs while reverse-engineering the upload step.

Much more attention needs to be paid to security; as recent work has shown [26], the internal communications infrastructure of modern automobiles can be relatively easily breached, and a capability for wirelessly re-programming the ECU can amplify potential security holes. Our current implementation, being a proof-of-concept prototype, has minimal security provisions — password-based authentication for Bluetooth communication, and this is clearly insufficient. Furthermore, while our efforts have been focused mainly on creating the programming infrastructure, it will be necessary to create a developer ecosystem in the form of an appstore. Additional safety measures could be enforced within this ecosystem, such as application certification or mandatory kill-switches. Such measures would work together with the guarantees already provided by CARMA’s layering and API restrictions, thus minimizing the likelihood of damage by a misguided or malicious application.
### Table 2. ModAPI Tunable Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Affected Goals</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DisableEGR</td>
<td>✓</td>
<td>Disables exhaust gas recirculation to improve performance; re-enabling optimizes fuel efficiency.</td>
</tr>
<tr>
<td>EnrichMixture</td>
<td>✓</td>
<td>Enriches full throttle air-to-fuel ratio. When applied, increases horsepower and torque; factory defaults optimize for fuel efficiency.</td>
</tr>
<tr>
<td>EnrichPartThrottle</td>
<td>✓</td>
<td>Enriches cruise and heavy throttle air-to-fuel mixture, thus increasing horsepower and torque; factory defaults optimize for fuel efficiency to some extent.</td>
</tr>
<tr>
<td>IncreaseSparkTiming</td>
<td>✓</td>
<td>Increases power output by igniting the spark plugs slightly earlier; factory defaults are conservative and may allow for increases.</td>
</tr>
<tr>
<td>AdjustShiftDownPoints</td>
<td>✓ ✓ ✓</td>
<td>Adjusts down-shift points. Raising them makes the car feel more responsive; lowering provides greater comfort and fuel economy.</td>
</tr>
<tr>
<td>AdjustShiftUpPoints</td>
<td>✓ ✓ ✓</td>
<td>Adjusts shift points. Raising them makes the car feel more responsive; lowering provides greater comfort and fuel economy.</td>
</tr>
<tr>
<td>LimitRPM</td>
<td>✓ ✓ ✓</td>
<td>Sets an RPM limit.</td>
</tr>
<tr>
<td>LimitSpeed</td>
<td>✓ ✓ ✓</td>
<td>Sets a speed limit.</td>
</tr>
<tr>
<td>DecreaseShiftTime</td>
<td>✓</td>
<td>Makes the car shift faster; factory defaults provide greater comfort.</td>
</tr>
<tr>
<td>IncreaseShiftPressure</td>
<td>✓</td>
<td>Makes the car shift faster by increasing the shift pressure; factory defaults provide greater comfort.</td>
</tr>
<tr>
<td>SimulateManTransmission</td>
<td>✓</td>
<td>Prefers lower gears, emulating a manual transmission in hilly terrain and curvy routes; factory defaults provide greater comfort.</td>
</tr>
</tbody>
</table>

#### 4 CARMA Apps

Using the ModAPI, we have developed several smartphone apps that illustrate the power and flexibility of CARMA. These apps generally aim to achieve one or more of the following goals. Some apps improve fuel efficiency, which can have environmental and economic benefits. Car manufacturers already have to meet regulatory limits on fuel efficiency (e.g. CAFE in the United States [6]), which imposes strict design constraints. CARMA’s fuel efficiency apps can help make “run-time” mods towards the same goals. These apps attempt to shape driver behavior by appropriately setting trip-specific or driver-customized RPM and speed limits, or transmission shift points. Other apps maximize the car’s responsiveness, which enhances user satisfaction at the possible expense of fuel efficiency. To achieve this, multiple parameters can be altered including an increase in shift points, gear change speed, and speed limit. Finally, safety apps improve overall public safety, by enforcing RPM and speed limits. Such apps can be used to prevent rash driving by teens or valets. The following paragraphs describe some of these applications.

