Engineering Computations

Evaluating configuration and parametric diversity in simulation-driven powertrain computational synthesis with autonomous control tuning

Received 22 August 2024
Revised 4 February 2025
21 October 2025
Accepted 3 November 2025

Rohail Malik, Muhammad Ahmad and Jari Vepsäläinen
Department of Energy and Mechanical Engineering, Aalto University, Espoo, Finland

Abstract

Purpose – Computational design synthesis (CDS) provides a systematic means to explore the design space of complex systems. However, the scope of exploration in many CDS studies is biased by limited parametrization, where component parameters remain fixed or arbitrarily assigned. This paper investigates the influence of configuration and parametric diversity on the effectiveness of design-space exploration in vehicle powertrain synthesis.

Design/methodology/approach – A simulation-driven CDS framework is developed with an autonomous control tuning mechanism integrated to ensure consistent evaluation of the synthesized topologies. Powertrain topologies are synthesized by randomly interconnecting port-compatible components and assigning parameters from predefined ranges. Two topology sets are generated, one emphasizing configuration diversity and another emphasizing parametric diversity, to analyze their impact on exploration.

Findings – The results indicate that parameter initialization strongly influences perceived performance. A topology's parameter assignment defines its effective position in design space, and poor initialization can bias evaluation outcomes even with structural diversity.

Research limitations/implications – The framework is limited to sequential topologies, which makes it easier to explore the entire design space but also makes the results harder to generalize.

Practical implications – The groundwork established here impacts the development of generative topology synthesis frameworks, enabling autonomous generation, control tuning and simulation of systems.

Originality/value – The novelty is the autonomous control tuning integrated in a CDS workflow, alongside the investigation into the influence of configuration and parametric diversity on the effectiveness of design-space exploration.

Keywords Generative design, Genetic algorithms, Computational design synthesis, Powertrain design, Automatic control tuning

Paper type Research article

Nomenclature

η	Efficiency of the transmission or gearbox
η_{d}	Efficiency of the drivetrain
η_{gen}	Efficiency of the generator
η_{ice}	Efficiency of the IC engine.
η_{mot}	Efficiency of the motor
ω_{ice}	Rotational speed of the IC engine.
ω_{in}	Input rotational speed to a component (e.g. generator or transmission)

Funding: This work was supported by Business Finland (award no: Dnr 31/31/2024)

Engineering Computations
Emerald Publishing Limited
e-ISSN: 1758-7077
p-ISSN: 0264-4401
DOI 10.1108/EC-08-2024-0767

ω_{max}	Maximum speed of the IC engine.
$\omega_{mot,base}$	Base speed of the motor
ω_{mot}	Rotational speed of the motor
ω_{out}	Output rotational speed from a transmission or gearbox
ω_{stall}	Stall speed of the IC engine.
ρ	Air density
θ	Road grade angle
A	Frontal area of the vehicle
C_D	Drag coefficient
f_{rr}	Rolling resistance coefficient
i_g	Gear ratio
K_p	Proportional control constant
m	Vehicle mass
P_{elec}	Electrical power consumption or generation
P_{fuel}	Fuel power consumption
$P_{mot,abs,max}$	Maximum absolute power output of the motor
r_t	Effective wheel radius
SOC	State of charge of the battery
$T_{ice,max}$	Maximum torque output of the IC engine.
T_{ice}	Torque produced by the internal combustion engine (IC engine)
T_{in}	Input torque to a component (e.g. generator or transmission)
$T_{mot,abs,max}$	Maximum absolute torque output of the motor
$T_{mot,max}$	Maximum torque output of the motor
T_{mot}	Torque produced by the motor
T_{out}	Output torque from a transmission or gearbox
T_{req}	Requested torque
T_{tr}	Torque transmitted to the wheels
v	Vehicle speed

1. Introduction

Engineering design is a systematic process that combines scientific principles, technical knowledge, and creativity to develop innovative solutions. However, evolving requirements, designer bias, and complex interdisciplinary dependencies often restrict design-space exploration and, consequently, innovation.

Computational design synthesis (CDS) methods aim to automate early-stage design by generating and evaluating alternative system topologies. Yet, the scope of exploration in many CDS frameworks remains constrained: while topologies are varied, their parameters (e.g. component sizes, capacities) are often fixed or narrowly defined. In addition, the control schemes when simulating the topology are basic and not adapted to the system. This limits the diversity of candidate solutions and may bias the search toward certain architectures.

This paper integrates an autonomous optimal-control tuning module within a simulation-driven CDS workflow for vehicle powertrain topologies. The module automatically adjusts control parameters for each generated design before evaluation to ensure fair and consistent evaluation of synthesized topologies. In addition, it investigates the relative influence of configuration diversity and parametric diversity on design-space exploration. Two sets of topologies are synthesized: one emphasizing distinct configurations, and another featuring repeated configurations with varied parameters. Their performances are compared to analyze how parametrization affects the perceived design space and synthesis outcomes. Based on this, a strategy is proposed for balancing parametrization and configuration variation in computational design workflows.

The remainder of this paper is organized as follows. section 2 reviews related work on computational synthesis and the treatment of design parameters within these workflows. section 3 describes the proposed framework, including the synthesis algorithm, simulation architecture, and autonomous control-tuning module. section 4 presents the comparative experiments, followed by discussion in section 5, and conclusion in section 6.

2. Literature review

2.1 Background

Formal engineering design usually involves multidisciplinary teams managing diverse performance metrics, trade-offs, and customer needs. The increasing complexity of design parameters and portfolio interdependencies limits manual exploration of the entire design space. Reliance on the designers' existing knowledge and prior experiences inadvertently introduces design bias, restricting solution diversity and hindering innovation. In addition, the design process involves tedious, cyclic, and repetitive tasks, which add to the challenge.

Generative design is a computational methodology that seeks to address these challenges by systematically generating a wide range of design options that adhere to user-defined constraints (Ramesh *et al*., 2024). In contrast to traditional design processes, where designers actively generate and refine solutions, it employs iterative optimization to find potential designs (Kazi *et al*., 2017) independently. By automating the tedious task of design space exploration, the error margin can be reduced, efficiency increased (Tkachenko and Wang, 2024), and time spared for more creative endeavors (Helms *et al*., 2009). However, human designers remain crucial for defining parameters, making trade-off decisions, and addressing subjective aspects such as aesthetic requirements (Demirel *et al*., 2023).

When aimed at assembling components to form topological networks, this automation process is termed Computational Design Synthesis (CDS) (Gadeyne *et al*., 2014). Synthesis involves configuration design, where only the arrangement of components is developed, and components themselves are not designed. Once the configuration is fixed, setting optimal specifications turns it into a skeletal design problem (Wielinga and Schreiber, 1997). As it creates the basic structure of a product, synthesis is a creative step for generating novel solutions (Cagan, 2001) and strongly overlaps with the conceptual design stage in the product development process. As such, computational synthesis carries the potential of uncovering innovative solutions beyond human biases (Chakrabarti *et al*., 2011).

2.2 State-of-the-art

A purely randomized approach to topology synthesis, sometimes referred to as naïve creation, can produce novel configurations but is computationally inefficient because most generated solutions are infeasible or irrelevant (Helms *et al*., 2009). To achieve a better balance between creativity and efficiency, CDS methods structure the generation process through systematic procedures rather than random exploration.

Shea (2003) and Cagan *et al*. (2005) describe CDS as an iterative process built around four key phases that define how design knowledge is encoded, explored, assessed, and refined within a computational framework:

(1) **Representation** defines how design knowledge is captured for synthesis. It typically includes a metamodel of components, functional hierarchies, and the rules or constraints that describe allowable relationships.

(2) **Generation** produces candidate solutions based on the defined representation. This can be done through rule-based or search-based strategies, depending on how domain knowledge is encoded.

(3) **Evaluation** assesses generated solutions to identify feasible and promising candidates. Depending on the level of available knowledge, evaluation can range from expert judgment to automated simulation-based performance analysis. When large sets of designs are produced, embedding evaluation directly within the computational loop is most effective (Shea, 2003).

(4) **Guidance** uses the outcomes of evaluation to inform the next cycle. This feedback can be automatic or through designer intervention to steer exploration toward more promising regions of the design space.

Representation and generation form the constructive core of the synthesis process, while evaluation and guidance create an adaptive loop. No generation can occur without a proper representation, and meaningful guidance requires evaluation. When the guidance phase is absent, synthesis operates as an open-loop process in which designs are generated (sometimes with evaluation) but without systematic refinement or feedback.

Rule-based approaches, such as design grammars, provide a structured way to generate configurations by applying symbolic rules that define how components or functions can combine (Chakrabarti *et al*., 2011). This is conceptually similar to natural language, where a finite set of symbols and rules can yield diverse expressions. A metamodel typically defines available components, constraints, and relationships, ensuring that generated designs remain valid.

In topological CDS, grammar-based synthesis often begins with a functional decomposition of the problem, from which structural concepts are derived (Pahl *et al*., 2007). The Function–Behavior–Structure (FBS) framework (Umeda and Tomiyama, 1995) extends this logic by deriving structures from functions through the behaviors that realize them. It has been applied to the synthesis of electric and hybrid vehicle powertrain architectures (Helms *et al*., 2009; Helms and Shea, 2012), using hierarchical graph grammars that define rules linking functions, behaviors, and structures with each other and among themselves. Application of these rules builds the functional topology, then assigns behaviors, and finally instantiates concrete components until a complete design graph emerges.

