Graph-Based System Architecture Optimisation

Study of Graph-Based Surrogate Modelling Approaches for Black-Box Optimization of Aerospace System Architectures

by

Franciszek Latała

Thesis Supervisor:	Jasper Bussemaker
Thesis Advisor:	Elvin Isufi
Project Duration:	February, 2026 - September, 2026
Faculty:	Electrical Engineering, Mathematics and Computer Science

TU Delft

Contents

1 Introduction

Complex engineered systems must satisfy increasingly demanding and often conflicting requirements concerning performance, cost, safety, and environmental impact [1]. At the same time, designers can choose from a growing range of technologies and integration concepts. Aircraft design provides a clear example, where technologies intended to reduce fuel consumption or emissions may increase mass, cost, and integration complexity or introduce new safety constraints [2, 3]. Many of the most consequential decisions are therefore made during conceptual design, when the system architecture is defined—that is, when designers decide which functions the system must perform, which components will be used to fulfill them, and how these selected components will interact with each other. These early architectural decisions shape much of the system's eventual performance and can be costly to revise once detailed development has begun [4].

It is therefore important to identify a high-performing system architecture before committing to further development. However, doing so can be difficult because architectural decisions are interdependent and combinatorial. A system's function may be fulfilled by different components or combinations of components, and each such decision may enable different downstream choices concerning component-specific parameters and feasible connections. Consider, for example, an aircraft propulsion architecture in which the function of generating thrust may be fulfilled by either a jet engine or a turboprop. Selecting a jet engine makes its bypass ratio a relevant sizing parameter to consider, whereas selecting a turboprop instead requires optimizing its propeller blade count. The selected component must then be connected to the fuel system, where the permissible number of fuel-line connections may also depend on the specific engine choice. Even this small example admits many distinct architectures. For more complex systems, the number of feasible alternatives can grow rapidly, making it impractical or even infeasible to enumerate and compare them all [4, 5].

To address this challenge, System Architecture Optimization (SAO) has emerged within systems engineering as a way to automate the process of generating and comparing potential architectures by using numerical optimization algorithms, rather than relying solely on experience and established design principles [1, 5]. At the core of SAO lies a formally defined architecture design space, which is the set of all feasible architectures that can be constructed from the available component, parameter, and connection choices while satisfying the rules governing their combination. During the numerical search, candidate architectures from this space are assessed using some quantitative evaluation model, which returns objective and constraint values that are then used to guide the optimization algorithm towards high-performing solutions.

In realistic SAO problems, the evaluator often consists of multidisciplinary or physics-based simulation workflows that effectively act as black boxes, which means that they can provide objective and constraint values for a candidate architecture, but not the gradients with respect to specific choices. Moreover, many of the optimized decisions are discrete and their feasible ranges can be highly interconnected. Together, this makes gradient-based optimization impractical, and most frequently infeasible [1, 5]. SAO consequently relies on gradient-free methods. For example, genetic algorithms iteratively evolve populations of candidate architectures by selecting, combining, and modifying the designs that perform best when evaluated. However, such population-based methods typically require thousands of evaluations to identify high-performing solutions. When each evaluation involves a

physics-based multidisciplinary analysis, the resulting computational cost can become prohibitive [5].

To reduce this computational burden, surrogate-based optimization (SBO) has been successfully adopted in SAO [1, 5]. In SBO, a comparatively inexpensive predictive model, known as a surrogate, is fitted using previously evaluated architectures and their performance values. The surrogate then estimates the performance of unevaluated candidates and guides the optimizer towards promising regions of the design space. Only the selected architectures are assessed using the expensive architecture evaluator, after which the new results are added to the data and the surrogate is updated. In this way, the available evaluation budget can be used more selectively. In particular, Bayesian optimization (BO) with Gaussian-process (GP) surrogates has been adopted in SAO due to its sample efficiency—that is, its ability to identify high-performing solutions using few expensive evaluations [1, 5, 6].

In this setting, the architecture representation used by the surrogate model is particularly important, since it determines how readily the model can identify performance-relevant properties and similarities between architectures. System architectures are inherently relational, as system functions may be fulfilled by different components and those components may be connected in different ways. They consequently admit a natural graph representation, in which functions, components, and other system elements are represented as nodes and their relationships as edges. Accordingly, graphs serve as the primary architecture representation adopted in SAO [1, 5].

However, current SAO approaches do not rely on surrogates that take architecture graphs as inputs for their predictions. Instead, architecture graphs are first encoded into fixed-length design vectors that record the architectural choices and the associated parameter values. Such a representation has two potential limitations. First, architectures need not contain the same components, resulting in variables that are active for some architectures but inactive for others. To account for this, existing methods rely on Gaussian processes with specialized kernels that incorporate variable activeness information [5, 7]. Second, structural information can be difficult to preserve. Encoding an entire connection pattern as a categorical variable does not distinguish between a one-edge modification and a completely different structure. More expressive encodings can preserve finer differences, but increase dimensionality and can still fail to capture global patterns such as paths or redundancy motifs. Together, these limitations motivate investigating surrogate models that operate directly on architecture graphs [1].

Related work suggests that graph-based surrogates can be effective in settings similar to SAO, where candidate solutions are represented as graphs and the objective is to identify those with the best performance. The closest analogue is neural architecture search (NAS), where candidate neural networks are modeled as graphs and their performance can only be determined through expensive training and validation. Bayesian optimization with graph-kernel Gaussian processes and graph-neural-network surrogates has been successfully applied in this setting [8, 9]. Graph-based surrogates have also been investigated for more general attributed-graph optimization problems, where they can exploit shared substructures when predicting the performance of unevaluated candidates [10, 11, 12].

Although these settings share SAO's central objective of identifying high-performing graphs using few evaluations, SAO has several distinctive characteristics. First, SAO covers a much broader range of systems, from sensor networks to propulsion architectures, and even NAS can itself be represented as an SAO problem in which neural-network operations and connections are architectural choices. A suitable surrogate therefore requires enough flexibility to adapt to different domains and ways in which structure affects performance, while remaining sample efficient. Second, SAO graphs can differ in size and node labels, while individual nodes may carry entirely different attributes or none at all. Compared with most related settings, this heterogeneity can place greater demands on the surrogate's ability to generalize from few evaluated architectures. Third, SAO commonly involves multiple objectives and constraints, whereas most of the related methods optimize a single scalar objective. Consequently, it remains unclear how well graph-based surrogates developed for other domains transfer to SAO and which surrogate family is most suitable. Graph-kernel Gaussian processes provide principled predictive uncertainty, are well suited to learning from limited data, and require relatively little problem-specific tuning, whereas graph-neural surrogates offer greater representational flexibility and potentially lower computational cost.

Because architecture evaluations often dominate the computational cost of SAO, sample efficiency is central. This thesis therefore set out to investigate whether surrogate models operating directly on architecture graphs could improve sample efficiency compared with existing vector-based approaches. This investigation was guided by the following overarching research question and two supporting subquestions.

Research question
To what extent can graph-based surrogate models improve the sample efficiency of System Architecture Optimization compared with existing vector-based approaches?

- **Subquestion 1:** *How does the relative sample efficiency of graph-based and vector-based surrogate models vary with the characteristics of the SAO problem, particularly the extent to which its objectives depend on connectivity and other structural patterns?*
- **Subquestion 2:** *How do graph-kernel Gaussian-process surrogates compare with graph-neural surrogate models in terms of sample efficiency across SAO problems?*

To answer these questions, I carried out the following work:

1. Developed a graph-kernel Gaussian-process surrogate that combines architecture structure, sizing parameters, and connection multiplicities, with extensions that learn which structural scales and levels of node-label detail are most relevant to the modeled response.
2. Adapted graph-neural surrogate approaches to SAO and compared them with the graph-kernel approach.
3. Evaluated graph-based and existing vector-based methods across SAO problems with different structural characteristics.

Thesis structure and reading guide
The next chapter presents the scientific article, which introduces the graph-kernel surrogates for SAO and constitutes the main contribution of this work. The main conclusions are presented within the article itself. The concluding thesis chapter returns to the broader research question and its subquestions, drawing on the article's findings and the supplementary GNN analysis to address them. Supplementary material is provided in two appendices. Appendix A offers an extended background on SAO, Bayesian optimization, and graph-based surrogate models, which readers seeking a more gradual introduction may consult before reading the self-contained article. Appendix B details the adaptation and additional analysis of the graph-neural surrogates considered in the article.

2

Scientific Article

The core research of this thesis is presented in the following scientific article. It introduces graph-kernel Gaussian-process surrogate models for System Architecture Optimization and evaluates their performance across a range of architecture optimization problems. The article is currently being prepared for submission to the *Journal of Global Optimization*[1].

[1] https://link.springer.com/journal/10898

System Architecture Optimization via Graph-Kernel Gaussian Process Surrogates

Franciszek Latała[1,2*], Jasper Bussemaker[1] and Elvin Isufi[2]

[1] Institute of System Architectures in Aeronautics, German Aerospace Center (DLR), Hein-Saß-Weg 22, Hamburg, 21129, Germany .

[2] Faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology, Mekelweg 4, Delft, 2628 CD, The Netherlands .

*Corresponding author(s).
Contributing authors:

Abstract

System Architecture Optimization (SAO) problems are typically mixed-discrete, hierarchical, and expensive to evaluate, motivating the application of Bayesian optimization, which is known for its sample efficiency. Current approaches for SAO use Gaussian processes (GPs) operating on encoded design vectors. These encodings can obscure similarities in component connectivity, making it harder for the GP to capture performance correlations between evaluated and unevaluated architectures and potentially reducing optimization sample efficiency. To address this, we propose a GP surrogate that incorporates similarities between architecture graphs through the Weisfeiler–Lehman optimal assignment kernel to better capture shared structural patterns. We also introduce extensions that facilitate adapting the neighborhood scale at which architecture graphs are compared, as well as the specificity with which structural patterns are distinguished. This allows learning those properties during GP fitting rather than tuning them beforehand, which is infeasible for SAO, as new problems typically lack prior simulation data. Across four benchmarks against encoding-based GPs, an alternative graph-kernel, graph neural networks, and model-free baselines, our method achieves the best optimization performance on connectivity-driven problems. It reduces regret by up to 40% and final normalized hypervolume error by up to 50% relative to the strongest encoding-based baseline. This comes at a higher computational cost, but the additional overhead is offset once architecture evaluations take approximately 4–25 seconds. On the remaining problems, our method remains competitive with state-of-the-art encoding-based approaches. These findings suggest that graph-based approaches could help engineers find better system architectures faster when simulations are expensive.

Keywords: System Architecture Optimization, Bayesian Optimization, Gaussian Processes, Graph Kernels, Weisfeiler–Lehman Kernel, Mixed-Discrete Optimization

1 Introduction

System architecture concerns the early design decisions that determine how complex systems meet stakeholder needs, including which components fulfill the required functions and how these components interact. These decisions are strongly interdependent, as choosing certain functions influences which components are needed, and those component choices in turn shape the possible connections and the overall system structure. Due to the combinatorial nature of architecture choices, it is difficult to design and identify a feasible architecture that is likely to perform well [1, 2], let alone find the best one. Moreover, selecting a non-optimal architecture early on can lead to costly redesigns later in development [3].

System Architecture Optimization (SAO) has recently emerged within systems engineering as a way to formulate architecting as an optimization problem [2, 4]. It combines a formal design space with quantitative evaluation of candidate architectures and numerical search algorithms in order to identify promising architectures without exhaustively evaluating all possibilities. SAO problems are typically mixed-discrete, hierarchical, constrained, and often multi-objective, with performance evaluated through expensive black-box physics-based simulations, which generally provide performance values but not gradients with respect to the input architecture. Together with the discrete design choices, this makes gradient-based optimization infeasible [2].

These characteristics make SAO well suited to surrogate-based optimization (SBO), in which a cheap surrogate model approximates the expensive objective function and is iteratively refined using newly evaluated architectures. This allows the optimizer to focus expensive evaluations on regions of the design space that the surrogate model identifies as promising. In particular, Bayesian optimization (BO) with Gaussian-process surrogate models is sample-efficient and does not require gradients [2, 4, 5]. These methods have already demonstrated strong performance in SAO [2, 4].

System architectures are inherently relational. System functions may be fulfilled by different components, which can themselves be connected in different ways. Therefore, they admit a natural graph representation, with functions, components, and other system entities represented as nodes and their relationships as edges. Accordingly, architecture graphs are the primary representation used in SAO [2, 4].

For optimization, however, existing GP-based methods map these graphs to fixed-length design vectors [2, 6]. This creates two problems. First, architectures need not contain the same components, resulting in inactive vector variables that require special treatment. Second, representing an entire connection pattern as a categorical value does not distinguish a one-edge modification from a completely different structure. More expressive encodings can preserve finer structural differences, but increase dimensionality and still do not allow conventional vector-based kernels to directly recognize global structures such as paths, redundancy structures, and other motifs.

Related work suggests that graph-based surrogates can be effective when candidate solutions are represented as graphs and the objective is to identify those with the best performance. A close analogue to SAO is neural architecture search (NAS), where candidate networks are modeled as graphs, with the nodes representing neural operators, edges describing the flow of information, and the objective being to identify the

lowest-error ones. Because evaluating a candidate network requires expensive training and validation, Bayesian optimization with graph-kernel GPs and graph neural network (GNN) surrogates has been widely and successfully adopted in NAS [7–11]. Another field in which graph-based surrogates have been used is molecular discovery. Here, molecules are represented as attributed graphs, with atoms as nodes and chemical bonds as edges, and the objective is to identify molecules with desirable properties [12, 13]. Beyond such domain-specific applications, graph-based Bayesian optimization has also been studied in broader attributed-graph settings incorporating node-, edge-, and graph-level features, for instance urban road-network design [14].

While these settings share SAO's central challenge of identifying high-performing graphs using few evaluations, they differ in several ways. First, SAO covers a much broader range of systems, from sensor networks and hydraulic systems to propulsion architectures. A suitable surrogate therefore requires enough flexibility to adapt to different domains and ways in which structure affects performance, while remaining sample efficient. Second, SAO graphs can be heterogeneous. For instance, nodes representing different component types may have entirely different sets of attributes. Third, SAO commonly involves multiple objectives and constraints, whereas most of the aforementioned methods optimize a single scalar objective. Consequently, applying graph-based surrogates to SAO requires adapting them to heterogeneous graphs and diverse relationships between structure and performance. This motivates developing surrogates tailored to SAO that can accommodate these diverse requirements while remaining sample efficient. Graph-kernel GPs, already successful in domains such as NAS, are a promising choice for this development because they can incorporate structural similarity while retaining principled uncertainty estimates required for Bayesian optimization. Moreover, their simplicity and small number of tunable parameters make them well suited to the limited training data available in SAO, while also making their behavior easier to analyze and interpret than that of more complex GNNs.

Because architecture evaluations often dominate the computational cost of SAO, sample efficiency is central. This work therefore investigates whether graph-kernel-based Gaussian-process surrogates can improve the sample efficiency of System Architecture Optimization. The contributions of this work are as follows:

- We formulate a graph-kernel-based Gaussian-process surrogate model for SAO architecture graphs that uses a composite kernel construction to capture multiple drivers of architecture variability, such as structure, design parameter values, and edge multiplicities.
- We introduce learned depth and multi-granularity variants of the Weisfeiler–Lehman optimal assignment kernel, which facilitate learning the relevant structural scale and semantic specificity during GP fitting. This removes the need for problem-specific tuning, which is generally infeasible in SAO since new problems typically lack prior simulation data.
- Through benchmarking against encoding-based GP and GNN surrogates, as well as model-free baselines, we provide empirical evidence that the proposed graph-kernel surrogate can improve sample efficiency on topology-driven SAO problems.
- We propose practical criteria for selecting an SAO approach based on the structural nature of the problem, particularly the importance of connectivity, and the cost of evaluating candidate architectures relative to surrogate-model overhead.

2 Background

This section covers the background relevant to this research. It first introduces SAO and the architecture design space graph (ADSG) as the graph-based representation of the feasible architecture space, as well as how concrete architecture graphs are derived from it. It then discusses surrogate-based optimization, followed by modeling methods that rely on vector encodings alongside their limitations.

2.1 SAO and architecture design space graphs

SAO formulates system architecting as an optimization problem over an *architecture design space*, that is the set of all system architectures admissible under the specified modeling assumptions. These design spaces arise from the core architecting tasks [1], namely allocating functions to components, characterizing these components, and defining how they are connected. The resulting design spaces have three defining properties that any suitable model must capture. First, they are *mixed-discrete*: they combine discrete architectural choices with architecture-specific sizing parameters; these choices include whether the system fulfills a given function, which components it uses and how many (e.g. the engine type and number of engines on an aircraft), and how various components connect; the sizing parameters may be continuous, integer, or categorical. Second, they are *hierarchical*, in that some choices govern whether other downstream ones are active at all and may restrict the options available to them. Third, they are also *constrained*, as not every combination of otherwise-valid choices yields a feasible architecture [4].

To model the architecture design space, we adopt Bussemaker's Architecture Design Space Graph (ADSG) [4], a graph formalism developed specifically for SAO and supported by an open-source Python implementation. It represents the design space as a directed graph with typed nodes and edges. Figure 1 shows an example alongside a resolved architecture instance. The concepts most relevant to this work are:

- **Function and component nodes.** Function nodes, denoted as `FUN`, specify what functions the system must perform; component nodes (`COMP`) represent the elements that fulfill those functions. Fulfillment is expressed by a derivation edge from the function to the component, and a selected component may in turn induce further functions, just like the `PROPULSION SYSTEM` component induces the `GENERATE THRUST` and `STORE FUEL` functions in Figure 1.
- **Selection choices.** A function may admit multiple realization options: a choice between different components, joint fulfillment by multiple components, or non-fulfillment. In such cases, a selection choice node is inserted between the function and the available alternatives. Resolving a selection choice keeps one branch and prunes the others with their derived subgraphs. This is shown as the resolution **R1** in Figure 1, where `COMP:JET ENGINE` is selected to fulfill `FUN:GENERATE THRUST`, and consequently `COMP:TURBOPROP` gets pruned. Pruning propagates only as far as the discarded branch is the sole justification for a node's presence: any node that is also derived from another part of the graph is retained. Note that selection choices may contain other downstream choices in their derived subgraphs. These also get pruned, which further reduces the size of the remaining selection space.

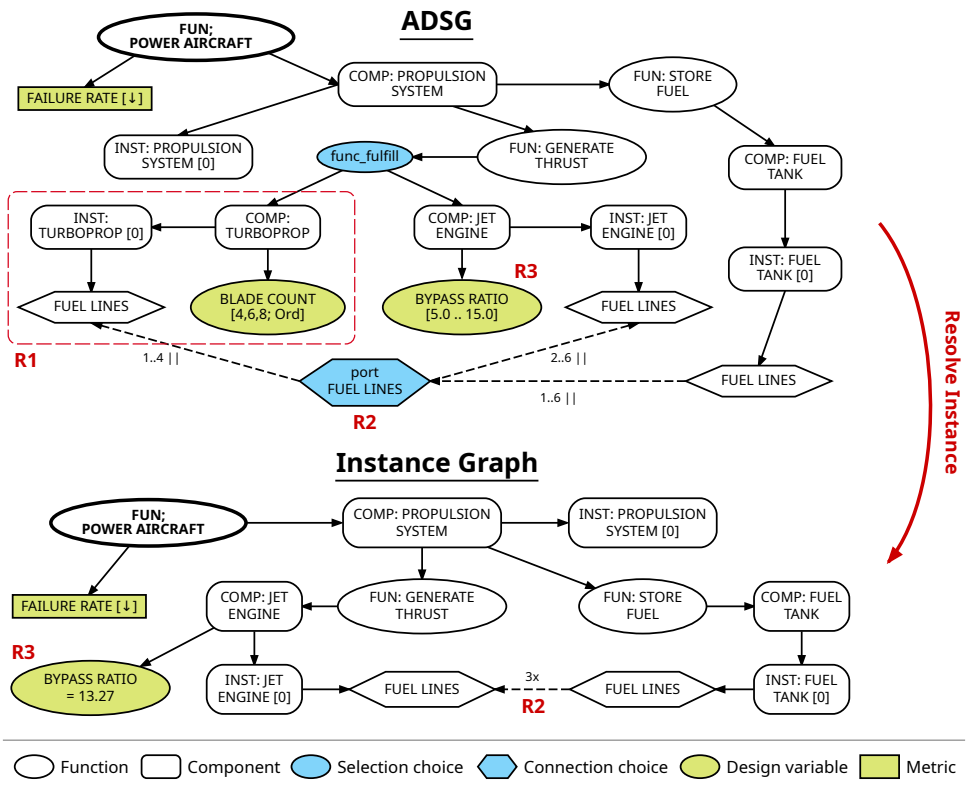


Fig. 1 Example ADSG (top) and the instance graph (bottom) obtained by resolving all of the choices. The ADSG models the propulsion system of an aircraft. Three resolutions are highlighted. **R1** resolves the selection choice `func_fulfill` by selecting `COMP: JET ENGINE` to fulfill `FUN: GENERATE THRUST`, pruning `COMP: TURBOPROP` together with its derived subgraph (dashed red box). **R2** resolves the connection choice between the remaining `FUEL LINES` ports of the jet engine and fuel tank, whose cardinalities are 2..6 and 1..6, respectively; here three connections are established. **R3** assigns a value to the continuous design variable `BYPASS RATIO`. The resulting instance graph describes one architecture candidate, from which the metric `FAILURE RATE` can be evaluated.

- **Connection choices.** Component instances can expose input and output ports; a connection choice node gathers the admissible edge sets between source and target ports, subject to per-port multiplicity constraints (e.g. each source supplying 0–$*$ connections, each target receiving 1 or 2). Resolving it deletes the choice node and instantiates one admissible set as explicit port-to-port edges. In Figure 1, this is denoted as the resolution **R2**, where a triple `FUEL LINES` connection is selected between `INST:JET ENGINE` and `INST:FUEL TANK`.
- **Design-variable (and attribute) nodes.** Using these nodes, one can optionally define architecture-specific sizing parameters for components, such as the continuous `BYPASS RATIO` or discrete `BLADE COUNT`. They are resolved by assigning a value from the specified range, as shown by the resolution **R3** in Figure 1.

- **Metric nodes.** To represent the evaluation outputs used to formulate the SAO problem, metric nodes are used. Since SAO aims to identify optimal or Pareto-optimal architectures while satisfying constraints, these nodes may represent minimization/maximization objectives (e.g. `FAILURE RATE`) or inequality constraints. Once an instance is evaluated, they are set to the obtained values.

Additional modeling mechanisms in the ADSG include incompatibility edges and choice constraints that restrict component combinations, as well as grouping nodes that remove permutation-equivalent selection and connection resolutions [4]. Finally, resolving all the selection and connection choices while respecting any additional constraints yields a single architecture instance, as seen at the bottom of Figure 1.

