Developing Heuristics for Resource Allocation and Utilization in Systems Design: A Hierarchical Reinforcement Learning Approach

Vikranth S. Gadi
School of Mechanical Engineering,
Purdue University,
West Lafayette, Indiana 47907
email: vgadi@purdue.edu

Zoe Szajnfarber
Engineering Management and Systems Engineering,
The George Washington University,
Washington, DC 20052
email: zszajnfa@gwu.edu

Jitesh H. Panchal[1]
School of Mechanical Engineering,
Purdue University,
West Lafayette, Indiana 47907
email: panchal@purdue.edu

Systems design involves decomposing a system into interconnected subsystems and allocating resources to teams responsible for designing each subsystem. The outcomes of the process depend on how well limited resources are allocated to different teams, and the strategy each team uses to design the subsystems. This paper presents an approach based on hierarchical Reinforcement Learning (RL) to generate heuristics for solving complex design problems under resource constraints. The approach consists of formulating systems design problems as hierarchical multi-armed bandit (MAB) problems, where decisions are made at both the system level (allocating budget across subsystems) and subsystem level (selecting heuristics for sequential information acquisition). The approach is demonstrated using an illustrative example of a race car optimization in The Open Racing Car Simulator (TORCS) environment. The results indicate that the RL agent can learn to allocate resources strategically, prioritize the subsystems with the greatest influence on overall performance, and identify effective information acquisition heuristics for each subsystem. For example, the RL agent learned to allocate a larger portion of the budget to the Gearbox subsystem, which has a higher-dimensional design space compared to other subsystems. The results also indicate that the extracted heuristics lead to convergence to high-performing car configurations with greater efficiency when compared to using Bayesian Optimization for design.

Keywords: Design heuristics, decision making, reinforcement learning, systems design

1 Introduction

In engineering organizations, complex design projects are often undertaken by teams with specialized expertise in various subsystems or components. A typical automotive company, for instance, has dedicated teams working on powertrain systems, chassis design, aerodynamics, and other vehicle subsystems [1]. These teams operate under the guidance of a program manager responsible for overseeing the entire vehicle development process, ensuring collaboration between teams, and integrating their respective contributions into a cohesive final product.

Product development processes dictate how organizations approach the design task, starting with partitioning the overall problem into manageable subsystems [2]. Resources, such as budgets, personnel, and computational tools, are then allocated to each team according to their specific requirements and the relative importance of their subsystems. Subsequently, teams work independently to optimize their respective subsystems within the allocated resources. Finally, the outputs from each team are combined and integrated to produce the complete system design.

The field of Multidisciplinary Design Optimization (MDO) has developed systematic approaches for optimizing complex systems involving multiple coupled disciplines, such as aircraft wing design where aerodynamics, structures, and controls are strongly coupled [3]. MDO techniques aim to find optimal solutions for specific problem instances by considering the interactions between various subsystems and employing optimization algorithms. For example, Kim et al. [4] illustrate the application of analytical target cascading (ATC) in automotive engineering through a ride and handling optimization problem. In their framework, the "higher-level" systems were defined as the vehicle-level ride and handling targets. These targets were then cascaded down to the "lower-level" systems, which comprised the front and rear suspension components, as well as the vertical and cornering stiffness parameters. Similarly, Kang et al. [5] implemented ATC for suspension design of a heavy-duty truck and body structure design of a bus, optimizing for weight and structural stiffness and explored the use of an aggregate objective function (AOF) to solve MDO problems in automotive design. Their work on suspension design for commercial vans considered ride comfort, controllability, and stability, using both hierarchical and non-hierarchical formulations of ATC. However, MDO methods are primarily focused on finding optimal designs for a given set of objectives and constraints for a specific problem, rather than extracting general design heuristics that can be applied across a broad class of problems. Heuristics represent distilled practical knowledge that guides actions in different situations, allowing designers to navigate complex design spaces and make timely decisions as requirements evolve. As organizations strive for more agile product development processes, the role of effective heuristics becomes increasingly crucial.

Heuristics, or experience-based guidelines and rules of thumb, have played a vital role in engineering design processes, guiding designers through the complexities of exploring vast design spaces and navigating trade-offs between conflicting objectives [6]. Traditionally, heuristics have been derived from the accumulated wisdom and insights of practitioners, distilled through years of hands-on experience in tackling specific design challenges [7,8]. Heuristics are employed across various stages of the design pro-

[1]Corresponding Author.
February 27, 2026

cess, from the creation of new design concepts to decision-making during optimization tasks [9]. However, while heuristics can be useful in managing design complexity, they can also lead to unintended constraints and cognitive biases if not carefully evaluated [10]. Unlike MDO, heuristics are non-optimal but satisficing solutions that can be applied to a class of related problems within a domain [6]. As engineering systems grow increasingly complex, with interdependencies between subsystems and rapidly evolving design requirements, the manual derivation of effective heuristics becomes an arduous and time-consuming task.

Large-scale systems design problems often defy closed-form solutions and analytical techniques, necessitating iterative exploration and evaluation of alternatives. To address this challenge, reinforcement learning (RL) offers a powerful computational framework for learning heuristics from experience through iterative interactions with the design environment [11]. Reinforcement learning [12] is a computational approach to learning from interaction on how to map situations to actions to maximize a scalar reward signal. RL agents can learn optimal policies through trial-and-error interactions with their environment, making them suitable for situations where the underlying problem is initially unknown or changes over time. A key advantage of RL algorithms is their theoretical guarantees of convergence to optimal solutions as the number of interactions with the environment increases. This provides a foundation for learning design heuristics that can outperform manually-derived rules. In our previous work, we showed how reinforcement learning can be used to extract heuristics for solver assignment in complex system design problems [11]. We used a Markov Decision Process formulation solved via Q-learning to systematically explore the space of possible solver-module assignments and identify effective heuristics. In this paper, we extend the approach to hierarchical design.

Beyond problem decomposition of the overall system into manageable sub-problems and assignment of specialized teams for solving each sub-problem, the design of modern complex engineering systems requires judicious allocation of limited budgets and resources across those sub-problems. Heuristics for decision-making at the system level for resource allocation, as well as at the subsystem level for design exploration and optimization, can help reduce the cost of systems design while increasing the performance of the resulting designs. To automate the extraction of heuristics, we present a hierarchical reinforcement learning framework that addresses this question by algorithmically generating heuristics based on interactions with the design environment. While black-box optimization techniques are a powerful tool for optimizing decision-making in complex systems, their application to large-scale, interdependent systems presents unique challenges that hierarchical organization is well-suited to address. A hierarchical structure enables effective resource distribution across interdependent subsystems, ensuring that decisions made at higher levels account for system-wide objectives while allowing lower-level components to focus on tailoring optimization strategies for each sub-problem based on its complexity.

The ability of Reinforcement Learning (RL) to dynamically allocate finite computational resources has been demonstrated in simulation-based optimization frameworks. For instance, RL has been employed in multi-fidelity frameworks to select between low- and high-fidelity simulations, minimizing computational cost while maximizing efficiency[13,14]. Bayesian Optimization (BO) is recognized for its ability to efficiently optimize expensive-to-evaluate functions by leveraging surrogate models [15]. Decision-making frameworks that integrate BO with Value of Information (VoI) formalism have shown promise for resource-aware, uncertainty-informed decisions[16]. Furthermore, the integration of RL with acquisition functions has enabled adaptive exploration-exploitation trade-offs in BO. RL-assisted Bayesian Optimization (RLABO), which formulates acquisition function selection as a Markov Decision Process (MDP), exemplifies this integration by dynamically switching strategies to improve performance [17,18]. Recent advancements have explored integrating RL with BO to address complex design problems. For instance, mixed-initiative Bayesian subgoal optimization combines RL and BO to refine subgoal setting in high-dimensional environments [19]. Simulation budget allocation is critical in optimizing resource use during the design process. Optimal Computing Budget Allocation (OCBA) has been validated for prioritizing simulation resources with efficient variance reduction and high accuracy in ranking/selecting designs under fixed budgets [20,21]. Dynamic extensions of OCBA incorporate Bayesian or Markov approaches for adaptive reallocation of resources [16,22].

While many BO based techniques for engineering design optimization treat the entire system as one black-box [15], our approach leverages the decomposition of complex systems into manageable subsystems. Extensions to ATC have facilitated non-hierarchical decompositions, enabling multidisciplinary system handling through parallelization methods such as Diagonal Quadratic Approximation [23]. Augmented Lagrangian Coordination (ALC) complements ATC by improving constraint handling and subsystem coordination. Pareto-optimal partitioning strategies further balance subproblem simplicity with coordination costs [24,25].

The proposed hierarchical RL framework builds on these foundations by addressing both system-level resource allocation and subsystem-level optimization strategies. By formulating systems design problems as multi-armed bandit (MAB) problems, this approach learns effective heuristics for budget allocation across subsystems while optimizing subsystem-specific information acquisition strategies. For example, the use of Thompson Sampling at both system and subsystem levels enables efficient exploration-exploitation trade-offs, reflecting advancements in RL-assisted BO methods such as RLABO [13,26]. [27] introduced a Reinforcement Learning-Based Sequential Batch-Sampling for Bayesian Optimal Experimental Design framework that leverages RL and Gaussian Process Regression (GP) to optimize expensive black-box functions under budget constraints but optimizes the black-box functions as a single unit without any decomposition. Additionally, the hierarchical decomposition parallels ATC's principles while extending applicability to non-convex problems through learned heuristics. The integration of dynamic budget allocation strategies further complements OCBA's objectives by incorporating learning-based heuristics. Chen and Heydari [28] established a deep RL framework for dynamic resource distribution in cooperative-competitive Systems-of-Systems (SoS), decoupling system-level resource management from subsystem learning while preserving agent autonomy. Their interpretable deep RL policy optimized operational coordination through strategic allocation of energy and bandwidth resources, though focused on real-time operational decisions rather than design optimization.