While grammars provide expressive generative power, their combinatorial growth limits scalability. To address this, Münzer *et al*. (2013) reformulated synthesis as a Boolean satisfiability (SAT) problem, where component types, ports, and interconnections are expressed as logical constraints. Solving the SAT model yields all valid system topologies, after which Constraint Satisfaction Problem (CSP) solving assigns consistent numerical parameters. These topologies are automatically transformed into executable bond-graph simulation models (Münzer and Shea, 2017), evaluated under representative conditions, and further refined through a Simulated Annealing loop to optimize performance (Münzer and Shea, 2016). This integrated pipeline, from topology generation to simulation-based evaluation, covers the full CDS process and is domain agnostic, though it remains open-loop and computationally intensive since all topologies are exhaustively generated and evaluated.

A related constraint-driven approach by Anderson *et al*. (2017) generated electronic circuits and firmware from behavioral specifications. Rather than graph transformations, it used a declarative model where components and behaviors were matched through dependency resolution. This demonstrated that constraint-based reasoning can achieve generative synthesis without formal grammar systems.

Beyond generating designs from scratch, several studies focus on deriving alternatives from existing valid configurations. These methods emphasize iterative improvement and search efficiency rather than full generative exploration. Bolognini *et al*. (2007) proposed a stochastic, simulation-guided method called CNS-burst, where electro-mechanical designs are represented as Connected-Node Systems. At each step, a design is randomly but selectively modified, simulated, and retained only if it remains Pareto-optimal. Repeating this process yields a diverse set of optimized configurations that balance exploration and performance.

Starling and Shea (2005) introduced a more structured approach using parallel grammars that define both function and geometry. Grammar rules are applied stochastically among valid options, and each generated configuration is automatically simulated and evaluated. A hybrid pattern search algorithm then biases rule selection toward sequences that previously improved performance, combining stochastic variety with directed learning.

Recent advances in artificial intelligence have introduced new directions for CDS, particularly through machine learning (ML) techniques that enable data-driven modeling and automated design exploration (Peckham *et al*., 2025). These methods generally serve two complementary roles: as surrogate models that accelerate evaluation, and as generative models that propose new design candidates.

To alleviate the significant computational effort associated with CDS, surrogate models — also known as metamodels, response surface models, or emulators — are developed as computationally efficient approximations of high-fidelity models. These surrogate approaches span a wide spectrum, from machine learning techniques such as neural networks to classical methods like polynomial response surfaces, radial basis functions, Kriging, polynomial chaos expansions, and support vector machines (Simpson *et al*., 2001; Sudret *et al*., 2017). In recent years, physics-informed and hybrid physics-machine learning models have further enhanced surrogate modeling by embedding domain knowledge and governing equations into data-driven architectures, improving generalizability and physical consistency even with limited training data (Karniadakis *et al*., 2021). As an example of physics-informed machine learning in the automotive domain, Tran *et al*. (2024) employed a nonlinear autoencoder to learn a compact latent design space from car body geometries and their drag coefficients. The model could predict aerodynamic performance directly from latent variables, thereby allowing for faster evaluation of synthesized designs.

The application of AI to topological generation itself remains limited, as data-driven models tend to reproduce existing design patterns rather than discover novel configurations. However, significant progress has been made in geometric generative design, where design spaces are continuous and novelty is more naturally defined. Zang *et al*. (2024) trained a conditional GAN on vehicle silhouette data to generate body shapes consistent with stylistic descriptors. Wang *et al*. (2025) combined a convolutional neural network with a multimodal large language model (LLM) to generate car designs aligned with target brand identities and to provide explainable feedback on stylistic features. Similarly, Oh *et al*. (2018) used GANs to generate concept images of wheel designs, which were then refined through topology optimization to produce manufacturable, performance-oriented geometries.

Most recently, LLM-based frameworks have been introduced as agents within design pipelines. Elrefaie *et al*. (2025) presented a multi-agent architecture where one agent translated sketches or prompts into renderings, another constructed (or retrieved) 3D geometries based on these renderings, and a third used surrogate simulations to provide aerodynamic feedback on generated geometries. Massoudi and Fuge (2025) investigated the use of these systems for improving early-stage engineering design. They compared a two-agent and a nine-agent system for an end-to-end pipeline from requirement extraction to conceptual design, and from there to simulation model generation, all embedded into a JSON-based representation. Both showed strong adherence to syntax and format, but heavily struggled with requirement coverage and physical correction. These approaches show how generative AI can go beyond achieving tasks themselves by acting as an orchestrator of designated tools, but also indicate their inability to properly understand the scientific knowledge and domain-specific heuristics.

2.3 Challenges and research gap

Across the body of work reviewed, it is clear that various CDS methods, ranging from rule-based grammars to AI-driven techniques, have achieved considerable progress, especially within the automotive domain. Yet several fundamental challenges persist that restrict their scalability, generality, and practical applicability.

(1) **Human Dependence for Setup:** Rule-based systems such as design grammars rely on manually engineered rule sets that encode expert knowledge about components, interfaces, and constraints within a narrowly defined domain (Chakrabarti *et al*., 2011; Alber and Rudolph, 2003). Creating or extending such grammars demands substantial effort to determine what constitutes a valid transformation and to formalize it accordingly (Caldas, 2008; Kriegman *et al*., 2020). While limited attempts at self-learning grammars exist (Orsborn *et al*., 2008; Yogev *et al*., 2010), these remain highly task-specific. More recently, LLMs have been researched as a grammar development and interpreting "partner" to reduce the extent of human effort required (Shea *et al*., 2025).

(2) **Lack of Generalization:** AI-based approaches can reduce human dependence, but inherit the same issue in a different form: the learned representations are constrained by their training data and therefore generalize poorly beyond the domain from which the data were drawn. Cross-domain transfer usually requires retraining on new datasets or embedding explicit physics-based constraints, both of which raise computational cost and model complexity (Peckham *et al*., 2025). Efforts such as Münzer's Boolean satisfiability (SAT) formulation (Münzer *et al*., 2013) attempt to define a domain-agnostic synthesis framework by expressing design constraints in a unified logical form, but at the expense of extremely high computational demand and extensive simulation requirements (reporting an overall time of 2,500 h for generating, simulating, and optimizing all feasible powertrain architectures to produce around a hundred optimal ones).

(3) **Conversion to technical models:** Although topology generation is central to CDS, the resulting configurations must ultimately be converted into mathematical or simulation-ready models for quantitative evaluation. This translation step remains largely manual, as the exact transformation from the inputs to outputs within a component, on a system-level, strongly depends on the domain, use-case, physical phenomenon, utilized language/tool, and modeling fidelity. Each CDS implementation incorporating simulation based evaluation requires predefined component models and tool-specific libraries (Münzer and Shea, 2017; Starling and Shea, 2005; Bolognini *et al*., 2007). Wu *et al*. (2008) partially addresses this gap through automated bond-graph construction, yet the approach is still tied to the physical domain for which the component templates are defined. Consequently, there is still no general mechanism for automatically deriving simulation-ready models across different physical domains or fidelity levels. This missing link prevents full automation of the synthesis–evaluation loop and reinforces dependence on human intervention during the modeling phase.

(4) **Balancing Novelty with Efficiency:** There is a persistent difficulty in balancing exploratory generation with guided optimization. Open-loop or exhaustive methods, such as Münzer's SAT-based framework, guarantee coverage of feasible design spaces but at extreme computational cost. Conversely, biased grammar rules or heuristic guidance can improve efficiency but risk overlooking novel configurations (Starling and Shea, 2005). AI-driven methods, when considered for generation purposes, can also be biased towards existing solutions.

(5) **Combined impact of configuration, parameters, and control:** The evaluation phase is often simplified (Starling and Shea, 2005). Most implementations fix or randomly assign parameters and employ generic control strategies, meaning that a topology may appear suboptimal simply because it was paired with non-ideal parameter settings or control logic. In complex systems such as powertrains, topology, parameter values, and control behavior are tightly coupled, and neglecting this

coupling can lead to misleading assessments of design quality. Münzer and Shea (2016) included parametric optimization of generated powertrains via simulated annealing, but the control strategies were left static and simplistic. While the comprehensive way would be to optimize each generated configuration in terms of its parameters, tune a perfect control strategy, and then compare it with others, it would overall be very inefficient. This imbalance between exploration, guidance, and evaluation fidelity remains a central obstacle to efficient yet innovative computational synthesis.

This research focuses on the complex inter-dependencies of topology configuration, parameters, and control. In particular, we analyze the impact of parameterization during simulation-based evaluation of computationally synthesized powertrain topologies. The goal is to find out if parametric variation affects topological performance during the evaluation phase, and if and where parametric optimization should be incorporated within a synthesis workflow. Further, we present a methodology for incorporating autonomous optimal control into simulation-driven synthesis workflows. This naturally builds on prior simulation-driven workflows such as Münzer and Shea (2017), which utilized a simplistic and constant-parameter control loop for simulating their powertrain topologies. The novelty of the work lies in the incorporation of automatic control tuning within a simulation-driven evaluation framework for computational synthesis, alongside the utilization of MATLAB-Simulink for simulation purposes.

3. Methodology

To investigate the influence of parameterization on performance evaluation within a CDS workflow, a simulation-driven framework was developed that autonomously generates and evaluates vehicle powertrain topologies. The framework also integrates autonomous control tuning, representing the second contribution of this work.

The central question addressed in this study is: Given an automatically synthesized topology, to what extent does its parameterization affect its evaluated performance relative to its structural layout? In other words, within a synthesis loop, does a topology's performance ranking depend primarily on its configuration, or can suboptimal parameter choices cause otherwise promising topologies to be overlooked? This question is significant because most existing CDS workflows either fix parameters or optimize all candidates exhaustively, which could lead to inefficiencies or biased evaluations. To address this, the framework is structured around the following key objectives:

(1) Develop a topology generation strategy that allows controlled variation in both layout and parameters, enabling comparative analysis of their respective effects on topology selection.