SAO then searches over such architecture instances, represented here by resolved instance graphs, to find designs that optimize the selected objective metrics while satisfying the imposed constraints. Importantly, these instance graphs can be heterogeneous. Depending on the choices taken they differ in node types, labels, and attributes (one instance may carry `COMP:JET ENGINE` with `BYPASS RATIO`, another has `COMP:TURBOPROP` with `BLADE COUNT`), they can vary in size, and may carry edge multiplicities. Together these properties make SAO a challenging problem.

2.2 Surrogate-based optimization

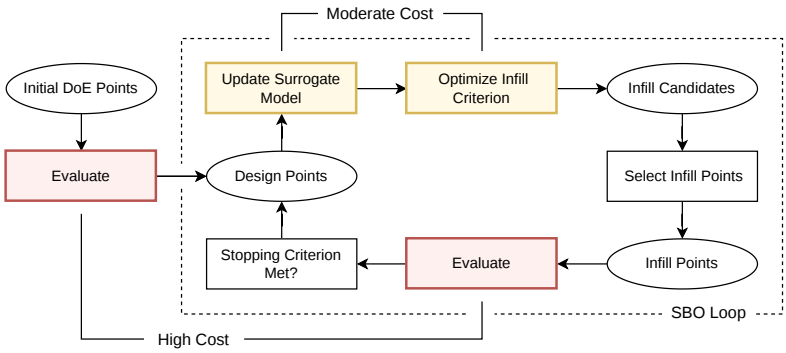


Fig. 2 A representative example of a Surrogate-Based Optimization loop. Modified figure from [15].

Evaluating specific architecture instances in terms of their objective and constraint values typically involves translating them into higher-fidelity models, which are then evaluated through expensive physics-based simulations, where gradients are generally unavailable. Gradient-free methods such as genetic algorithms can handle such black-box problems, but typically require thousands of evaluations to converge, which can become prohibitive under realistic evaluation costs [4]. Surrogate-based optimization (SBO) methods address this by relying on a cheaper approximation of the expensive black-box function to guide the search, so that the simulation budget is spent only on the most promising candidates. A representative SBO procedure is depicted in Figure 2. An initial set of evaluated points, also referred to as the Design of Experiments (DoE), is sampled and used to fit the surrogate model. The surrogate then

provides cheap estimates of objective values, which an infill criterion converts into a score indicating how promising each candidate is. An auxiliary optimizer, such as a genetic algorithm, is then used to search for a subset of candidates that maximize this cheaper score. The best ones are selected and evaluated using the expensive black-box function, added to the training data set, and the surrogate is updated. This process is repeated until a stopping criterion is reached [5, 16].

Bayesian optimization

Bayesian optimization is an uncertainty-aware variant of SBO. Instead of only predicting the objective value, the surrogate model also provides an uncertainty estimate at every point. This allows using acquisition functions that balance exploitation against exploration of high-uncertainty regions [5, 16]. BO is therefore particularly sample-efficient and suitable when function evaluations are expensive, since the most informative candidates for evaluation can be selected [5, 17].

Gaussian processes

Gaussian processes [18], also known as Kriging models, are commonly used as BO surrogates due to their sample efficiency and well-calibrated uncertainty estimates, and are also adopted in Bussemaker's foundational SAO framework [2, 4]. A GP defines a prior distribution over functions $f(\cdot) \sim \mathcal{GP}\big(m(\cdot), k(\cdot,\cdot)\big)$, characterized by the mean function m and a covariance kernel k. Conditioning this prior on observed evaluations yields a posterior predictive distribution that serves as the surrogate model.

To be more specific, let $X = \{x_i\}_{i=1}^n$ denote the evaluated designs, and let $y = [y_1, \ldots, y_n]^\top$ denote their standardized objective values, where $^\top$ denotes the transpose. Let $K \in \mathbb{R}^{n\times n}$ be the kernel matrix whose (i,j)th entry is $K_{ij} = k(x_i, x_j)$, and define

$$k_* = [k(x_1, x_*), \ldots, k(x_n, x_*)]^\top.$$

The noise-free GP posterior at a test point x_* is Gaussian with mean and variance

$$\mu(x_*) = k_*^\top K^{-1} y, \qquad \sigma^2(x_*) = k(x_*, x_*) - k_*^\top K^{-1} k_*. \tag{1}$$

As seen above, GP inference requires factorizing and solving linear systems involving the kernel matrix K, incurring $\mathcal{O}(n^3)$ training cost and $\mathcal{O}(n^2)$ memory [18], which can become limiting at larger sample sizes. However, in realistic SAO settings evaluation budgets are small precisely because each evaluation can be very expensive, making this modeling cost acceptable [4].

Kernels

The kernel is the main modeling component of a GP. It defines the assumed similarity between designs and determines how observations at evaluated designs inform predictions at test points [18]. A valid kernel can be expressed as an inner product in a reproducing kernel Hilbert space (RKHS) $\mathcal{H}$,

$$k\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}, \qquad k(x, x') = \langle \phi(x), \phi(x') \rangle_{\mathcal{H}},$$

where $\mathcal{X}$ is the design space, $\phi\colon \mathcal{X} \to \mathcal{H}$ is an implicit feature map to the RKHS, and $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ denotes the inner-product operation [18, 19]. Such a kernel is therefore positive semidefinite (PSD), which is the required condition for a valid GP covariance function. As per the generalized representer theorem [20], for a broad class of regularized learning problems in an RKHS, a function minimizing the training loss can be written as

$$\hat{f}(x) = \langle w, \phi(x) \rangle_{\mathcal{H}} = \sum_{i=1}^{n} \alpha_i k(x_i, x),$$

where $w \in \mathcal{H}$ parameterizes the linear predictor and $\alpha_i \in \mathbb{R}$ is the weight coefficient associated with training design x_i. The GP posterior mean in Equation 1 has exactly this form, since

$$k_*^\top K^{-1} y = k_*^\top \alpha = \sum_{i=1}^{n} \alpha_i k(x_i, x_*).$$

The kernel therefore defines both the feature space in which the objective is modeled and the model's inductive bias. When present, kernel parameters can be learned by maximizing the log marginal likelihood of the training data [18]. This kernel-centric view motivates kernels that capture architecture-level similarity directly.

2.3 Vector encodings and their limitations

While Bussemaker's foundational SAO framework represents architecture instances as graphs, it relies on vector-based optimization methods [2, 6]. To facilitate this, architecture instance graphs get mapped to fixed-length numerical *design variable vectors* [4, 21], as shown in Figure 3, while correction mechanisms facilitate mapping candidate vectors back to closest valid architecture graphs. In this section, the general workings of those encoding mechanisms are introduced, and their potential limitations are discussed, which motivates the remainder of this research.

Firstly, selection choice nodes get encoded into discrete design variables, whose value ranges correspond to the set of feasible choices. For example, the choice between a turboprop and a jet engine from Figure 1 would be encoded as follows:

$$\mathbf{x} = [\cdots,\ x_{\text{eng}},\ \cdots], \qquad x_{\text{eng}} \in \{0, 1\}, \qquad \begin{cases} x_{\text{eng}} = 0 \ \Rightarrow \ \textsc{Turboprop}, \\ x_{\text{eng}} = 1 \ \Rightarrow \ \textsc{Jet Engine}. \end{cases}$$

As discussed in Section 2.1, resolving selection choices may prune downstream nodes. The resulting hierarchy can be represented by an *activeness vector*, indicating which entries of the encoded design vector remain active. Existing SAO approaches use this information in hierarchy-aware GP kernels, such as the one of Saves et al. [6].

Whereas selection-choice encodings need to capture abstract categorical decisions, connection-choice encodings must represent connection topology in vector form. Since connection choices may involve many source and target connector instances, as well as parallel connections, they yield many feasible connection patterns. Encoding each pattern as an unrelated categorical option would preserve validity, but discard any meaningful notion of similarity, as any two distinct encodings would yield the same categorical distance.

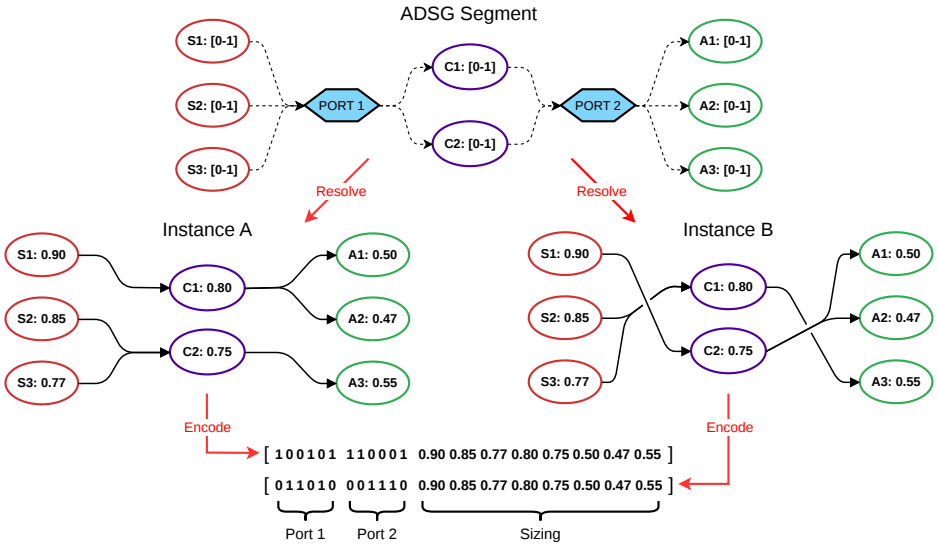


Fig. 3 Example illustrating the failure to capture topological similarity between instance graphs when relying on isolated vector encodings of connection choices. The encodings of the `PORT 1` and `PORT 2` choices use direct vectorization of their adjacency matrices, so an ideal D_{corr} is preserved.

Bussemaker therefore relies on pattern-based encoders to address this issue. These encoders are selected to maximize distance correlation denoted as

$$D_{\text{corr}} = \text{pearsonr}\big(\{d_{\text{manh}}(x_i, x_j)\}_{(i,j)\in\mathcal{P}}, \{d_{\text{manh}}(M_i, M_j)\}_{(i,j)\in\mathcal{P}}\big)\,,$$

where x_i is the encoded vector of a feasible pattern, M_i is its source–target connection matrix, $\mathcal{P}$ is a set of feasible patterns, d_{manh} is the Manhattan distance, and pearsonr is the Pearson correlation. A high correlation means that distances between encoded vectors reflect those between realized connection matrices [4]. However, that may require larger encodings, creating a trade-off in terms of encoding quality and size. Moreover, even $D_{\text{corr}} = 1$ within each connection choice does not guarantee meaningful distances between complete architectures composed of multiple choices.

In Figure 3, `PORT 1` and `PORT 2` are encoded using their respective flattened connection matrices, yielding $D_{\text{corr}} = 1$. The two resolved instances receive maximally different connection encodings, with $d_{\text{manh}}(x_A^{\text{conn}}, x_B^{\text{conn}}) = 12$ out of a maximum of 12, despite being structurally isomorphic under relabeling of C_1 and C_2. Similarity is preserved locally within each encoded connection choice, but not at the graph level.

This limitation plays a role when an objective is largely topology driven, and less dependent on the sizing parameters; redundancy- or failure-related metrics are intuitive examples. It also extends beyond isomorphism, as architectures may share relevant motifs despite differing strongly in individual connections. Such relationships must then be inferred from limited data. This can be restrictive for Gaussian processes, as their representation is fixed by the kernel-induced RKHS and remains relatively rigid even when kernel hyperparameters are learned [18, 22].

Directly comparing resolved architecture graphs offers a complementary representation that can capture neighborhoods, paths, and higher-order motifs spanning multiple connection choices. It also avoids encoder trade-offs between size, redundancy, and distance preservation. Moreover, inactive elements are simply absent from the graph rather than represented by activeness vectors and imputed entries.

3 Related Work

This section reviews graph-based optimization methods, focusing on those that share the SAO setting in which each candidate solution is represented as a graph, its evaluation is expensive, and the objective is to identify high-performing candidates. While graph-based learning has been widely applied to combinatorial optimization problems posed on a given graph, including routing, node selection, graph partitioning, and graph modification [23, 24], these methods typically construct a solution within, or optimize a process over, a given graph. Comparatively fewer methods address the setting considered here, in which each feasible candidate is a complete graph.

Within this setting, NAS provides the closest analogue to SAO. Candidate neural networks are represented as graphs, with nodes denoting neural operations, edges describing information flow, and evaluation requiring expensive training of each candidate beforehand. Bayesian optimization with graph-kernel GP and GNN surrogates has been widely adopted in NAS [7–11]. Similar methods have also been proposed for more general optimization over attributed graphs, network design, molecular design, and virtual screening [12–14, 25, 26].

3.1 Graph-kernel Gaussian process surrogates

A major line of work relies on Gaussian-process surrogates with graph kernels [7, 8, 12, 13, 27]. Such approaches retain the sample efficiency and principled uncertainty estimates of GPs (Section 2.2) while replacing vector-based kernels with graph-based ones. A graph kernel is a kernel function over an input space of graphs $\mathcal{G}$, whose value corresponds to an inner product in an associated RKHS [28]:

$$k_G \colon \mathcal{G} \times \mathcal{G} \to \mathbb{R}, \qquad k_G(G, G') = \langle \phi(G), \phi(G') \rangle_{\mathcal{H}}.$$

Graph kernels provide a notion of similarity between two graphs by comparing the structural patterns occurring within them, such as shared substructures, local neighborhoods, paths, or walks [28, 29]. They may operate on labeled graphs or additionally incorporate continuous attributes [28]. Attributed graph kernels could therefore include sizing parameters directly as node attributes. While SAO graphs contain node attributes, their heterogeneous nature complicates direct use of attributed graph kernels, as sizing parameters and attributes are only present for dedicated design-variable or attribute nodes, and their meaning and dimensionality can differ depending on the associated component type. A common attribute representation would therefore have to include all possible sizing variables for every node, resulting in high-dimensional vectors dominated by inactive or padded entries.

A complementary approach is to construct a hybrid kernel in which a label-based graph kernel captures structural similarity and a separate kernel handles

non-structural information. Graph-BO methods such as BoGrape and NAS-GOAT adopt this strategy with strong empirical results [8, 13]. A common multiple-kernel construction is

$$k_{\text{comp}}(G, G') = \sum_{r=1}^{R} \alpha_r k_r(G, G'), \qquad \alpha_r \geq 0, \tag{2}$$

where each component captures a different type of structural, attribute, or edge-level similarity [30]. This means that the sizing parameters could be handled through an imputed vector-based kernel, which avoids the aforementioned sparse attributed representation. Moreover, this allows selecting the structural component from the broad family of label-based graph kernels [28, 29].

Weisfeiler–Lehman (WL) kernels are particularly attractive for two reasons. First, they achieve state-of-the-art performance on related tasks such as graph classification and NAS [7, 28]. Second, many WL kernels can be computed in $\mathcal{O}(Hm)$, where H is a constant depth and m is the number of edges [31, 32]. The proven performance and computational efficiency make WL kernels a promising choice for SAO.

Weisfeiler–Lehman optimal assignment

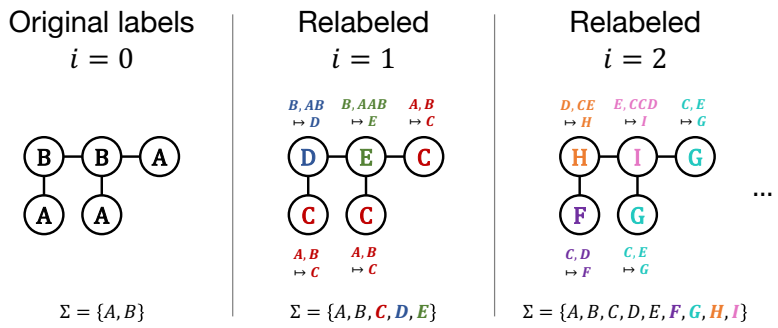


Fig. 4 Weisfeiler–Lehman relabeling over two iterations. At each iteration, each node's label is combined with the multiset of its neighbors' sorted labels and mapped to a new discrete label. For example, in Figure 4, all `A`-nodes with a single `B` neighbor at $i = 0$ receive the same label `C` at $i = 1$. The mapping is shared across graphs, so identical combinations receive the same label. Successive iterations produce labels that capture progressively larger rooted neighborhoods. Use of sorted multisets makes the procedure invariant to node ordering, meaning that all isomorphic structures receive the same label. Reproduced from Kriege et al. [29], Fig. 3, under the CC BY 4.0 licence.

Weisfeiler–Lehman kernels compare graphs by matching their nodes based on the labels first obtained through the iterative Weisfeiler–Lehman neighborhood relabeling. Figure 4 shows how this relabeling procedure works.

In SAO architecture graphs, labels obtained that way can capture shared component semantics and structural patterns at increasing neighborhood depths, as shown in Figure 5. At depth $d = 0$, matching labels reflect only shared node types and component semantics. After one WL iteration, a match requires the corresponding one-hop neighborhoods to agree, so the refined labels begin to encode local connectivity rather than node identity alone. At $d = 2$, the labels represent larger neighborhoods spanning

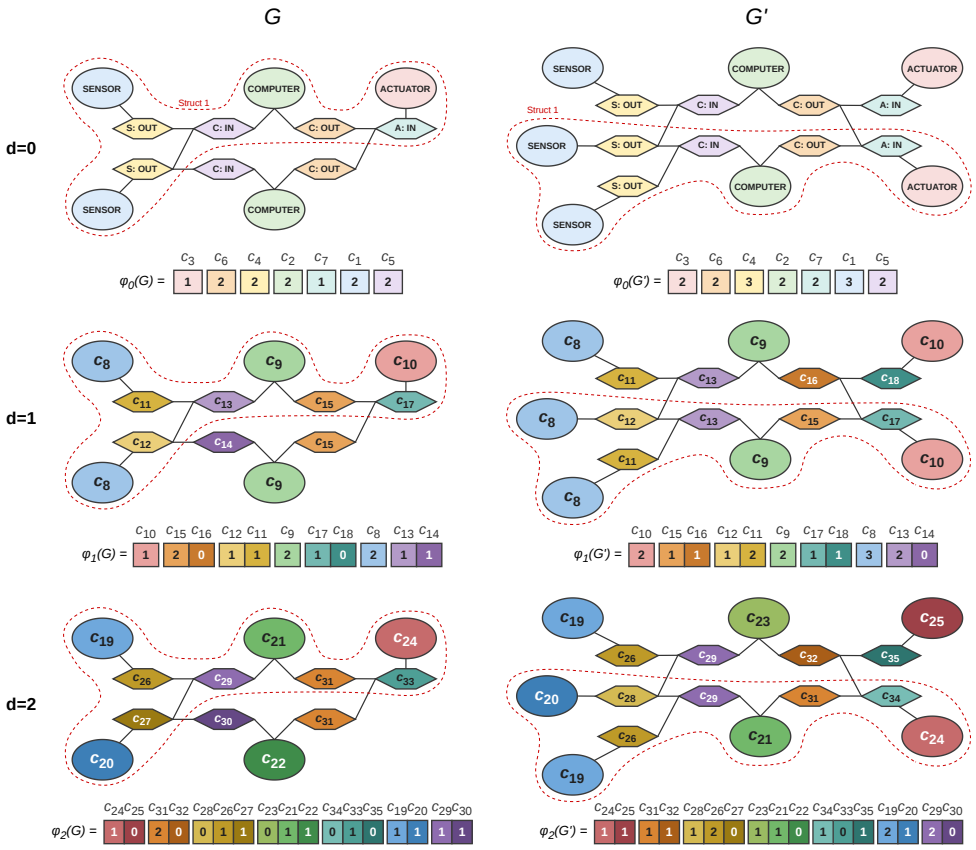


Fig. 5 Illustration of Weisfeiler–Lehman relabeling for two simplified architecture instance graph segments G and G'. Initial labels encode ADSG node types and component semantics. Each WL iteration incorporates one additional neighborhood hop, while the feature vectors ϕ_h count the labels occurring at each depth. The highlighted regions illustrate a common architectural structure spanning two resolved connection choices.

several nodes and connections. In the highlighted regions, both graphs contain the same `SENSOR–COMPUTER–ACTUATOR` pattern spanning two resolved connection choices. The shared labels c_{29}, c_{21}, and c_{31} arise from this common structure, showing how WL can identify the same architectural motif even when the graphs differ elsewhere.

The Weisfeiler–Lehman optimal assignment (WLOA) kernel is currently the state-of-the-art [28, 32]. The idea behind it is to compare two nodes u and v by the number of WL depths at which their labels agree, up to a chosen cutoff depth H:

$$k_{\mathrm{WLOA}}(u, v) = \sum_{h=0}^{H} \delta(\tau_h(u), \tau_h(v)) \,, \tag{3}$$

where $\tau_h(u)$ is the label of node u at depth h, and δ is the Kronecker delta. The graph-level kernel finds the assignment maximizing the total node similarity between

two graphs G and G',

$$K_{\mathrm{WLOA}}(G, G') = \max_{B \in \mathcal{B}(V,V')} \sum_{(u,v)\in B} k_{\mathrm{WLOA}}(u, v),$$

where $\mathcal{B}(V, V')$ denotes the set of all valid injective assignments between their respective node sets V and V'. Since a match at depth h implies matches at all preceding depths, this can be evaluated directly as the histogram intersection [32]

$$K_{\mathrm{WLOA}}(G, G') = \sum_{h=0}^{H} \sum_{a \in \mathcal{L}_h} \min\left(c_h^G(a), c_h^{G'}(a)\right),$$

where $\mathcal{L}_h$ denotes the set of WL labels observed at depth h, and $c_h^G(a)$ counts the nodes in G with label a at depth h. This can be efficiently evaluated in $\mathcal{O}(Hm)$, where m is the number of edges processed during relabeling [32].

When structural selections and connections are encoded as separate design variables, an encoding-based kernel can learn relevance parameters for each choice. WLOA is less flexible in this respect, since all matching WL patterns up to depth H contribute uniformly. However, this trade-off may be beneficial when performance depends on structural patterns spanning many connection choices, akin to the graphs in Figure 3. Moreover, deep WL kernels could recover the lost flexibility by learning separate weights for individual WL labels [33]. This, however, can introduce hundreds or thousands of parameters, making it challenging in the low-sample SAO setting.

3.2 Graph neural network surrogates

A second line of work uses graph neural networks (GNNs) as surrogate models [9–11, 14]. GNNs learn a representation for each node by repeatedly aggregating its current representation with those of its neighbors and applying learned transformations, such as neural networks. The resulting node representations can be pooled to obtain a graph-level embedding, for example by summation. DGBO and DeepGBO-NAS fit a Bayesian linear regressor on these graph embeddings to obtain predictions and uncertainty estimates required for Bayesian optimization [11, 14, 34]. Methods such as Arch2Vec decouple representation learning from surrogate fitting by pretraining graph embeddings on unlabeled architectures and fitting a smaller supervised surrogate in the resulting latent space, potentially improving data efficiency [10].