To illustrate the proposed approach, we use the design optimization of a race car in The Open Racing Car Simulator (TORCS) environment [29] as an example application. TORCS provides a realistic simulation platform where different car configurations can be evaluated, enabling the study of heuristics for design under resource constraints. The race car design problem involves parameters across multiple subsystems such as the gearbox, brakes, and suspension while working within a fixed budget for design evaluations.

The paper is structured as follows. Section 2 provides the necessary background and preliminaries, including problem representation, sequential information acquisition in decision-making (SIADM), and the multi-armed bandit (MAB) framework for reinforcement learning. Section 3 details our approach incorporating systems engineering principles and hierarchical RL for systems design problems. Section 4 presents an example application of our methodology to the optimization of car setup parameters in the TORCS environment. Section 5 reports the experimental results, analyzing the learned heuristics and comparing our approach's performance to a Bayesian optimization method. Finally, Section 6 presents the implications of our findings, potential limitations, and opportunities for future research.

2 Preliminaries

Engineering design problems involve finding solutions that optimize performance while adhering to constraints. This section outlines a model for representing such problems, breaking them into manageable sub-problems, and problem-solving in Section 2.1. The sequential information acquisition decisions for problem-solving are described in Section 2.2. We describe reinforcement learning in the multi-armed bandit problem in Section 2.3.

2.1 Problem Representation. A design problem is denoted as $\mathscr{P}$. Problem-solving involves finding a solution X within the design space $\mathscr{X}$. The design space encompasses all possible combinations of design variables (e.g., material choices, geometric parameters, component configurations). Each solution $X \in \mathscr{X}$ maps to a point $Y \in \mathscr{Y}$ within the performance space. This performance space comprises the metrics used for evaluation. A preference function $V(Y)$ encodes trade-offs between potentially conflicting objectives, ranking the desirability of different solutions. For example, in designing a powerplant, the performance space $\mathscr{Y}$ could include dimensions like power output, fuel efficiency, emissions, and cost. The design space $\mathscr{X}$ encompasses variables such as engine type (internal combustion, turbine), displacement, valve timing, turbocharger specifications, etc. The preference function $V(Y)$ would balance power, efficiency, and emissions targets, likely informed by regulations and market demands.

A problem $\mathscr{P}_i$ can be broken down into sub-problems $\mathscr{P}_{ij}$, which carry their own subset of objectives. For example, an automobile design problem might be decomposed into sub-problems such as drivetrain (gearbox ratios, differential selection, driveshaft sizing) and chassis (structural rigidity, suspension configuration, material choices) [1]. Complex design problems are often solved through hierarchical decomposition into a system-level problem and subsystem-level problems. A systems designer is typically tasked with allocating a fixed budget across the various subsystems that constitute the overall system. At the subsystem level, teams of designers employ the Sequential Information Acquisition Decision-Making (SIADM) process outlined in Section 2.2 to iteratively explore design alternatives and optimize their respective subsystems.

2.2 Sequential Information Acquisition in Decision-Making (SIADM). The subsystem designers use an abstraction of the engineering design process as a sequential decision-making process [30,31]. The subsystem team's goal is to locate the design solution within the design space $\mathscr{X}$ that yields the optimal performance as defined by the preference function. This is achieved by strategically acquiring information about the performance of various design alternatives throughout the search process. At each iterative stage, the designer is tasked with making several decisions such as determining the next design $X \in \mathscr{X}$ to evaluate, selecting the method for evaluating the selected design's performance, and deciding whether sufficient information has been collected to terminate the search process. The process of information acquisition for iterative design often involves leveraging multiple sources (e.g., prototypes, simulations) with varying costs and levels of uncertainty.

The design exploration process is fundamentally constrained by a fixed budget B during each iteration. This budget might represent financial limitations, computational resources, energy constraints, or other restrictions on the total number of evaluations permissible. Within this sequential decision-making process, the designer begins with an initial state of knowledge $\mathscr{K}$ informed by prior beliefs and existing observations $D = \{(X_i, Y_i)\}_{i=1}^{n}$. In each iteration, the designer makes the following decisions:

- **Decision to choose function learning model**: Selects a model to represent the relationship between design variables and the observations D.
- **Decision to choose next design point**: Applies a heuristic to select the next design $X_i \in \mathscr{X}$ for evaluation, and obtains a performance estimation ($Y \in \mathscr{Y}$).
- **Decision to choose when to stop the information acquisition process**: Decides whether to continue exploration based on the updated state of knowledge ($\mathscr{K}$) and any constraints imposed by the remaining budget B.

The design process is constrained by a predetermined budget, which imposes limitations on the total number of design evaluations that can be performed.

2.3 Reinforcement Learning (RL) using Multi-Armed Bandit (MAB). Multi-armed bandit (MAB) [12] is a classic Reinforcement Learning (RL) problem, where the environment is represented by a set of K arms. Each arm k is associated with a fixed but unknown reward distribution R. The decision-maker (agent) interacts with this environment by repeatedly choosing arms to pull. When an arm is pulled, the agent receives a reward. The goal of the agent is to devise a strategy that maximizes the total reward obtained over time. To achieve this, a balanced approach is necessary. An effective strategy must exploit the arms that are currently believed to be the most promising to yield high rewards. However, it must also intelligently explore other arms that have the potential to uncover better long-term solutions.

To be modeled using the MAB framework an environment should satisfy the following conditions:

- **Stationary Reward Distributions:** The underlying reward distribution for each arm is assumed to be fixed over time.
- **Independent Arms:** The reward for pulling one arm does not directly affect the reward distributions of other arms (with some exceptions in contextual bandit variations).

Thompson Sampling [12] provides a powerful Bayesian approach to tackle the MAB problem. The fundamental idea is to maintain belief distributions over the expected rewards of each arm. Typically, beta distributions are used to represent these beliefs due to their flexibility and computational advantages. During each decision step, the Thompson Sampling algorithm samples a potential reward value from the belief distribution of all arms. The arm with the highest sampled reward is then chosen for evaluation. After the actual reward is observed, the belief distribution for the chosen arm is updated using Bayes' rule.

Let the true (but unknown) parameter representing the reward distribution of arm k (e.g., the mean for a Gaussian) be θ_k. The history of interactions (arm pulls and rewards) up to time step t is d_t. The posterior distribution of θ_k after observing data d_t is $P(\theta_k \mid d_t)$. This belief distribution is typically initialized using a prior distribution. First, for each arm k, draw a sample $\hat{\theta}_k$ from its current posterior distribution $P(\theta_k \mid d_t)$. Then, we choose the arm a_t with the highest sampled value: $a_t = \text{argmax}_k \hat{\theta}_k$. We pull arm a_t and obtain reward R_t. Next, we update the posterior distribution $P(\theta_{a_t} \mid d_{t+1})$ using the observed reward R_t, incorporating the new data via Bayesian inference.

The benefits of Thompson Sampling include its inherent encouragement of exploration as arms with less certainty (broader belief distributions) have a non-zero probability of being selected. Additionally, Thompson Sampling is adaptive, with the belief distributions evolving based on new information, allowing it to adjust to potential changes in true reward distributions.

3 Approach

In this section, we propose an approach that integrates systems engineering principles with reinforcement learning (RL), particularly MAB, to generate heuristics. We begin by outlining the application of systems engineering methodologies in managing large-scale design tasks, followed by a discussion on how RL can optimize decision-making processes at both system and subsystem levels.

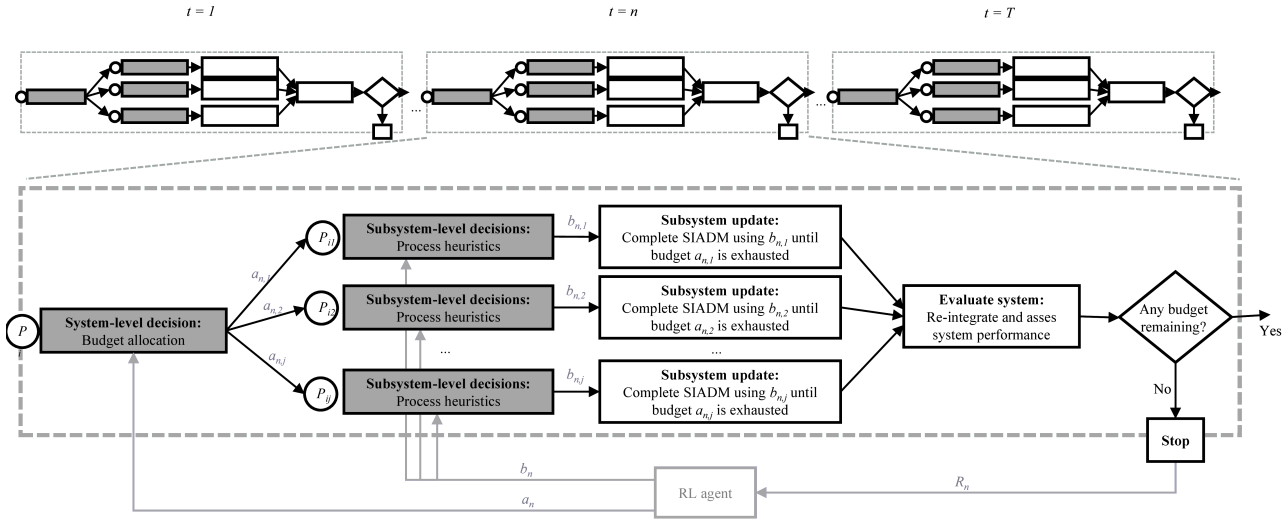

Fig. 1 Resources are allocated by the system engineers and the subsystems engineers use SIADM to carry out their design tasks, and the RL agent learns heuristics.