(2) Establish a method for automatically converting generated topologies into executable simulation models for quantitative performance evaluation.

(3) Integrate an autonomous control optimization scheme within the simulation process to ensure consistent and realistic dynamic assessment.

(4) Define performance metrics and evaluation criteria for analyzing and comparing simulated topologies.

This paper focuses on developing and demonstrating this framework using vehicle powertrain architectures, a domain particularly suited to computational synthesis due to the modularity and combinatorial diversity of its subsystems (Helms and Shea, 2012). The code for this entire framework, alongside the utilized powertrain component library and parameter sets, can be found in Malik *et al.* (2024).

3.1 Topology generation

For topological synthesis, the variation in layouts and parameters can be controlled through the following strategy;

(1) If structural uniqueness is enforced, only distinct layouts are generated.

(2) Otherwise, repeated layouts can be produced, albeit with different parameters, emphasizing parametric variation within the same topological structures.

To analyze the influence of topology versus parameterization on performance throughout the design space, it should be explored exhaustively. However, since this is computationally infeasible, we adopt a strategy that maximizes coverage rather than convergence. Within CDS, the design space can be conceptualized as comprising two main dimensions: (1) topological structure, and (2) system parameters. We constrain the topological dimension by considering sequential powertrain layouts, a simplified subset that reduces combinatorial growth and therefore a smaller design space, which can be fully explored easily. At the same time, we enable broad exploration by adopting random parameter assignment and random combination-based synthesis. This represents a basic naïve generation approach (Cagan *et al*., 2005) which allows maximum exploration of the design space. Not only does this simpler approach prioritize evaluation functionality over advanced synthesis capabilities, but it also avoids the complexities of integrating design grammars. In the following text, we present the implementation details of this approach.

3.1.1 Ports as constraints. A primary characteristic of CDS is the modularity of constituent components, which are defined as stand-alone structures with specified inputs and outputs, commonly referred to as ports. These ports are characterized by attributes such as the type of energy flowing through them, allowing only compatible components to interconnect. They therefore reduce the solution space to a feasible region (Mittal and Frayman, 1989). Further, representing designs as configurations of port-based objects is useful at the conceptual design stage when the geometry and spatial layout are still ill-defined (Liang and Paredis, 2004).

Prior works on computational synthesis of electric (Helms *et al*., 2009) and hybrid (Helms and Shea, 2012) vehicle powertrain topologies have used the nature of energy flow as the basis of defining ports. Since engineering components generally involve energy conversion, storage, or manipulation, this method is broadly applicable. Therefore, the same approach has been utilized in this framework, with ports defined by the type and direction of energy flow through them. Only ports with the same energy types and opposing flow direction can interconnect, as illustrated in Figure 1. Table 1 provides detailed information on the ports and other attributes of the considered powertrain components. These components are chosen as they represent the most essential systems in common powertrain types (i.e. combustion, electric, and hybrids).

3.1.2 Sequential synthesis. Since the layouts are sequential, they are represented as sequences of words, with each word denoting a component. Each word also has mapped metadata, which includes the type and direction of its ports. The synthesis process begins with the selection of a vehicle block (the load block) from the library as the initial element.

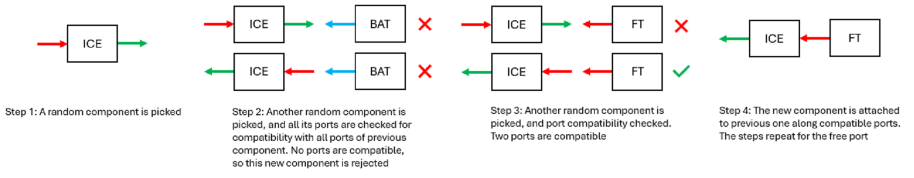


Figure 1. Working of the randomized synthesis algorithm. Source: Authors' own work. The arrows and their color indicate direction and nature of energy flow (Blue is electrical, Green is mechanical, and Red is chemical energy.)

Table 1. Attributes of the component blocks in the library. Only ports related to energy flow are mentioned

Component	ID	Class	Ports	Ports domain	Ports direction
Fuel Tank	FT	Storage	1	Chemical	Output
Battery	BAT	Storage	1	Electrical	Input
IC Engine	ICE	Actuator	2	Chemical, Mechanical	Input, Output
Motor	MOT	Actuator	2	Electrical, Mechanical	Input, Output
Generator	GEN	Generator	2	Mechanical, Electrical	Input, Output
Transmission	TR	Transmitter	2	Mechanical, Mechanical	Input, Output
Gearbox	GB	Transmitter	2	Mechanical, Mechanical	Input, Output
Vehicle	VEH	Load	1	Mechanical	Input

Port-compatible components are then sequentially added to each available port until no free ports remain. Ports are considered compatible if they share the same energy domain and have opposite energy flow directions. For each available port, the algorithm randomly selects a component from the library and checks its compatibility. If compatible, the component is placed adjacent to the previous one, and its free port becomes the next connection point. If not, another component is randomly selected, and the process repeats. The algorithm is also constrained to have the generated layouts contain at least one actuator. This, coupled with the fact that the generation begins from the vehicle block, ensures that the generated topologies are actually powertrain systems, e.g. avoiding a layout where two vehicle blocks are connected via a gearbox.

3.2 Model generation

Once a topology has been synthesized, it needs to be converted into a mathematical model for simulation and analysis. For this framework, the MATLAB environment is used, offering both text-based (MATLAB) and visual (Simulink) programming tools. Simulink is suitable for system-level design, and is widely utilized in powertrain modeling research (Tomar *et al*., 2021; Gangurde *et al*., 2019) due to its dynamic system modeling capabilities. Furthermore, its compatibility with MATLAB allows easy integration of the text-based generation algorithm.

3.2.1 Component modularity. Translating modular synthesis units with defined ports into functional technical models presents several challenges. Mathematical models of components often require complex and diverse input and output variables that may not align neatly with the simplified energy-based notion of input and output ports. For instance, the model of an electric motor may include several variables such as voltage, current, torque, speed, and control signal. To address this, mathematical models must be adapted to group related variables under unified energy flow ports.

Simulink's Simscape package offers models for engineering components interconnected through bond graphs, which provide a unified approach for representing energy transfer. The package's built-in compatibility across engineering domains also facilitates translating synthesized topologies. However, the models' complexity makes it more prone to errors, especially for cases such as computational synthesis, where unconventional topologies may be present. Furthermore, modifying or developing custom Simscape models is challenging due to limited documentation.

To mitigate these issues, the framework uses custom Simulink-based mathematical models developed from scratch for the component library. To unify related input and output variables into single energy-based ports, a two-way connection block is utilized in the modeling process. For instance, a motor providing torque to a vehicle requires feedback of the vehicle's speed (or the corresponding rotational speed at the motor shaft) to adjust torque output based on its specifications. Although this approach is primarily necessary for mechanical energy ports, it is

applied uniformly across all energy flow ports for consistency. For ports where one flow direction is unused, the unused direction is terminated at both ends, as illustrated in Figure 2.

The component modularity is facilitated by modeling components as masked Simulink blocks to allow seamless integration. Once developed and saved, they can then be utilized in a system by requiring only essential parameter adjustments without internal mathematical modeling. The component's external attributes (detailed in Table 1, required for the top-level configuration by the topology generation algorithm) are stored as a read-only parameter for each component block.

3.2.2 Mathematical modeling. Component modeling is designed to support generation of forward-facing powertrain models. These models simulate from energy source to sink (e.g. from fuel to engine to wheels), followed by a feedback to control this energy flow. Since physical causality makes models and simulations more realistic, they are more suited to evaluate a vehicle's performance and dynamic behavior (Mohan *et al.*, 2013). This contrasts with backward facing models, which pre-assume feasibility, and trace energy flow in reverse (e.g. from vehicle to engine to fuel).

A basic summary of the components' mathematical modeling for this framework, is given in this section.

(1) Fuel Tank: Essentially a sink for power consumption output from the IC engine. Power consumption is translated into fuel consumption through the fuel's energy density.

$$m_{\text{fuel}} = \frac{P_{\text{fuel}}}{E_{\text{d}}} \tag{1}$$

(2) Battery: A simple Thevenin model (Salazar and Garcia, 2022) with ideal battery efficiency of 1. The relationship between battery's open source voltage and State of Charge (SOC) is modeled by scaled data from Panasonic NCR18650B (Li, 2024), a lithium-ion battery cell used in older Tesla models.

$$V_{\text{bat}} = V_{\text{oc, bat}}(\text{SoC}(t)) - R_{\text{int}} i \tag{2}$$

$$\text{SoC}(t) = \text{SoC}_0 - \frac{1}{C_{\text{rated}}} \int_0^t i \, dt \tag{3}$$

$$i = \frac{P_{\text{elec}}}{V_{\text{bat}}} \tag{4}$$

(3) IC Engine: The simplified engine model works by limiting the demand torque according to the maximum torque at the current speed from the engine torque curve. The clutch is simulated by saturating engine speed at stall and maximum speeds, while

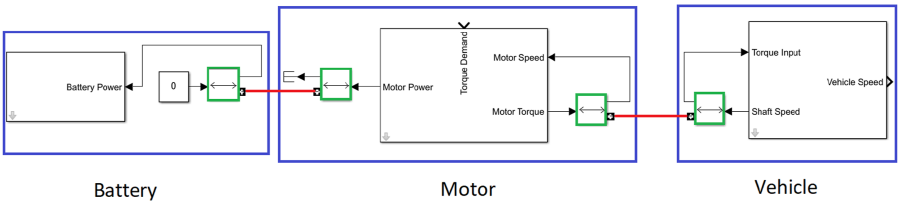


Figure 2. The two way connection strategy as utilized in an EV architecture. Source: Authors' own work). The two-way connection block (green) connects through a single connection (red) the EV components (blue)

zeroing torque output. Braking is simulated by forwarding the negative torque demand as manual braking force, but is not counted during power consumption calculations.