GNN- and graph-kernel-based surrogates offer different trade-offs. First, prediction with a GNN requires only a forward pass, whereas graph-kernel GPs require kernel evaluations against the training graphs. This can make GNNs faster during infill search, although the difference matters less when architecture evaluation dominates the optimization cost. Second, GNNs can learn their structural representation end-to-end, whereas graph-kernel GPs adapt only a limited number of hyperparameters within a fixed feature construction. This gives GNNs greater representational flexibility, but also introduces many trainable parameters that can be difficult to estimate reliably from the limited samples typical of SAO. Graph kernels are therefore less flexible, but potentially better suited to this data regime. Moreover, when uncertainty is

modeled only in the output layer, it may become unreliable for graphs whose learned embeddings are poorly supported by the training data. Graph kernels can also offer greater interpretability, since their similarity measures are based on explicitly defined graph features or structural comparisons. When parameters control the contributions of specific substructures or feature groups, their values have a direct interpretation, whereas attributing the representations learned by GNNs to specific graph properties is generally more difficult.

3.3 Implications for SAO

This research follows the graph-kernel line because it retains the sample efficiency of GPs, which suits the low-evaluation regime of SAO, and has proven effective in the closely related NAS setting [7, 8]. The surrogate also remains a Gaussian process, as in Bussemaker's BO approach for SAO [2, 4], allowing performance differences to be attributed primarily to the representation rather than the surrogate class.

However, SAO is a broader setting than those considered by prior graph-BO methods. It spans diverse system domains and commonly involves multiple objectives and constraints, requiring greater surrogate flexibility than narrower settings such as NAS. ADSGs further combine heterogeneous node semantics, optional sizing variables, and edge multiplicities, increasing the complexity of the relationships that a surrogate must learn from limited data. The graph instances are also substantially larger than in the closest prior setting of NAS. The test problems considered here contain 32–82 nodes, compared with at most seven and four in NAS-Bench-101 and NAS-Bench-201 [35, 36]. Since graph-kernel evaluations are repeated during GP fitting and infill search, this makes computational scalability more important. General graph-BO methods consider larger graphs [13, 26], but not this combination of characteristics. Applying graph-based BO to SAO therefore requires flexible, computationally practical kernel formulations and validation across this broader setting.

4 Method

This section presents the proposed graph-kernel Gaussian-process surrogate and its integration into Bussemaker's SAO framework. We first outline the Bayesian optimization backbone, followed by the composite architecture kernel and its structural, sizing, and edge components. Finally we explain how various parameters are learned. The implementation of the method is available in a public repository.[1]

4.1 Optimization backbone

To facilitate direct benchmarking against existing SAO methods, this work uses a modified variant of the Bayesian optimization SAO backbone of Bussemaker et al. [2], as implemented in SBArchOpt[2] [37].

The procedure is shown in Figure 6. An initial design of experiments is generated using hierarchical sampling of [2] and evaluated with the expensive architecture evaluator. The resulting data are then used to fit the graph-based GP surrogate, which

[1] https://github.com/flatala/sao_graph_sbo
[2] https://github.com/jbussemaker/SBArchOpt

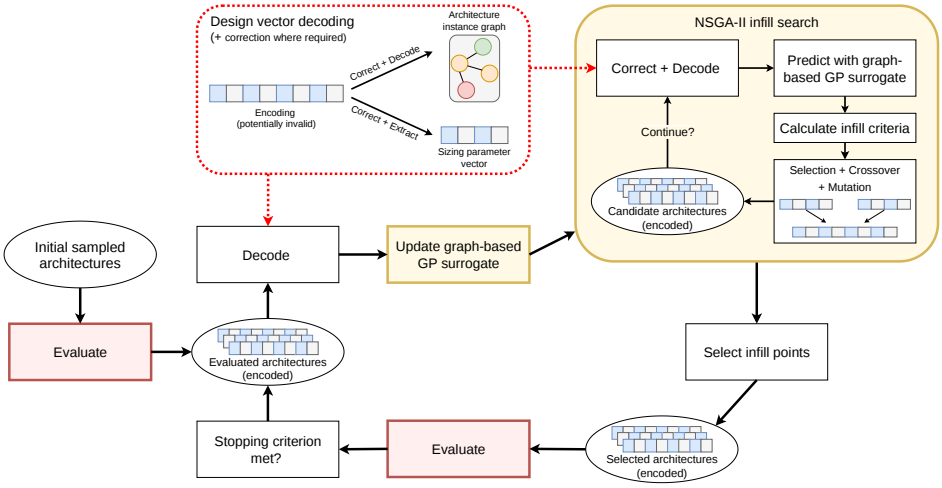


Fig. 6 Overview of the optimization workflow. Evaluated encoded architectures are decoded to instance graphs to update the graph-based GP surrogate. NSGA-II then searches for new infill candidates by repeatedly correcting and decoding candidate vectors, evaluating them with the graph-based surrogate, and applying selection, crossover, and mutation among the strongest candidates. Selected architectures are evaluated with the expensive objective function and the process repeats.

first requires decoding the design vectors into architecture instance graphs. Promising infill points are then found with the NSGA-II [38] optimization algorithm. During this search, each candidate population is first corrected and decoded into architecture graphs to be evaluated by the graph-based GP surrogate. Its predictions are used to compute the infill-criterion values, which determine which candidates are retained and evolved through crossover and mutation. Once NSGA-II terminates, the final set of selected candidates is evaluated with the expensive evaluator, added to the data set, and the surrogate is refitted. This process repeats until a maximum number of evaluations is reached or another stopping criterion is met.

The decoding step uses the *Generate Architecture* procedure of [4], which effectively inverts the encoding procedure described in Section 2.3. During this step, any invalid selection- and connection-choice encodings produced by the evolutionary operators are corrected to the closest valid ones, before the architecture graph is reconstructed. The infill criteria, batch-selection procedure, and hidden-constraint handling are taken directly from [2] and [39].

4.2 Composite-kernel Gaussian process

We replace the standard vector-based GP with one whose kernel incorporates graph-based structural similarity between resolved architecture instances. Each architecture instance is represented by an instance graph G_x resolved from the problem's ADSG. These instance graphs capture the functions, components, connections, sizing parameters and attributes that make up the corresponding candidate architecture. To model structural similarity, we adopt label-only WL kernels that do not incorporate continuous sizing parameters or edge multiplicities. We therefore model structure, sizing, and

edge multiplicity through separate kernel components,

$$k_{\mathrm{g},\vartheta_{\mathrm{g}}}(G_x, G_{x'}), \qquad k_{\mathrm{s},\vartheta_{\mathrm{s}}}(G_x, G_{x'}), \qquad k_{\mathrm{e},\vartheta_{\mathrm{e}}}(G_x, G_{x'}),$$

where k_{g} models structural similarity, k_{s} sizing similarity, and k_{e} edge-multiplicity similarity. The parameters ϑ_{g}, ϑ_{s}, and ϑ_{e} denote the learnable parameters of the corresponding kernels. The edge component is included only for problems permitting parallel connections. This decomposition is similar to the approach in BoGrape [13].

Let $\mathcal{Q} \subseteq \{\mathrm{g}, \mathrm{s}, \mathrm{e}\}$ denote the kernel components available for a given problem and $\vartheta = \{\vartheta_q\}_{q\in\mathcal{Q}}$ their parameters. We combine their normalized forms using a weighted second-order ANOVA-style kernel [40]:

$$\begin{aligned} k_{\mathrm{arch},\vartheta,\alpha}(G_x, G_{x'}) = &\sum_{q\in\mathcal{Q}} \alpha_q \, \bar{k}_{q,\vartheta_q}(G_x, G_{x'}) \\ &+ \sum_{\substack{q,r\in\mathcal{Q} \\ q<r}} \alpha_{qr} \, \bar{k}_{q,\vartheta_q}(G_x, G_{x'}) \bar{k}_{r,\vartheta_r}(G_x, G_{x'}). \end{aligned} \tag{4}$$

Here, $\bar{k}_{q,\vartheta_q}$ denotes the normalized form of component kernel k_{q,ϑ_q}, whereas α_q and α_{qr} denote the weights of the corresponding first- and second-order branches. These weights are learned jointly with the kernel parameters during GP fitting, constrained to be nonnegative, and normalized across all active branches such that

$$\sum_{q\in\mathcal{Q}} \alpha_q + \sum_{\substack{q,r\in\mathcal{Q} \\ q<r}} \alpha_{qr} = 1.$$

The first-order branches allow each information source to contribute independently, whereas the second-order branches capture interactions between information sources. For example, the structure–sizing branch

$$k_{\mathrm{gs},\vartheta}(G_x, G_{x'}) = \bar{k}_{\mathrm{g},\vartheta_{\mathrm{g}}}(G_x, G_{x'}) \bar{k}_{\mathrm{s},\vartheta_{\mathrm{s}}}(G_x, G_{x'})$$

is large only when two architectures are similar in both structure and sizing. Thus, unlike either a purely additive or purely multiplicative composition, the composition from Equation 4 can represent both independent and joint similarity effects.

Each component kernel is normalized by its self-similarities:

$$\bar{k}_{q,\vartheta_q}(G_x, G_{x'}) = \frac{k_{q,\vartheta_q}(G_x, G_{x'})}{\sqrt{k_{q,\vartheta_q}(G_x, G_x) k_{q,\vartheta_q}(G_{x'}, G_{x'})}}. \tag{5}$$

This gives $\bar{k}_{q,\vartheta_q}(G_x, G_x) = 1$ and places all component kernels and their products on a common scale. Therefore, the composition weights reflect relative contributions to the covariance rather than differences in raw kernel magnitude. Since every branch has unit self-similarity and the branch weights sum to one, the composite kernel also satisfies $k_{\mathrm{arch},\vartheta,\alpha}(G_x, G_x) = 1$. Moreover, normalization from Equation 5 preserves

positive semidefiniteness, while products and nonnegative weighted sums of kernels remain positive semidefinite [30]. Therefore, Equation 4 is a valid covariance kernel.

Given a set of n evaluated architecture graphs $\{G_{x_i}\}_{i=1}^{n}$, and a test graph G_*, the covariance matrix $K_{\vartheta,\alpha}$ and cross-covariance vector k_* are defined as

$$[K_{\vartheta,\alpha}]_{ij} = k_{\text{arch},\vartheta,\alpha}(G_{x_i}, G_{x_j}), \qquad [k_*]_i = k_{\text{arch},\vartheta,\alpha}(G_{x_i}, G_*).$$

These quantities take the place of K and $\mathbf{k}_*$ in Equation 1, yielding the predictive mean and variance for G_*.

4.3 Proposed Weisfeiler–Lehman structural kernels

As discussed in Section 3.1, Weisfeiler–Lehman kernels provide a graph-based alternative to encoding-based kernels by comparing labeled neighborhood patterns directly between architecture graphs. The Weisfeiler–Lehman optimal assignment kernel achieves state-of-the-art performance while remaining computationally efficient [28, 32]. However, WLOA's standard formulation is relatively rigid, as it requires fixing key hyperparameters beforehand. We discuss the resulting limitations below, as well as the proposed extensions that aim to address them. The extended variant is used as the structural kernel $k_{\text{g},\vartheta_\text{g}}$.

Learned-depth WLOA

First, standard WLOA fixes a maximum relabeling depth H and weights all included depths equally. Shallow depths capture component counts and local connections, whereas deeper depths capture larger substructures. The relevant structural scales can vary with the objective: system mass may depend primarily on component counts and types, whereas failure rate may depend more strongly on connectivity patterns. Moreover, the appropriate depth can be hard to establish a priori, and separate tuning is often infeasible as new SAO problems often lack evaluated samples.

To address this limitation while avoiding the large parameter count of deep WL kernels [33], we propose the learned-depth WLOA (LD-WLOA) kernel, which can learn the contribution of each depth independently:

$$k_{\text{LD}}(u, v) = \sum_{h=0}^{H} w_h \delta\left(\tau_h(u), \tau_h(v)\right), \qquad w_h \geq 0, \tag{6}$$

where the weights w_h are optimized as GP hyperparameters. This allows matches at uninformative structural scales to be downweighted and those at relevant scales to be emphasized. Standard WLOA with any fixed depth $d \leq H$ is recovered by setting $w_h = 1$ for $h \leq d$ and $w_h = 0$ otherwise. Thus, increasing H enlarges the model family without forcing deeper levels to contribute, while also permitting nonuniform combinations of structural scales. This formulation also preserves the efficient histogram-intersection computation:

$$K_{\text{LD-WLOA}}(G, G') = \sum_{h=0}^{H} w_h \sum_{a \in \mathcal{L}_h} \min\left(c_h^G(a), c_h^{G'}(a)\right).$$

For nonnegative depth weights, LD-WLOA remains positive semidefinite. A more detailed explanation and a proof can be found in Section A.

Multi-granularity LD-WLOA

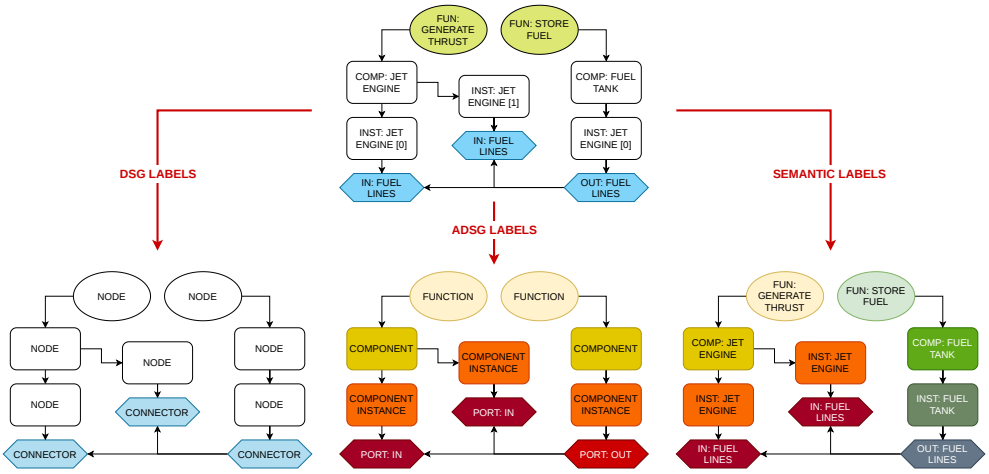


Fig. 7 Initial node-label granularities for the same resolved architecture instance graph segment, ordered from coarse to fine. DSG labels distinguish generic graph roles, ADSG labels distinguish architecting roles, and semantic labels additionally distinguish specific function and component types. Equal colors indicate equal initial labels and thus nodes that may match at WL depth 0.

Second, standard WLOA requires choosing how graph nodes are labeled before carrying out WL relabeling. At one extreme, every node could be given the same label; at the other, nodes could be labeled according to their specific architectural semantics, such as `COMP:FUEL TANK` or `FUN:GENERATE THRUST`. We refer to this level of specificity as the label *granularity*. Because WL relabeling preserves initial label distinctions, the chosen granularity determines which nodes, and therefore which substructures, match. A coarser granularity allows structurally analogous subtrees to match across node types that a finer one keeps apart. For example, similar connectivity patterns in IoT mesh architectures could imply largely similar throughput despite differing router and gateway types, whereas cost could depend strongly on the specific device types used. The appropriate granularity is therefore problem- and objective-dependent and generally unknown a priori.

For SAO architecture graphs, we therefore consider three label types of varying granularity, which are illustrated in Figure 7 and listed below:

- *DSG labels*, which arise from the generic graph formalism underlying the ADSG [4], retain only basic node classes: ordinary, connector, and design-variable nodes.
- *ADSG labels* additionally distinguish the node types described in Section 2.1, including function, component, instance, grouping, port, and connector nodes.

- *Semantic labels* further distinguish elements by their architectural semantics, such as different functions (e.g. `FUN:GENERATE THRUST` and `FUN:STORE FUEL`) or components (e.g. `COMP:JET ENGINE` and `COMP:FUEL TANK`).

Rather than selecting a single granularity a priori, we evaluate several in parallel and combine the resulting normalized LD-WLOA kernels. This allows the surrogate to learn how strongly structural similarity should depend on specific component types for a given objective, or whether broader patterns, such as redundancy across different component types, are sufficient. The resulting multi-granularity variant (MG-LD-WLOA) is defined as:

$$K_{\text{MG-LD-WLOA}}(G, G') = \sum_{r \in \mathcal{R}} \beta_r \, \bar{K}_{r,\mathbf{w}}(G, G'), \qquad \beta_r \geq 0, \qquad \sum_{r \in \mathcal{R}} \beta_r = 1, \tag{7}$$

where $\mathcal{R}$ is the set of selected granularities and $\bar{K}_{r,\mathbf{w}}$ is the normalized LD-WLOA kernel under granularity r. The mixture weights β_r are learned during GP fitting, while the depth weights $\mathbf{w}$ are shared across granularities to limit the parameter count. Every graph pair is compared under all selected granularities, and the fitted β_r determine the contribution of each comparison. As the relabelings are performed jointly within the same graph traversal, each additional granularity incurs little additional overhead.

4.4 Design parameter kernel

Since the structural WLOA kernels operate on discrete node labels, and not node attributes, we model the similarity between the attributes and sizing parameters of two architectures through a separate kernel component $k_{\text{s},\vartheta_\text{s}}$. In contrast to the structural kernel, which operates on the graphs directly, this component acts on the sizing part of the corrected design vector, which is reflected in Figure 6. We write x_s for the sizing subvector of the architecture represented by G_x. Continuous and ordinal variables are normalized, while categorical variables retain their encoded levels. For each sizing parameter j, the distance between two compared architectures is defined as

$$d_j(x_\text{s}, x'_\text{s}) = \begin{cases} |x_{\text{s},j} - x'_{\text{s},j}|, & j \text{ continuous or ordinal,} \\ \mathbb{I}\left[x_{\text{s},j} \neq x'_{\text{s},j}\right], & j \text{ categorical.} \end{cases}$$

The sizing kernel then follows from the above distance as

$$k_{\text{s},\vartheta_\text{s}}(G_x, G_{x'}) = \exp\left(- \sum_{j=1}^{D_\text{s}} \theta_j \, d_j(x_\text{s}, x'_\text{s})^p \right), \quad \theta_j > 0, \tag{8}$$

where D_s is the number of sizing parameters and $p = 1.9$ is fixed in the final configuration, following the default from [41]. The dimension-specific parameters θ_j allow the surrogate to learn which sizing variables are informative for the corresponding output. This construction is inspired by [6].

Any sizing parameters that are inactive in the respective architecture instance, because their branch was pruned, are assigned a canonical value [2]: inactive continuous variables receive the midpoint of their bounds before normalization, while inactive ordinal and categorical variables receive a predefined constant encoded value, typically zero. Similarity is then computed directly on the resulting fixed-length vectors.

4.5 Edge-weight kernel

Connection choices in ADSGs allow for parallel connection edges between ports. However, none of the preceding kernel components captures edge multiplicity, as WL kernels treat adjacency as binary. For problems in which such parallel edges are present, we use a separate edge-weight (EW) kernel component $k_{\mathrm{e},\vartheta_\mathrm{e}}$.

Each port node u arising from a resolved connection choice is assigned a semantic context $\lambda(u)$, consisting of its port type and the type of component to which it belongs. Looking at the instance graph in Figure 1, one such port context would be `JET ENGINE: FUEL LINES`, whereas another would be `FUEL TANK: FUEL LINES`. Let $\mathcal{P}_c(G)$ denote the set of distinct connected port pairs whose source and target ports have context $c = (\ell_s, \ell_t)$. We then define a per-context multiplicity feature as

$$f_c(G) = \sum_{(u,v)\in\mathcal{P}_c(G)} \log(1 + m_{uv}),$$

where m_{uv} is the number of parallel connections between ports u and v. Thus, $f_c(G)$ provides a measure of how strongly ports of one semantic component context are connected to ports of another. The logarithmic transformation compresses the range of multiplicity values, yielding a more balanced comparison across endpoint contexts.

The resulting feature vectors are compared using a Laplacian kernel,

$$k_{\mathrm{e},\gamma}(G, G') = \exp\left(-\gamma \sum_{c\in\mathcal{C}} |f_c(G) - f_c(G')|\right), \qquad \gamma > 0,$$

where $\mathcal{C}$ is the set of port contexts. This deliberately coarse kernel captures the sparse multiplicity signal using only one learned decay coefficient, making it suitable for the low-data setting considered here.

4.6 Kernel parameter learning

The hybrid ANOVA formulation and its kernel components contain learnable parameters, including the weights assigned to composition branches, WL depths, and label granularities, as well as the sizing- and edge-kernel coefficients. We learn these jointly by maximizing the log marginal likelihood of the observed objective values.

For n evaluated architecture graphs and their standardized objective values y, the zero-mean GP log marginal likelihood is

$$\mathcal{L}(\vartheta, \alpha) = \log p\left(\mathbf{y} \mid \{G_i\}_{i=1}^n, \vartheta, \alpha\right) = -\frac{1}{2}\mathbf{y}^\top \widetilde{K}_{\vartheta,\alpha}^{-1}\mathbf{y} - \frac{1}{2}\log\det \widetilde{K}_{\vartheta,\alpha} - \frac{n}{2}\log(2\pi), \quad (9)$$

where ϑ collects the component-kernel parameters and α the composition weights. Here, $\widetilde{K}_{\vartheta,\alpha} = K_{\vartheta,\alpha}+\eta I$, where $K_{\vartheta,\alpha}$ is the covariance matrix constructed by evaluating the composite architecture kernel between all pairs of evaluated graphs, and $\eta > 0$ is a fixed numerical nugget added for stability.

We maximize this criterion using COBYLA, a derivative-free constrained optimizer [42]. The sizing-kernel coefficients θ_j and edge-kernel coefficient γ are optimized in base-10 logarithmic space, since suitable similarity-decay rates can span several orders of magnitude. The WL depth weights and raw granularity and composition weights are optimized in linear space. In preliminary experiments, logarithmic parameterization produced sharp changes between successive COBYLA proposals that hindered convergence, whereas linear parameterization enabled more gradual adjustments and substantially improved optimization behavior.

5 Experimental setup

This section first introduces the test problems and their evaluation budgets, followed by the competing surrogate and optimization baselines. Finally, we define the evaluation metrics and the repeated-run protocol.

5.1 Test problems

We evaluate the proposed methods on four SAO benchmarks that differ in how strongly their objectives depend on graph structure. The mixed-discrete Guidance, Navigation and Control (MDGNC) problems contain explicit topology-dependent behavior, with the edge-failure variant (MDGNC-EF) additionally involving parallel connections, whereas the ENGINE and ROCKET problems are dominated by hierarchical choices. Their main characteristics and optimization budgets are summarized in Table 1.

Table 1 Test-problem characteristics and optimization budgets used in the final comparison. The total budget includes the initial design of experiments.