3.1 The System Design Process. Systems engineering provides a structured methodology for addressing the inherent complexity of systems design problems. At its core, systems engineering emphasizes the decomposition of complex systems into manageable subsystems or components, facilitating parallel problem-solving efforts and efficient resource allocation [32]. It emphasizes the decomposition of a complex problem $\mathscr{P}_i$ into interconnected sub-problems $\mathscr{P}_{ij}$, which can be addressed by specialized teams or subsystems. The first step in the systems design process is the decomposition of the overall design problem into interconnected sub-problems or subsystems. Each subsystem is responsible for addressing specific aspects of the design, allowing for focused problem-solving efforts and modular design approaches. A key task is the allocation of the budget B across the various sub-problems. This budget allocation must consider the estimated cost of solving each sub-problem and its relative importance to the overall system's performance.

While this work does not directly address how to decompose the problem into subsystems, it builds upon previous work in Solver-Aware System Architecting (SASA) [11]. Our focus is on the allocation of budget to pre-defined subsystems, optimizing resource distribution within an already decomposed problem structure. The decomposition itself is assumed to be derived from prior domain knowledge or established organizational structures.

Once sub-problems are defined and resources are allocated, the teams responsible for each sub-problem face the sequential decision-making process outlined in Section 2.2. They employ various heuristics to choose the function learning model, choose the next design point, and decide when to stop. After the teams work through their allocated budget to address their respective sub-problems, the systems engineer re-integrates the proposed solutions. This re-integration allows for the assessment of the overall system's performance. This process is shown in Figure 1.

3.2 Hierarchical Reinforcement Learning to Learn Heuristics. In the problem-solving process outlined above, decisions are made at two different levels: (i) at the system level by the system engineer, and (ii) at the subsystem level by the subsystem teams. The problem is formulated at both system and subsystem levels to reflect the hierarchical nature of complex engineering design processes. This two-level approach allows for efficient resource allocation at the system level, mirroring how organizations typically distribute budgets across different departments or teams, as well as, specialized optimization within each subsystem, acknowledging that different components may require different strategies for improvement. The lower-level decision is formulated as choosing a process heuristic such as the information acquisition function because this choice directly impacts how efficiently each subsystem explores its design space within its allocated budget. Specifically, we consider three well-established acquisition functions as our process heuristics: Upper Confidence Bound (UCB) [33], Expected Improvement (EI) [34], and Probability of Improvement (PI) [35]. Points are chosen where the predictive mean is high (exploitation) and where the variance is large (exploration). Different information acquisition functions have varying exploration-exploitation trade-offs, which can significantly affect the quality of solutions found within resource constraints. By learning which of these heuristics performs best for each subsystem, our approach adapts its optimization strategy to the specific characteristics of each component of the overall system.

Hierarchical reinforcement learning (RL) using the Thompson Sampling policy provides a framework to learn effective decision-making heuristics at multiple levels of a complex system. At the system level, hierarchical RL can guide overarching decisions that impact the entire system, while at the subsystem level, it can optimize decisions within individual components or subsystems. RL, as described in Section 2.3, can be initiated as follows to solve MABs at both levels, where we learn all policies concurrently.

3.2.1 System-level MAB. The arms correspond to all possible combinations of budget allocations, and the reward corresponds to the quality of improvement (performance) achieved by each allocation. Let $a_{t,i}$ represent the allocation of budget for subsystem i at time step t, where $i \in \{1, 2, ..., N\}$ and N is the number of subsystems. Let $\theta_{a_{t,i}}$ denote the true parameter representing the reward distribution for allocating budget $a_{t,i}$ to subsystem i. At each iteration t, the Thompson Sampling algorithm samples a potential reward value from the belief distribution of all possible budget allocations for each subsystem:

$$\hat{\theta}_{a_{t,i}} \sim P(\theta_{a_{t,i}} | d_{t,i}),$$

where $d_{t,i}$ represents the history of interactions (budget allocations and rewards) for subsystem i up to time step t. The budget allocation $a_{t,i}$ with the highest sampled reward is then chosen for evaluation

$$a_{t,i} = \arg\max_{a} \hat{\theta}_{a_{t,i}}.$$

After the actual reward is observed, the belief distribution for the chosen budget allocation is updated using Bayes' rule.

3.2.2 Subsystem-level MAB. The arms correspond to different process heuristics available for each subsystem, and the reward corresponds to the quality of improvement (performance) achieved by each heuristic. Let $b_{t,j}$ denote the choice of process heuristic j at time step t, where $j \in \{1, 2, ..., M\}$ and M is the number of combinations of process heuristic. Let $\theta_{b_{t,j}}$ denote the true parameter representing the reward distribution for selecting process heuristic $b^i_{t,j}$ for subsystem i. At each iteration t, the Thompson Sampling algorithm samples a potential reward value from the belief distribution of all possible process heuristics for each subsystem:

$$\hat{\theta}_{b_{t,j}} \sim P(\theta_{b_{t,j}} | d_{t,j})$$

where $d_{t,j}$ represents the history of interactions (process heuristic selections and rewards) for process heuristic j up to time step t. The process heuristic $b_{t,j}$ with the highest sampled reward is then chosen for evaluation within each subsystem:

$$b_{t,j} = \arg\max_b \hat{\theta}_{b_{t,j}}.$$

After the actual reward is observed, the belief distribution for the chosen process heuristic is updated using Bayes' rule. The conditions for the MAB framework in Section 2.3 are satisfied in our approach as follows:

- **Stationary Reward Distributions:** The reward distributions associated with each action (i.e., budget allocation or heuristic selection) can be assumed to be stationary. This assumption is reasonable because the underlying design problem and the mapping between design variables and performance metrics remain fixed throughout the optimization process. While the agent's knowledge and beliefs about these reward distributions may evolve over time, the true distributions themselves do not change.
- **Independent Arms:** The formulation of our approach ensures that the rewards obtained from selecting one action (e.g., a specific budget allocation or heuristic) do not directly impact the reward distributions of other actions. This independence condition holds because each subsystem is optimized independently within its allocated budget, and the heuristic selections within a subsystem do not directly influence the performance of other subsystems.

By concurrently learning policies at both levels, the hierarchical RL agent can efficiently explore the decision spaces, adapt strategies based on observed outcomes, and learn effective decision-making heuristics to optimize system performance. Figure 1 visually represents the proposed framework, which integrates resource allocation, subsystem design tasks, and learning through feedback. At each time step t, system engineers allocate resources $a_{t,i}$ to each subsystem i, where $i = 1, 2, \ldots, N$. These allocations are informed by the engineers' belief about the potential performance of each subsystem, which is updated iteratively based on observed outcomes. The resource $a_{t,i}$ represents the budget or effort assigned to subsystem i at time t. Each subsystem engineer utilizes the allocated resources $a_{t,i}$ to perform design tasks aimed at improving subsystem performance. Subsystem engineers use Sequential Information Acquisition in Decision-Making (SIADM) informed by $b^i_{t,j}$ to optimize their respective design tasks. The system engineers employ a probabilistic learning mechanism, such as Thompson Sampling, to guide resource allocation decisions. At each time step, they sample from the posterior distributions of rewards for each subsystem and allocate resources based on expected performance improvements. This mechanism balances exploration (allocating resources to less-explored subsystems and information acquisition functions) with exploitation (allocating resources to subsystems and their corresponding information acquisition functions with known high rewards). After observing the rewards r_t at time t, the system engineers update their belief distributions. This updated belief informs resource allocations in subsequent iterations, creating a feedback loop that improves decision-making over time.

Table 1 Design parameters and subsystems for car setup optimization in TORCS.