$$T_{\text{ice}} = \begin{cases} \leq T_{\text{ice,max}}, & \text{if } T_{\text{req}} \geq 0, \\ 0, & \text{if } T_{\text{req}} < 0. \end{cases} \tag{5}$$

$$T_{\text{ice,max}} = \begin{cases} f(\omega_{\text{ice, max}}), & \text{if } w_{stall} \leq w_{\text{ice}} \leq w_{\text{max}}, \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

$$P_{\text{fuel}} = \eta_{\text{ice}} \eta_{\text{d}} T_{\text{ice}} w_{\text{ice}} \tag{7}$$

(4) Motor: Similar to the IC engine, the motor works by limiting the requested torque according to the torque-curve of the motor (Yang, 2022). The motor is assumed to work at the maximum torque at speeds lower than a base speed, and then decaying torque as speed increases. Braking is simulated by providing regenerative torque by the motor and forwarding the remaining of the requested as manual braking force.

$$T_{\text{mot}} \begin{cases} \leq T_{\text{mot,max}}, & \text{if } T_{\text{req}} \geq 0, \\ \geq -T_{\text{mot,max}}, & \text{if } T_{\text{req}} < 0. \end{cases} \tag{8}$$

$$T_{\text{mot,max}} = \begin{cases} T_{\text{mot,abs,max}}, & \text{if } \omega_{\text{mot}} \leq \omega_{\text{mot,base}}, \\ \dfrac{P_{\text{mot,abs,max}}}{\omega_{\text{mot}}}, & \text{if } \omega_{\text{mot}} > \omega_{\text{mot,base}}. \end{cases} \tag{9}$$

$$\omega_{\text{mot,base}} = \frac{P_{\text{mot,abs,max}}}{T_{\text{mot,abs,max}}} \tag{10}$$

$$P_{\text{elec}} = \eta_{\text{mot}} \eta_{\text{d}} T_{\text{mot}} w_{\text{mot}} \tag{11}$$

(5) Generator: A simple generator model with the battery model incorporated. The generator simply calculates the provided mechanical power and converts it into electrical power at a constant efficiency.

$$P_{\text{elec}} = \eta_{\text{gen}} T_{\text{in}} \omega_{\text{in}} \tag{12}$$

(6) Transmission and Gearbox: A simple multiplication of gear ratio with the provided torque and feedback speed. The gear ratios are modified based on provided threshold speeds. For gearbox, the model is the same but the gear ratio is fixed.

$$T_{\text{out}} = \eta i_g T_{\text{in}}, \quad \omega_{\text{in}} = i_g \omega_{\text{out}} \tag{13}$$

(7) Vehicle: A simple longitudinal vehicle dynamics model with resistive and tractive forces.

$$m\dot{v} = T_{\text{tr}} r_t - mgf_{\text{rr}} - \frac{1}{2}\rho C_D A v^2 - mg \sin\theta \tag{14}$$

(8) Driver: Essentially a PID (only proportional control is used) speed controller for the provided drive cycle. The actuators are controlled by providing a proportional torque demand proportional to the drive cycle speed. In case of a series hybrid layout, the engine is controlled based on the SOC of the battery. It is only turned on once SOC falls below a specified threshold, and then is kept on for a fixed time.

$$T_{\text{req}} = K_p v \tag{15}$$

3.2.3 Translation into model. Once models of all powertrain components have been developed in accordance with the presented approach, they are saved in a local Simulink library, ready to be used by the framework. After a topology is synthesized, it is translated into a Simulink model by selecting and arranging blocks from this library. A script reads the generated topology sequence and selects Simulink blocks based on the external attributes defined in Table 1. The driver controller block is then added to complete the system model. Parameters for each block are assigned by randomly selecting a parameter set from a predefined collection and applying it to the corresponding components. Finally, global simulation parameters are configured, preparing the system model for simulation.

3.3 Autonomous control tuning

After the formation of a forward-facing architecture, the PID controller is to be tuned to develop an optimal control strategy for the developed powertrain model. This is the second contribution of this paper, and ensures that an otherwise promising but novel topology is not discarded due to a poorer (but universal) control scheme. For this purpose, we utilize genetic algorithms, and this section details their implementation for this use case.

Genetic algorithms belong to the class of evolutionary algorithms and are inspired by genetics and natural selection. They are primarily used to find approximate solutions to optimization and search problems. Unlike Monte Carlo simulations, where the whole search process is based on randomness and the likelihood of reaching optimal values diminishes with increasing dimensions, the genetic algorithm has a structured approach, making them scale well to high-dimensional search spaces. (Mantri and Kulkarni, 2013; Ahmmed *et al*., 2020; Yusoff *et al*., 2015) Genetic Algorithms start with a random initial guess and then pivot towards the optimal values as they converge.

A genetic algorithm starts by initializing a population of possible solutions (referred to as individuals or chromosomes). Each individual in the population has a genome that encodes the solution. For example, to optimize PID gains, a genome would be [10, 2, 5] or [4 1 1]. The set of all the current solutions at a given point is called a generation. The starting generation is populated randomly or based on some heuristic. Next, a fitness function is used to simulate the process of natural selection, i.e. survival of the fittest. Each individual in the population is then evaluated using a cost function, called the fitness function, which quantifies how close the solution is to the optimal solution. Based on this evaluation, there are four phenomena through which the next generation is populated from the previous one, visualized in Figure 3.

(1) Elitism: The individuals with the highest fitness, i.e. least cost, are chosen unaltered for the next generation.

(2) Cross-over: Some individuals exchange values among themselves to create new ones for the next generation.

(3) Mutation: Some individuals undergo random changes in their values and pass them on to the next generation. It helps discover new solutions that were not possible otherwise with the starting gene pool and also prevents premature convergence to a local optimum.

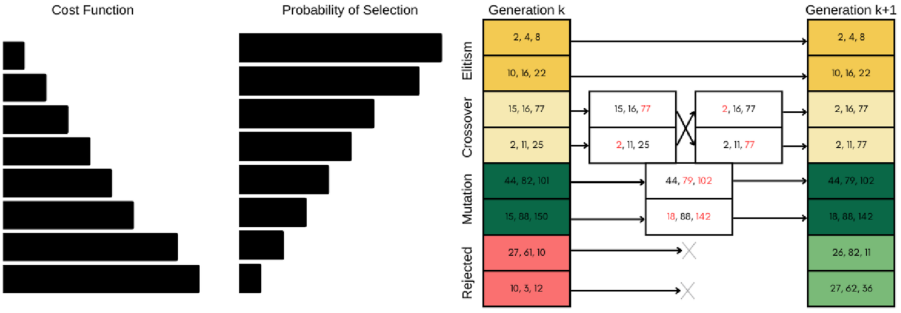


Figure 3. Working of a Genetic algorithm. Source: Authors' own work

(4) Rejection: The individuals with the highest cost are rejected and not selected for the next generation.

The algorithm loops either till the desired fitness score is achieved or until the maximum number of generations is reached. While forming a generation, a probabilistic approach guides operations following elitism. This makes each generation statistically more efficient than its predecessor.

Utilization of a genetic algorithm for PID tuning of a control system first requires the selection of a fitness function. Mirzal *et al.* (2012) used Integral square error (ISE), Integral absolute error (IAE), Integral time absolute error (ITAE), and Integral time square error (ITSE) to analyze different performance metrics (overshoot, settling time, rise time, etc.). ISE has a faster response but requires more control effort, while IAE has a smooth but slower response. ITSE leads to a fast response and easy control, but usually results in more oscillatory behavior. In the context of the problem at hand, a fast response and settling time are suitable, so ITAE is used. ITAE calculates the integral of the absolute error weighted by time and hence penalizes errors that persist longer. ITAE was chosen because it not only minimizes the error but also the time taken to minimize the desired response.

To improve computational efficiency for tuning, optimal input parameters for the genetic algorithm need to be selected. These include the number of generations, populations, and bounds for the values. For this purpose, the MATLAB Experiment Manager App was used to design an experiment aimed at deriving optimal PI gains for a hybrid powertrain operating on the WLTP cycle. 1728 parameter combinations were formed and evaluated based on average percentage error and tuning time. Based on its results, another refined experiment was then run. Details of parameters for both experiments are given in Table 2.

3.4 Evaluation

The evaluation process assesses the viability of the generated topologies and is a crucial step within the whole framework. The topology's Simulink model is simulated over the entire drive

Table 2. Parameter values for the genetic algorithm experiment

Parameter	Experiment 1	Experiment 2
Drive Cycle Length (s)	[30, 100, 200, 1800]	[30, 150, 1800]
Lower Bounds for PID gains	[0, 70]	[0]
Upper Bounds for PID gains	[No bound, 95, 100, 500]	[250, 500]
Generations	[2, 3, 5, 10]	[3, 5, 10]
Population	[5, 10, 20]	[5, 10]

cycle, and its performance is first evaluated based on the desired requirement of Mean Absolute Error (MAE). If the powertrain model achieves the provided drive cycle within an acceptable range of MAE, its features are saved and then assessed based on two numerical metrics: costs and emissions. These metrics are then visualized on a Cost vs. Emissions graph.

(1) **Cost:** For simplicity, only direct costs are considered, encompassing prices for individual components. Each component is assigned a cost based on current market prices and technical specifications. The total cost of the entire architecture is estimated by summing the costs of all its constituent components.