Problem	Objectives	Constraints	DoE size	Batch size	Total budget
ENGINE	1	5	75	10	305
MDGNC	2	0	30	10	200
MDGNC-EF	2	0	30	10	250
ROCKET	2	3	100	10	750

The initial set of evaluated architectures used to fit the surrogate is referred to as the design of experiments (DoE). Its size is guided by the heuristic of [4], $n_{\text{doe}} = k_{\text{doe}} n_x/(1 - \text{FR}_{\text{exp}})$, where k_{doe} is the DoE multiplier, n_x is the dimensionality of the full design-vector encoding, and FR_{exp} is the expected failure rate. The DoE is generated in the design-vector space and shared across methods, ensuring identical initialization; the total evaluation budget is also identical across methods for each benchmark. For ENGINE, its hidden constraint motivates the failure-rate correction from [4], with $k_{\text{doe}} = 2$ and $\text{FR}_{\text{exp}} = 60\%$ giving $n_{\text{doe}} = 75$. MDGNC and MDGNC-EF

both use $n_{\text{doe}} = 30$ because they share the same underlying architectural decisions, while MDGNC-EF receives a larger total budget to account for its additional parallel connections and edge multiplicities. ROCKET uses a larger DoE and the largest total budget because of slower convergence across all methods.

Mixed-discrete GNC

The mixed-discrete Guidance, Navigation and Control (MDGNC) problem is adopted from the GNC benchmark of Bussemaker et al. [2]. It represents a control system where sensors collect data, computers process it, and actuators execute the resulting commands. For each component type, a selection choice determines whether one, two, or three instances are present. Each instance v has a continuous sizing parameter $p_v \in [0, 1]$, which determines its mass $m_v(p_v)$ and failure probability $q_v(p_v)$ through fixed nonlinear functions, with the general trend being that heavier components are more reliable. Component instances are connected by two consecutive many-to-many connection choices, akin to the ADSG segment in Figure 3. The objective of the problem is to minimize the system's mass and failure rate, which are in conflict with each other, yielding a multi-objective optimization problem.

Total mass is obtained by summing the masses $m_v(p_v)$ of all instantiated components and adding a unit mass penalty for every connection edge. System failure probability is computed by enumerating component-survival scenarios layer by layer: a component is functional only if it itself does not fail and at least one upstream component connecting to it remains functional. For any component set U, components fail independently, so a subset $T \subseteq U$ survives with probability

$$\pi(T; U) \;=\; \prod_{v \in T} (1 - q_v) \prod_{v \in U \setminus T} q_v.$$

Conditioning on a surviving sensor set $T_S \subseteq S$, only the computers $C(T_S)$ where each is connected to at least one sensor in T_S remain relevant; conditioning next on surviving computers $T_C \subseteq C(T_S)$ determines the reachable actuator set $A(T_C)$. The system then fails if and only if all reachable actuators fail, so summing over all scenarios yields the following objective:

$$P_{\text{fail}} \;=\; \sum_{T_S \subseteq S} \pi(T_S; S) \sum_{T_C \subseteq C(T_S)} \pi(T_C; C(T_S)) \prod_{a \in A(T_C)} q_a.$$

Two architectures containing the same components and number of connections can therefore have different failure probabilities if those connections are arranged differently, making reliability dependent on the global sensor–computer–actuator topology. This is precisely the type of setting in which encoding-based surrogates may be limited, as discussed in Section 2.3.

Edge-failure MDGNC

We extend MDGNC by assigning each connection edge e an independent failure probability q_e and by allowing parallel connections, which provide redundancy. Component and edge survival are propagated jointly: a downstream component remains reachable

only through at least one operational connection from a functional upstream component. The system again fails when no operational sensor–computer–actuator path remains. The complete failure-probability formulation is given in Section B.

Since additional edges increase mass, the extension creates a trade-off between redundancy and weight. This also makes connection topology more influential and introduces edge multiplicity as an explicit performance factor, which facilitates evaluating the edge kernel proposed in Section 4.5.

Jet-engine architecture

The jet-engine problem is adopted from the conceptual aircraft-engine benchmark of Bussemaker et al. [43]. It considers turbojet and turbofan architectures, with discrete choices such as fan, gearbox, and mixed-nozzle inclusion, together with continuous parameters such as bypass ratio and fan and overall pressure ratios.

The design space is strongly hierarchical: fan-related variables are active only for turbofan architectures, gearbox ratio only when a gearbox is selected, and shaft-related variables depend on the selected number of shafts. The objective of the problem is to minimize thrust-specific fuel consumption, while fulfilling five inequality constraints pertaining to exhaust-jet speeds and compressor pressure ratios. Evaluations may also fail when no valid thermodynamic solution is found, introducing a hidden constraint. For efficient repeated experiments, the original physics-based evaluation is approximated using random-forest regressors fitted to its outputs, following [2].

Unlike MDGNC, the problem is primarily driven by hierarchical component and sizing choices. Its limited connection choices concern bleed-air and power offtakes and do not form consecutive many-to-many patterns. It therefore evaluates the graph-based surrogate on a realistic hierarchical problem in which explicit connection topology is less influential.

Multi-stage rocket

As the final benchmark we use the multi-stage rocket problem from García Sánchez [44]. It considers rocket architectures with one to three stages, with hierarchical choices governing stage presence, engine type and count, stage dimensions, and nose geometry. The problem minimizes launch cost while maximizing payload mass, subject to constraints on dynamic pressure, payload volume, and achievable velocity increment. It is strongly constrained: only a few percent of the architectures sampled in an initial DoE are feasible, so the surrogate is fitted on a training set in which feasible designs are heavily outnumbered, and constraint prediction rather than objective prediction limits optimization progress. Its ADSG contains no connection choices, so graph-based methods are not expected to benefit from explicit topology modeling, but it nevertheless provides a useful benchmark for evaluating them in a purely hierarchical setting.

5.2 Baseline methods

We compare the proposed WLOA-based kernels against four families of baselines.

- **An alternative graph kernel** that operates on the same architecture instance graphs as our method: the exponential shortest-path kernel obtained through the successive-embeddings construction of Nikolentzos and Vazirgiannis [45]. Let a map

$\phi_{\mathrm{SP}}(G)$ contain the counts of directed shortest paths in the graph G, aggregated by source label, target label, and length. The shortest path (SP) kernel is then:

$$k_{\mathrm{SP}}(G, G') = \langle \phi_{\mathrm{SP}}(G), \phi_{\mathrm{SP}}(G') \rangle.$$

Applying an RBF kernel to the induced feature-space distance yields

$$k_{\text{SP-RBF}}(G, G') = \exp\left(-\frac{k_{\mathrm{SP}}(G, G) + k_{\mathrm{SP}}(G', G') - 2k_{\mathrm{SP}}(G, G')}{2\ell_{\mathrm{SP}}^2} \right),$$

with learned length scale ℓ_{SP}. We include it because its inductive bias is aligned with the structure of hierarchical design spaces. A directed path from the root to a function node records where in the decision hierarchy that function is introduced, and a path from a function node to a component node records which component type was selected to fulfill it, and through which intermediate decisions. Two architectures that share many such root–function and function–component paths should therefore share large parts of their decision hierarchy. Shortest-path computation is cubic in the worst case, but our instance graphs are sparse (average node degree ≈ 2), keeping it fast in practice; comparable sparsity is likely in other SAO problems with structured functional and component relationships. We evaluate SP-RBF using semantic labels only.

- **Encoding-based surrogates**, operating on the encoded design vectors of Section 2.3 rather than on architecture graphs.
  - *HIER*: the hierarchy-aware Kriging (GP) model of Bussemaker et al. [2], whose kernel accounts for variable activeness and penalizes variables active in one design but inactive in the other [6].
  - *KPLS*: partial least squares projects the encodings onto ten response-informed latent components, in which a Kriging model is then fitted [46]. Also considered in Bussemaker et al. [2].
  - *BLR*: Bayesian linear regression on the design encoding, as a simple probabilistic baseline.
- **GNN-based surrogates**, operating on attributed architecture graphs. Node features encode the node family, semantic type, and any associated design-variable value. Continuous variables use a sparse encoding indexed by canonical component type c and variable name d:

$$[\boldsymbol{\phi}_v]_{(c,d)} = \begin{cases} (x_d - l_d)/(u_d - l_d), & \text{if node } v \text{ represents variable } d \text{ of type } c, \\ 0, & \text{otherwise,} \end{cases}$$

with l_d, u_d the variable bounds. Repeated instances of a component type thus share the overlapping feature dimensions across all node rows; categorical variables use one dimension per option.

  - *DGBO*: the graph-neural surrogate of Cui et al. [14].

– *Arch2Vec*: extended from adjacency and discrete label reconstruction [10] to joint structure–feature embeddings, as our problems additionally involve continuous sizing parameters.

- **Model-free optimizers**, constructing no surrogate.
  – *Random*: samples feasible designs directly, giving a reference with asymptotic global coverage — when the sampling distribution has full support, it eventually samples arbitrarily close to any feasible design [47].
  – *GA*: NSGA-II [38] on the multi-objective problems and a single-objective genetic algorithm on ENGINE. NSGA-II also serves as the infill optimizer within the surrogate-based methods (Section 4.1); here it is applied directly to the expensive evaluations.

To ensure that the baselines are not disadvantaged by arbitrary configurations, HIER and KPLS use the established implementations and previously validated settings from SBArchOpt [2, 37]. DGBO follows the architecture of Cui et al. [14] and is lightly tuned through preliminary experiments, primarily with respect to its learning rate. As our Arch2Vec adaptation introduces several new configuration choices, we conduct preliminary ablations to assess reconstruction quality and downstream BO performance and select a well-performing configuration. SP-RBF and BLR learn their model hyperparameters during fitting, while Random and GA require no tuning.

5.3 Evaluation details

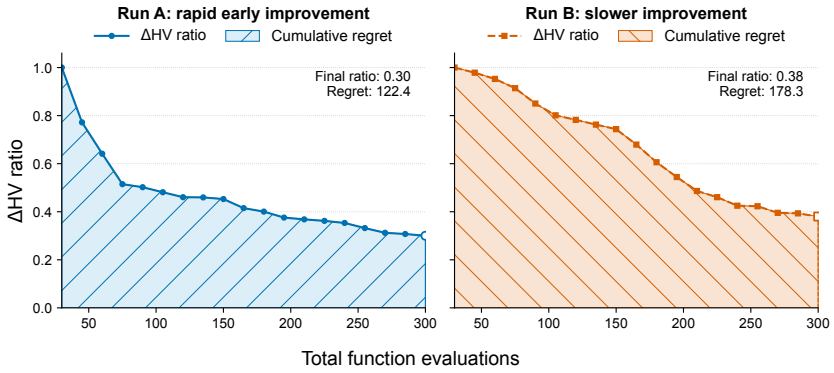


Fig. 8 Illustration of the ΔHV ratio trajectories and corresponding cumulative regret for two synthetic optimization runs. In each panel, the hatched area between the trajectory and zero over the evaluation budget represents that run's cumulative regret. Run A (left) improves rapidly, reaching a final ΔHV ratio of 0.30 and regret of 122.4; Run B (right) improves more slowly, reaching 0.38 and 178.3, respectively.

We compare final solution quality and convergence over the complete evaluation budget using the metrics defined below. The evaluation protocol follows [2].

We additionally report total optimization wall-clock time, comprising surrogate fitting, infill search, one-off pretraining where applicable (Arch2Vec), benchmark

evaluation, and other optimization overhead. Individual evaluations of the adopted benchmark problems take on the order of milliseconds and therefore contribute negligibly to the total wall-clock time. The reported runtime thus primarily reflects surrogate-fitting and infill-search costs.

The performance metrics defined below require a reference optimum. Since the exact optimal solutions are generally unavailable, we approximate them with genetic-algorithm runs involving between 30,000 and 192,000 function evaluations. This vastly exceeds the evaluation budgets used in the final experiments. For single-objective problems, $f^\star$ denotes the best feasible objective value found across these runs. For multi-objective problems, their combined non-dominated solutions define the reference Pareto front and its hypervolume $HV^\star$. For each problem, all hypervolumes are computed using the same fixed worst-case reference point r, also obtained using GA.

For a single-objective minimization problem, let $f_{\text{best},i}$ denote the best feasible objective value found at checkpoint i. The remaining optimality gap and its ratio relative to the initial DoE are

$$g_i = f_{\text{best},i} - f^\star, \qquad \text{GapRatio}_i = \frac{g_i}{g_0}.$$

A gap ratio of 1 means that none of the initial gap has been closed, 0.2 means that 20% remains, and 0 indicates that the reference optimum has been reached.

For multi-objective problems, let HV_i denote the hypervolume of the obtained non-dominated set at checkpoint i. The normalized hypervolume error is

$$\Delta HV_i = \frac{HV^\star - HV_i}{HV^\star},$$

which measures the fraction of the reference-front hypervolume that has not yet been recovered. To account for variation in the quality of the initial DoE across runs, this error is normalized by its initial value, yielding the ΔHV ratio:

$$\Delta HV\text{Ratio}_i = \frac{\Delta HV_i}{\Delta HV_0}.$$

As all methods use matched DoE seeds, comparisons remain fair, and we verified that absolute errors preserve method rankings. As illustrated in Figure 8, the final value $\Delta HV\text{Ratio}_T$ measures the fraction of the initial hypervolume error that remains at the end of the evaluation budget. A value of 1 indicates no improvement over the initial DoE, while 0 indicates that the reference-front hypervolume has been recovered.

For both problem types, convergence over the complete optimization run is summarized by the trapezoidal area under the corresponding error-ratio curve:

$$\text{Regret}_i = \text{Regret}_{i-1} + \frac{q_{i-1} + q_i}{2}\,\Delta n_i,$$

where q_i denotes GapRatio_i for single-objective problems and $\Delta HV\text{Ratio}_i$ for multi-objective problems, while Δn_i is the number of function evaluations since the previous

checkpoint. As shown in Figure 8, reducing the remaining gap earlier yields lower regret, even when final solution quality is similar.

For the ablation studies, each configuration is evaluated over 20 independent runs using different random seeds. The final comparison is performed over 40 additional non-overlapping seeds, providing an independent and more robust estimate of optimization performance.

6 Results

This section first presents the ablation studies used to select the final model configurations, followed by the main benchmark results obtained using these settings. Each experiment is discussed in a separate subsection. Additionally, in Section C we investigate the predictive performance of the surrogate models directly, outside of the optimization setting.

6.1 Kernel composition study

Table 2 Kernel interaction-composition ablation for depth-8 MG-LD-WLOA and SP-RBF. Results are reported as mean ± standard deviation across 20 matched runs. Lower is better for all metrics; the best value per column within each kernel is shown in bold.

MDGNC				
Kernel	Composition	Time (s)	Regret	ΔHV ratio
MG-LD-WLOA	Additive	3463 ± 217	**63.20** ± 10.83	**0.215** ± 0.051
MG-LD-WLOA	Multiplicative	**3348** ± 245	65.62 ± 8.47	0.223 ± 0.049
MG-LD-WLOA	ANOVA	3525 ± 293	64.25 ± 7.16	0.227 ± 0.042
SP-RBF	Additive	1622 ± 92	**76.57** ± 13.51	0.335 ± 0.070
SP-RBF	Multiplicative	**1565** ± 108	77.92 ± 13.20	0.324 ± 0.056
SP-RBF	ANOVA	1705 ± 93	77.35 ± 12.45	**0.320** ± 0.077
ENGINE				
Kernel	Composition	Time (s)	Regret	Gap ratio
MG-LD-WLOA	Additive	19 164 ± 2486	81.86 ± 54.22	0.177 ± 0.180
MG-LD-WLOA	Multiplicative	**16 924** ± 2941	**53.34** ± 33.74	**0.082** ± 0.074
MG-LD-WLOA	ANOVA	19 751 ± 1628	54.81 ± 28.74	**0.082** ± 0.100
SP-RBF	Additive	9521 ± 172	81.63 ± 47.55	0.202 ± 0.188
SP-RBF	Multiplicative	**8726** ± 375	54.23 ± 31.30	**0.079** ± 0.067
SP-RBF	ANOVA	9202 ± 450	**53.28** ± 30.32	0.102 ± 0.103

Here, we compare the learned additive, multiplicative, and ANOVA-style kernel composition described in Section 4.2. Each configuration is evaluated over 20 reruns on the MDGNC and ENGINE problems. As can be seen in Table 2, for MDGNC the

composition choice has little effect for either SP-RBF or MG-LD-WLOA. However, for ENGINE the multiplicative and ANOVA compositions perform substantially better than the additive composition.

The learned ANOVA weights provide a better insight into this behavior. For ENGINE, the graph–sizing interaction term receives the largest average weight for four of the five constraints, as shown in Figure 9, consistent with the strong performance of the multiplicative compositions for that benchmark. On MDGNC, the graph–sizing interaction dominates the failure-rate objective, whereas the isolated sizing term dominates mass, in line with the problem formulation described in Section 5.1. Overall, the results do not show a clear performance advantage of ANOVA over the best fixed composition. Instead, they show that the optimal kernel composition can be problem-dependent, as is the case for ENGINE, whereas ANOVA remains competitive across both benchmarks by learning the relative importance of additive and interaction effects rather than requiring this choice to be made a priori.

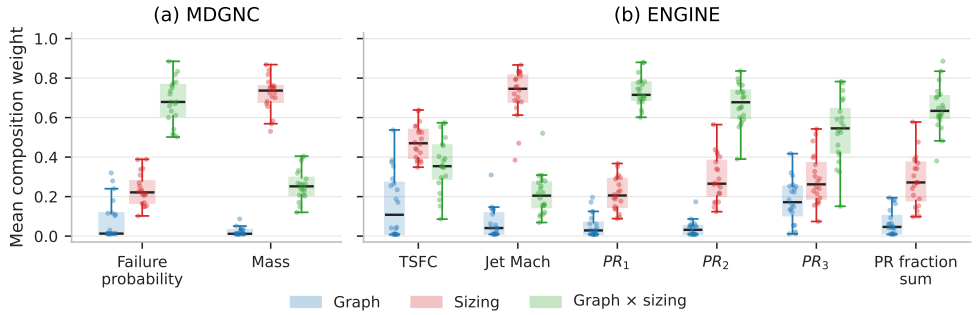


Fig. 9 Learned kernel-composition weights for MG-LD-WLOA on (a) MDGNC and (b) ENGINE. Each point is the mean normalized weight over all Bayesian optimization updates in one of 20 runs; boxes summarize the distributions across runs. The graph, sizing, and interaction weights sum to one at every update.

6.2 LD-WLOA depth study

The motivation for the learned-depth WLOA kernel is that it removes the need to select a single WL refinement depth in advance. To isolate this choice, we compare fixed-depth WLOA with $d \in \{0, \ldots, 8\}$ against LD-WLOA with $d_{\max} = 8$, while fixing the remaining configuration. We set the maximum WL depth to 8, as it provides a structural reach comparable to the maximum graph diameter of 17 edges encountered in the considered benchmarks, with most graphs being substantially smaller. For this study, all variants use the fixed semantic-type labeling and the ANOVA-style composition. Each configuration is evaluated using the same 20 random seeds.

As shown in Table 3, the strongest fixed depth depends on both the problem and the performance metric. LD-WLOA remains close to the best fixed-depth results across both benchmarks. In terms of regret, Holm-corrected paired Wilcoxon tests identify neither a significant improvement nor a significant degradation of LD-WLOA

Table 3 WLOA depth ablation results, reported as mean ± standard deviation over 20 independent runs. Fixed-depth variants use $d \in \{0, \ldots, 8\}$, while LD-WLOA uses $d_{\max} = 8$. Runtime is given in seconds. For MDGNC, regret and gap ratio are based on hypervolume. Bold values indicate the best mean for each metric and problem.

Problem	Method	Regret	Gap / ΔHV ratio	Runtime [s]
MDGNC	WLOA ($d = 0$)	111.45 ± 15.06	0.516 ± 0.114	$\mathbf{1288 \pm 131}$
	WLOA ($d = 1$)	88.97 ± 13.56	0.362 ± 0.078	1363 ± 159
	WLOA ($d = 2$)	78.11 ± 13.23	0.279 ± 0.068	1455 ± 115
	WLOA ($d = 3$)	66.04 ± 8.38	0.210 ± 0.038	1497 ± 107
	WLOA ($d = 4$)	65.02 ± 7.49	0.205 ± 0.027	1471 ± 107
	WLOA ($d = 5$)	$\mathbf{64.57 \pm 6.34}$	0.203 ± 0.032	1646 ± 86
	WLOA ($d = 6$)	66.92 ± 6.91	0.188 ± 0.025	1682 ± 108
	WLOA ($d = 7$)	67.75 ± 10.28	0.196 ± 0.037	1764 ± 124
	WLOA ($d = 8$)	68.53 ± 7.31	$\mathbf{0.186 \pm 0.035}$	1859 ± 117
	LD-WLOA ($d_{\max} = 8$)	67.61 ± 8.39	0.214 ± 0.042	2767 ± 138
ENGINE	WLOA ($d = 0$)	57.26 ± 32.96	0.088 ± 0.084	7025 ± 639
	WLOA ($d = 1$)	51.07 ± 28.54	0.087 ± 0.095	7855 ± 701
	WLOA ($d = 2$)	$\mathbf{45.60 \pm 27.35}$	$\mathbf{0.069 \pm 0.090}$	$\mathbf{7008 \pm 1273}$
	WLOA ($d = 3$)	56.30 ± 36.05	0.114 ± 0.128	8373 ± 418
	WLOA ($d = 4$)	62.18 ± 39.81	0.128 ± 0.145	8922 ± 511
	WLOA ($d = 5$)	50.49 ± 27.57	0.076 ± 0.084	9598 ± 619
	WLOA ($d = 6$)	54.59 ± 31.72	0.090 ± 0.115	$10\,188 \pm 938$
	WLOA ($d = 7$)	49.48 ± 30.62	0.090 ± 0.110	$10\,658 \pm 762$
	WLOA ($d = 8$)	49.18 ± 29.25	0.071 ± 0.084	$11\,336 \pm 877$
	LD-WLOA ($d_{\max} = 8$)	52.34 ± 24.85	0.079 ± 0.076	$14\,595 \pm 669$

relative to any fixed depth at $\alpha = 0.05$. Thus, although LD-WLOA does not clearly outperform the strongest fixed-depth variants, it removes the need for manual depth selection while remaining close to their performance across both benchmarks. This is useful since an unsuitable fixed depth can incur a substantial penalty. For instance, on ENGINE selecting $d = 4$ instead of $d = 2$ increases mean regret from 45.60 to 62.18 and the final gap ratio from 0.069 to 0.128.

Figure 10a summarizes the learned depth distribution over repeated optimization runs by its weighted mean depth $\bar{h}_w$; for example, equal weights for depths 0 through 5 yield $\bar{h}_w = 2.5$. The mean increases from approximately 1.4 after 40 evaluations to 2.1–2.4 between 50 and 70 evaluations, after which it changes more gradually. LD-WLOA therefore initially favors coarse, local information and becomes more discriminative by incorporating larger neighborhood patterns as more data become available.

Learning the depth additionally provides insight into which structural scales are relevant to each objective. As discussed in Section 5.1, MDGNC failure probability depends on connectivity and redundancy across multiple components, and Figure 9 shows that graph-based similarity contributes strongly to its surrogate. Accordingly, Figure 10b shows that LD-WLOA meaningfully weights depths 0 through 5, which is consistent with the importance of multi-hop connectivity and redundancy patterns.