Subsystem	Parameter	Unit	Min	Max
Gearbox	Gears/2 ratio	SI	0	5
Gearbox	Gears/3 ratio	SI	0	5
Gearbox	Gears/4 ratio	SI	0	5
Gearbox	Gears/5 ratio	SI	0	5
Gearbox	Gears/6 ratio	SI	0	5
Aerodynamics	Rear wing angle	deg	0	18
Aerodynamics	Front wing angle	deg	0	12
Brake system	Front-rear brake repartition	SI	0.3	0.7
Brake system	Max pressure	kPa	100	150000
Suspension	Front anti-roll bar spring	lbs/in	0	5000
Suspension	Rear anti-roll bar spring	lbs/in	0	5000
Wheel	Front ride height	mm	100	300
Wheel	Front toe	deg	-5	5
Wheel	Front camber	deg	-5	3
Wheel	Rear ride height	mm	100	300
Wheel	Rear camber	deg	-5	2

4 Example Application: Race Car Design in the TORCS Environment

In this section, we present the application of our proposed approach to the optimization of car setup parameters in the TORCS (The Open Racing Car Simulator) [29] environment. TORCS supports two different challenging tasks which include developing an adaptive racing controller and optimizing the car setup design. Additionally, it runs on various operating systems and contains many different cars and tracks, a simple damage model, aerodynamics, etc. TORCS provides a realistic simulation platform for evaluating different car configurations and tuning parameters, making it an ideal testbed for our methodology. Table 1 presents the design parameters and subsystems relevant to car setup optimization in TORCS, including gearbox ratios, aerodynamics settings, brake system parameters, and suspension characteristics. We use the following settings in TORCS to perform experiments: car (car1-stock1), driver (berniw-1), and race mode (quickrace). To instantiate an environment, we randomly select one track from the three available categories: road (E-Track 6), oval (C-Speedway) and dirt (Dirt-3). In the output of each TORCS simulation, we receive information regarding the time taken to complete the race, damage to the vehicle and top speed. Performance is given by the average speed. The car operates using a pre-defined policy designed to optimize racing performance based on sensor inputs and a control strategy. This policy integrates components such as track sensors for maintaining alignment with the racing line, opponent sensors for collision avoidance and overtaking, and speed and angle sensors for dynamic control. The control strategy includes proportional steering adjustments, acceleration and braking based on detected track curvature, and gear shifting according to predefined thresholds. Additionally, a finite state machine governs decision-making in scenarios like overtaking or recovery from off-track situations, ensuring adaptive and consistent performance during races. The damage is modeled through a physics-based collision system that calculates deformation and mechanical degradation based on impact forces. High-speed collisions result in significant structural damage, while repeated minor impacts accumulate over time. This damage directly affects vehicle dynamics, such as reduced speed or impaired handling, simulating realistic consequences of collisions.

For the simulations, we make the following simplifying assumptions.

Assumption 1 (Continuous design space). The design performance $f(x)$ is an 11-dimensional continuous function of design

parameters, or input, $x \in \mathcal{X}$, where $\mathcal{X} = [0,1]^{11}$.

Assumption 2 (No noise in the output, y_i). We assume that there is no measurement error in design evaluations, such that $y_i = f(x_i) + \epsilon_m$ and $\epsilon_m = 0$.

Assumption 3 (Single information source). Typically in a SIADM process, design evaluations are performed using multiple information sources with different uncertainties and costs. Here, we use a single information source with a fixed cost per evaluation.

These assumptions limit the scope of applicability of the results from the experimental study to the specific scenario. However, the overall approach described in this paper and decision models in the next section are general and can be applied in the future to more complex design problems with more information sources, and added noise.

We use a Gaussian process regression (GPR) to map the inputs x_i onto the outputs y_i [36, Chapter 3]. GPR is a non-parametric regression method that is completely defined by its mean, $\mu(x)$, and covariance function, or kernel, $k(x,x')$ [37, Chapter 2]. It defines a distribution over functions $f : \mathcal{X} \to \mathbb{R}$ that maps inputs, x_i, to outputs y_i as a random draw from a Gaussian distribution:

$$f \sim GP(\mu_n, k_n), \tag{1}$$

where μ_n and k_n are the posterior mean and covariance functions after n observations, respectively.

Since GPR provides a flexible approach to prediction, we utilize it with an RBF kernel to associate which inputs x_i are likely to have similar outputs y_i. In choosing the next design point, three strategies are used: upper confidence bound [33], expected improvement [35], and probability of improvement [35].

The upper confidence bound model is defined as:

$$\mathrm{UCB}_i(x) = \mu_i(x) + \xi\sigma_i(x), \tag{2}$$

where $\xi \geq 0$ is the exploration parameter, and $\mu_i(x)$ and $\sigma_i(x)$ are the predictive mean and standard deviation, respectively [33]. Higher values of ξ emphasize exploration, while $\xi = 0$ represents a pure exploitation strategy. The expected improvement model computes the expectation of improvement (EI) for each design point relative to the currently best-observed point, y_i^* [34].

The EI values are calculated by:

$$\mathrm{EI}_i(x) = \frac{\mu_i(x) - y_i^*}{\sigma_i(x)}\Phi\left(\frac{\mu_i(x) - y_i^*}{\sigma_i(x)}\right) + \sigma_i(x)\phi\left(\frac{\mu_i(x) - y_i^*}{\sigma_i(x)}\right), \tag{3}$$

where Φ and ϕ are the cumulative distribution function and probability density functions of the standard normal distribution, respectively.

The Probability of Improvement (PI) heuristic chooses the next design point x based on the probability that its performance y will be better than the current best-observed performance y^*. The PI value is calculated as:

$$\mathrm{PI}_i(x) = \Phi\left(\frac{\mu_i(x) - y_i^*}{\sigma_i(x)}\right) \tag{4}$$

Where $\Phi(\cdot)$ is the cumulative distribution function of the standard normal distribution, $\mu_i(x)$ is the predicted mean of the Gaussian process at x, and $\sigma_i(x)$ is the predicted standard deviation at x.

In all strategies, a $GP(0, k_{\mathrm{RBF}})$ is used to learn the objective function and predict the mean and variance at every $x \in \mathcal{X}$ based on the observed input-output pairs.

We formulate the car setup optimization problem in TORCS as a hierarchical decision-making process, where decisions are made at both the system and sub-system levels. At the system level, the decision-making process involves allocating a fixed budget B across the various subsystems of the car setup optimization problem. This budget allocation determines the resources (e.g., computational time, simulations) dedicated to optimizing each subsystem. The available actions at the system level correspond to different combinations of budget allocation across the subsystems. Let N be the total number of subsystems, and $a_{t,i}$ denote the budget allocation for subsystem i at time step t, where $i \in \{1, 2, ..., N\}$. The **system-level actions** are represented as arrays of length N, with each element specifying the budget allocation for the corresponding subsystem. The budget allocations represented by these arrays determine the resources available for optimizing each subsystem during the design process. By strategically selecting the budget allocation at each time step, the system-level decision-making aims to maximize the overall performance of the car setup design while adhering to the fixed budget constraint.

Let $b_{t,j}$ denote the choice of information acquisition heuristic j for subsystem at time step t, where $j \in \{1, 2, ..., M\}$ and M is the number of information acquisition heuristics. The **subsystem-level actions** corresponding to selecting one of the available heuristics (UCB, EI, or PI) for each subsystem i within the allocated budget. For selecting the next design point x_{i+1}, an acquisition function assigns a value to every design based on the observed history. We explore three decision models based on different acquisition functions.

In the TORCS car setup optimization environment, the reward signal is derived from the performance of the car under the chosen configuration. Specifically, the **reward** is defined as the normalized average speed achieved by the car during a race, contingent upon the absence of damage. If the car sustains damage during the race, the reward is set to zero. Mathematically, the reward R_t at time step t can be expressed as:

$$R_t = \begin{cases} v_{avg} & \text{if no damage occurs} \\ 0 & \text{if damage occurs} \end{cases} \tag{5}$$

where v_{avg} is the normalized average speed achieved by the car during the race. The rewards are provided to the system and subsystem-level decision-makers after the allocated budget has elapsed, reflecting the overall performance of the car setup configuration resulting from the selected actions. The goal is to learn effective decision-making policies that maximize the cumulative reward, thereby optimizing the car setup for high-speed performance while avoiding damage.

In the context of the TORCS car setup optimization problem, the conditions for the MAB framework mentioned in Section 2.3 are justified as follows:

- **Stationary Reward Distributions:** The reward distribution for each action (i.e., budget allocation or heuristic selection) can be considered stationary. This is because the underlying simulation environment, including the track characteristics, car dynamics, and performance metrics, remains fixed throughout the optimization process. While the agent's knowledge of the reward distributions evolves, the true distributions themselves do not change, as they are determined by the simulation environment.

- **Independent Arms:** The reward obtained from selecting a specific budget allocation or heuristic for one subsystem (e.g., gearbox or aerodynamics) does not directly impact the reward distributions of other subsystems. Each subsystem is optimized independently within its allocated budget, and the heuristic selections within a subsystem do not directly influence the performance of other subsystems.

To learn these policies, we employ Thompson Sampling, a Bayesian approach to the multi-armed bandit problem. Thompson Sampling leverages Bayesian inference to balance exploration and exploitation, enabling efficient learning of optimal car setup configurations within the constrained budget environment. The budget represents computational resources dedicated to evaluating different car configurations in The Open Racing Car Simulator (TORCS). It indicates the total number of evaluations permissible during the optimization process of that subsystem. Subsystems

must optimize their performance while adhering to the budget constraint. Each subsystem operates within its allocated share of the overall budget. This allocation determines how much computational effort corresponding to the number of times TORCS can be simulated on exploring its design space.

5 Results

We conduct experiments to evaluate the performance of our approach across 50 runs, each starting with a random initialization of the state. The total budget allocated for each episode is set to 1250. We present the results and analysis of our experiments, highlighting the impact of different budget allocations, information acquisition strategies, and subsystem-level decision models on the performance of car setups in TORCS.