**TuneWizard** asks the user to qualitatively specify the characteristics of the route, and the desired performance goal, and automatically performs the appropriate mods. Specifically, users can specify the type of terrain (flat, low hills, or steep hills), the degree of congestion expected on the route (low, medium, or high), the traffic light density (no lights, few lights, and many lights), and the maximum speed limit on the route. Users then select one or more of three goals: fuel efficiency, performance, or safety; conflicting goals are resolved using a static preference order among these goals. Based on these inputs, TuneWizard applies the recommended mods for that route and the desired performance goals. TuneWizard’s mod choices are derived from discussions with a domain expert.

Many of TuneWizard’s choices can be automated, using the capability of the smartphone to access online databases. To demonstrate this, we developed a RouteEvaluator app, which, given a specific route, estimates the type of terrain and the speed limit. It does this by consulting the Google Directions API and Elevation API, which are both available as a web service. Given the start and end points of a trip, the web service returns a route specification as a list of polylines, with each polyline encoding a chain of waypoints approximating the actual route. Associated with each polyline is an estimated travel time. RouteEvaluator uses this information to derive an approximate speed limit, the maximum grade and the elevation gain along the route. These latter pieces of information are used to classify the terrain. Other online services, such as Navteq’s MapTP routing service [17], might be able to provide more accurate estimates, but we have left these to future work. Finally, RouteEvaluator is not yet integrated into TuneWizard, because we are currently exploring algorithms for estimating expected congestion and traffic.
light density along the route.

The ValetMode app only allows driving in the two lowest gears and enforces an RPM limit of 2000, thus preventing misuse of the car. The app’s name refers to well-publicized incidents of joy riding by valets.

Finally, the DriverCustomizer app is a reflective application that analyzes the past performance of the driver and automatically modifies car behavior to curb aggressive driving. This app analyzes the locally stored scan results from previous trips (recall that our ScanApp, Section 3.3, stores scan results in a database on the phone). It then determines the percentage of trip time for which a driver’s throttle position was over a certain preset limit (40% for our test car). If this percentage exceeds 5%, the driver is deemed aggressive (Figure 3) and an RPM cutoff limit of 3000 is applied if the user intends to drive on a flat terrain. Actively curbing aggression can reduce fuel consumption and relieve engine stress. More sophisticated analytics are possible, but we have left these to future work.

5 Experiments

In this section, we quantify, through experiments on our CARMA prototype, several features of CARMA. All our experiments on CARMA are conducted on a 1998 Pontiac Grand Prix, one of the models supported by our Tuner. Specifically, our experiments demonstrate:

- that CARMA has relatively low overhead, and provides a high enough level of abstraction to support flexible personalized tuning apps,
- that personalized tuning can provide significant fuel efficiency benefits, and
- that CARMA is expressive enough to design apps that promote safety or improve a car’s responsiveness, analyze driver behavior to curb aggressiveness, and simplify per-trip tuning through automation.

We conclude this section with a discussion of the implications of our results.

5.1 CARMA Microbenchmarks

In this section, we quantify some aspects of CARMA performance, and the ease of app development in CARMA. Table 3 summarizes the execution times for basic ECU operations used by the Scanner and Tuner. The high values for reading or writing an ECU image are mainly a result of bandwidth limitations of the connection between the tool and the ECU. In our test environment we used the SAE J1850 VPW protocol which supports a bandwidth of 10.4 kbps in standard mode and 41.6 kbps in high speed mode (see SAE J2534 [41]). In our experiments, we are actually able to achieve the nominal four-fold speed-up of the high speed mode, so we believe that these times can be reduced by more than an order of magnitude in newer cars equipped with the new ISO 15765-4 (CAN) bus which provides a bandwidth of up to 500 kbps [34].