(2) **Emissions:** The environmental impact of the powertrain layouts is evaluated by calculating carbon emissions, which are divided into two categories:

- Cradle-to-Gate Emissions: These emissions are produced during the entire production process of a component, from the extraction of raw materials to the point of delivery. Each component is assigned a specific value, quantified in tons of CO2 emissions, based on broadly accepted estimation ranges. The total Cradle-to-Gate emissions for the overall architecture are determined by summing the emissions values of all constituent components. For the sake of simplicity, emissions associated with fuel production and battery charging are excluded from this calculation.
- Tank-to-Wheel Emissions: These emissions result from fuel combustion within the relevant powertrains. The simulation allows for the estimation of fuel consumption per unit distance. By applying an average value for a typical road vehicle's lifetime (in terms of total distance traveled) and the emissions per unit mass of fuel, the total emissions over the vehicle's life cycle are calculated.

3.5 Overall workflow

The workflow of the resulting framework is visualized in Figure 4, and described below.

(1) **Requirement Provision:** The framework provides a driving cycle, and the requirement is for a powertrain to achieve it within a certain range of Mean Absolute Error (MAE).

(2) **Topology Generation:** The topology is generated in the form of a string sequence, through a random, port compatibility-constrained combination of individual components. If the uniqueness constraint is applied, the topology is checked if it is distinct from the previously created ones, and discarded otherwise.

(3) **Model Generation:** A MATLAB script translates the topology to a mathematical model by fetching Simulink blocks from the custom component library. The model is forward-facing with PID control.

(4) **Parameterization:** Technical parameters are randomly assigned from a specified set to Simulink blocks in the layout.

(5) **Controller Tuning:** The Simulink model's PID speed controller is tuned using a Genetic algorithm against the provided drive cycle.

(6) **Evaluation:** The powertrain topology model is simulated for the entire drive cycle, and the results are presented in accordance with specified criteria. The topology is rejected if it does not meet the criteria; otherwise, the model and results are saved. This whole process iterates until the number of requested topologies is produced. Finally, the performance of all topologies is presented in the form of a Costs vs. Emissions graph.

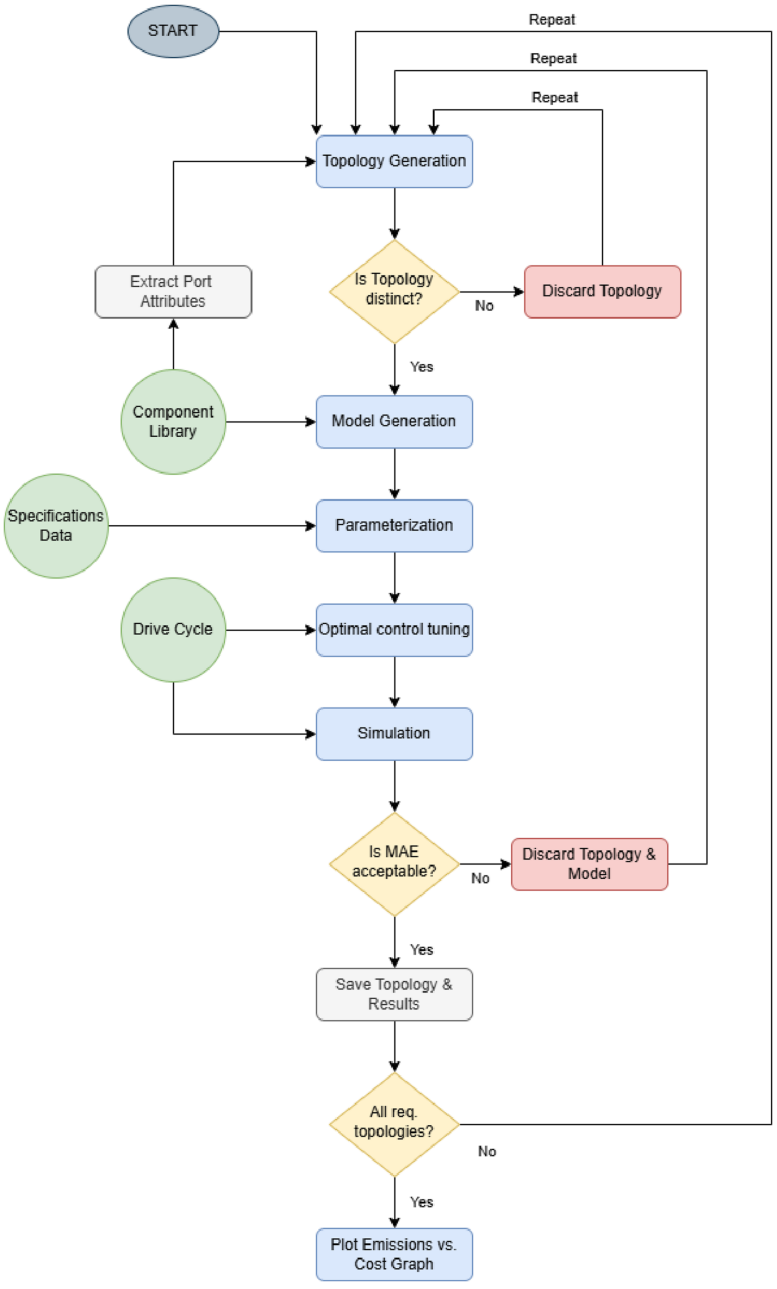


Figure 4. Workflow of the Proposed Framework. Source: Authors' own work

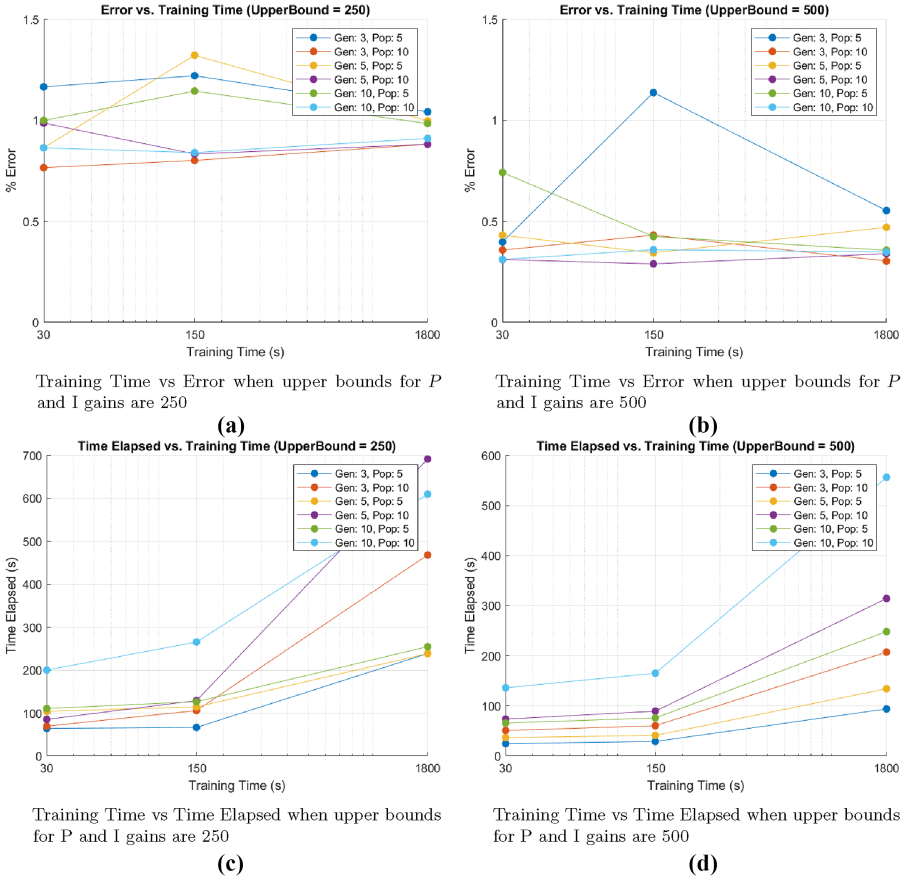
4. Results

4.1 Optimal auto-tuning hyper-parameters

The first experiment results helped in identifying that genetic algorithm when trained on smaller speed profiles and larger bounds for PID gains performed better in general. So, the parameters were updated for a second experiment and 72 new input parameter combinations were run (see Table 2). The results for the second experiment (Figure 5) showed that the algorithm with only a 30 s snippet of the speed profile, gains between 0 and 500, with 3 generations and a population of 5 individuals per generation yielded the most optimal results in terms of error minimization and computational efficiency (time).

Worldwide Harmonized Light Vehicles Test Procedure (WLTP) drive Cycle (1800s) was then run with the obtained PID gains. The mean percentage absolute error observed was 0.31% and it took 27 s for the algorithm to find the gains. It is to be noted that, the minimum error (0.29%) is obtained when the parameters for the algorithm are set to 5 generations with a population of 10 but the computation time is above 100 s. However, with the input parameters of 3 generations with a population of 5 the error obtained was 0.31% but the computation time was just 27 s. Therefore, the latter parameters were chosen.

Furthermore, the percentage error is higher for all the generations, population and simulation time combinations when the bounds are between 0 and 250. The reason is that the



Training Time vs Error when upper bounds for P and I gains are 250

(a)

Training Time vs Error when upper bounds for P and I gains are 500

(b)

Training Time vs Time Elapsed when upper bounds for P and I gains are 250

(c)

Training Time vs Time Elapsed when upper bounds for P and I gains are 500

(d)

Figure 5. Results of the genetic algorithm parameter optimization experiment

values for optimal gains are above 250. Consequently, it is better to have bigger bounds to make sure that the algorithm has the correct search space to finds the gains. Lastly, to validate the performance of the algorithm, the gains were determined for a different drive cycle (New York City Cycle) and a simplified DC motor model. The resultant errors were 0.34% and 0.12% respectively.