5.1 Reinforcement Learning. We consider the TORCS car setup optimization problem with four subsystems (gearbox, aerodynamics, brake system, and suspension). The available actions at the **system-level** are:

(1) $a_{t,1} = [50, 25, 25, 25]$ (High budget for gearbox subsystem)
(2) $a_{t,2} = [25, 50, 25, 25]$ (High budget for aerodynamics subsystem)
(3) $a_{t,3} = [25, 25, 50, 25]$ (High budget for suspension subsystem)
(4) $a_{t,4} = [25, 25, 25, 50]$ (High budget for brake system subsystem)

The actions at the **subsystem-level** correspond to choosing one of the following process heuristics for each subsystem i within the allocated budget:

(1) $b^i_{t,1}$ = EI
(2) $b^i_{t,2}$ = PI
(3) $b^i_{t,3}$ = UCB

The subsystem-level decision-making process operates as follows: At the start of each episode, the system-level RL agent allocates a fixed budget for information acquisition to each subsystem (e.g., [50, 25, 25, 25]). Subsystems optimize their parameters within this budget while keeping others fixed. After completing their allocated steps using the chosen information acquisition function, all subsystems are re-integrated, updating their parameter values. This process repeats (10 times) within the episode, with updated parameters used as fixed inputs for subsequent iterations. Subsystem-level actions remain fixed throughout the episode. The final parameter values for each subsystem are determined by their optimized parameters after all iterations. The action taken by the subsystem in each episode is determined by the subsystem level multi-armed bandit. Figure 2 illustrates the learning process of the RL agent by visualizing reward R_t against episodes (t) for 400 episodes. It is observed that the reward increases with the number of episodes and plateaus after around 150, indicating convergence of the RL algorithm.

5.2 Heuristics Learned.

5.2.1 System-level. Figure 3 illustrates the probability of choosing each arm at the system level $P(a_{t,i})$ versus episodes (t) for 400 episodes. Here, i denotes the subsystem, with Gearbox ($d = 5$), Aerodynamics ($d = 2$), Brakes ($d = 2$), and Suspension ($d = 2$). Initially, all probabilities start from 0.25 (random). It is observed that the probability of picking the Gearbox increases with the number of episodes, eventually reaching 1 after around 150 episodes. Conversely, the probabilities for Aerodynamics, Brakes, and Suspension reach 0. This is possibly due to the higher-dimensional design space of the Gearbox subsystem, requiring more budget allocation. The heuristic derived from the system-level decision-making process is to prioritize budget allocation to the Gearbox subsystem.

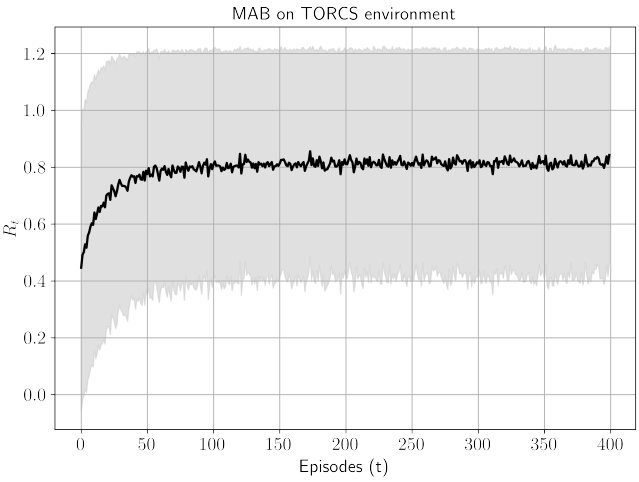

Fig. 2 RL agent reward over time averaged across 50 runs.

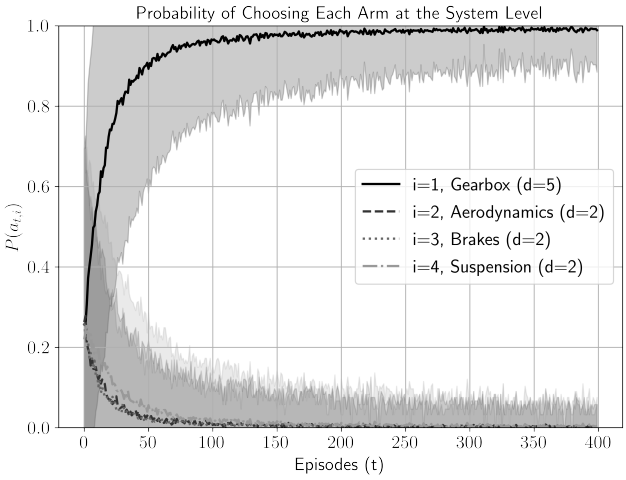

Fig. 3 Probability of choosing each arm at the system level $P(a_{t,i})$.

5.2.2 Process-level. Figure 4 presents the probability of choosing each arm at the subsystem level $P(a_{t,j})$ versus episodes t for three different heuristics: EI, PI, and UCB. For Gearbox and Suspension, the probability of selecting UCB increases with time, indicating it as the preferred information acquisition decision. However, for Aerodynamics and Brakes, it remains inconclusive, although the probability of selecting PI reduces with time in Aerodynamics.

The experimental results indicate the following heuristics:

(1) For the Gearbox and Suspension subsystems, employ the Upper Confidence Bound (UCB) heuristic for information acquisition.
(2) For the Aerodynamics subsystem, avoid using the Probability of Improvement (PI) heuristic for information acquisition.
(3) For the Brake subsystem, there is insufficient evidence to formulate a definitive heuristic, as the probabilities for different information acquisition strategies do not show clear convergence.

5.3 Analysis of Different Subsystem Groupings. To further investigate the impact of budget allocation on car setup optimiza-

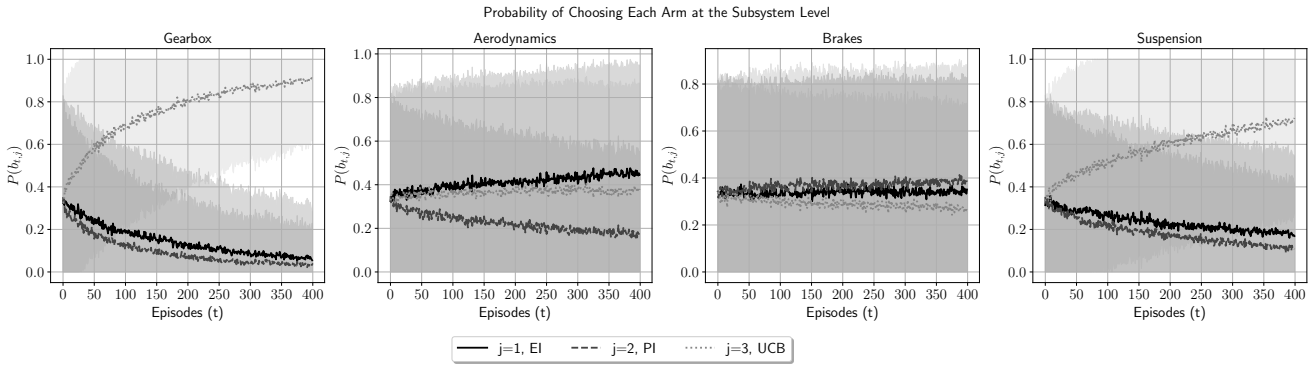

Fig. 4 Probability of choosing each arm at the subsystem level $P(b_{t,j})$.

tion, we conducted experiments considering different groupings of subsystems. We represent the computations as a graph and calculate the total number of nodes that need to be traversed to complete a computation, as shown in Appendix A. This analysis aims to understand how the subsystem complexity, represented by the number of parameters and computational graph depth, affects performance when budget priorities are shifted. We consider the TORCS car setup optimization problem with two subsystems, the available **system-level actions** are:

(1) $a_{t,1} = [50, 25]$ (High budget for first subsystem)
(2) $a_{t,2} = [25, 50]$ (High budget for second subsystem)

We perform experiments with four pairings of subsystems, considering different combinations of subsystems with varying numbers of parameters and computational graph depths. Table 2 summarizes the results for each pairing, showing the average speed when one subsystem has a higher budget (50) than the other (25). These experiments highlight the influence of subsystem complexity (in terms of number of parameters and average depth of the computational graph) on performance. Across all experiments, we observe that when a subsystem with more parameters and greater computational graph depth (e.g., Wheel with 5 parameters and depth 4) is allocated a higher budget, the average speed tends to improve. For example, in the Wheel and Aerodynamics/Brake System/Suspension grouping, when the Wheel is allocated a higher budget, the average speed decreases to 9.633 m/s, compared to 9.781 m/s when the Aerodynamics/Brake System/Suspension grouping receives a higher budget. This suggests that subsystems with greater complexity benefit more from increased resource allocation. Our analysis demonstrates that optimizing the car setup requires a balance in budget allocation, particularly favoring subsystems with higher parameter counts and more complex computational structures. The results suggest that, in general, prioritizing budget to such subsystems leads to higher overall performance, as they can better utilize the additional resources to explore their larger design space more effectively.

6 Discussion

In the TORCS example, the RL agent recommends the system designer focus on optimizing the higher-dimensional Gearbox subsystem by allocating most resources to it, allocate fewer resources to Aerodynamics, Brakes, and Suspension subsystems as they have a lower impact, and adapt resource allocation based on evolving knowledge of subsystem importance and impact on overall performance. The RL agent recommends that Gearbox and Suspension subsystem designers use the Upper Confidence Bound (UCB) heuristic for information acquisition and Aerodynamics subsystem designers, and avoid using Probability of Improvement (PI).