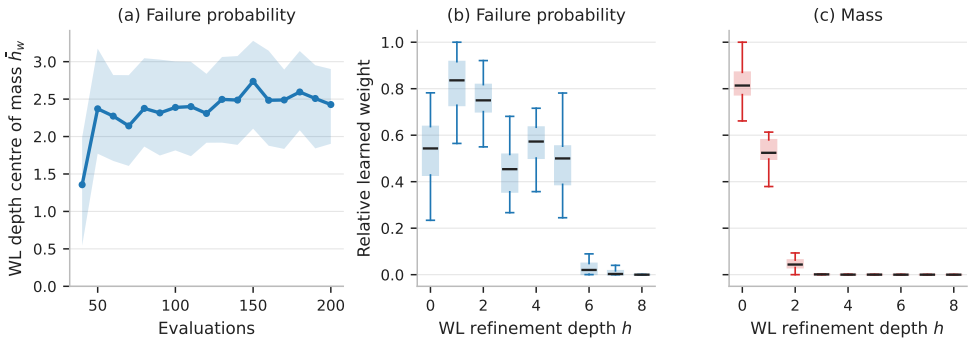


Fig. 10 Learned refinement depths of LD-WLOA on MDGNC. (a) Evolution of the depth centre of mass $\bar{h}_w$ for the failure-probability objective; the line and shaded region denote the mean and central 75% interval over 20 runs. Panels (b) and (c) show the distributions across runs of the weights learned for the failure-probability and mass objectives, respectively. At each BO update, weights are normalized to sum to one and subsequently averaged over updates within each run. For visual comparison, each panel is scaled by its largest non-outlying value.

For mass, which is primarily sizing dependent, the graph component contributes less overall. Nevertheless, the learned depths remain consistent with the problem structure. As shown in Figure 10c, the kernel captures overlap in terms of component counts ($d = 0$) and depth-one subtrees ($d = 1$). A match at $d = 1$ requires agreement in the number and types of neighboring nodes, and therefore also reflects local connection counts, which contribute directly to mass.

This adaptivity requires additional computation because LD-WLOA constructs representations up to $d_{\max}$ and optimizes their weights. The overhead is most readily justified in the intended expensive-evaluation setting, where surrogate fitting constitutes only a small fraction of the total optimization time.

6.3 Label granularity study

Table 4 Effect of label granularity on cumulative regret and total runtime. Values are mean ± standard deviation over 20 independent runs; lower is better for all columns, and the best value per column is shown in bold.

	ENGINE		MDGNC	
Label Granularity	Regret	Runtime [h]	HV Regret	Runtime [h]
DSG	**48.70** ± 27.16	**3.75** ± 0.19	**62.25** ± 8.44	0.75 ± 0.04
ADSG	52.60 ± 30.31	3.85 ± 0.19	68.53 ± 10.15	**0.74** ± 0.03
Semantic	52.34 ± 24.85	4.05 ± 0.19	67.61 ± 8.39	0.77 ± 0.04
ADSG + Semantic	54.81 ± 28.74	5.49 ± 0.45	64.25 ± 7.16	0.98 ± 0.08

This study compares using DSG-level, ADSG-level, semantic labels, and a multi-granularity kernel variant. In particular, we compare with the ADSG + Semantic

hybrid. DSG is omitted from this combination because its generic labels can make locally similar patterns match despite having different architecting roles. For example, at shallow depths, a function-decomposition pattern could be matched with a component connection pattern.

As shown in Table 4, DSG achieves the lowest regret on both problems, despite its potential for matching architecturally unrelated patterns at shallower WL depths. However, two-sided Welch tests with $\alpha = 0.05$ detect no significant difference between DSG and the multi-granularity representation on either ENGINE ($p = 0.494$) or MDGNC ($p = 0.423$). The added flexibility of MG-LD-WLOA comes at a computational cost, increasing runtime by 1.74 h on ENGINE and 0.23 h on MDGNC.

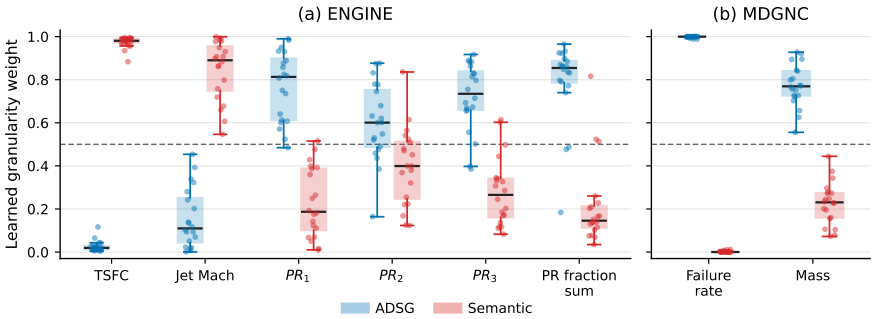


Fig. 11 Granularity weights learned by MG-LD-WLOA throughout Bayesian optimization. Each point is the mean across all BO updates in one run, and each box summarizes 20 runs. The dashed line denotes equal weighting. The response-dependent distributions show that the surrogate adapts its label granularity to individual objectives and constraints.

Inspecting the learned weights, as shown in Figure 11, confirms that the additional flexibility of the multi-granularity variant is used. The MDGNC failure-rate surrogate assigns an average ADSG weight of 0.998, compared with 0.773 for mass, while ENGINE exhibits still greater variation across its responses. The failure-rate preference for MDGNC is consistent with the problem formulation, as additional connections generally improve reliability by creating alternative paths through the sensor–computer–actuator network. Although the exact placement of connections matters given how the failure rate is calculated, with similarity in upstream sensor–computer connection patterns being slightly more relevant than similarity in downstream computer–actuator patterns, generic connectivity and redundancy should already provide a strong signal.

Overall, the multi-granularity representation does not outperform the fixed labelings on these benchmarks and incurs additional computational cost. Nevertheless, it remains competitive with DSG, and the learned weights show that the preferred granularity varies across objectives. The multi-granularity representation may therefore provide greater robustness across SAO design spaces, as relying on any single labeling could degrade performance when its distinctions do not align with the problem. For instance, although DSG performs well here, it could perform poorly on ROCKET where objectives depend on the specific component types, which DSG does not capture.

We therefore retain ADSG + Semantic for the final method, as these complementary label granularities should together capture the key structural patterns across SAO problems while allowing objective-dependent adaptation, without incurring the cost of additional labelings.

6.4 Edge-weight kernel study

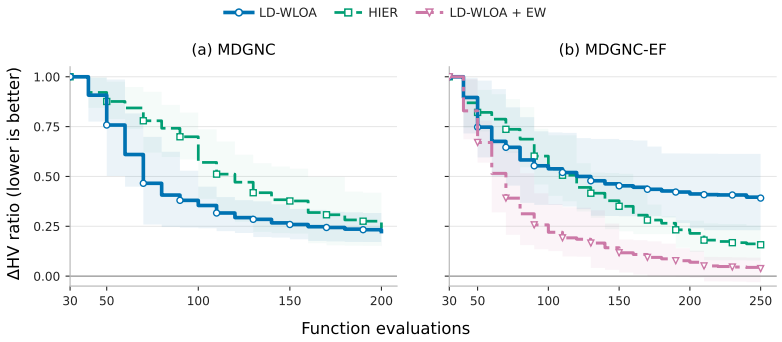


Fig. 12 Effect of edge-weight information on LD-WLOA, with hierarchical (HIER) kernel GP shown as a reference. Standard LD-WLOA performs well on MDGNC (a) but degrades under edge failures (b), where adding the edge-weight kernel (EW) substantially improves convergence.

We evaluate the effect of explicitly modeling connection multiplicity across 20 matched reruns, using the kernel proposed in Section 4.5. To that end, we compare the optimization performance on the MDGNC-EF benchmark with and without including it in the kernel composition. The optimization curves are shown in Figure 12.

As shown in Table 5, LD-WLOA performs well on the standard MDGNC problem, outperforming HIER on both reported metrics. This changes on MDGNC-EF, where failure rate depends on the number of parallel connections. Without modeling multiplicity information, LD-WLOA reaches a final ΔHV ratio of 0.392 and an HV regret of 115.1, compared with 0.157 and 97.0 for HIER. Unlike LD-WLOA, the design-vector encoding used by HIER retains connection-multiplicity information, and HIER performs significantly better than LD-WLOA on both metrics ($p < 0.03$).

Adding the edge-weight kernel reverses this degradation. Relative to LD-WLOA without edge weights, it reduces the final ΔHV ratio by over 90%, whereas regret decreases by 55.3%. Both improvements are significant ($p < 0.001$). LD-WLOA + EW also achieves significantly better performance than HIER on both metrics.

These results provide strong evidence that the deterioration of LD-WLOA on MDGNC-EF results from omitting connection-multiplicity information, and that the proposed edge-weight component supplies a useful signal when parallel-edge redundancy directly affects the objective. We therefore include the edge-weight kernel for all problems in which edge multiplicities arise.

Table 5 Impact of including edge multiplicity information on the MDGNC-EF problem. Values are mean ± standard deviation over 20 independent runs; lower is better and the best value per problem is shown in bold.

Problem	Method	ΔHV ratio	HV regret	Runtime [s]
MDGNC	HIER	0.240 ± 0.068	94.0 ± 16.7	$\mathbf{1150} \pm 174$
	LD-WLOA	$\mathbf{0.224} \pm 0.066$	$\mathbf{68.8} \pm 12.7$	2539 ± 170
MDGNC-EF	HIER	0.157 ± 0.099	97.0 ± 16.5	$\mathbf{1740} \pm 201$
	LD-WLOA	0.392 ± 0.148	115.1 ± 31.1	4041 ± 349
	LD-WLOA + EW	$\mathbf{0.038} \pm 0.078$	$\mathbf{51.4} \pm 17.9$	4053 ± 621

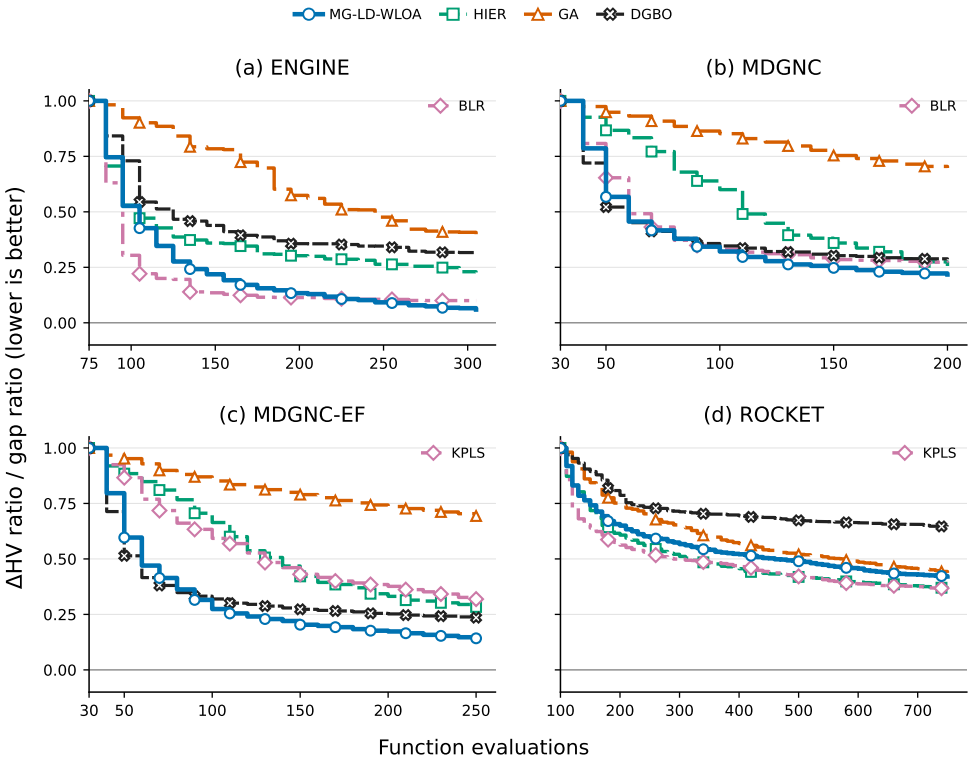


Fig. 13 Optimization performance over 40 independent runs. Curves show the mean normalized objective-gap ratio for ENGINE and the mean normalized ΔHV ratio for the multi-objective benchmarks; lower values are better. For each problem, an additional curve shows the strongest of the remaining encoding-based BO baselines (BLR and KPLS), selected by the lower mean ratio at the final evaluation budget: BLR for ENGINE and MDGNC, and KPLS for MDGNC-EF and ROCKET. Standard deviations and complete final results are reported in Table 6.

6.5 Optimization performance evaluation

We compare MG-LD-WLOA with the considered baselines across 40 independent reruns. The final MG-LD-WLOA configuration is shared across all problems. We use learned depths up to $H = 8$, the multi-granularity ADSG + Semantic representation, and the ANOVA-style kernel composition; the edge-weight kernel is additionally included for the MDGNC-EF problem. While individual fixed variants occasionally achieve better mean performance in the ablations, this configuration remains general and avoids problem-specific parameter tuning, which is generally infeasible in SAO.

Figure 13 shows the mean normalized objective-gap ratio for ENGINE and the mean normalized ΔHV ratio for the multi-objective problems. The figure includes the hierarchical kernel GP (as adopted in Bussemaker's framework [2, 4]), the strongest encoding-based BO method for each problem, DGBO, the genetic algorithm, and MG-LD-WLOA. Exact final ratios and cumulative regrets are reported in Table 6. We additionally assess statistical significance using two-sided paired Wilcoxon signed-rank tests with Holm correction.

Table 6 Final optimization performance after the problem-specific evaluation budget. Values are mean ± standard deviation over 40 independent runs; lower is better. Bold and underlined entries denote the best and second-best result in each column, respectively. The MDGNC-EF runs for SP-RBF and MG-LD-WLOA included the EW kernel from Section 4.5.

	ENGINE		MDGNC	
Method	Regret	Gap ratio	HV regret	ΔHV ratio
Random	191.01 ± 39.71	0.646 ± 0.281	145.03 ± 12.31	0.783 ± 0.082
GA	148.85 ± 52.59	0.388 ± 0.247	140.74 ± 11.67	0.698 ± 0.082
HIER	81.62 ± 48.08	0.227 ± 0.186	92.39 ± 17.67	<u>0.256 ± 0.128</u>
KPLS	67.51 ± 47.25	0.166 ± 0.171	84.64 ± 12.25	0.273 ± 0.075
BLR	**40.21 ± 24.17**	0.100 ± 0.101	66.99 ± 13.14	0.271 ± 0.063
SP-RBF	<u>47.91 ± 24.09</u>	<u>0.080 ± 0.099</u>	77.27 ± 13.31	0.319 ± 0.072
Arch2Vec	96.62 ± 63.57	0.270 ± 0.278	85.31 ± 16.69	0.394 ± 0.090
DGBO	99.39 ± 68.11	0.316 ± 0.323	<u>65.89 ± 15.76</u>	0.286 ± 0.083
MG-LD-WLOA	50.34 ± 22.37	**0.060 ± 0.066**	**61.34 ± 8.32**	**0.218 ± 0.045**
	MDGNC-EF		ROCKET	
Method	HV regret	ΔHV ratio	HV regret	ΔHV ratio
Random	191.34 ± 11.62	0.810 ± 0.056	413.36 ± 89.06	0.492 ± 0.150
GA	178.88 ± 11.03	0.694 ± 0.064	388.89 ± 97.29	0.437 ± 0.138
HIER	118.19 ± 19.64	0.284 ± 0.106	<u>318.77 ± 76.83</u>	<u>0.369 ± 0.117</u>
KPLS	116.47 ± 16.79	0.319 ± 0.074	**309.18 ± 83.34**	**0.367 ± 0.110**
BLR	108.16 ± 19.95	0.353 ± 0.089	528.10 ± 51.22	0.739 ± 0.094
SP-RBF	89.38 ± 11.52	0.279 ± 0.065	401.08 ± 86.90	0.472 ± 0.156
Arch2Vec	119.47 ± 23.01	0.425 ± 0.101	531.74 ± 54.61	0.728 ± 0.108
DGBO	<u>73.49 ± 16.61</u>	<u>0.235 ± 0.066</u>	466.11 ± 103.48	0.646 ± 0.188
MG-LD-WLOA	**64.95 ± 11.76**	**0.142 ± 0.039**	352.59 ± 83.50	0.421 ± 0.131

MG-LD-WLOA obtains the best mean regret and final ratio on both MDGNC and MDGNC-EF, the problems in which performance depends explicitly on component connectivity. On MDGNC, it reduces regret by approximately 7% relative to DGBO and the final ratio by 15% relative to hierarchical kernel GP. Across the two metrics, MG-LD-WLOA significantly outperforms every comparator on at least one metric (all $p \leq 0.000341$), while no comparator significantly outperforms it on either metric. On MDGNC-EF, the advantage is stronger: MG-LD-WLOA reduces regret by approximately 12% and the final ratio by 40% relative to DGBO. It significantly outperforms every comparator on both metrics; even against DGBO, the closest competitor, the differences remain significant for regret ($p = 0.00696$) and final error ($p = 5.06 \times 10^{-8}$).

These results seem to align well with the representation mismatch discussed in Section 2.3. In MDGNC and MDGNC-EF, the reliability objective depends on connectivity patterns spanning multiple connection choices, and while the vector encodings preserve connection similarity within each choice separately, they do not directly preserve similarity of global patterns spanning several choices. Graph-based surrogates instead operate on the architecture graphs and can capture such higher-level structural patterns directly. The fact that two substantially different graph surrogates, namely MG-LD-WLOA and DGBO, both perform particularly strongly on these two problems appears consistent with a broader benefit arising from their ability to capture higher-level connectivity structure. For MG-LD-WLOA, this interpretation is further supported by the predictive study in Section C, where it generalizes best among the evaluated surrogates to isomorphic and structurally unseen architectures.

On ENGINE, MG-LD-WLOA obtains the best final gap ratio and is not significantly outperformed by any method on either metric. It achieves significantly lower regret than hierarchical kernel GP and DGBO (both $p \leq 0.000269$), and significantly lower final error than hierarchical kernel GP, KPLS, and DGBO (all $p \leq 0.000216$). Although BLR and SP-RBF achieve lower mean cumulative regret, neither difference is significant (all $p \geq 0.135$). Unlike MDGNC, ENGINE does not contain consecutive connection choices and is driven primarily by hierarchical architectural decisions and sizing parameters, so the benefit of graph-level structural similarity is less pronounced.

ROCKET contains no connection choices and no objectives that clearly depend on higher-order connectivity patterns, so the potential advantage of graph-based representations is less pronounced than on MDGNC and MDGNC-EF, although the graph still represents hierarchical structure that the kernel can distinguish. A plausible factor is that MG-LD-WLOA can only adjust the importance of each WL depth, weighting all patterns at that depth equally, whereas encoding-based kernels can learn separate sensitivities for individual selection choice encodings. This reduced flexibility may contribute to its comparatively weaker performance on a predominantly hierarchical problem. However, ROCKET also differs in other respects, including its constraint and feasibility characteristics, making it difficult to confidently attribute the result to any single factor. Nevertheless, MG-LD-WLOA significantly outperforms BLR and DGBO on both metrics ($p \leq 4.05 \times 10^{-9}$) and outperforms random search and the genetic algorithm in mean performance on all four benchmarks.

Computational cost

MG-LD-WLOA has substantially greater optimization overhead, as shown in Table 7. Because the test problem evaluations are inexpensive, these measurements primarily reflect surrogate training and infill search costs. However, in a realistic setting where evaluation is the most expensive step, the relevant quantity is the wall-clock time required to reach a given solution quality.

Table 7 Total optimization wall-clock time in seconds. Values are mean ± standard deviation over 40 runs. Arch2Vec includes its one-off encoder pretraining cost. GNN methods used GPU nodes, whereas the remaining methods used CPU nodes; times are therefore not hardware-normalized.

Method	ENGINE	MDGNC	MDGNC-EF	ROCKET
Random	29.6 ± 0.5	4.3 ± 0.1	6.9 ± 0.1	47.7 ± 1.6
GA	34.7 ± 1.2	5.6 ± 0.2	7.3 ± 0.3	39.3 ± 5.1
HIER	$3\,286 \pm 523$	974 ± 183	$1\,627 \pm 253$	$40\,089 \pm 6\,250$
KPLS	$1\,550 \pm 217$	332 ± 24	545 ± 33	$4\,719 \pm 480$
BLR	329 ± 32	79.5 ± 4.3	83.8 ± 5.0	212 ± 62
SP-RBF	$9\,276 \pm 391$	$1\,700 \pm 94$	$3\,049 \pm 192$	$37\,589 \pm 3\,812$
Arch2Vec	$1\,320 \pm 131$	597 ± 47	755 ± 45	$1\,609 \pm 356$
DGBO	$2\,062 \pm 226$	735 ± 128	$1\,120 \pm 257$	$8\,719 \pm 3\,470$
MG-LD-WLOA	$19\,548 \pm 1\,489$	$3\,561 \pm 286$	$6\,373 \pm 492$	$139\,481 \pm 27\,374$

MG-LD-WLOA reaches the final mean performance of BLR after approximately 130 rather than 200 evaluations on MDGNC, and that of KPLS after 90 rather than 250 evaluations on MDGNC-EF. The relevant overhead is the cumulative MG-LD-WLOA runtime at these earlier matching points: approximately 1 809 s at 130 evaluations on MDGNC and 1 144 s at 90 evaluations on MDGNC-EF. Dividing the excess over the baseline's full-run overhead by the evaluations saved gives break-even evaluation times of $(1809 - 79.5)/(200 - 130) \approx 25$ s and $(1144 - 545)/(250 - 90) \approx 4$ s, respectively. These thresholds are modest in the context of SAO. A single physics-based evaluation of the jet-engine architecture problem underlying the ENGINE benchmark takes on the order of two minutes [2], approximately five times the MDGNC threshold and thirty times the MDGNC-EF threshold. For realistic simulation-based evaluations, the gain in sample efficiency offsets the additional surrogate overhead; for inexpensive analytical evaluations, however, this overhead can become limiting.

7 Practical selection criteria

The results, together with the computational timings in Table 7, suggest the following practical criteria for selecting an optimization approach:

- *Topological content, expensive evaluations.* The proposed surrogate should be considered when optimized objectives depend on connectivity, redundancy, or other graph-level structural patterns. The break-even evaluation times reported in Section 6.5 are on the order of seconds to tens of seconds, indicating that

the additional optimization overhead can be recovered for realistically expensive simulation-based evaluations. Guidance and control systems, sensor networks, IoT meshes, and hydraulic or electrical distribution networks plausibly fall into this category.

- *Limited topological dependence, expensive evaluations.* When objectives are not clearly connectivity-driven, the advantage of graph-based surrogates is less clear. Encoding-based Bayesian optimization remains an attractive default in such cases given its lower computational cost.
- *Cheap evaluations.* Low-overhead methods such as BLR or evolutionary optimization become increasingly attractive regardless of representation, since time saved on surrogate fitting and infill search can instead be spent evaluating more designs. This is consistent with the findings of Bussemaker et al. [2].