Table 4 shows the size of the CARMA implementation, broken down by component and measured in lines of code (LOC). This table gives a rough measure of the complexity of the various subsystems. The Scanner, ModAPI and OBD-II logic change little across multiple ECU models, but the

<table>
<thead>
<tr>
<th>Interface</th>
<th>Operation</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensing</td>
<td>Scan a PID (single)</td>
<td>0.085 s</td>
</tr>
<tr>
<td></td>
<td>Scan a PID (streamed)</td>
<td>0.025 s</td>
</tr>
<tr>
<td>Tuning</td>
<td>Read a full ECU image</td>
<td>195 s</td>
</tr>
<tr>
<td></td>
<td>Write data section of an ECU image</td>
<td>51 s</td>
</tr>
</tbody>
</table>

Table 3. Processing times for basic ECU operations

<table>
<thead>
<tr>
<th>Component</th>
<th>LOC (w/ UI)</th>
</tr>
</thead>
<tbody>
<tr>
<td>OBD-II Logic</td>
<td>894</td>
</tr>
<tr>
<td>Scanner API</td>
<td>550</td>
</tr>
<tr>
<td>ModAPI</td>
<td>333</td>
</tr>
<tr>
<td>Binary Reader/Writer</td>
<td>411</td>
</tr>
<tr>
<td>Binary Mod. Subsystem</td>
<td>579</td>
</tr>
<tr>
<td>ScanApp</td>
<td>98 (453)</td>
</tr>
<tr>
<td>TuneWizard</td>
<td>214 (462)</td>
</tr>
<tr>
<td>RouteEvaluator</td>
<td>408</td>
</tr>
<tr>
<td>ValetMode</td>
<td>60 (214)</td>
</tr>
<tr>
<td>DriverCustomizer</td>
<td>76 (234)</td>
</tr>
</tbody>
</table>

Table 4. Implementation code size of CARMA components and sample applications. The numbers in parentheses denote total lines-of-code values including UI logic.

5.2 The Benefits of Personalized Tuning

CARMA was motivated by personalized tuning, the capability to tune a car at the granularity of a single trip. In this section, we quantify the benefits of personalized tuning by conducting experiments on a variety of traffic conditions and routes.

5.2.1 Methodology

All our experiments were conducted on a 1998 Pontiac Grand Prix belonging to one of the authors. In Section 2, we used two other car models (mainly for ease of experimentation) to motivate CARMA; these models are not supported by our Tuner. Our test car also shows variability with router and traffic conditions (Section 2), as discussed below.

Our experiments are conducted on several trips, each with qualitatively different characteristics. The trips covered different types of routes, levels of congestion, traffic light density, and terrain. Each trip is unidirectional from a fixed starting point to a fixed end point. The trips and their characteristics are summarized in Table 5.

6Trip 4a and 4b share the same route but exhibit different congestion levels.
For each trip, we conducted two runs: a baseline run on the test car without any modifications, and a mod run which uses a fuel efficiency modification suggested by TuneWizard. Recall that TuneWizard takes as input the characteristics of a route and automatically suggests the appropriate modifications. On a subset of these trips, we conducted baseline and mod runs with additional drivers to quantify the impact of driver behavior. All in all, our experiments logged over 1100 miles. Moreover, during each trip we continuously measured 13 PIDs using our ScanApp at a frequency of 4 Hz.

Our metric of performance is the improvement in fuel economy of the mod run over the baseline run. The fuel economy on a trip is measured using the scan data as discussed in a companion technical report [30].

Each of our trips uses a public thoroughfare along which it is difficult to conduct a controlled experiment. We were careful to conduct the baseline and mod runs at the same time of day, and under similar temperature and humidity (since these can affect engine air intake). Furthermore, all our trips are at least 20 min long, so that initial and final transients have minimal effect, and minor variability evens out. We discarded runs of the experiment where the experimenter noticed significant differences in traffic conditions (e.g., congestion) between the baseline and the mod run. To measure the variability across multiple runs, we repeated the baseline run for one of our trips 3 times and found that the mean deviation in fuel economy across those three runs was less than 1.54%. As we show below, our fuel economy improvements are well above this variability.

Finally, the experimenters were aware of the possibility of bias skewing the results and were instructed to drive normally. Also, our tests are conducted on only one vehicle. Conducting blind trials to remove bias or testing on other vehicles has insurance and legal implications that extend well beyond the scope of this work.