6.1 Comparison with Bayesian Optimization. In this comparison, we evaluate the effectiveness of our hierarchical reinforcement learning approach in finding heuristics and the performance of these heuristics against a Bayesian Optimization (BO) method. Specifically, we compare our Multi-Armed Bandit (MAB) based approach to full Bayesian Optimization where the entire 11-dimensional design space is treated as a black box. The full BO approach considers all 11 parameters (gearbox ratios, aerodynamic settings, brake system parameters, and suspension characteristics) as inputs X, with the car's performance (average speed in km/s) as the output Y. This method employs a single Gaussian Process model with Expected Improvement (GP-EI) as the acquisition function to navigate the entire design space simultaneously. In contrast, our hierarchical MAB approach decomposes the problem into system and subsystem levels, learning budget allocation strategies at the system level while concurrently optimizing subsystem-specific information acquisition heuristics. This approach both discovers effective heuristics (e.g., prioritizing budget allocation to the gearbox subsystem using UCB for certain subsystems) and applies these heuristics to optimize the car's performance.

The performance (average speed) versus time elapsed (in seconds) is compared between Bayesian Optimization (BO) and Multi-Armed Bandit (MAB) in Figure 5. The results in Figure 5 demonstrate that our MAB-based approach outperforms full BO in achieving higher average speeds as the computational time increases. This superior performance can be attributed to two factors: (i) the effectiveness of the learned heuristics in guiding the optimization process, and (ii) the computational efficiency gained from problem decomposition, where fitting and acquiring information in multiple lower-dimensional Gaussian Processes (5-dimensional for gearbox, 2D each for aerodynamics, brakes, and suspension) is less expensive than fitting a single 11-dimensional GP in the full BO approach. We include a non-learning-based method where the policy is based on random sampling rather than Thompson Sampling. This baseline serves as an intermediate comparison between BO and our proposed method. The results as shown in Figure 5 demonstrate that the non-learning-based random sampling approach performs better than BO due to the computational efficiency gained from problem decomposition. By focusing on smaller subsystems, even random sampling benefits from reduced dimensionality compared to BO's single high-dimensional Gaussian Process. However, this method still underperforms compared to our hierarchical reinforcement learning approach, which effectively learns and exploits heuristics (e.g., prioritizing resource allocation to critical subsystems like the gearbox) for improved performance. This additional comparison highlights the advantages of both the hierarchical structure and learning-based methods. Specifically, The hierarchical structure reduces computational complexity by breaking down the problem into manageable subsystems and the learning-based approach further enhances performance by adaptively refining strategies based on observed outcomes.

Table 2 Performance when different groupings of subsystems are considered for car setup optimization in TORCS.

Subsystem 1			Subsystem 2			Average speed when subsystem 1 has higher budget (m/s)	Average speed when subsystem 2 has higher budget (m/s)
Components	Number of parameters	Computational graph average depth	Components	Number of parameters	Computational graph average depth		
Gearbox	5	6	Aerodynamics, Brake System and Suspension	6	4.33	9.778	9.623
Gearbox	5	6	Wheel	5	4	9.724	9.617
Wheel	5	4	Aerodynamics, Brake System and Suspension	6	4.33	9.633	9.781
Wheel	5	4	Aerodynamics	2	4	9.701	9.560

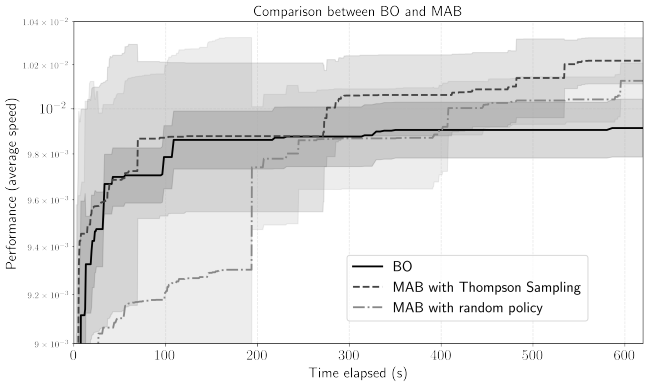

Fig. 5 Comparison between BO (GP-EI) and MAB.

6.2 Expanded Action Space. To further investigate the stability and performance of our joint learning approach, we conducted additional experiments with an expanded action space. Specifically, we increased the number of budget allocation actions from 4 to 16 while keeping the same 3 actions for information acquisition. Considering the TORCS car setup optimization problem with four subsystems (gearbox, aerodynamics, brake system, and suspension), the new set of budget allocation actions includes combinations described as follows:

$$a_{t,1} = [50, 25, 25, 25],\ a_{t,2} = [25, 50, 25, 25],$$

$$a_{t,3} = [25, 25, 50, 25],\ a_{t,4} = [25, 25, 25, 50],$$

$$a_{t,5} = [65, 30, 15, 15],\ a_{t,6} = [65, 15, 30, 15],$$

$$a_{t,7} = [65, 15, 15, 30],\ a_{t,8} = [30, 65, 15, 15],$$

$$a_{t,9} = [30, 15, 65, 15],\ a_{t,10} = [30, 15, 15, 65],$$

$$a_{t,11} = [15, 65, 30, 15],\ a_{t,12} = [15, 65, 15, 30],$$

$$a_{t,13} = [15, 30, 65, 15],\ a_{t,14} = [15, 30, 15, 65],$$

$$a_{t,15} = [15, 15, 65, 30],\ a_{t,16} = [15, 15, 30, 65].$$

We evaluated our approach over 50 independent runs for 5000 episodes. We observe that at the system level, no single action had the highest probability of selection, and it kept varying between $a_{t,1}, a_{t,5}, a_{t,6}, a_{t,7}$ as seen in Figure 6. These actions allocate a relatively higher budget to the gearbox (e.g., 65 in three cases and 50 in one case) compared to other subsystems (e.g., budgets of 15, 25, or 30). However, no single action among them dominated as observed in smaller action spaces. At the subsystem level, we observed that for the gearbox subsystem specifically (where higher budgets are allocated), the UCB heuristic was preferred after 2000 episodes. However, for other subsystems no clear preference emerged.

To address potential instability caused by joint learning and ensure convergence to stable policies, we replace the Thompson Sampling exploration strategy with an epsilon-greedy policy. The epsilon value is defined as an exponential decay function: $\epsilon = \exp(-k \cdot t)$ where t is the episode number. For system-level policies: $k = 0.05$ (faster decay to stabilize budget allocation early). For subsystem-level policies: $k = 0.01$ (slower decay to allow sufficient exploration). This modification ensures that the system-level policy stabilizes early in training to provide consistent budget allocations. We plot the ϵ values in Figure 8.

We run the 50 experiments for 5000 episodes and visualize the resulting heuristics in Figure 6 for the system-level actions and in Figure 7 for the gearbox subsystem-level actions. With this modification, we observe that at the system level, Action $a_{t,i=5}$ emerged as the dominant choice with the highest probability of selection across runs. At the subsystem level, the heuristics followed patterns observed in smaller action spaces. Specifically, UCB was preferred for gearbox and suspension subsystems and PI was consistently avoided for aerodynamics subsystem. The results demonstrate that our proposed method can indeed handle a larger action space, enabling more detailed exploration of budget allocation strategies. However, the expanded action space increases the number of episodes required for the RL agent to converge to effective policies. This is due to the larger number of potential actions that need to be explored and evaluated. The resulting heuristics at the also become harder to interpret compared to those derived from a smaller action space.

The results demonstrate that the proposed RL approach can effectively learn heuristics that facilitate the efficient exploration of complex design spaces under resource constraints. By strategically allocating resources and selecting appropriate information acquisition strategies, the RL agent outperformed the Bayesian Optimization (BO) method in rapidly converging to high-performing car configurations. The learned heuristics allowed the agent to prioritize the most impactful subsystems and adapt its decision-making processes based on evolving knowledge, thereby maximizing the overall system performance within the given budget. The hierarchical formulation aligns with complex system decomposition, guiding resource allocation across subsystems while optimizing localized processes within each subsystem. Automating heuristic derivation alleviates designers' cognitive burden and enables organizations to navigate complex design spaces, reducing development cycles and costs while rapidly delivering innovative products.

The heuristics derived in this study demonstrate efficacy across the diverse track types available in the TORCS environment, including dirt, oval, and road configurations. This adaptability suggests that the learned strategies for budget allocation and information acquisition are robust to varying racing scenarios, each presenting distinct challenges in vehicle dynamics and performance optimization. While these specific heuristics are tailored to the TORCS simulation, the methodology employed for their identification has broader applicability to a wide class of complex engineering design problems.

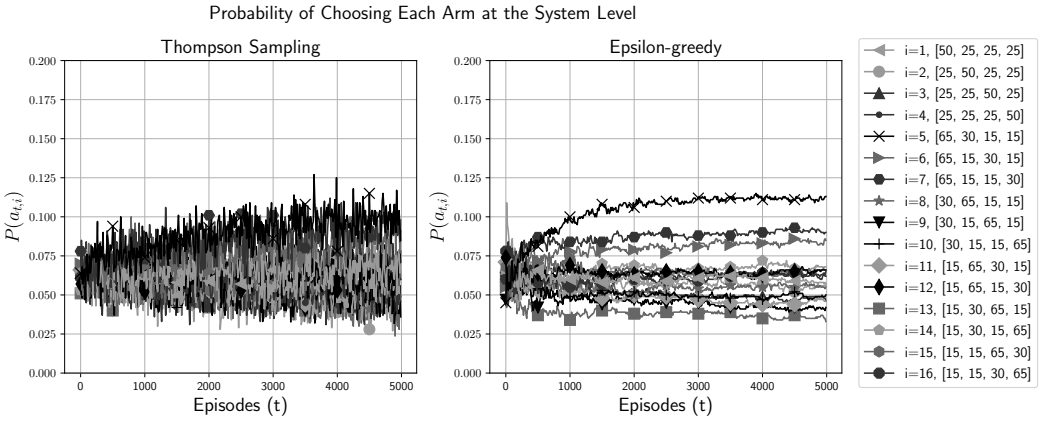

Fig. 6 Comparison of system-level policies using Thompson Sampling and epsilon-greedy with exponential decay.