8 Conclusions

This work investigated whether Gaussian-process surrogates incorporating graph-based structural similarity between resolved architecture instances can improve the sample efficiency of System Architecture Optimization. We introduced a composite graph-kernel surrogate combining structural, sizing, and, where applicable, edge-multiplicity information. We further proposed learned-depth and multi-granularity extensions of the Weisfeiler–Lehman optimal assignment kernel.

Across the benchmark problems, our proposed method, MG-LD-WLOA, achieves the strongest performance on the topology-dependent MDGNC and MDGNC-EF problems, remains competitive with encoding-based methods on the predominantly hierarchical and sizing-driven ENGINE problem, but performs less strongly on ROCKET. GNN-based DGBO also performs well on MDGNC and MDGNC-EF. The strong performance of two distinct graph-based surrogates appears consistent with a broader benefit from graph representations in connectivity-driven problems. However, MDGNC and MDGNC-EF are related variants rather than fully independent benchmarks. At the same time, readily reusable realistic SAO benchmarks are limited. While the results are therefore promising, validation on additional topology-sensitive cases, ideally including real-world engineering problems with full physics-based evaluations, is needed to establish how generally this pattern holds.

If the promise shown by these initial results carries over to other topology-sensitive SAO problems, graph-based methods could become increasingly valuable as SAO matures and moves toward larger industrial systems, where connectivity can be orders of magnitude richer than in the benchmarks considered here. For example, the A380 electrical wiring interconnection system contains approximately 100,000 wires and 40,300 connectors [48].

Finally, the proposed kernels provide a degree of interpretability through their learned depth, granularity, and composition weights. Across the ablation studies, these weights vary systematically across responses and generally align with known problem structure. This form of model transparency may help practitioners validate learned similarity relationships against engineering knowledge and identify which structural scales and information sources are relevant to different responses.

8.1 Future work

The most immediate direction for future work is broader validation, as discussed above. A larger number of SAO benchmarks, and ideally full-scale engineering applications with physics-based simulations, would provide a stronger test of the extent to which the performance pattern observed on MDGNC and MDGNC-EF generalizes.

Scalability and kernel expressiveness remain important directions for future work. The present approach deliberately adopts WL-based kernels because of their strong empirical performance and computational efficiency on labeled graphs [28, 31]. Nevertheless, repeated graph-kernel evaluations during GP fitting and infill search introduce substantial overhead, while GP inference itself scales as $\mathcal{O}(n^3)$. Larger evaluation budgets could therefore benefit from sparse or variational GP approximations [49] and low-rank kernel-matrix approximations such as the Nyström method [50]. More compact architecture graphs that remove structurally redundant ADSG elements while retaining performance-relevant topology could provide another alternative.

There is also scope for richer structural representations. Attributed graph kernels, such as GraphHopper [51] and propagation kernels [52], could associate sizing information directly with structural context. Their application to SAO is non-trivial, however, because architecture graphs are heterogeneous and different node types carry different sets of attributes. Hash graph kernels [53] provide one possible direction for efficiently incorporating continuous attributes.

The infill search presents another opportunity. NSGA-II currently operates on encoded design vectors, with every candidate decoded into an architecture graph before surrogate evaluation (Section 4.1). Evolutionary operators acting directly on architecture instances could remove this round-trip and make the graph representation consistent throughout the optimization loop.

Finally, the strong performance and short runtime of DGBO on the connectivity-driven problems suggest that graph-neural surrogates warrant closer study in SAO, as such models can in principle learn more flexible structural representations than graph kernels.

Statements and Declarations

Data and code availability

The source code, experimental configurations, and data supporting the results of this study are publicly available at https://github.com/flatala/sao_graph_sbo. The repository includes instructions and scripts for reproducing the experiments and generating the reported figures and tables.

Appendix A Validity of the LD-WLOA kernel

The relabeling procedure in Figure 4 induces a hierarchy, which is illustrated in Figure A1. Since every refined label $\tau_h(u)$ is dependent on the node's preceding label $\tau_{h-1}(u)$, two vertices that match at depth h must also match at all earlier depths. Equivalently, if two subtrees of depth h match, the subtrees of depth $h-1$ with the same roots must also match. Each label at depth h therefore has a unique parent at

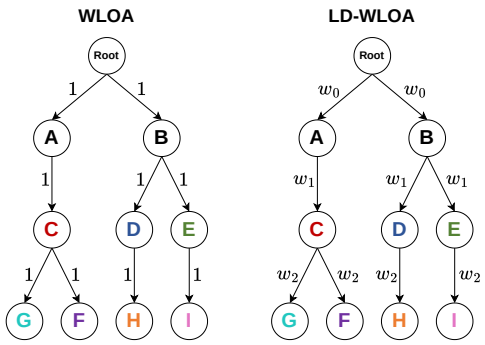


Fig. A1 WL label hierarchy induced by the relabeling example in Figure 4. Standard WLOA assigns unit contribution to each shared hierarchy level, whereas LD-WLOA assigns the learned nonnegative weight w_h to depth h.

depth $h-1$, and every vertex u follows the path

$$\text{Root},\ \tau_0(u),\ \tau_1(u),\ \ldots,\ \tau_H(u)$$

through the hierarchy. The similarity between two vertices is consequently determined by the deepest level up to which their paths coincide. As defined in Equation 3, standard WLOA assigns unit contribution to every shared level. LD-WLOA instead assigns the nonnegative weight w_h to the shared level at depth h, as defined in Equation 6.

Optimal-assignment kernels are not positive semidefinite for arbitrary base kernels. However, as Kriege et al. [32] show, optimal-assignment kernels induced by a strong base kernel are positive semidefinite. The following proposition shows that the nonnegative depth weighting preserves the strong-kernel property.

Proposition 1 *For nonnegative depth weights w_h, the LD-WLOA kernel is positive semidefinite.*

Proof A kernel k is strong if, for any $x, y, z \in \mathcal{X}$,

$$k(x,y) \geq \min\{k(x,z), k(z,y)\}.$$

We show that the weighted WL base kernel k_{LD} is strong. WL refinement is hierarchical: if two vertices have equal labels at depth h, then they also have equal labels at all earlier depths. Let $p(u,v)$ be the number of consecutive depths, starting from depth 0, at which u and v have equal WL labels. Then

$$k_{\mathrm{LD}}(u,v) = \sum_{h=0}^{H} w_h \delta\left(\tau_h(u), \tau_h(v)\right) = \sum_{h=0}^{p(u,v)-1} w_h,$$

where the sum is zero when $p(u,v) = 0$. Now take any vertices x, y, z, and define

$$a = p(x,z), \qquad b = p(z,y), \qquad m = \min(a,b).$$

The pairs (x, z) and (z, y) match at all depths $0, \ldots, m-1$. Therefore, by transitivity of label equality, x and y also match at these depths, so $p(x, y) \geq m$. Since all weights are nonnegative, these prefix sums are monotone. Hence,

$$k_{\mathrm{LD}}(x, y) = \sum_{h=0}^{p(x,y)-1} w_h \geq \sum_{h=0}^{m-1} w_h = \min\left\{\sum_{h=0}^{a-1} w_h, \sum_{h=0}^{b-1} w_h\right\} = \min\{k_{\mathrm{LD}}(x, z), k_{\mathrm{LD}}(z, y)\}.$$

Thus, k_{LD} is strong. Consequently, by [32], the optimal-assignment kernel induced by k_{LD}, namely LD-WLOA, is positive semidefinite. □

Appendix B Edge-failure MDGNC formulation

For an edge set E, let

$$\pi_E(R; E) = \prod_{e \in R} (1 - q_e) \prod_{e \in E \setminus R} q_e$$

denote the probability that exactly the subset $R \subseteq E$ remains operational. Let $R_{SC} \subseteq E_{SC}$ and $R_{CA} \subseteq E_{CA}$ be the surviving sensor–computer and computer–actuator edges. Furthermore, let $C(T_S, R_{SC})$ denote the computers reachable from the surviving sensors T_S, and $A(T_C, R_{CA})$ the actuators reachable from the surviving computers T_C. The failure probability is then

$$P_{\mathrm{fail}}^{\mathrm{edge}} = \sum_{T_S \subseteq S} \pi(T_S; S) \sum_{R_{SC} \subseteq E_{SC}} \pi_E(R_{SC}; E_{SC}) \times \sum_{T_C \subseteq C(T_S, R_{SC})} \pi(T_C; C(T_S, R_{SC})) \sum_{R_{CA} \subseteq E_{CA}} \pi_E(R_{CA}; E_{CA}) \prod_{a \in A(T_C, R_{CA})} q_a.$$

As in the original formulation, an empty actuator product equals one, so any state with no reachable operational actuator is counted as system failure.

Appendix C Structural generalization study

The experiments in Section 6 assess end-to-end optimization performance. Here we instead examine the surrogates' predictive behavior directly, and specifically how it generalizes as the structural distance between a test architecture and the training data grows. We focus on the failure-probability objectives of MDGNC and MDGNC-EF, which depend on the encountered connection structures (Section 5.1).

Held-out predictions are partitioned by their structural relationship to the training data. An *exact* architecture has the same discrete architecture as a training sample; an *isomorphic* architecture has a distinct encoded representation but an isomorphic graph structure; a *novel* structure is isomorphic to no training architecture. This partitioning considers only the discrete connection structure and disregards the continuous sizing variables, as the connection structure is a major driver of the failure probability. These distinct groups probe memorization, invariance to structural isomorphism, and extrapolation, respectively.

For each problem, 5,000 architectures were generated using stratified random sampling that preserved structural diversity. They were then divided into 10 disjoint folds of 500. In each split, one fold supplied the training candidates and the remaining 4,500

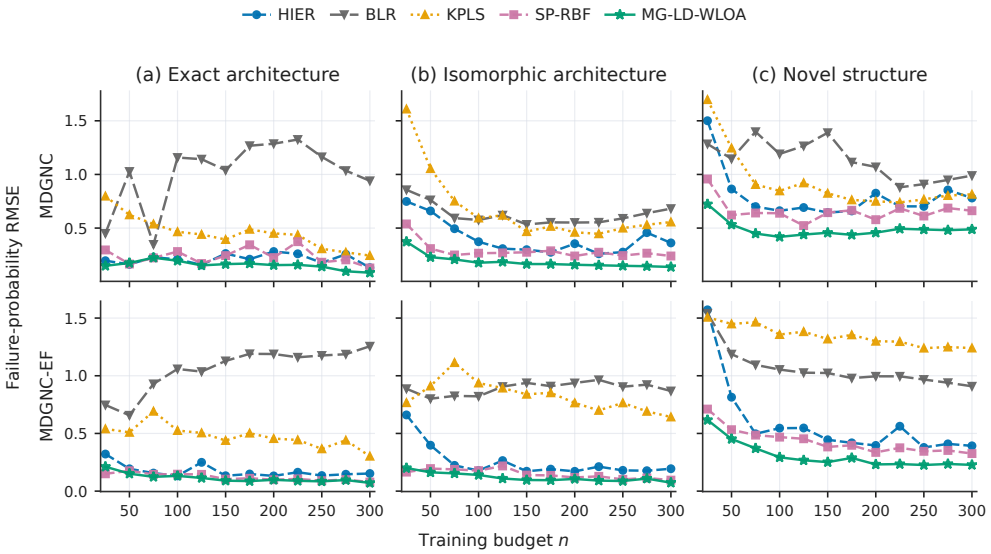


Fig. C2 Predictive RMSE for the graph-dependent failure-probability objective for MDGNC and MDGNC-EF, partitioned by the relationship between each held-out architecture and the training set, as a function of training budget.

samples formed the test set. Nested subsets of size $n \in \{25, 50, \ldots, 300\}$ were then drawn greedily from the training fold. All surrogates used the same sample selections, so differences between curves are attributable to the surrogate alone. Each point in Figure C2 is the mean of 10 fold-level RMSEs.

MG-LD-WLOA achieves the lowest mean RMSE in every regime on both problems. Its mean absolute exact–isomorphic gap is only 0.039 on MDGNC and 0.009 on MDGNC-EF, compared with 0.195 and 0.080 for HIER, the strongest encoding-based baseline. SP-RBF, which is also graph-kernel based, yields the second best gaps of 0.080 and 0.027. MG-LD-WLOA is invariant to graph isomorphism by construction, which follows from the WL procedure producing identical label histograms for isomorphic graphs, as discussed in Section 3.1.

The advantage extends to unobserved structures: MG-LD-WLOA attains the lowest novel-structure RMSE in all comparisons. MG-LD-WLOA therefore supplies a useful inductive bias for the considered objective and transfers better to unseen structures, as compared to encoding-based methods.

References

[1] Crawley, E.F., Cameron, B.G., Selva, D.: System Architecture: Strategy and Product Development for Complex Systems. Pearson, Harlow, UK (2015)

[2] Bussemaker, J.H., Saves, P., Bartoli, N., Lefebvre, T., Lafage, R.: System architecture optimization strategies: Dealing with expensive hierarchical problems. Journal of Global Optimization **91**(4), 851–895 (2025) https://doi.org/10.1007/

s10898-024-01443-8

[3] Madni, A.M., Purohit, S.: Economic analysis of model-based systems engineering. Systems **7**(1) (2019) https://doi.org/10.3390/systems7010012

[4] Bussemaker, J.H.: System architecture optimization: Function-based modeling, optimization algorithms, and multidisciplinary evaluation. Phd thesis, Delft University of Technology and ISAE-SUPAERO (2025). https://doi.org/10.4233/uuid:246b18f9-1f8c-4ff7-b824-2b1786cf9d14

[5] Shahriari, B., Swersky, K., Wang, Z., Adams, R.P., Freitas, N.: Taking the human out of the loop: A review of bayesian optimization. Proceedings of the IEEE **104**(1), 148–175 (2016) https://doi.org/10.1109/JPROC.2015.2494218

[6] Saves, P., Hallé-Hannan, E., Bussemaker, J., Diouane, Y., Bartoli, N.: Modeling hierarchical spaces: A review and unified framework for surrogate-based architecture design. Structural and Multidisciplinary Optimization **69**(3), 65 (2026) https://doi.org/10.1007/s00158-026-04249-2

[7] Ru, B., Wan, X., Dong, X., Osborne, M.A.: Interpretable neural architecture search via bayesian optimisation with weisfeiler-lehman kernels. In: International Conference on Learning Representations (2021)

[8] Xie, Y., Zhang, S., Qing, J., Misener, R., Tsay, C.: Global optimization of graph acquisition functions for neural architecture search. In: International Conference on Learning Representations (2026)

[9] Shi, H., Pi, R., Xu, H., Li, Z., Kwok, J.T., Zhang, T.: Bridging the gap between sample-based and one-shot neural architecture search with BONAS. In: Advances in Neural Information Processing Systems, vol. 33 (2020)

[10] Yan, S., Zheng, Y., Ao, W., Zeng, X., Zhang, M.: Does unsupervised architecture representation learning help neural architecture search? In: Advances in Neural Information Processing Systems, vol. 33, pp. 12486–12498 (2020)

[11] Ma, L., Cui, J., Yang, B.: Deep neural architecture search with deep graph bayesian optimization. In: Proceedings of the 2019 IEEE/WIC/ACM International Conference on Web Intelligence. WI '19, pp. 500–507 (2019). https://doi.org/10.1145/3350546.3360740

[12] Korovina, K., Xu, S., Kandasamy, K., Neiswanger, W., Poczos, B., Schneider, J., Xing, E.: ChemBO: Bayesian optimization of small organic molecules with synthesizable recommendations. In: Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics. Proceedings of Machine Learning Research, vol. 108, pp. 3393–3403 (2020)

[13] Xie, Y., Zhang, S., Qing, J., Misener, R., Tsay, C.: BoGrape: Bayesian optimization over graphs with shortest-path encoded. In: International Conference on

Learning Representations (2026)

[14] Cui, J., Yang, B., Hu, X.: Deep bayesian optimization on attributed graphs. Proceedings of the AAAI Conference on Artificial Intelligence **33**(01), 1377–1384 (2019) https://doi.org/10.1609/aaai.v33i01.33011377

[15] Bussemaker, J.H., Bartoli, N., Lefebvre, T., Ciampa, P.D., Nagel, B.: Effectiveness of surrogate-based optimization algorithms for system architecture optimization. In: AIAA AVIATION 2021 Forum (2021). https://doi.org/10.2514/6.2021-3095

[16] Jones, D.R., Schonlau, M., Welch, W.J.: Efficient global optimization of expensive black-box functions. Journal of Global Optimization **13**(4), 455–492 (1998) https://doi.org/10.1023/A:1008306431147

[17] Frazier, P.I.: Bayesian optimization. In: Gel, E., Ntaimo, L. (eds.) Recent Advances in Optimization and Modeling of Contemporary Problems. INFORMS TutORials in Operations Research, pp. 255–278. INFORMS, Catonsville, MD (2018). https://doi.org/10.1287/educ.2018.0188

[18] Rasmussen, C.E., Williams, C.K.I.: Gaussian Processes for Machine Learning. MIT Press, Cambridge, MA (2006). https://doi.org/10.7551/mitpress/3206.001.0001

[19] Berlinet, A., Thomas-Agnan, C.: Reproducing Kernel Hilbert Spaces in Probability and Statistics, 1st edn. Springer, New York, NY (2004). https://doi.org/10.1007/978-1-4419-9096-9

[20] Schölkopf, B., Herbrich, R., Smola, A.J.: A generalized representer theorem. In: Helmbold, D., Williamson, B. (eds.) Computational Learning Theory. Lecture Notes in Computer Science, vol. 2111, pp. 416–426. Springer, Berlin, Heidelberg (2001). https://doi.org/10.1007/3-540-44581-1_27

[21] Apaza, G., Selva, D.: Automatic composition of encoding scheme and search operators in system architecture optimization. In: Volume 2: 41st Computers and Information in Engineering Conference (CIE). American Society of Mechanical Engineers, Virtual (2021). https://doi.org/10.1115/DETC2021-71399

[22] Wilson, A.G., Hu, Z., Salakhutdinov, R., Xing, E.P.: Deep kernel learning. In: Gretton, A., Robert, C.C. (eds.) Proceedings of the 19th International Conference on Artificial Intelligence and Statistics. Proceedings of Machine Learning Research, vol. 51, pp. 370–378. PMLR, Cadiz, Spain (2016)

[23] Cappart, Q., Chételat, D., Khalil, E.B., Lodi, A., Morris, C., Veličković, P.: Combinatorial optimization and reasoning with graph neural networks. In: Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, pp. 4348–4355 (2021). https://doi.org/10.24963/ijcai.2021/595

[24] Darvariu, V.-A., Hailes, S., Musolesi, M.: Graph reinforcement learning for combinatorial optimization: A survey and unifying perspective. Transactions on Machine Learning Research (2024)

[25] Wang-Henderson, M., Soyuer, B., Kassraie, P., Krause, A., Bogunovic, I.: Graph neural bayesian optimization for virtual screening. In: NeurIPS 2023 Workshop on New Frontiers of AI for Drug Discovery and Development (2023). https://openreview.net/forum?id=3OW9rqKGKy

[26] Cui, J., Tan, Q., Zhang, C., Yang, B.: A novel framework of graph bayesian optimization and its applications to real-world network analysis. Expert Systems with Applications **170**, 114524 (2021) https://doi.org/10.1016/j.eswa.2020.114524

[27] Kandasamy, K., Neiswanger, W., Schneider, J., Póczos, B., Xing, E.P.: Neural architecture search with bayesian optimisation and optimal transport. In: Advances in Neural Information Processing Systems, vol. 31, pp. 2020–2029 (2018)

[28] Nikolentzos, G., Siglidis, G., Vazirgiannis, M.: Graph kernels: A survey. Journal of Artificial Intelligence Research **72**, 943–1027 (2021) https://doi.org/10.1613/jair.1.13225

[29] Kriege, N.M., Johansson, F.D., Morris, C.: A survey on graph kernels. Applied Network Science **5**(1), 6 (2020) https://doi.org/10.1007/s41109-019-0195-3

[30] Gönen, M., Alpaydin, E.: Multiple kernel learning algorithms. Journal of Machine Learning Research **12**(64), 2211–2268 (2011)

[31] Shervashidze, N., Schweitzer, P., Leeuwen, E.J., Mehlhorn, K., Borgwardt, K.M.: Weisfeiler-lehman graph kernels. Journal of Machine Learning Research **12**(77), 2539–2561 (2011)

[32] Kriege, N.M., Giscard, P.-L., Wilson, R.C.: On valid optimal assignment kernels and applications to graph classification. In: Advances in Neural Information Processing Systems, vol. 29, pp. 1615–1623 (2016)

[33] Kriege, N.M.: Deep weisfeiler–lehman assignment kernels via multiple kernel learning. In: Proceedings of the 27th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN), pp. 107–112. i6doc.com, Bruges, Belgium (2019)

[34] Snoek, J., Rippel, O., Swersky, K., Kiros, R., Satish, N., Sundaram, N., Patwary, M.M.A., Prabhat, Adams, R.P.: Scalable bayesian optimization using deep neural networks. In: Proceedings of the 32nd International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 37, pp. 2171–2180 (2015)

[35] Ying, C., Klein, A., Christiansen, E., Real, E., Murphy, K., Hutter, F.: NAS-bench-101: Towards reproducible neural architecture search. In: Chaudhuri, K., Salakhutdinov, R. (eds.) Proceedings of the 36th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 97, pp. 7105–7114 (2019)

[36] Dong, X., Yang, Y.: NAS-bench-201: Extending the scope of reproducible neural architecture search. In: International Conference on Learning Representations (2020)

[37] Bussemaker, J.H.: SBArchOpt: Surrogate-based architecture optimization. Journal of Open Source Software **8**(89), 5564 (2023) https://doi.org/10.21105/joss.05564

[38] Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation **6**(2), 182–197 (2002) https://doi.org/10.1109/4235.996017

[39] Bussemaker, J.H., Saves, P., Bartoli, N., Lefebvre, T., Nagel, B.: Surrogate-based optimization of system architectures subject to hidden constraints. In: AIAA AVIATION 2024 FORUM, Las Vegas, NV, USA (2024). https://doi.org/10.2514/6.2024-4401

[40] Hofmann, T., Schölkopf, B., Smola, A.J.: Kernel methods in machine learning. The Annals of Statistics **36**(3), 1171–1220 (2008) https://doi.org/10.1214/009053607000000677

[41] Saves, P., Lafage, R., Bartoli, N., Diouane, Y., Bussemaker, J., Lefebvre, T., Hwang, J.T., Morlier, J., Martins, J.R.R.A.: SMT 2.0: A surrogate modeling toolbox with a focus on hierarchical and mixed variables gaussian processes. Advances in Engineering Software **188**, 103571 (2024) https://doi.org/10.1016/j.advengsoft.2023.103571

[42] Powell, M.J.D.: A direct search optimization method that models the objective and constraint functions by linear interpolation. In: Gomez, S., Hennart, J.-P. (eds.) Advances in Optimization and Numerical Analysis, pp. 51–67. Springer, Dordrecht (1994). https://doi.org/10.1007/978-94-015-8330-5_4

[43] Bussemaker, J.H., De Smedt, T., La Rocca, G., Ciampa, P.D., Nagel, B.: System architecture optimization: An open source multidisciplinary aircraft jet engine architecting problem. In: AIAA AVIATION 2021 Forum (2021). https://doi.org/10.2514/6.2021-3078

[44] García Sánchez, R.: Adaptation of an MDO platform for system architecture optimization. Master's thesis, Delft University of Technology, Delft, The Netherlands (January 2024). https://resolver.tudelft.nl/uuid:f2ad48f1-b960-4d36-a9fe-20486d3f0645

[45] Nikolentzos, G., Vazirgiannis, M.: Enhancing graph kernels via successive embeddings. In: Proceedings of the 27th ACM International Conference on Information and Knowledge Management. CIKM '18, pp. 1583–1586. Association for Computing Machinery, New York, NY, USA (2018). https://doi.org/10.1145/3269206.3269289

[46] Bouhlel, M.A., Bartoli, N., Otsmane, A., Morlier, J.: Improving kriging surrogates of high-dimensional design models by partial least squares dimension reduction. Structural and Multidisciplinary Optimization **53**(5), 935–952 (2016) https://doi.org/10.1007/s00158-015-1395-9

[47] Solis, F.J., Wets, R.J.-B.: Minimization by random search techniques. Mathematics of Operations Research **6**(1), 19–30 (1981) https://doi.org/10.1287/moor.6.1.19

[48] Zhu, Z., La Rocca, G., Tooren, M.J.L.: A methodology to enable automatic 3d routing of aircraft electrical wiring interconnection system. CEAS Aeronautical Journal **8**(2), 287–302 (2017) https://doi.org/10.1007/s13272-017-0238-3

[49] Titsias, M.: Variational learning of inducing variables in sparse gaussian processes. In: Dyk, D., Welling, M. (eds.) Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics. Proceedings of Machine Learning Research, vol. 5, pp. 567–574 (2009)

[50] Williams, C.K.I., Seeger, M.: Using the nyström method to speed up kernel machines. In: Advances in Neural Information Processing Systems, vol. 13, pp. 682–688 (2001)

[51] Feragen, A., Kasenburg, N., Petersen, J., Bruijne, M., Borgwardt, K.M.: Scalable kernels for graphs with continuous attributes. In: Advances in Neural Information Processing Systems, vol. 26, pp. 216–224 (2013)

[52] Neumann, M., Garnett, R., Bauckhage, C., Kersting, K.: Propagation kernels: Efficient graph kernels from propagated information. Machine Learning **102**(2), 209–245 (2016) https://doi.org/10.1007/s10994-015-5517-9

[53] Morris, C., Kriege, N.M., Kersting, K., Mutzel, P.: Faster kernels for graphs with continuous attributes via hashing. In: 2016 IEEE 16th International Conference on Data Mining (ICDM), pp. 1095–1100 (2016). https://doi.org/10.1109/ICDM.2016.0142

3

Discussion and Conclusions

This thesis investigated whether graph-based surrogate models can improve the sample efficiency of System Architecture Optimization (SAO). The scientific article in chapter 2 examined the application of graph-kernel Gaussian-process surrogates to SAO, developing specialized kernels for architecture graphs and evaluating their impact on sample efficiency against established methods and GNN-based alternatives. Building on that analysis and the supplementary observations from Appendix B, this chapter returns to the overarching research question and supporting subquestions that motivated this thesis, as stated in chapter 1, and considers how the findings address them.