5.2.2 Results

For each of our trips, TuneWizard suggested the mods shown in Table 6. As discussed earlier, this app encapsulates domain expertise and suggests modifications based on the characteristics of the route chosen. All of these mods limit both the speed and the RPM based on the trip’s characteristics. For example, trip 1 is over hilly terrain with steep grades which may require higher RPM to surmount. Similarly, trip 4b in rush-hour traffic on a relatively flat route can be navigated using a much lower RPM limit.

Figure 6 plots the fuel efficiency for the baseline and the mod runs across each of our trips. First, observe that there is significant variability in baseline fuel economy across the different trips. This is qualitatively consistent with results presented, using other makes and models, in Section 2.

More important, the mod run’s fuel economy was at least 10% higher than that of the baseline run for a majority of trips. This is a significant improvement, especially considering that manufacturers engineer their cars for fuel efficiency to comply with fuel economy standards. It is also the central result of our paper, suggesting that personalized tuning may have significant economic benefits. The U.S. Bureau of Transportation Statistics [13] estimates that the annual fuel consumption of passenger cars is about 70 billion gallons. A 10% increase in fuel economy would save approximately 6.3 billion gallons, reducing fuel costs by over $19 billion per year and significantly reducing carbon emissions.

To understand the reasons for fuel economy improvements, we present a more detailed analysis of the scan data obtained from the experiments. Specifically, we focus on the results of the experiments on trip 3 and 4b since each of them illustrates a qualitatively different reason for fuel economy gains. On other trips, fuel economy gains are achieved through a combination of these two factors, and we omit a detailed analysis of these trips for space reasons.

As Table 6 shows, we enforce a speed limit of 55 mph on the mod run on trip 3, a high-speed interstate highway with little congestion. As a result, the average and maximum speed observed during the mod run drops significantly on this trip, as shown in Figure 7.

On this graph, the baseline run has fewer data points
since the driver reaches the destination earlier. Fuel efficiency gains are obtained in this scenario through two means. First, an enforced speed limit leads to a smoother speed curve especially on a decongested highway, since the driver is unlikely to encounter a slower car, thereby avoiding extra deceleration and unnecessary propulsive work, resulting in improving fuel efficiency [39]. Second, lower speeds are more fuel efficient than higher speeds [24]: the Transportation Energy Data Book [27] reports an average fuel economy loss of 17.1% when driving at a speed of 70 mph instead of 55 mph. Some of these fuel efficiency gains can be obtained through driver education; we discuss this in Section 5.4.

However a reduction of the speed is not the only reason for a better fuel economy. To see this, we turn to an analysis of the experiments for trip 4b. Since these experiments were conducted during rush hour, our speed limit was rarely reached; instead, congestion induced frequent stop-and-go behavior. On this trip, the RPM limit contributes to improved fuel economy, by preventing spikes in RPM (caused by frequent starting and stopping), or short bursts of high RPM values caused by frequently pushing the throttle by a large extent. Figure 8 shows the relative RPM frequencies for the baseline and the mod run. The distribution of RPM is clearly different for the mod run compared to the baseline run. In general, substantially more fuel is consumed at high RPM values. By preventing these situations we were able to improve the fuel efficiency by 13.7% for this trip.

Finally, we demonstrate that the relative improvements in fuel economy can vary significantly for different drivers in Figure 9. As the figure shows, for trip 1, one of the drivers exhibits slightly higher improvements in fuel efficiency than the other, but for trip 2 the roles are reversed and the differences are much more significant. These differences arise from variations in driving habits. CARMA attempts to impose a specific driver behavior with its RPM and speed limits. On some types of roads, one of our drivers may already closely match the targeted behavior (e.g., driving steadily on a highway) while the other may not, resulting in a higher potential for fuel efficiency gains in the latter case.