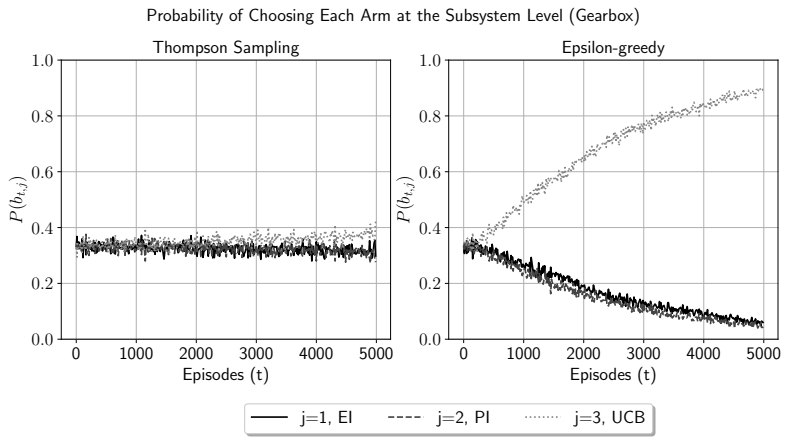

Fig. 7 Comparison subsystem-level policies using Thompson Sampling and epsilon-greedy with exponential decay.

The hierarchical reinforcement learning approach, which combines system-level resource allocation with subsystem-level optimization strategies, represents a generalizable framework for deriving heuristics. The generalizability of this approach is rooted in its fundamental components: the multi-armed bandit (MAB) formulation for both system and subsystem levels, the use of Thompson Sampling for policy learning, and the hierarchical decomposition of the problem space. These elements are not specific to vehicle dynamics or racing simulations, but rather represent a general approach to solving complex, multi-level optimization problems. For instance, the system-level MAB, which learns to allocate budget across subsystems, could be analogous to resource distribution in fields such as aerospace system design, where budgets must be allocated across propulsion, structural, and avionics subsystems. Similarly, the subsystem-level MABs, which learn optimal information acquisition strategies, could be applicable in any domain where different optimization approaches (e.g., UCB, EI, PI) might be more or less effective for different components of the overall system. The key strength of this approach lies in its ability to systematically uncover effective decision-making strategies in complex, hierarchical design spaces, regardless of the specific domain. Learning both resource allocation and optimization strategies concurrently addresses a fundamental challenge in many engineering design processes: how to efficiently distribute limited resources across interdependent subsystems while simultaneously optimizing within each subsystem.

Several opportunities for further research and improvements exist. One avenue for future work is to extract more comprehensive subproblem heuristics from the learned decision policies. Similar to the Solver-Aware System Architecting approach [11], inclusionary and exclusionary heuristics for different confidence thresholds could be derived, providing insights into the most influential design variables and guiding principles for effective subproblem decomposition. Additionally, incorporating a cost component for evaluating process heuristics would enable a more nuanced trade-off analysis between exploration and exploitation strategies within subsystems. This could lead to more efficient resource allocation and improved convergence rates. As design problems evolve and requirements change over time, transitioning from the multi-armed bandit framework to temporal difference methods [12] could facilitate the extraction of state-based heuristics. These heuristics could capture the dynamic nature of design processes and adapt to changing objectives or constraints. While the TORCS environment provided a suitable testbed for our methodology, further experiments with different tracks, car models, and a broader range of design parameters would enhance the generalizability of our approach. Increasing the number of design variables, for instance, to the 22 parameters used in [38], would better reflect the high-dimensional nature of real-world design problems. Mechanical gearboxes typically offer discrete gear ratios, whereas a belt drive would offer a continuous transmission ratio. While our approach assumes continuous design parameters, some, like gear ratios in a mechanical gearbox, are inherently discrete. In reality, the design space for a mechanical gearbox would be discrete, limited by the number of feasible gear ratio combinations. Our approach, while assuming continuous variables, can be adapted to incorporate dis-

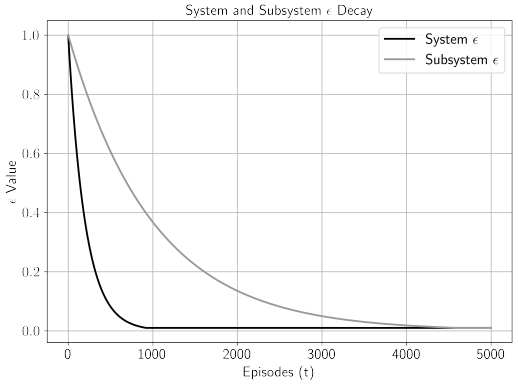

Fig. 8 Exponential decay of epsilon values for system-level ($k = 0.05$) and subsystem-level ($k = 0.01$) policies. Faster decay stabilizes system-level budget allocation early, while slower decay ensures sufficient exploration at the subsystem level.

crete variables, which could simplify the optimization process by reducing the search space. Future work could involve applying the identified heuristics to other resource allocation scenarios, such as project management tasks where subsystems differ in complexity or impact. This would test their applicability as starting points for resource allocation strategies in diverse domains. Note that the decomposition of design problems into subsystems is often an arbitrary process that may not fully capture interdependencies between all variables. Approaches like turning black-box functions into white functions [39] or cooperative co-evolution with differential grouping [40] could be explored to identify more effective problem decompositions. Moreover, our current approach lacks communication mechanisms between subsystems and a centralized knowledge database, which are crucial for resolving interdependencies and facilitating information exchange during the design process. Finally, it is important to acknowledge that real-world engineering organizations are often already decomposed based on previous experience and domain knowledge.

Acknowledgments

We gratefully acknowledge the financial support from the National Science Foundation through NSF CMMI Grants 2129539 (Purdue University) and 2129574 (The George Washington University). Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

[1] Bhise, V. D., 2017, *Automotive Product Development: A Systems Engineering Implementation*, CRC Press.

[2] Freriks, H., Heemels, W., Muller, G., and Sandee, J., 2006, "5.3. 2 On the Systematic Use of Budget-Based Design: Sixteenth Annual International Symposium of the International Council On Systems Engineering (INCOSE) 8-14 July 2006," *INCOSE International Symposium*, Vol. 16, Wiley Online Library, Paper No. 1, pp. 788–803.

[3] Martins, J. R. and Lambe, A. B., 2013, "Multidisciplinary design optimization: a survey of architectures," AIAA journal, **51**(9), pp. 2049–2075.

[4] Kim, H. M., Rideout, D. G., Papalambros, P. Y., and Stein, J. L., 2003, "Analytical target cascading in automotive vehicle design," J. Mech. Des., **125**(3), pp. 481–489.

[5] Kang, N., Kokkolaras, M., Papalambros, P. Y., Yoo, S., Na, W., Park, J., and Featherman, D., 2014, "Optimal design of commercial vehicle systems using analytical target cascading," Structural and Multidisciplinary Optimization, **50**, pp. 1103–1114.

[6] Fu, K. K., Yang, M. C., and Wood, K. L., 2016, "Design Principles: Literature Review, Analysis, and Future Directions," Journal of Mechanical Design, **138**(10).

[7] Yilmaz, S., Daly, S. R., Seifert, C. M., and Gonzalez, R., 2015, "How do designers generate new ideas? Design heuristics across two disciplines," Design Science, **1**.

[8] Yilmaz, S. and Seifert, C. M., 2011, "Creativity through design heuristics: A case study of expert product design," Design Studies, **32**(4), pp. 384–415.

[9] Fillingim, K. B., Nwaeri, R. O., Borja, F., Fu, K., and Paredis, C. J. J., 2019, "Design Heuristics: Extraction and Classification Methods with Jet Propulsion Laboratory's Architecture Team," Journal of Mechanical Design, p. 1.

[10] Deshmukh, A. P., Thurston, D. L., and Allison, J. T., 2016, "Heuristics for Formulating Design Optimization Models: Their Uses and Pitfalls," *5th Internations Engineering Systems Symposium, CESUN 2016*, Washington, D.C., USA.

[11] Gadi, V. S., Topcu, T. G., Szajnfarber, Z., and Panchal, J. H., 2023, "Heuristics for Solver-Aware Systems Architecting (SASA): A Reinforcement Learning Approach," *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, Vol. 87318, American Society of Mechanical Engineers, p. V03BT03A001.

[12] Sutton, R. S. and Barto, A. G., 2018, *Reinforcement learning: An introduction*, MIT press.

[13] Tao, S., Van Beek, A., Apley, D. W., and Chen, W., 2020, "Bayesian optimization for simulation-based design of multi-model systems," *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, Vol. 84010, American Society of Mechanical Engineers, p. V11BT11A022.

[14] Tao, S., Sharma, C., and Devanathan, S., 2024, "Resource-Aware Multi-Fidelity Multi-Objective Multidisciplinary Design Optimization," *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, Vol. 88377, American Society of Mechanical Engineers, p. V03BT03A031.