3.1. Main findings

Main research question
To what extent can graph-based surrogate models improve the sample efficiency of System Architecture Optimization compared with existing vector-based approaches?

The results support a conditional advantage. The strongest graph-based method overall, MG-LD-WLOA, achieved the lowest mean cumulative regret and final normalized hypervolume error on MDGNC and MDGNC-EF. It reached the final mean performance of BLR on MDGNC using approximately 35% fewer evaluations (130 instead of 200), and that of KPLS on MDGNC-EF using approximately 64% fewer evaluations (90 instead of 250), providing direct evidence of improved sample efficiency in these settings. The GNN-based DGBO also showed faster convergence on these problems. Observing improvements with two substantially different graph-based surrogates suggests that exploiting architectural structure may contribute to these gains. However, this advantage was not consistent across all four benchmarks. On ENGINE, the method achieved the lowest mean final gap ratio but not the lowest cumulative regret, while on ROCKET, encoding-based methods performed better. Thus, the experiments support graph-based modeling as a useful extension to established SAO methods, without establishing it as a generally superior replacement.

Subquestion 1: dependence on problem characteristics
How does the relative sample efficiency of graph-based and vector-based surrogate models vary with the characteristics of the SAO problem, particularly the extent to which its objectives depend on connectivity and other structural patterns?

Graph-based surrogates showed the clearest improvements over encoding-based methods on MDGNC and MDGNC-EF problems, where reliability depends on structural redundancy patterns spanning across many nodes and edges. This aligns with the observation that design-vector encodings do not explicitly preserve similarities between structural patterns spanning multiple connection choices. Encoding-based surrogates must instead infer their relevance from the limited evaluated samples, whereas graph-based surrogates can incorporate such similarities directly. The structural generalization study supports this interpretation, as the graph-based surrogates predicted more accurately for architectures that were structurally equivalent to training examples but encoded differently, and for structurally non-equivalent architectures with varying degrees of similarity to previously observed designs.

The results were more mixed on ENGINE and ROCKET. On ENGINE, MG-LD-WLOA achieved a better final gap ratio but higher regret than the strongest encoding-based methods, while on ROCKET it was outperformed by them. Both problems place greater emphasis on hierarchical selection and sizing. Encoding-based surrogates can learn the relevance of individual decisions, whereas the WL component of MG-LD-WLOA weights all patterns at a given depth equally. This rigidity may limit its ability to emphasize influential component choices. Other graph-based approaches offered no consistent advantage either, with the GNN surrogates performing considerably worse. Overall, the results suggest that graph-based surrogates are most beneficial when the objective depends strongly on graph structure, while their advantage is less clear for problems dominated by hierarchical or sizing decisions. However, since MDGNC and MDGNC-EF belong to the same benchmark family, this interpretation requires further validation on additional problems.

Subquestion 2: graph-kernel versus graph-neural surrogates
How do graph-kernel Gaussian-process surrogates compare with graph-neural surrogate models in terms of sample efficiency across SAO problems?

In our experiments, MG-LD-WLOA was more sample-efficient than both considered graph-neural surrogates across the studied SAO problems. Admittedly, the kernel-based method was the central focus of this thesis and received substantially more development and investigation, so these results do not exclude graph-neural surrogates as a viable direction. DGBO, in particular, remained reasonably competitive on the structurally driven benchmarks, although its performance deteriorated substantially on ENGINE and ROCKET.

Arch2Vec performed poorly overall, and its downstream DNGO regression head was particularly fragile. Longer training could reduce the estimated uncertainty without improving held-out predictive accuracy, while changes in representation quality did not translate consistently into better optimization performance. Together with the large variation in DGBO performance across problems, these observations point to a broader difficulty with graph-neural surrogates in the small-data SAO setting. They introduce several interacting choices, including network architecture, learning rate, training duration, and representation dimensionality, whose appropriate settings may depend strongly on the problem and modeled response. Under the small evaluation budgets typical of SAO, extensive hyperparameter sweeps or validation-based tuning are generally not practical. The kernel-based approach developed here was deliberately designed to avoid much of this training complexity by imposing a structured notion of similarity and learning a comparatively small set of kernel parameters directly from the available observations. While the present results do not establish that graph-neural surrogates are inherently inferior, if anything, they point to practical difficulties in making such methods comparably robust under small evaluation budgets.

3.2. Outlook

The results obtained in this work demonstrate that graph-based surrogates can improve sample efficiency on the studied connectivity-dependent SAO problems. The observed gains suggest that incorporating structural similarities directly into the surrogate may also benefit other problems with similar architectural characteristics. While these findings are promising, they represent an initial step towards establishing graph-based surrogates as a practical approach to SAO. Their applicability to larger and more complex engineering systems remains to be demonstrated. Such systems may introduce greater computational demands and more varied relationships between architecture and performance, exposing limitations in both the scalability of the approach and the kernels' ability to capture the relevant structural similarities.

Broader validation on independent engineering problems with realistic evaluations is therefore a priority. Beyond testing whether the observed gains persist, this would help identify which problem characteristics favor graph-based modeling and where the proposed approach requires further development. The directions concerning scalability and kernel expressiveness discussed in chapter 2 provide starting points for addressing these challenges. Together, broader validation and methodological development would build on the evidence provided here and help establish what is needed to bring graph-based surrogates into wider engineering practice.

Final Remarks

Impact statement

This work aims to improve the efficiency of system architecture optimization for engineering applications, with particular relevance to aerospace systems. By reducing the number of expensive evaluations needed to identify promising architectures, such methods could support the design of systems with improved performance, reliability, or resource efficiency. The resulting benefits depend on the objectives and constraints used to guide the optimization.

These methods also have dual-use potential. Although applicable to civilian engineering, they could also support the design and optimization of military systems. Their broader societal impact therefore depends on the applications in which they are used and the goals they serve.

Use of generative AI

Generative AI tools were used during this thesis for the following purposes:

- **Literature exploration and brainstorming:** ChatGPT and Claude were used to help identify potentially relevant literature and brainstorm research ideas.
- **Coding assistance:** Codex and Claude Code were used to assist with general implementation, debugging, and preparing plots.
- **Writing assistance:** ChatGPT and Claude were used for language editing, paragraph drafting and revision, and feedback on the clarity and organization of the text.

Acknowledgements

Working on this thesis and studying at TU Delft have been deeply rewarding experiences. I am grateful for everything I have learned and the opportunities I have had, but above all, for the people who have made these years so special. I would like to take this opportunity to thank them.

I would like to thank Jasper for always being willing to help with my day-to-day questions, and for his kindness and warm welcome in Hamburg. You made my time there so much more enjoyable. Thank you also for the plane ride—it was a wonderful experience.

I am grateful to Elvin for taking me on for this project despite a tight schedule, for the time and care he devoted to reviewing my work, and for being available to discuss whenever I needed guidance.

To my friends, thank you for your kindness and companionship throughout my years in Delft. You made this time so much better, and I could not have wished for better people to share it with.

Finally, I want to thank my mom, my dad, and my brother for always believing in me, supporting me through the ups and downs, and being there for me. I cannot fully express how grateful I am for everything you have done for me over the years. I would not be where I am today without you.

A

Extended Background

This chapter introduces the background required to understand the remainder of this thesis. It extends the background already provided in the scientific article in chapter 2, resulting in some unavoidable overlap, but presents the relevant concepts more gradually and with less assumed prior knowledge. It serves as an accessible technical reference and does not repeat the article's research argument or related-work review. It covers System Architecture Optimization, graph-based architecture representations, surrogate-based optimization, Gaussian processes and kernels, and graph-neural surrogate models.

A.1. System Architecture Optimization

System Architecture Optimization (SAO) aims to identify high-performing system architectures that belong to a formally defined architecture design space, by formulating architecting as a numerical optimization problem [1, 5]. It therfore combines automated generation of architecture instances with quantitative evaluation of their performance. The resulting objective and constraint values are used to guide the optimization algorithm towards high-performing solutions.

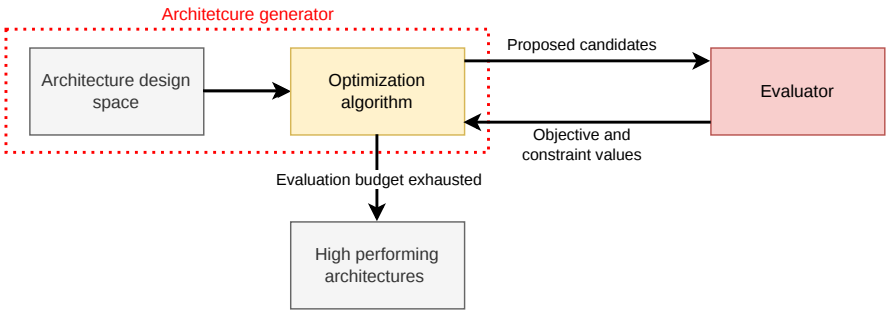


Figure A.1: Overview of the System Architecture Optimization process. The architecture generator proposes candidate architectures, while the evaluator returns their objective and constraint values.

Conceptually, SAO can be divided into two main components: the *architecture generator* and the *architecture evaluator*, as illustrated in Figure A.1 [1]. The architecture generator is responsible for proposing promising candidate architectures. To that end it requires a formal model of the architecture design space, which defines the available choices, design parameters, and their constraints and dependencies. This model specifies what constitutes a valid architecture and enables concrete architecture instances to be generated. An optimization algorithm then searches the specified design space, using feedback from previous evaluations to determine which candidate architectures should be evaluated next.

The architecture evaluator determines the performance of each proposed candidate, often through physics-based or multidisciplinary simulation workflows, and returns the calculates objective and constraint value. From the perspective of the optimizer, these workflows are typically black-box and do not provide gradients with respect to any architectural decisions. This is why global, gradient-free

optimization methods are used in SAO [1, 5]. The process is repeated until the available evaluation budget is exhausted, after which the best architecture or set of architectures found is returned.

The focus in this work is on the architecture generator and primarily on the optimization algorithm. However, understanding how the architecture design space and its individual instances are represented is a prerequisite for understanding the optimization methods developed in this work. Therefore, the following section introduces the graph-based design-space model used in SAO and explains how it can be used to generates graphs that represent specific architectures.

A.2. Modeling SAO Problems

This section covers the first of the two tasks of the *architecture generator*, namely representing the feasible architecture design space in a machine-readable form, together with a corresponding representation of the concrete architecture instances that belong to it. In particular, this work adopts the graph-based representations introduced by Bussemaker [1]. The section first defines the architecture design space, then presents the graph-based representations used to model both the design space and the concrete architecture instances generated from it.

A.2.1. Architecture Design Space and Architectural Choices

The architecture generator operates witin an architecture design space, that is, the set of all architectures that can be generated for a given problem under the adopted modeling assumptions [1, 4]. Bussemaker introduces a graph-based modeling framework for this purpose. It is important to first understand key properties the underlying architecture design spaces have, and which properties any suitable representation must capture. The most important of these are outlined below:

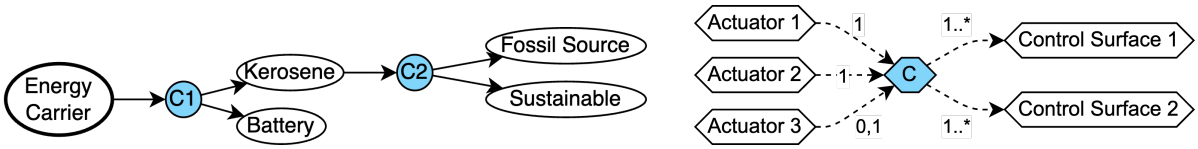
- **Defined by Architectural Choices and Constraints:** An architecture design space specifies the architectural choices that can be made, the available options for each choice, and constraints on which combinations of options are permitted [4, 1]. For example, a propulsion-system design space may allow thrust to be generated by either a turbofan or a propeller and may offer several possible energy carriers. An architectural constraint can exclude an incompatible combination, such as a battery-powered turbofan [1]. A feasible architecture is obtained by making a concrete set of choices that satisfy these constraints.
- **Architectural Choices are Mixed-Discrete:** SAO design spaces are naturally mixed-discrete, since architecture design combines discrete architectural choices with continuous architecture-specific design parameters [1]. For instance, in the jet engine architecting problem of [13] some choices are categorical, such as whether a fan is included, some are integer-valued, such as the number of shafts, and others are continuous, such as the bypass ratio and fan pressure ratio.
- **Architectural Choices are Hierarchically Dependent:** Another central property of SAO design spaces is that the choices are organized hierarchically [1]. Some architectural decisions determine whether other choices and parameters are present at all. For instance, in the aforementioned jet engine architecting problem [13], the decision whether to include a fan determines if the architecture is a turbojet or a turbofan. If a fan is included, additional choices such as the bypass ratio and fan pressure ratio become active, whereas they remain inactive otherwise [1].

In addition to the key properties discussed above, SAO design spaces may involve further constraints. For instance some combinations of design and parameter choices may be infeasible even when this is not captured by simple downstream activation, so broader feasibility relations may need to be represented explicitly [1]. Moreover, not all constraints can be identified from the architecture representation alone. Some only become apparent during simulation or evaluation rather than being directly encoded in the design space model. In short, design spaces are heterogeneous, hierarchical, and mixed-discrete, which makes optimization over those spaces a complex task, since two candidate architectures do not necessarily share the same active variables or bounds [7].

A.2.2. Graph Representations

Having outlined the key properties of architecture design spaces, the next step is to consider how they can be represented as graphs, and how specific architecture instances can be obtained from them. The Design Space Graph (DSG) is introduced first, followed by its architecting-specific extension, namely

the Architecture Design Space Graph (ADSG). This is followed by an algorithm for resolving specific architecture instances from these graph-models of the design space. Finally, vector-based encodings of instances and their limitations are covered.



(a) Example of a DSG with two unresolved selection choices, denoted by vertices C_1 and C_2.

(b) Example of a connection choice node C and the edges defining feasible component connections.

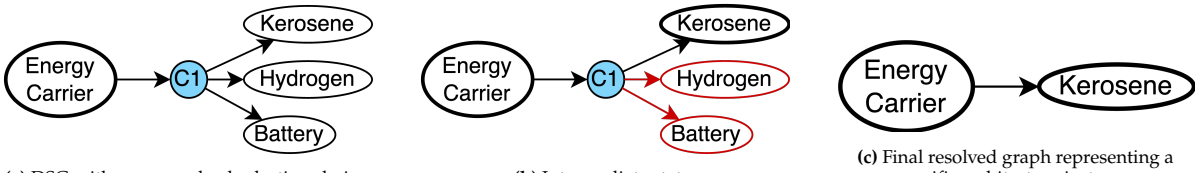
Figure A.2: Examples of selection and connection choices in the DSG. Reproduced from [1].

Design Space Graph To represent the underlying structural scaffolding of the design space, Bussemaker introduces the Design Space Graph (DSG), a directed graph formalism [1].

To capture the hierarchical dependencies that arise in architecture design spaces, the DSG defines a mechanism for modeling selections. Three primitives define its structure: *start nodes* anchor the fixed portion of the graph; *derivation edges* encode implication — the presence of one element triggers another; and *selection choice nodes* model discrete architectural decisions, where exactly one option from a set can be chosen. An example of such a selection choice is shown in Figure A.2a. Crucially, one can see that choosing the patter power source at node $C1$ would eliminate an entire branch of the graph, rendering the downstream choices $C2$ unreachable.

The DSG also includes a mechanism for choosing how the remaining architectural elements should be connected. This part is defined through *connector nodes* and *connection choice nodes*, which, alongside attributed edges, specify feasible connection patterns between source and target elements. An example of a connection choice is shown in Figure A.2, where one can see the choice node C and its incident edges annotated with patterns representing the admissible connection multiplicities.

To support the remaining properties of architecture design spaces, the DSG also includes constructs such as connector grouping nodes, exclusion edges, and flags for allowing parallel connections [1]. Nevertheless, the two mechanisms described above are enough to understand some relevant points later. In short, the DSG is a graph formalism that uses various node and edge types to represent the hierarchical structure of feasible architectural choices and their compatible interconnections, which exactly defines the set of all feasible graphs that can be constructed.



(a) DSG with an unresolved selection choice.

(b) Intermediate state.

(c) Final resolved graph representing a specific architecture instance.

Figure A.3: Example resolution of a selection choice into a concrete graph instance. Reproduced from [1].

Choice Resolution To generate concrete graphs from the DSG, Bussemaker introduces the following choice resolution mechanism [1]. First, selection choices are resolved by confirming the chosen option nodes and their dependencies, while pruning unselected alternatives and their derived subgraphs. An example is shown in Figure A.3. Once all the active architectural elements are determined, the connection choices are resolved. This involves removing the abstract choice nodes (like node C in Figure A.2b) and instantiating explicit connection edges between the active source and target elements, strictly adhering to all defined connection multiplicity constraints. Importantly, whereas selection choices resolve to a single, mutually exclusive option, connection choice resolution supports complex mappings, such as one-to-many or many-to-many connections between the involved nodes. For instance, in Figure A.2b a single control surface can be connected to two actuators. Furthermore, under specific multiplicity constraints, parallel edges can be instantiated between the exact same source and target pair, which could be used to represent properties such as redundancy and capacity.

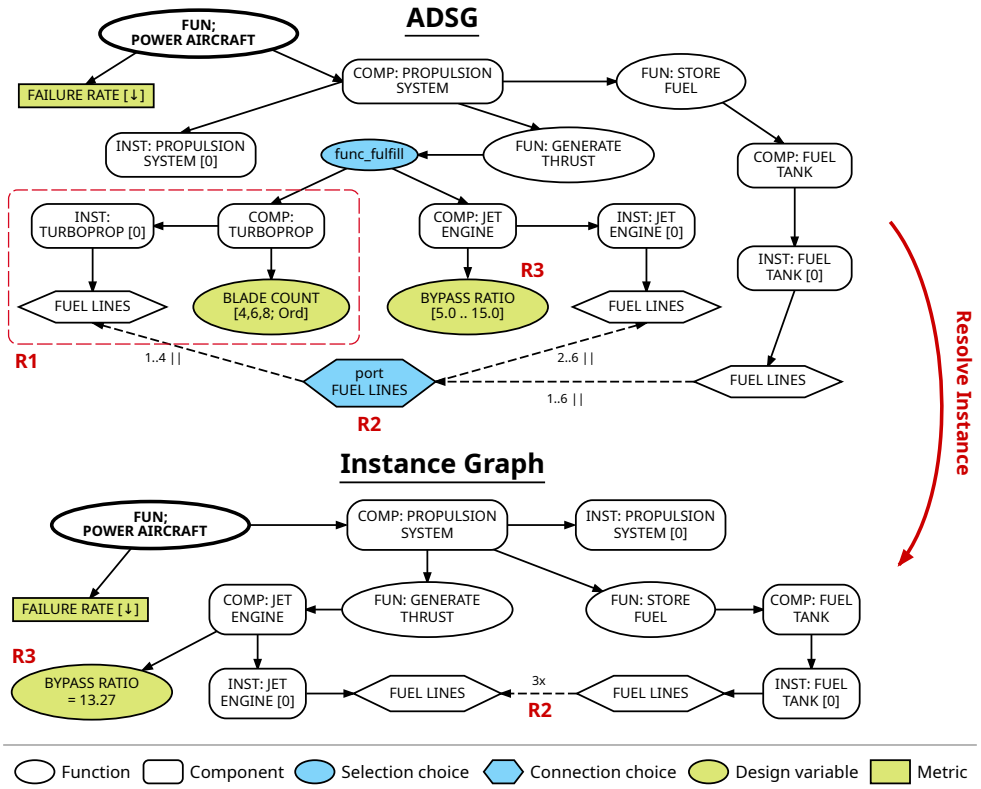


Figure A.4: Example ADSG (top) and the instance graph (bottom) obtained by resolving all of the choices. The ADSG models the propulsion system of an aircraft. Three resolutions are highlighted. **R1** resolves the selection choice `func_fulfill` by selecting `COMP: JET ENGINE` to fulfill `FUN: GENERATE THRUST`, pruning the alternative `COMP: TURBOPROP` together with its derived subgraph (dashed red box). **R2** resolves the connection choice on the `FUEL LINES` ports, whose source and target cardinalities (1..4 and 2..6) admit several valid connection patterns; here three connections are established. **R3** assigns a value to the continuous design variable `BYPASS RATIO`. The resulting instance graph describes one architecture candidate, from which the metric `FAILURE RATE` can be evaluated.