### 5.3 The Expressivity of CARMA

CARMA has uses beyond improving fuel efficiency. It can be used to increase responsiveness of a car. On each of our trips, we also conducted a responsiveness mod run in addition to the runs discussed above. TuneWizard increases responsiveness by changing gear shift timings and transmission shift points. This has the effect of staying at a lower gear until higher RPM values than normal, and shifting gears more quickly than normal. On terrain with frequent turns, these changes can make the car feel much more responsive. These changes, especially the gear shift timings, occur at finer time scales than supported by the scanning protocol our test vehicle uses, so we cannot quantify them; with the faster CAN protocol (found in newer cars) [34], they may be visible. However, one of our additional drivers (not an author of the paper) reported a perceived increase in responsiveness, and enjoyed being able to drive more aggressively. We have left to future work a user study that validates responsiveness mods.

CARMA can also promote safety. ValetMode limits the maximum RPM and prevents the car from shifting into higher gears thus implicitly enforcing a speed limit.\(^7\) To demonstrate this app, we ran an experiment where the driver was asked to maintain a constant throttle position on a car to which ValetMode had been applied. Figure 10 illustrates the behavior of ValetMode, and plots the RPM distribution during a test run as a function of time. Ordinarily, at a constant throttle position, the car should continuously accelerate and increase the revolution frequency. As the figure shows, the car’s RPM value drops once the limit is reached (a bouncing effect), demonstrating the impact of the RPM limiter. A similar behavior is observed for speed since the car cannot accelerate once a certain speed is reached with the gear and RPM limiter in force.

CARMA can also be used to design reflective applications that analyze driver behavior and tune engine parameters based on this analysis. DriverCustomizer (Section 4) curbs aggressive driving by establishing an RPM limit for drivers who exceed a fixed throttle position relatively frequently. Figure 11 shows the results from an experiment in which a driver was asked to drive aggressively. Without the curbs, the driver produces spikes in the RPM using high throttle positions. When the curbs are in place, the ECU cuts fuel to the engine when the RPM limit is reached even at high throttle position: this prevents excessive acceleration and fuel consumption. In this experiment, the DriverCus-

\(^7\)The LimitSpeed mod does not apply to this scenario since speed limits are not enforced below a certain threshold.
route calculations to automate the process of tuning based on driver behavior or those that perform complex oriented apps as well as those that customize car behavior also enables the development of responsiveness and safety's significant economic and social consequences:

- Efficiency gains: Even small gains in fuel economy can have
dramatic impact. Personalized tuning can often achieve more than what we allow for drivers who are not required to change their driving habits at all. Exploring this trade-off is beyond the scope of this paper.

- The RPM and speed cutoff limits in TuneWizard were suggested by our domain expert. These RPM limits were set generously to allow drivers to accelerate for safety reasons, such as when entering an interstate highway. Other safety concerns, such as an investigation of possible negative effects of using CARMA’s apps, safe though they may be, on a vehicle’s lifetime, is left to future work. That said, we believe the risk of such effects on lifetime to be negligible since car enthusiasts and fleet operators have been experimenting such mods for quite some time and with far more aggressive settings than what we allow. Moreover, many of the mods CARMA allows, such as the RPM and speed limiters, have no foreseeable side effects and simply enforce the kinds of behavior drivers may use under certain circumstances.

- The emergence of hybrids does not diminish the importance of CARMA. Extrinsic variability in the form of driver behavior is a crucial factor determining the fuel efficiency gains with hybrids. To ensure the regenerative braking, drivers must drive hybrids differently than they would normal cars [25]. Personalized tuning of hybrids to enforce some of these “hyper-miling” tricks on appropriate route segments can improve the achieved fuel efficiency. Whether personalized tuning can improve the energy efficiency of electric vehicles is an open question.

- Eventually, a much finer grained form of personalized modifications done on it. One consequence was a lower fuel economy at low RPM values compared to an unmodified car: for example, our test car would only achieve 35 mpg instead of the 40 mpg in an unmodified car. Thus, our results likely underestimate the gains achievable in unmodified cars.

CARMA’s personalized tuning capability effectively shapes driver behavior by changing the way the engine performs. Some of its benefits, such as fuel efficiency, could have been achieved with driver education (e.g., the eco-driving [9, 38] movement). While education can certainly help, CARMA can be an additional tool for ensuring socially responsible driving. With CARMA’s mods, drivers do not have to consciously practice responsible driving and may not need to change their driving habits at all. Exploring this trade-off is beyond the scope of this paper.