4.2 Topology synthesis and evaluation

The developed framework was tested with WLTP drive cycle due to its widespread utilization in designing different types of vehicle powertrains. The threshold for MAE for an acceptable architecture was set to 10 km/h. Additional constraints include utilization of each item only once, therefore preventing items to be repeated. This was done primarily to prevent generation of complex series hybrid layouts e.g. a topology containing two generators, which would require a much more complex controller. Two tests were conducted, once without the uniqueness constraint (thereby allowing repeated topologies) and once with it.

The results in Table 3 show that while the uniqueness constraint increase processing time (due to rejections of repeated layouts), the overall impact on simulation time is small since duplicates are rejected and not simulated.

Table 4 and Table 5 show the accepted and rejected powertrain topologies from run 1 and run 2, respectively. The table categorizes layouts based on included components as Internal Combustion Engine Vehicle (ICEV), Electric Vehicle (EV), and Series Hybrid Electric Vehicle (SHEV). Topologies labeled in parentheses were rejected due to simulation results for one of two reasons:

(1) Failure to achieve the given drive cycle within acceptable MAE.

(2) Duplication (under the uniqueness constraint), ensuring distinct topologies.

Performance of select powertrain topologies as plots of reference and achieved vehicle speeds is presented in Figure 6. Figure 7 presents the Cost vs. Emissions graphs for both runs, allowing an overview of the generated architectures usability. The labeled points refer to the corresponding accepted topology which can be located in Table 4 and Table 5. The green line marks an approximate Pareto Optimal front that can be established for this design space, and solutions below this front (i.e. lower costs and emissions) are not possible.

5. Discussion

5.1 Automatized control tuning

Efficient PID tuning with the genetic algorithm required balanced parameter bounds. Narrow bounds restricted the search space, while overly broad or unbounded ranges increased computation time and led to unstable or sub-optimal results. Choosing an appropriate

Table 3. Summary of the framework runs

	Run 1	Run 2
MAE Requirement	10	10
Uniqueness Constraint	No	Yes
Generated Topologies	17	23
Rejected Topologies	2	8
Required Topologies	15	15
ICEV Topologies	4	9
EV Topologies	7	4
SHEV Topologies	6	9
Simulation Time (minutes)	35	41

Table 4. Topologies generated during run 1 (without uniqueness constraint i.e. duplicate topologies possible). Note that two same topologies may have different component specs

No.	Powertrain topology	Accepted	Type
1	BAT – MOT - TR – VEH	Yes	EV
2	FT – ICE - TR – GB - VEH	Yes	ICEV
3	FT – ICE - TR – GEN - MOT – VEH	Yes	SHEV
(4)	FT – ICE - VEH	No	ICEV
4	BAT – MOT - TR – VEH	Yes	EV
5	FT – ICE - GEN – MOT - VEH	Yes	SHEV
6	BAT – MOT - VEH	Yes	EV
7	FT – ICE - TR – GB - GEN – MOT - VEH	Yes	SHEV
8	FT – ICE - GEN – MOT - VEH	Yes	SHEV
9	FT – ICE - GEN – MOT - VEH	Yes	SHEV
10	BAT – MOT - TR – VEH	Yes	EV
(11)	FT – ICE - VEH	No	ICEV
11	BAT – MOT - TR – GB - VEH	Yes	EV
12	FT – ICE - TR – GEN - MOT – GB - VEH	Yes	SHEV
13	BAT – MOT - GB – VEH	Yes	EV
14	BAT – MOT - GB – VEH	Yes	EV
15	FT – ICE - VEH	Yes	ICEV

Table 5. Topologies generated during run 2 (with uniqueness constraint i.e. all accepted topologies are distinct)

No.	Powertrain topology	Accepted	Type
1	BAT – MOT - TR – VEH	Yes	EV
(2)	FT – ICE - VEH	No, failed	ICEV
2	FT – ICE - GB – VEH	Yes	ICEV
3	FT – ICE - TR – GEN - MOT – VEH	Yes	SHEV
4	FT – ICE - GEN – MOT - VEH	Yes	SHEV
(5)	BAT – MOT - VEH	No, failed	SHEV
5	BAT – MOT - VEH	Yes	EV
6	FT – ICE - TR – GB - VEH	Yes	ICEV
7	FT – ICE - GB – TR - VEH	Yes	ICEV
(8)	FT – ICE - VEH	No, failed	ICEV
(8)	FT – ICE - VEH	No, failed	ICEV
8	FT – ICE - TR – VEH	Yes	ICEV
9	BAT – MOT - GB – VEH	Yes	EV
10	FT – ICE - TR – GEN - MOT – GB - VEH	Yes	SHEV
11	FT – ICE - GEN – MOT - GB – VEH	Yes	SHEV
(12)	FT – ICE - VEH	No, duplicate	ICEV
(12)	BAT – MOT - GB – VEH	No, duplicate	EV
12	FT – ICE - TR – GB - GEN – MOT - VEH	Yes	SHEV
13	BAT – MOT - TR – GB - VEH	Yes	EV
(14)	FT – ICE - TR – GB - GEN – MOT - VEH	No, duplicate	SHEV
14	FT – ICE - GEN – MOT - TR – GB - VEH	Yes	SHEV
(15)	FT – ICE - GEN – MOT - TR – GB - VEH	No, duplicate	SHEV
15	FT – ICE - TR – VEH	Yes	ICEV

population size and generation count was also critical. Larger populations improved diversity and convergence but increased computational cost. In practice, starting with moderate values and adjusting based on convergence behavior gave the best balance of performance and efficiency. The control tuning is still limited by manual exploration and setting of the hyperparameters, which therefore were considered in a limited range and granularity. Future work could look into making this hyper-parameter tuning process more autonomous.

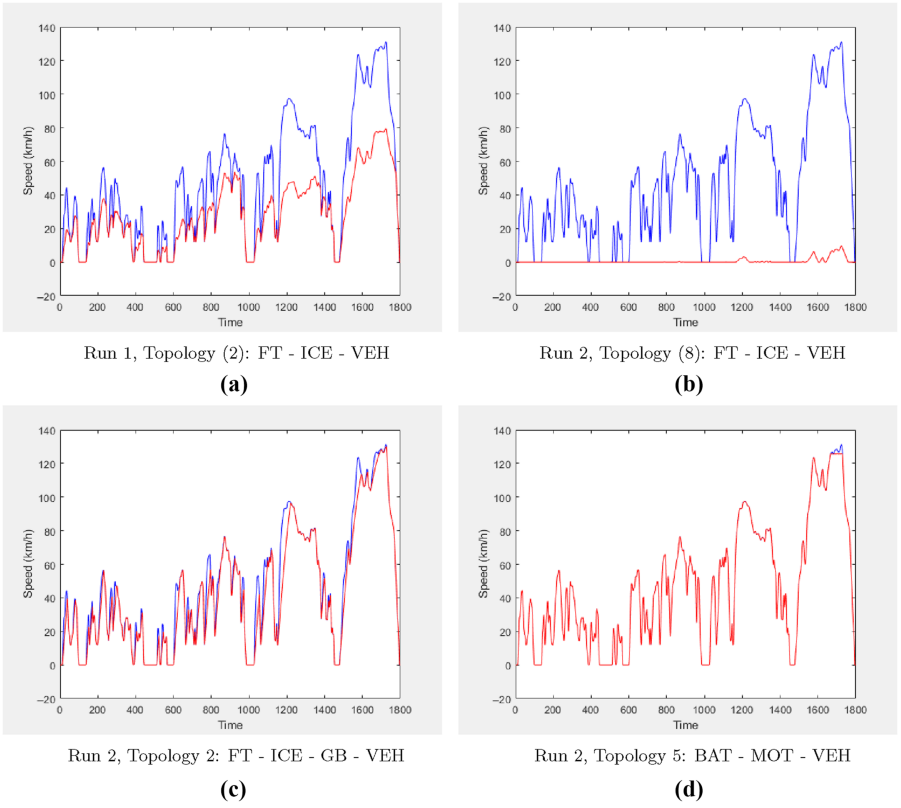


Figure 6. Performance plots of select topologies. Red is achieved speed and Blue is reference drive cycle speed. Topologies with numbered labels surrounded by parentheses are the rejected ones

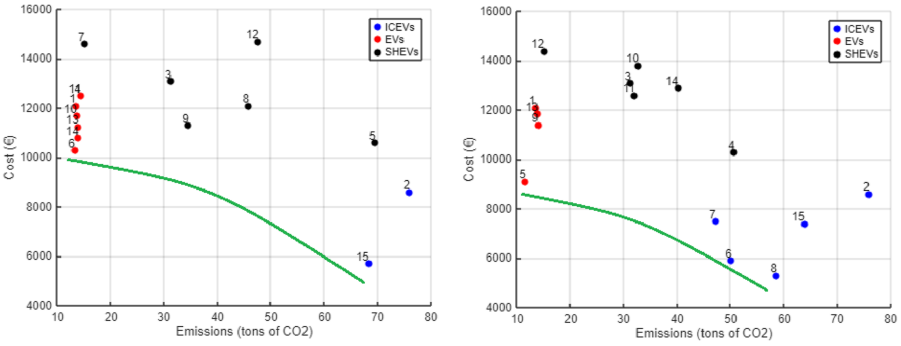


Figure 7. Plots of evaluation metrics for each layout generated during run 1 (left) and 2 (right). Green line marks the estimated Pareto-Optimality front

The complete framework, including automatic control tuning, was evaluated through the generated powertrain topologies and their simulated performance. The cost–emission plots in Figure 7 show the expected clustering for both runs. ICEVs occupy the lower-right region, showing low cost but high emissions. EVs appear in the upper-left, with zero tank-to-wheel

emissions but higher costs from battery systems. SHEVs, the only hybrid type allowed by the topology constraint, lie between the two. They show lower emissions than ICEVs but a higher cost due to combined engine and electric components. Some SHEVs appear as cost outliers due to the random selection of oversized engines intended for ICE layouts.