[15] Tran, A., Wildey, T., and McCann, S., 2020, "sMF-BO-2CoGP: A sequential multi-fidelity constrained Bayesian optimization framework for design applications," Journal of Computing and Information Science in Engineering, **20**(3), p. 031007.

[16] Xiao, H. and Wei, Z., 2023, "Efficient Dynamic Allocation Policy for Robust Ranking and Selection under Stochastic Control Framework," arXiv preprint arXiv:2305.07603.

[17] Hsieh, B.-J., Hsieh, P.-C., and Liu, X., 2021, "Reinforced few-shot acquisition function learning for bayesian optimization," Advances in Neural Information Processing Systems, **34**, pp. 7718–7731.

[18] Liu, Z., Qu, X., Liu, X., and Lyu, H., 2022, "Robust Bayesian optimization with reinforcement learned acquisition functions," arXiv preprint arXiv:2210.00476.

[19] Ma, H., Vo, T. V., and Leong, T.-Y., 2024, "Mixed-Initiative Bayesian Sub-Goal Optimization in Hierarchical Reinforcement Learning," *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems*, pp. 1328–1336.

[20] Chen, H.-C., Dai, L., Chen, C.-H., and Yücesan, E., 1997, "New development of optimal computing budget allocation for discrete event simulation," *Proceedings of the 29th conference on Winter simulation*, pp. 334–341.

[21] Xiao, H., Lee, L. H., and Chen, C.-H., 2015, "Optimal budget allocation rule for simulation optimization using quadratic regression in partitioned domains," IEEE Transactions on Systems, Man, and Cybernetics: Systems, **45**(7), pp. 1047–1062.

[22] Fan, Q. and Hu, J., 2013, "Adaptive simulation budget allocation for determining the best design," *2013 Winter Simulations Conference (WSC)*, IEEE, pp. 888–897.

[23] Michelena, N., Park, H., and Papalambros, P. Y., 2003, "Convergence properties of analytical target cascading," AIAA journal, **41**(5), pp. 897–905.

[24] Allison, J. T. and Papalambros, P. Y., 2007, "Optimal partitioning and coordination decisions in system design using an evolutionary algorithm," *Proceedings of the Seventh World Conference on Structural and Multidisciplinary Optimization, Seoul, South Korea, May*, pp. 21–25.

[25] Allison, J. T. and Papalambros, P. Y., 2010, "Consistency constraint allocation in augmented lagrangian coordination," .

[26] Miguel, F., Gómez, T., Luque, M., Ruiz, F., and Caballero, R., 2009, "A decomposition-coordination method for complex multi-objective systems," Asia-Pacific Journal of Operational Research, **26**(06), pp. 735–757.

[27] Ashenafi, Y., Pandita, P., and Ghosh, S., 2022, "Reinforcement learning-based sequential batch-sampling for bayesian optimal experimental design," Journal of Mechanical Design, **144**(9), p. 091705.

[28] Chen, Q. and Heydari, B., 2022, "Dynamic resource allocation in systems-of-systems using a heuristic-based interpretable deep reinforcement learning," Journal of Mechanical Design, **144**(9), p. 091711.

[29] Wymann, B., Espié, E., Guionneau, C., Dimitrakakis, C., Coulom, R., and Sumner, A., 2000, "Torcs, the open racing car simulator," Software available at http://torcs. sourceforge. net, **4**(6), p. 2.

[30] Chaudhari, A. M., Bilionis, I., and Panchal, J. H., 2020, "Descriptive models of sequential decisions in engineering design: An experimental study," Journal of Mechanical Design, **142**(8).

[31] Shergadwala, M., Bilionis, I., Kannan, K. N., and Panchal, J. H., 2018, "Quantifying the impact of domain knowledge and problem framing on sequential decisions in engineering design," Journal of Mechanical Design, **140**(10).

[32] Haskins, C., Forsberg, K., and Krueger, M., 2011, "Systems engineering handbook: A guide for system life cycle processes and activities," Incose San Diego, CA (US).

[33] Auer, P., 2002, "Using confidence bounds for exploitation-exploration tradeoffs," Journal of Machine Learning Research, **3**(Nov), pp. 397–422.

[34] Jones, D. R., Schonlau, M., and Welch, W. J., 1998, "Efficient global optimization of expensive black-box functions," Journal of Global optimization, **13**(4), pp. 455–492.
[35] Mockus, J., 1994, "Application of Bayesian approach to numerical methods of global and stochastic optimization," Journal of Global Optimization, **4**, pp. 347–365.
[36] Bishop, C. M., 2006, *Pattern recognition and machine learning*, springer.
[37] Rasmussen, C. E. and Williams, C. K., 2006, *Gaussian processes for machine learning*, The MIT Press.
[38] Kemmerling, M. and Preuss, M., 2010, "Automatic adaptation to generated content via car setup optimization in torcs," *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games*, IEEE, pp. 131–138.
[39] Shan, S. and Wang, G. G., 2011, "Turning Black-Box Functions Into White Functions," Journal of Mechanical Design, **133**(3), p. 031003.
[40] Omidvar, M. N., Li, X., Mei, Y., and Yao, X., 2014, "Cooperative Co-Evolution With Differential Grouping for Large Scale Optimization," IEEE Transactions on Evolutionary Computation, **18**(3), pp. 378–393.

Appendix A: Computation Graph of TORCS

The computation graph shown in Figure 9 provides a high-level abstraction of the car dynamics in the racing simulation system. This model is developed by following the architecture and implementation of TORCS open-source code. The graph captures the main physical components of a racing car and how various user inputs, such as throttle, gear selection, steering, and braking, influence the car's behavior on the track.

The engine block receives input from the throttle control. It is modeled using factors such as fuel consumption, torque-RPM data, and other engine-specific parameters (e.g., minimum and maximum RPM values). Based on the user throttle input and gear selection, it produces torque, which is then transmitted to the engine shaft. The engine shaft transmits the torque to the gearbox. The gearbox is controlled by the user input for gear selection and contains data about shift time, ratio, inertia, and efficiency of the gears. The differential governs the torque split between the wheels, factoring in inertia, efficiency, slip bias, and locking torque. In conjunction with the drive shaft, the axle transfers the torque to the wheels. The differential allows for varied wheel speeds, especially while cornering. The wheels' forces are influenced by their speeds, the transmission torque, and external forces such as aerodynamic drag. Additional parameters like the wheels' dimensions, inertia, and frictional properties (dynamic friction) affect the car's traction and handling. This subsystem models the car's drag and downforce, accounting for variables such as front area, drag coefficient, and the effects of front and rear wings on vehicle stability and speed. The suspension system influences the car's ride comfort and handling. It is modeled based on spring, bump, and rebound characteristics, allowing realistic reactions to track irregularities, bumps, and turns. The environment introduces external forces and conditions affecting the vehicle's performance. Variables like track temperature, wind, and road surface influence it. The primary user inputs – throttle, gear, steering, and braking – directly affect the car's position, speed, and other variables.

At each time step during the simulation, the car's position, speed, and fuel consumption are updated based on the user inputs and the internal states derived from the computational graph. The system performs these updates using the predetermined design variables, which remain constant for the duration of each race. The traversal of the graph represents the series of calculations required to update the car's state at each time step. To estimate the computational load, the average depth of the graph can be calculated by counting the number of nodes traversed during each update cycle.

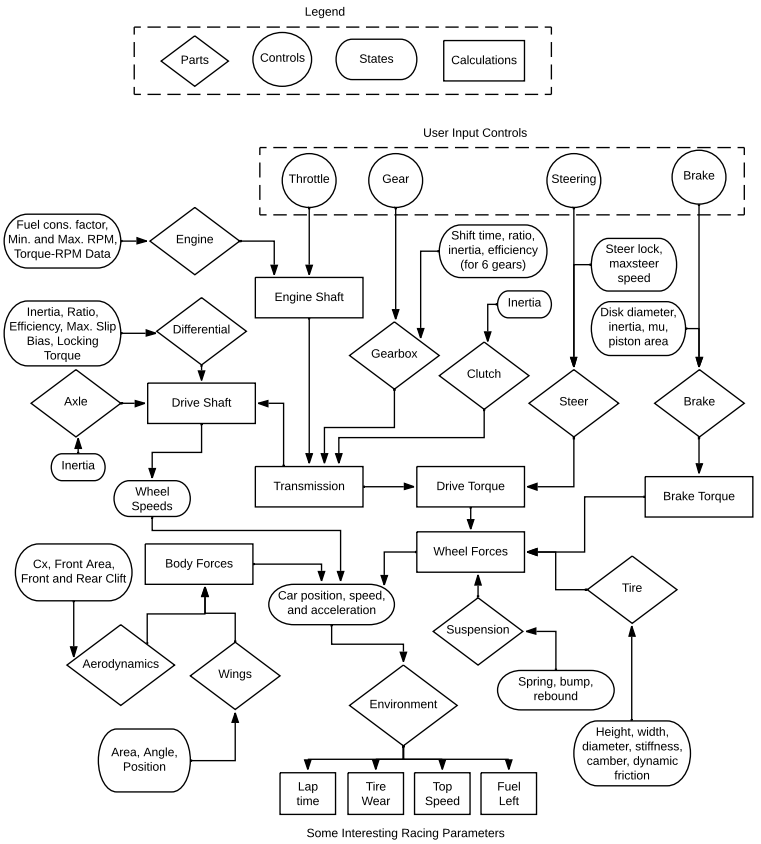

Fig. 9 TORCS Computation Graph.