**Summary and Discussion**

Across a wide variety of driving conditions, CARMA’s personalized tuning can often achieve more than 10% in fuel efficiency gains. Even small gains in fuel economy can have significant economic and social consequences. CARMA also enables the development of responsiveness and safety-oriented apps, as well as those that customize car behavior based on driver behavior, or those that perform complex route calculations to automate the process of tuning.

Our test car had hardware after-market performance enhancers and Internet connectivity can be used to simplify the process of tuning. Currently, TuneWizard requires the user to manually specify the characteristics of the route. Instead, our RouteEvaluator can calculate many of these characteristics using online databases and can automatically be integrated with a navigation application, requiring the user to only input start and end points. Table 7 shows preliminary results for the RouteEvaluator: its classification of the maximum grade on a route matches that of a human classifying the route from memory, while the estimation of the maximum speed deviates from the posted maximum. This deviation results from our estimation of speed limits from Google Maps’ travel times; using a web service which provides explicit speed limits such as Navteq’s MapTP Service [17] can give better estimates.

**5.4 Summary and Discussion**

Across a wide variety of driving conditions, CARMA’s personalized tuning can often achieve more than 10% in fuel efficiency gains. Even small gains in fuel economy can have significant economic and social consequences. CARMA also enables the development of responsiveness and safety-oriented apps, as well as those that customize car behavior based on driver behavior, or those that perform complex route calculations to automate the process of tuning.

Our test car had hardware after-market performance enhancers and Internet connectivity can be used to simplify the process of tuning. Currently, TuneWizard requires the user to manually specify the characteristics of the route. Instead, our RouteEvaluator can calculate many of these characteristics using online databases and can automatically be integrated with a navigation application, requiring the user to only input start and end points. Table 7 shows preliminary results for the RouteEvaluator: its classification of the maximum grade on a route matches that of a human classifying the route from memory, while the estimation of the maximum speed deviates from the posted maximum. This deviation results from our estimation of speed limits from Google Maps’ travel times; using a web service which provides explicit speed limits such as Navteq’s MapTP Service [17] can give better estimates.

**Table 7. Comparison of the RouteEvaluator estimations with the real values**

<table>
<thead>
<tr>
<th>Route</th>
<th>Max. Grade (est./real)</th>
<th>Speed Limit (est./real)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8% / 9%</td>
<td>40 mph / 50 mph</td>
</tr>
<tr>
<td>2</td>
<td>3% / 2%</td>
<td>55 mph / 55 mph</td>
</tr>
<tr>
<td>3</td>
<td>0.6% / &lt; 1%</td>
<td>70 mph / 55 mph</td>
</tr>
<tr>
<td>4</td>
<td>0.6% / &lt; 1%</td>
<td>65 mph / 50 mph</td>
</tr>
</tbody>
</table>

Figure 10. RPM time series for ValetMode

![Figure 10. RPM time series for ValetMode](image)

Figure 11. RPM vs. Throttle Position for the DriverCustomizer

![Figure 11. RPM vs. Throttle Position for the DriverCustomizer](image)
tuning, where the car continually adapts, in real-time, to driver behavior and changes in route characteristics, can provide significant gains. This Utopian vision might be achievable with relatively modest ECU re-engineering, together with sophisticated on-board diagnostics and analytics.

6 Related Work

CARMA is, to our knowledge, the first platform that provides high-level interfaces for scanning and tuning a vehicle, thereby enabling the development of applications on smartphones for vehicle sensing, analysis and control. Figure 12 puts CARMA in the context of some of the prior work in this area, which we discuss below.

With OBD-II standardization, scanning software and systems are widely available. For example, users of Android smartphones can find apps that work with the ELM327 adapter. One such app is Torque [22], which allows users to view OBD-II scan data in real-time or to log it for offline analysis. Unlike these apps, CARMA supports tuning in addition to scanning.

Car manufacturers are also seeing the value in providing such services to their customers. Two such systems, General Motors OnStar [14] and MyFord Touch [16], have incorporated some forms of vehicle diagnostics and monitoring, allowing users to visualize scan data on the car’s dash-board or on a smartphone. Fiat’s eco:Drive [12] service provides feedback to drivers on their driving habits and provides tips on how to increase their fuel efficiency. None of these systems support personalized tuning.