In run 1, all failed topologies (Table 4, (4), (11)) and three of four failed topologies from run 2 (Table 5, (2), (8), (8)) were ICEVs without a transmission or gearbox. Their rejection is technically justified: an engine directly coupled to the wheels cannot provide adequate torque to meet the drive cycle. In simulation, such vehicles show severe under-performance: as seen in Figure 6 (a) and (b), one moves slowly while the other remains nearly stationary.

An accepted topology in Figure 6 (c) includes an ICE with a gearbox but no transmission. It allows propulsion but prevents the engine from operating near its efficient speed range, resulting in high fuel use and emissions. This is reflected in Figure 7 (b), where Topology 2 appears far to the right, indicating inefficient operation.

Overall, the results align with expected technical behavior and confirm that the modeling fidelity is sufficient for evaluating automatically generated powertrain topologies. The successful operation of autonomous control tuning shows that control optimization can be effectively embedded within computational design synthesis. Although the framework was not benchmarked against an approach without tuning, the realistic and consistent simulation outcomes support confidence in the integrated approach.

A comparison of the two synthesis runs (Figure 7, Table 3) reveals how parameter assignment and structural diversity interact in simulation-driven design. In the unconstrained run 1, results cluster around certain architectures, especially EV layouts. These configurations are both simpler to generate and likely to satisfy acceptance thresholds, causing the algorithm to resample the same structure with different parameters. The spread within each cluster reflects performance variation driven almost entirely by parameter differences rather than topological change. When the uniqueness constraint is introduced (run 2), these repetitions are eliminated, forcing exploration of complex configurations such as ICEVs and hybrids (Table 3). The resulting point cloud is more evenly distributed across the cost–emission space. However, the apparently superior and lower Pareto front in this second case stems from a favorable parameter draw for an existing EV topology rather than from discovering a new structure.

This outcome highlights a broader issue in computational design synthesis: every generated topology must be instantiated with a parameter set before it can be evaluated, and this choice can dominate the perceived quality of the topology. In the joint topology–parameter design space, parameterization defines the starting position of each candidate along a high-dimensional performance surface. Poor parameter initialization can, therefore, misrepresent a topology's potential, while optimization for every candidate is computationally prohibitive.

To balance efficiency and fairness, a few practical directions can be considered:

(1) When the solution space is small, all possible topologies may be generated (even exhaustively) and only structurally unique or promising ones simulated and optimized. This reduces computational cost but introduces the risk of human bias, since superficially "uninteresting" layouts might still offer high latent performance.

(2) The topologies which do not meet the error threshold should be evaluated by testing on additional one or two sets of parameters to ensure a potentially good design is not rejected due to an unoptimal set of parameters.

(3) Knowledge transfer between similar topologies offers a third, more scalable avenue. As emerging AI and graph-based methods mature, parameter information from previously evaluated layouts could inform the initialization of new, structurally similar ones. This could be either through similarity metrics or learned embeddings. Such approaches could ultimately enable efficient reuse of parametric knowledge across large, combinatorial design spaces.

Future research could extend this analysis to non-sequential or branched topologies to test the observed trends in larger and more complex design spaces. In addition, the proposed strategies of evaluating topologies through enabling knowledge transfer between structurally similar layouts should be implemented and tested to quantify their impact on synthesis efficiency and bias reduction.

A key challenge in CDS research, as noted in subsection 2.3, lies in the manual translation of synthesized topologies into simulation-ready models. Addressing this step through automated model construction, potentially supported by emerging AI tools such as LLMs, represents a promising direction for improving the scalability and autonomy of CDS workflows.

6. Conclusion

In this work, a MATLAB-based, simulation-driven, CDS workflow with an integrated genetic algorithm-based PID control optimization module is proposed. The framework was applied to the synthesis and evaluation of sequential vehicle powertrain topologies. The topologies were constructed and parametrized randomly to encourage maximum exploration of the design space. The work also investigated the relative influence of configuration diversity and parametric diversity on design-space exploration by synthesizing and simulating sets of topologies with and without enforcing structural distinction.

Key findings from this study include:

(1) The generated layouts perform as anticipated when assessed using cost and emissions metrics, both collectively as a set and in terms of their individual performance. The realistic and consistent simulation outcomes thus support confidence in the integrated control tuning approach.

(2) A comparison of synthesis runs with and without a topology uniqueness constraint revealed key patterns. Parameter variation dominated performance clustering in the unconstrained case, while the uniqueness constraint promoted exploration of structurally distinct configurations. Though the constrained case resulted in a more favorable Pareto front, it resulted from more favorable parameter initialization. These findings indicate that parameter initialization strongly influences perceived performance. A topology's parameter assignment defines its effective position in the joint topology–parameter design space and poor initialization can bias evaluation outcomes even when structural diversity is maintained.

Future computational design synthesis workflows could consider the following three strategies. First, each rejected topology could be evaluated using multiple parameters to ensure a potentially good topology is not rejected due to stochastic bias. Second, topology generation and parameter optimization should be separated when possible. All topologies can be generated first, and only promising ones are simulated and optimized in detail. This method is suitable for small or well-bounded design spaces but may not scale efficiently. Third, parameter information should be transferred between similar topologies. Similarity can be defined using graph-based metrics or learned through data-driven methods. Such knowledge transfer can improve parameter initialization for new designs. The hyper-parameter tuning process for the genetic algorithm for a given application could be made more autonomous based on the reference drive cycle. In addition, research should tackle the persistent challenge of manually converting synthesized topologies into simulation models, potentially through emerging AI tools such as LLMs.

Data availability

In the interest of transparency, data sharing, and reproducibility, the author(s) of this article have made the code and data underlying their research openly available. It can be accessed by following the link here: https://github.com/rohailamalik/powertrain-cds-with-auto-control-tune

Acknowledgments

ChatGPT was used to rephrase some of the text in this paper. All the content has been originally written, with ChatGPT merely used to enhance the vocabulary and flow of the text.

References

Ahmmed, T., Akhter, I., Karim, S.R. and Ahamed, F.S. (2020), "Genetic algorithm based pid parameter optimization", *American Journal of Intelligent Systems*, Vol. 10 No. 1, pp. 8-13, doi: 10.5923/j.ajis.20201001.02.

Alber, R. and Rudolph, S. (2003), "'43' - a generic approach for engineering design grammars",

Anderson, F., Grossman, T. and Fitzmaurice, G. (2017), "Trigger-action-circuits: leveraging generative design to enable novices to design and build circuitry", *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*, pp. 331-342, doi: 10.1145/3126594.3126637.

Bolognini, F., Seshia, A. and Shea, K. (2007), "Exploring the application of a multi-domain simulation-based computational synthesis method in MEMS design", *Proceedings of 14th International Conference on Engineering Design (ICED '07)*, pp. 81-82.

Cagan, J. (2001), *Formal Engineering Design Synthesis*, Cambridge University Press, Cambridge.

Cagan, J., Campbell, M., Finger, S. and Tomiyama, T. (2005), "A framework for computational design synthesis: model and applications", *Journal of Computing and Information Science in Engineering - JCISE*, Vol. 5 No. 3, pp. 171-181, doi: 10.1115/1.2013289.

Caldas, L. (2008), "Generation of energy-efficient architecture solutions applying gene_arch: an evolution-based generative design system", *Advanced Engineering Informatics*, Vol. 22 No. 1, pp. 59-70, doi: 10.1016/j.aei.2007.08.012.

Chakrabarti, A., Shea, K., Stone, R., Cagan, J., Campbell, M., Vargas Hernandez, N. and Wood, K. (2011), "Computer-based design synthesis research: an overview", *Journal of Computing and Information Science in Engineering*, Vol. 11 No. 2, 021003, doi: 10.1115/1.3593409.

Demirel, H.O., Goldstein, M.H., Li, X. and Sha, Z. (2023), "Human-centered generative design framework: an early design framework to support concept creation and evaluation", *International Journal of Human-Computer Interaction*, Vol. 40 No. 4, pp. 1-12, doi: 10.1080/10447318.2023.2171489.

Elrefaie, M., Qian, J., Wu, R., Chen, Q., Dai, A. and Ahmed, F. (2025), "Ai agents in engineering design: a multi-agent framework for aesthetic and aerodynamic car design", *Volume 3B: 51st Design Automation Conference (DAC)*, doi: 10.1115/detc2025-169682, available at: https://arxiv.org/abs/2503.23315

Gadeyne, K., Pinte, G. and Berx, K. (2014), "Describing the design space of mechanical computational design synthesis problems", *Advanced Engineering Informatics*, Vol. 28 No. 3, pp. 198-207, doi: 10.1016/j.aei.2014.03.004.

Gangurde, L., Gawande, A., Khanvilkar, S., Khochare, A. and Dave, P. (2019), "Modelling and control of electric car powertrain", *2019 1st International Conference on Innovations in Information and Communication Technology (ICIICT)*, pp. 1-5, doi: 10.1109/iciict1.2019.8741417.

Helms, B. and Shea, K. (2012), "Computational synthesis of product architectures based on object-oriented graph grammars", *Journal of Mechanical Design*, Vol. 134 No. 2, 021008, doi: 10.1115/1.4005592.

Helms, B., Shea, K. and Hoisl, F. (2009), "A framework for computational design synthesis based on graph-grammars and function-behavior-structure", *Volume 8: 14th Design for Manufacturing and the Life Cycle Conference; 6th Symposium on International Design and Design Education; 21st International Conference on Design Theory and Methodology, Parts A and B*, Vol. 8, pp. 841-851, doi: 10.1115/detc2009-86851.

Karniadakis, G.E., Kevrekidis, I.G., Lu, L., Perdikaris, P., Wang, S. and Yang, L. (2021), "Physics-informed machine learning", *Nature Reviews Physics*, Vol. 3 No. 6, pp. 422-440, doi: 10.1038/s42254-021-00314-5.

Kazi, R.H., Grossman, T., Cheong, H., Hashemi, A. and Fitzmaurice, G. (2017), "Dreamsketch: early stage 3d design explorations with sketching and generative design", *'Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology', UIST '17*, Association for Computing Machinery, NY, USA, pp. 401-414, doi: 10.1145/3126594.3126662.

Kriegman, S., Blackiston, D., Levin, M. and Bongard, J. (2020), "A scalable pipeline for designing reconfigurable organisms", *Proceedings of the National Academy of Sciences*, Vol. 117 No. 4, pp. 1853-1859, doi: 10.1073/pnas.1910837117.

Li, H. (2024), "Research on calculating the internal resistance of battery cell", *The Frontiers of Society, Science and Technology*, Vol. 6 No. 6, pp. 81-85.

Liang, V.-C. and Paredis, C.J.J. (2004), "A port ontology for conceptual design of systems", *Journal of Computing and Information Science in Engineering*, Vol. 4 No. 3, pp. 206-217, doi: 10.1115/1.1778191.

Malik, R., Ahmad, M. and Vepäläinen, J. (2024), *Simulation-Driven Computational Synthesis of Vehicle Powertrain Topologies with Integrated Autonomous Control Tuning*, GitHub repository, available at: https://github.com/rohailamalik/powertrain-cds-with-auto-control-tune

Mantri, G. and Kulkarni, N. (2013), "Design and optimization of pid controller using genetic algorithm", *International Journal of Engineering Research and Technology*, Vol. 2 No. 6, pp. 926-930.

Massoudi, S. and Fuge, M. (2025), "Agentic large language models for conceptual systems engineering and design", *Journal of Mechanical Design*, pp. 1-20, doi: 10.1115/1.4070328, available at: https://arxiv.org/abs/2507.08619

Mirzal, A., Yoshii, S. and Furukawa, M. (2012), "Pid parameters optimization by using genetic algorithm", arXiv preprint arXiv:1204.0885 .

Mittal, S. and Frayman, F. (1989), "Towards a generic model of configuraton tasks", *'Proceedings of the 11th International Joint Conference on Artificial Intelligence - Volume 2', IJCAI'89*, Morgan Kaufmann Publishers, San Francisco, CA, USA, pp. 1395-1401.

Mohan, G., Assadian, F. and Longo, S. (2013), "Comparative analysis of forward-facing models vs backward-facing models in powertrain component sizing", *IET Hybrid and Electric Vehicles Conference 2013 (HEVC 2013)*, pp. 1-6.

Münzer, C. and Shea, K. (2016), "An integrated approach to automated synthesis, simulation and optimization of energy and signal-based design concepts", Vol. 7: *28th International Conference on Design Theory and Methodology of International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, doi: 10.1115/DETC2016-59816

Münzer, C. and Shea, K. (2017), "Simulation-based computational design synthesis using automated generation of simulation models from concept model graphs", *Journal of Mechanical Design*, Vol. 139 No. 7, 071101, doi: 10.1115/1.4036567.

Münzer, C., Helms, B. and Shea, K. (2013), "Automatically transforming object-oriented graph-based representations into boolean satisfiability problems for computational design synthesis", *Journal of Mechanical Design*, Vol. 135 No. 10, 101001, doi: 10.1115/1.4024850.

Oh, S., Jung, Y., Lee, I. and Kang, N. (2018), Design automation by integrating generative adversarial networks and topology optimization, Vol. 2, *44th Design Automation Conference of International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, p. V02AT03A008, doi: 10.1115/DETC2018-85506

Orsborn, S., Cagan, J. and Boatwright, P. (2008), "A methodology for creating a statistically derived shape grammar composed of non-obvious shape chunks", *Research in Engineering Design*, Vol. 18 No. 4, pp. 181-196, doi: 10.1007/s00163-007-0035-9.

Pahl, G., Beitz, W., Feldhusen, J. and Grote, K.-H. (2007), "Engineering design: a systematic approach",

Peckham, O., Raines, J., Bulsink, E., Goudswaard, M., Gopsill, J., Barton, D., Nassehi, A. and Hicks, B. (2025), "Artificial intelligence in generative design: a structured review of trends and

opportunities in techniques and applications", *Design*, Vol. 9 No. 4, p. 79, doi: 10.3390/designs9040079.

Ramesh, B., Rj, A., Singla, A.K., Chandra, P.K., Sethi, V.A. and Abood, A.S. (2024), "Utilizing generative design algorithms for innovative structural engineering solutions", *E3S Web of Conferences*, available at: https://api.semanticscholar.org/CorpusID:268701317

Salazar, D. and Garcia, M. (2022), "Estimation and comparison of soc in batteries used in electromobility using the thevenin model and Coulomb ampere counting", *Energies*, Vol. 15 No. 19, 7204, doi: 10.3390/en15197204.

Shea, K. (2003), "From discrete structures to mechanical systems: a framework for creating performance-based parametric synthesis tools",

Shea, K., Stanković, T., Agrawal, A., Cagan, J. and McComb, C. (2025), "Expanding the generative power of large language models (llms) for design through formal design grammars and languages", *Journal of Computing and Information Science in Engineering*, pp. 1-34, doi: 10.1115/1.4070095.

Simpson, T.W., Poplinski, J.D., Koch, P.N. and Allen, J.K. (2001), "Metamodels for computer-based engineering design: survey and recommendations", *Engineering with computers*, Vol. 17 No. 2, pp. 129-150, doi: 10.1007/pl00007198.

Starling, A.C. and Shea, K. (2005), "A parallel grammar for simulation-driven mechanical design synthesis", *31st Design Automation Conference, Parts A and B of International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, Vol. 2, pp. 427-436, doi: 10.1115/DETC2005-85414

Sudret, B., Marelli, S. and Wiart, J. (2017), "Surrogate models for uncertainty quantification: an overview", *2017 11th European conference on antennas and propagation (EUCAP)*, IEEE, pp. 793-797.

Tkachenko, A. and Wang, C.J. (2024), "Analysis of engineering structures and components created using the generative design", *Journal of Physics: Conference Series*, Vol. 2762 No. 1, p. 012089, doi: 10.1088/1742-6596/2762/1/012089, available at: https://api.semanticscholar.org/CorpusID:270240312

Tomar, V., Chitra, A., Krishnachaitanya, D., Rao, N.S.R., V, I. and Raziasultana, W. (2021), "Design of powertrain model for an electric vehicle using matlab/simulink", in *2021 Innovations in Power and Advanced Computing Technologies (i-PACT)*, pp. 1-7.

Tran, J., Fukami, K., Inada, K., Umehara, D., Ono, Y., Ogawa, K. and Taira, K. (2024), "Aerodynamics-guided machine learning for design optimization of electric vehicles", *Communications Engineering*, Vol. 3 No. 1, p. 174, doi: 10.1038/s44172-024-00322-0.

Umeda, Y. and Tomiyama, T. (1995), "Fbs modeling : modeling scheme of function for conceptual design", *9 Int. Workshop on Qualitative Reasoning About Physical Systemsth*.

Wang, Z., Lu, Z., Wang, J. and You, F. (2025), "IXAI: generative design of automotive styling based on inception convolution with explainable AI", *Journal of Engineering Design*, pp. 1-29, doi: 10.1080/09544828.2025.2481537.

Wielinga, B. and Schreiber, G. (1997), "Configuration-design problem solving", *IEEE expert*, Vol. 12 No. 2, pp. 49-56, doi: 10.1109/64.585104.

Wu, Z., Campbell, M. and Fernández Rodríguez, B. (2008), "Bond graph based automated modeling for computer-aided design of dynamic systems", *Journal of Mechanical Design*, Vol. 130 No. 4, 041102, doi: 10.1115/1.2885180.

Yang, Q. (2022), "Simulation of electric vehicles based on simulink", *Journal of Physics: Conference Series*, Vol. 2417 No. 1, 012012, doi: 10.1088/1742-6596/2417/1/012012.

Yogev, O., Shapiro, A.A. and Antonsson, E.K. (2010), "Computational evolutionary embryogeny", *IEEE Transactions on Evolutionary Computation*, Vol. 14 No. 2, pp. 301-325, doi: 10.1109/tevc.2009.2030438.

Yusoff, T.A.F.K., Atan, M.F., Rahman, N.A., Salleh, S.F. and Wahab, N.A. (2015), "Optimization of pid tuning using genetic algorithm", *Journal of Applied Science and Process Engineering*, Vol. 2 No. 2, pp. 97-106.

Zang, T., Yang, M., Liu, Y. and Jiang, P. (2024), "Text2shape: intelligent computational design of car outer contour shapes based on improved conditional wasserstein generative adversarial network", *Advanced Engineering Informatics*, Vol. 62, 102892, doi: 10.1016/j.aei.2024.102892.

Corresponding author
Rohail Malik can be contacted at: rohail.malik@aalto.fi