Scanners have also been used in such projects as CarTel [33] and GreenGPS [31]. The CarTel project focuses on the networking and query abstraction challenges in mobile vehicular environments. The GreenGPS project analyzes vehicle sensor data to obtain fuel efficiency metrics. This data is then geotagged and crowd-sourced to create a database of “green” routes that can be queried. CARMA goes a step further: beyond sensing and data analysis, it enables tuning to achieve fuel efficiency or other goals.

There is a smaller body of work on tuning. Several companies offer customized tuners, which either replace the ECU entirely [23] or are hardware add-ons with limited personalization capabilities [8]. Some cars, like the Cadillac CTS [5], support performance modes, selectable on the dash-board, which modify gear shift points in order to emulate a more sporty handling behavior. These tools do add a small level of customization, but do not offer the same flexibility as CARMA as they reduce the user’s choice to a limited set of modes. Complementary to CARMA are methods that attempt to adapt automatically to user driving habits, without any user intervention [36, 20]. These approaches get closer to CARMA’s goal of driver-specific tuning, but lack the broader programmability that CARMA permits.

The complexity and inherent risk involved in tuning has led to few software tuners, and these are targeted towards mechanics and car enthusiasts. The proprietary DHP PowrTuner [7] and the open-source RomRaider [19] offer a full tuning suite, allowing one to download a vehicle’s binary image, modify tables using a UI, and upload it back to the ECU all within a single program. Unlike CARMA, these have been developed for a PC platform, and lack programmability; however, we have used the PowrTuner in the development and validation of our tuning subsystem. Another tool, Tiny Tuner [21], only supports off-line modifications to tables in binary image files (unlike CARMA which also transfers and installs images). Because it is open-source, Tiny Tuner was helpful in developing our ModAPI. Finally, recent work [26] has exposed the security risks inherent in car modification. As we have discussed in Section 3.6, mechanisms to secure modification are necessary before personalized tuning becomes a reality. Autoplug [29] is an emulator for ECU code updates that would enable off-line testing of mods for safety.

7 Conclusion

In this paper we described CARMA, a smartphone-based system that provides high-level abstractions for safe automotive sensing and control. Such a system can enable an application ecosystem for innovative car tuning apps; we have illustrated this potential by developing several proof-of-concept applications which allow users to customize their vehicle for specific routes, increase fuel efficiency and responsiveness, or even limit the vehicle’s abilities to prevent rash driving. Through experiments on our prototype we show that our apps do, in fact, achieve their desired effect, and that driver and route specific personalization can, in most cases, achieve more than 10% improvement in fuel economy.

Much work remains. For CARMA to be useful to a broader audience, further work is needed to support tuning on a wider range of car makes and models. Additionally, even though we have already demonstrated simple apps that analyze scanned data and react to modify vehicle behavior accordingly, improved data analysis and modeling could be developed to better shape driver behavior. Applications like the RouteEvaluator can be expanded to estimate other route characteristics such as congestion and traffic light density; ideally, the user should be able to just specify a route using a navigation application and the system should consult online databases to automatically and accurately derive all of the route-relevant features. Usability studies are needed to quantify the effects of response mods. Finally, for the CARMA vision to become reality, app development must drive the iterative improvement of ModAPI abstractions to find a level that best balances the complexity-flexibility trade-off.

Acknowledgments

Ben Gertner and Hilda Aldana worked on initial feasibility studies. Timothy Riesz and Michele Trendler helped
as additional drivers of the test vehicle during evaluation. Michael Riley from Advanced Vehicle Technologies, Inc. helped us understand the ECU communication. Our shepherd, Cormac Sreenan, and the anonymous reviewers provided insightful suggestions on improving the technical content and presentation of the paper. We thank them all.

8 References
[34] International Organization for Standardization. Road vehicles - Diagnostics on Controller Area Networks (CAN) - Part 4: Requirements for emissions-related systems, 2011.