Architecture Design Space Graph The Architecture Design Space Graph (ADSG) introduces a layer on top of the DSG that extends its generic choice and constraint mechanisms with node types and rules specific to system architecting tasks [1]. As illustrated in the upper part of Figure A.4, an ADSG can formally represent a complete System Architecture Optimization (SAO) problem by defining the architecture design space and including additional problem-formulation nodes. The key concepts that the ADSG introduces are the following:

- **Function and component nodes:** ADSG provides nodes for representing high-level system functions, such as `FUN: PROVIDE_THRUST` and `FUN: STORE_FUEL`. While `STORE_FUEL` directly derives the `FUEL_TANK` component, `PROVIDE_THRUST` contains a selection choice (`func_fulfill`, in blue) between the alternative `TURBOPROP` and `JET_ENGINE` components.
- **Component connection choices via port nodes:** The ADSG specializes the DSG's generic connector nodes as component ports, through which connection choices between specific components are defined. The `port_FUEL_LINES` connection choice (in blue) defines valid connections between the active fuel-tank output port and the corresponding engine input ports, while enforcing connection multiplicity constraints such as `1..6` (i.e. between 1 and 6) and `2..6`.
- **Design variable nodes:** The ADSG represents component parameters with design-variable nodes, which may be either continuous or discrete. For example, selecting the `JET_ENGINE` activates the continuous design variable `BYPASS_RATIO [5.0 .. 15.0]` (in green), whereas selecting the `TURBOPROP` activates the ordinal variable `BLADE_COUNT [4,6,8; Ord]`.
- **Metric nodes:** To represent the evaluation outputs used to formulate the SAO problem, metric nodes are used. Since SAO aims to identify optimal architectures while satisfying constraints, these nodes may represent minimization/maximization objectives (e.g. `FAILURE RATE`) or inequality constraints. Once an instance is evaluated, they are set to the obtained values.

While the ADSG also includes additional elements—such as external boundary, connector grouping nodes, and non-fulfillment nodes—that exist to facilitate modeling more complex constraint structures and architectural logic, the key node types mentioned above capture the core idea well.

By adding explicit architecting semantics to the otherwise abstract DSG, the ADSG becomes a complete formal model that captures not only the feasible structural alternatives and their compatible connections, but also the associated architectural semantics, specific parameters, objectives, and constraints necessary to later translate these abstract choices into higher fidelity models. This adds the critical level of detail required for applying physics-based simulations and multidisciplinary evaluation models to each generated instance, which in turn facilitates fully automated SAO loops.

Finally, Figure A.4 also demonstrates how applying the DSG resolution mechanism described previously produces a concrete architecture instance. In **R1**, the selection choice is resolved in favour of `COMP: JET ENGINE`, causing the alternative `COMP: TURBOPROP` branch and the elements derived exclusively from it to be pruned. In **R2**, the connection choice is replaced by three explicit parallel `FUEL LINES` edges between the fuel-tank and jet-engine instances, consistent with the specified multiplicity constraints. Finally, in **R3**, a value is assigned to the active `BYPASS RATIO` design variable.

Summary In Bussemaker's framework [1, 5], the resolved instance graphs are encoded into design vectors over which the search is performed. As the objective of this work is to operate on those graphs directly, they are of primary interest. Importantly, the resulting architecture-instance graphs are heterogeneous and variable-sized. Depending on the choices made, two instances may contain different node types and attributes with unrelated meanings. For example, one may contain a `JET_ENGINE` with a `BYPASS_RATIO`, while another contains a `TURBOPROP` with a `BLADE_COUNT`, preventing direct comparison and posing an additional challenge. Their edges may also have different multiplicities. Finally, realistic instance graphs can be substantially larger than the simplified examples shown here, and their size may vary considerably across architecture instances.

A.3. Optimization in SAO

SAO aims to identify feasible architectures that perform well with respect to one or more objectives while satisfying the specified constraints. Architecture design spaces can contain a large number of feasible alternatives, making exhaustive enumeration impractical. In realistic applications, candidate architectures are often assessed using expensive multidisciplinary simulations that effectively act as black-box evaluators, returning objective and constraint values without derivatives with respect to the sizing parameters. Together with the presence of discrete architectural choices, this means that conventional gradient-based methods are not directly applicable.

Given these characteristics, relies on either direct population-based evolutionary optimization or surrogate-based optimization (SBO) [5, 1]. Evolutionary algorithms are effective, but typically require many architecture evaluations. When each evaluation is expensive, SBO can reduce the overall optimization time by using an inexpensive predictive model to select which candidates warrant evaluation. Therefore, it plays a central role in the optimization methodology adopted in this thesis. Before introducing SBO in more detail, we first formalize the optimization problem encountered in SAO.

A.3.1. Optimization Problem Formulation

Formally, SAO seeks architecture instances that optimize one or more objectives over a feasible design space. Let $\mathcal{X}$ denote this design space, and let $x \in \mathcal{X}$ denote a candidate architecture instance. Then a SAO problem can be defined as

$$\begin{aligned} \min_{x \in \mathcal{X}} \quad & f_m(x) && m = 1, \dots, M \\ \text{subject to} \quad & g_k(x) \leq 0 && k = 1, \dots, K \\ & c_l(x) = 0 && l = 1, \dots, L \\ & h(x) = 0. \end{aligned}$$

Here, f_m are the objectives, g_k are ordinary inequality constraints, c_l are value or configuration constraints imposed by the architecture design space, and h denotes hidden constraints. The objectives quantify architecture performance with respect to the selected criteria, such as fuel burn or weight, while inequality constraints typically encode some physical or operational requirements. By contrast,

value constraints determine whether a design vector corresponds to a valid architecture at all. They correspond precisely to the structural constraints enforced by the ADSG, and can therefore be handled through the resolution procedure described in subsection A.2.2. Hidden constraints correspond to cases where the underlying simulation or analysis fails [5].

Although single-objective formulations are possible, SAO problems are often multi-objective, since architectural decisions usually involve trade-offs between competing criteria [1, 5]. In that case, the goal is not a single optimum, but the Pareto set of non-dominated designs.

A.3.2. Surrogate-Based Optimization

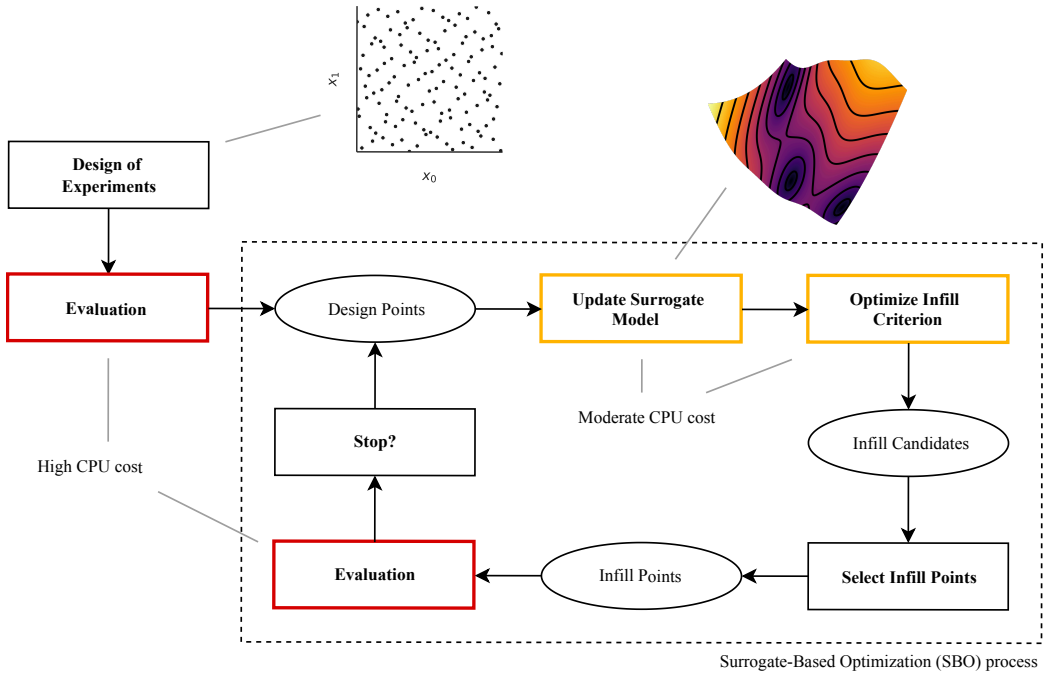


Figure A.5: Surrogate-Based Optimization loop. Adopted from [14].

Surrogate-based optimization is a class of methods that use a computationally inexpensive predictive model to identify promising candidates for evaluation, allowing the available budget of expensive evaluations to be used more selectively [5, 15]. A typical SBO loop is illustrated in Figure A.5.

The process begins by sampling an initial set of candidate designs, known as the design of experiments (DoE). These candidates are evaluated using the expensive high-fidelity model, and the resulting data are used to fit the initial surrogate. For any candidate design, this surrogate is used to predict its performance. This prediction is then passed to an infill criterion, which converts it into a score representing how promising the candidate is. Compared to the expensive physics-based evaluations, this is very cheap to compute. Therefore, an auxiliary optimizer, such as a genetic algorithm, can cheaply search for candidates that perform best according to this criterion. After a number of iterations is performed by the auxiliary search algorithm, the best identified candidates are again evaluated with the expensive high-fidelity model, added to the available set of evaluated design points, and used to update the surrogate model with more data. This procedure is repeated until the prescribed evaluation budget is exhausted or another stopping criterion is reached.

The main motivation for this procedure is sample efficiency. In realistic SAO problems, the computational bottleneck is usually not the optimization logic itself, but the repeated evaluation of candidate architectures through expensive simulation or analysis workflows [1, 5]. By using a surrogate to identify informative and promising evaluation points, SBO can reduce the number of true function evaluations needed to obtain good solutions. This means however, that sample efficiency and quality of the surrogate model are critical factors for overall optimization performance.

A.3.3. Bayesian Optimization

A central surrogate-based approach for expensive SAO problems is *Bayesian optimization* (BO) [1, 5]. It is an uncertainty-aware variant of SBO that relies on a probabilistic surrogate providing both predictions and their corresponding uncertainty estimates. In BO, the infill criterion is commonly referred to as an *acquisition function*. It scores candidate designs while accounting for both their predicted performance and the uncertainty of that prediction, balancing *exploitation*, which favors candidates proven to perform well, and *exploration*, which favors uncertain regions that may contain better solutions [16].

A widely used infill criterion is Expected Improvement (EI), which illustrates the benefit of such a probabilistic approach well [6]. Consider a single-objective minimization problem with Gaussian predictive distributions. Let $\mu_n(\mathbf{x})$ and $\sigma_n(\mathbf{x})$ denote the predictive mean and standard deviation for a candidate $\mathbf{x}$ after n evaluations, and let f_{best} denote the lowest objective value observed so far using the expensive evaluator. For $\sigma_n(\mathbf{x}) > 0$, define

$$\Delta_n(\mathbf{x}) = f_{best} - \mu_n(\mathbf{x}), \qquad z_n(\mathbf{x}) = \frac{\Delta_n(\mathbf{x})}{\sigma_n(\mathbf{x})}.$$

Here, $\Delta_n(\mathbf{x})$ measures how much better the predicted objective is than the best observed value, while $z_n(\mathbf{x})$ expresses this difference relative to the predictive uncertainty. The expected improvement is then

$$\mathrm{EI}(\mathbf{x}) = \underbrace{\Delta_n(\mathbf{x})\Phi\left(z_n(\mathbf{x})\right)}_{\text{predicted-improvement}} + \underbrace{\sigma_n(\mathbf{x})\phi\left(z_n(\mathbf{x})\right)}_{\text{uncertainty}},$$

where Φ and ϕ denote the standard normal cumulative distribution and probability density functions, respectively. The first term is positive when the predicted mean is better (lower) than the best observed value. The second accounts for predictive uncertainty, allowing a point to receive positive EI even when its predicted mean is worse than f_{best}, because its true objective could still be better. For a fixed predictive mean, increasing $\sigma_n(\mathbf{x})$ increases EI. When the uncertainty is zero, EI reduces to $\max\{f_{best} - \mu_n(\mathbf{x}), 0\}$. Although the objective is minimized, EI is maximized, as candidates are preferred when they are predicted to improve upon the current best solution or have sufficient uncertainty to plausibly do so.

A.4. Surrogate Modeling

This section covers the relevant surrogate modeling approaches that can operate directly on architecture graphs. It first covers Gaussian Processes, followed by a discussion of kernels and graph kernels, as they are the key modeling component of the GP. Finally, graph neural networks (GNN) are introduced.

A.4.1. Gaussian Process Regression

Gaussian process regression is a statistical model widely adopted in Bayesian optimization, due to its principled uncertainty estimates and sample efficiency [17]. Moreover, it's also adopted in Bussemaker's foundational SAO framework [1, 5]. To understand how they work, one has to first recognize that a finite set of observations does not uniquely determine the underlying objective function, as many different functions may agree at the evaluated points while disagreeing elsewhere. Predicting these unknown values therefore requires assumptions about how the function generalizes beyond the observations. A GP expresses these assumptions probabilistically through a prior distribution over possible functions, which is then conditioned on the observed data to obtain a posterior distribution.

The GP prior is fully specified by a mean function $m(\cdot)$ and a kernel function $k(\cdot,\cdot)$, both defined over the input domain $\mathcal{X}$:

$$f(\cdot) \sim \mathcal{GP}(m(\cdot), k(\cdot,\cdot)).$$

The mean function specifies the expected function value at each input, while the kernel specifies the covariance between the function values for any pair of inputs:

$$m(x) = \mathbb{E}[f(x)], \qquad k(x,x') = \mathrm{Cov}\left[f(x), f(x')\right].$$

The kernel thereby determines which inputs the GP considers related. In intuitive terms, it encodes the prior assumption that inputs with high kernel similarity tend to have similar function values. An observation at one input therefore provides more information about another input when their kernel similarity is high, whereas a kernel value close to zero implies little prior dependence between them.

More concretely, for any finite set of inputs $X = \{x_1, \ldots, x_n\}$, the GP defines the following n-dimensional Gaussian distribution:

$$\begin{bmatrix} f(x_1) \\ \vdots \\ f(x_n) \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} m(x_1) \\ \vdots \\ m(x_n) \end{bmatrix}, \begin{bmatrix} k(x_1, x_1) & k(x_1, x_2) & \cdots & k(x_1, x_n) \\ \vdots & \vdots & \ddots & \vdots \\ k(x_n, x_1) & k(x_n, x_2) & \cdots & k(x_n, x_n) \end{bmatrix} \right).$$

A draw from this distribution gives one possible set of function values at the selected inputs that is consistent with the prior over functions.

In practice, the observed outputs are commonly standardized by subtracting their sample mean and dividing by their sample standard deviation. This centers the observations around zero, making $m(x) = 0$ a natural prior mean for the standardized objective. Predictions can subsequently be transformed back to the original output scale by multiplying by the sample standard deviation and adding the mean.

Using this zero-mean convention, suppose that the standardized objective values $\mathbf{y} = \begin{bmatrix} f(x_1) \cdots & f(x_n) \end{bmatrix}^\top$ have been observed at the evaluated inputs $X = \{x_1, \ldots, x_n\}$. We now wish to predict the unknown function value $f_* = f(x_*)$ at an unevaluated test point x_*. Define

$$K = \left[k(x_i, x_j) \right]_{i,j=1}^{n}, \qquad \mathbf{k}_* = \begin{bmatrix} k(x_1, x_*) \\ \vdots \\ k(x_n, x_*) \end{bmatrix}, \qquad k_{**} = k(x_*, x_*).$$

Here, K contains the kernel similarities among the evaluated inputs, while $\mathbf{k}_*$ contains the similarities between the test point and each evaluated input. The value k_{**} represents the prior variance at the test point. Under the GP prior, the observed and unknown function values are jointly Gaussian:

$$\begin{bmatrix} \mathbf{y} \\ f_* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K & \mathbf{k}_* \\ \mathbf{k}_*^\top & k_{**} \end{bmatrix} \right).$$

Before conditioning, this distribution represents the possible values of both the evaluated and test points under the prior. Conditioning it on the observed values $\mathbf{y}$ restricts these possibilities to those consistent with the available evaluations. Because the joint distribution is Gaussian, the resulting posterior predictive distribution over function values f_* at test points, has the closed form

$$f_* \mid X, \mathbf{y} \sim \mathcal{N}\left(\mu_*, \sigma_*^2\right), \qquad \mu_* = \mathbf{k}_*^\top K^{-1} \mathbf{y}, \qquad \sigma_*^2 = k_{**} - \mathbf{k}_*^\top K^{-1} \mathbf{k}_*.$$

The posterior mean μ_* is a weighted sum of the observed objective values. The vector $\mathbf{k}_*$ describes how strongly each evaluated input is related to x_*, while K^{-1} adjusts the weights to account for overlapping information: observations at strongly correlated inputs are not treated as independent pieces of evidence.

The posterior variance σ_*^2 measures the uncertainty remaining at x_* after observing the data. It equals the prior variance k_{**} minus the reduction in uncertainty provided by the observations. When the evaluated inputs provide information about x_* through the kernel, uncertainty decreases. Conversely, when x_* is weakly related to the evaluated inputs, the variance remains close to its prior value.

A limitation of GPs is their scaling, as fitting has $O(n^3)$ time complexity where n is the number of training points. This can be restrictive for large datasets, although for low-sample SAO it is acceptable.

A.4.2. Kernel Functions

In a zero-mean Gaussian process, the kernel is the primary modeling component. It defines a notion of similarity between inputs, which the GP uses to inform its predictions. Importantly, graph kernels provide a way to construct a Gaussian process that operates directly on architecture graphs.

Formally, a kernel is a function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, where $\mathcal{X}$ is the input domain. Intuitively, $k(x, x')$ acts as a similarity score between $x, x' \in \mathcal{X}$, with larger values generally meaning that the inputs are more similar. Any valid GP kernel by definition corresponds to an inner product between representations of its inputs in some feature space:

$$k(x, x') = \langle \phi(x), \phi(x') \rangle_{\mathcal{H}}.$$

Here, ϕ maps each input into a Hilbert space $\mathcal{H}$, a complete inner-product space that may have finitely or infinitely many dimensions [18]. For example, consider a two-dimensional input $\mathbf{x} = (x_1, x_2)^\top$ and the feature map

$$\phi(\mathbf{x}) = \begin{bmatrix} x_1^2 & \sqrt{2}\, x_1 x_2 & x_2^2 \end{bmatrix}^\top.$$

This representation contains squared terms and interaction terms between the original coordinates. The dot product between two mapped inputs is

$$k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^\top \phi(\mathbf{x}') = x_1^2 x_1'^2 + 2x_1 x_2 x_1' x_2' + x_2^2 x_2'^2 = (\mathbf{x}^\top \mathbf{x}')^2.$$

Thus, the same value can be obtained implicitly by squaring the dot product of the original inputs, without explicitly constructing their feature vectors. More generally, an explicit feature representation need not even be known, provided that the kernel can be evaluated directly. This becomes particularly useful when the corresponding feature space is high-dimensional or infinite-dimensional.

Additionally, kernels can also be combined. If k_1 and k_2 are valid kernels, then

$$k_{\text{add}}(x, x') = w_1 k_1(x, x') + w_2 k_2(x, x'), \qquad k_{\text{mult}}(x, x') = k_1(x, x')\, k_2(x, x')$$

are also valid kernels for any nonnegative weights $w_1, w_2 \geq 0$ [17]. This allows kernels capturing different properties of the inputs to be combined within one model.

Finally, kernel functions can contain hyperparameters that control the behavior of the GP. For example, the radial basis function (RBF) kernel can be written as

$$k(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\ell^2}\right),$$

where ℓ is the length scale controlling how quickly similarity decreases with distance, and σ_f^2 controls the overall covariance magnitude. These hyperparameters can be learned from the available observations by finding the values under which the observed outputs are most probable according to the GP model.

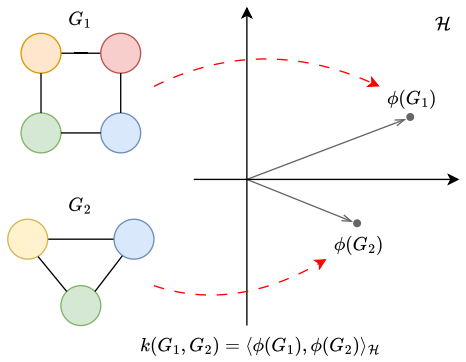


Figure A.6: Feature-space interpretation of a graph kernel. Dashed red arrows map graphs to feature vectors in $\mathcal{H}$, whose inner product gives the kernel value. The feature space is shown schematically in two dimensions. Illustration inspired by [19].

Graph kernels. A graph kernel takes two graphs as inputs and compares them by their similarity in terms of various properties such as node labels, paths, or neighborhood structures [19]. These comparisons correspond to inner products between explicit or implicit graph representations, as illustrated in Figure A.6. This allows graphs of different sizes and structures to be compared within the same feature space. A Gaussian process can therefore use architecture graphs directly as its inputs, with the GP's Gram-matrix entries equivalent to $K_{ij} = k(G_i, G_j)$, instead of relying on encoded design vectors, as current surrogate-based approaches for SAO do [1, 5, 7].

A.4.3. Graph Neural Network Surrogates

An alternative to Gaussian-process surrogates is to learn the objective function using a neural network. Since the candidate architectures considered in this are represented as graphs, Graph Neural Networks (GNNs) provide a way to operate directly on them.

Most GNNs follow the principle of *message passing*. Each node v in a graph is represented by a feature vector h_v, which is iteratively updated using information from its neighbors:

$$h_v^{(\ell+1)} = \text{UPDATE}^{(\ell)}\left(h_v^{(\ell)}, \text{AGG}^{(\ell)}\left(\{h_u^{(\ell)} : u \in \mathcal{N}(v)\}\right)\right).$$

Above, ℓ denotes the GNN layer, $h_v^{(\ell)} \in \mathbb{R}^d$ is the feature vector of node v at layer ℓ, and $\mathcal{N}(v)$ denotes the set of neighboring nodes of v. The operator AGG combines the neighboring node representations into a single message, while UPDATE combines this message with the current representation of node v to produce $h_v^{(\ell+1)}$. At the first layer, $h_v^{(0)}$ corresponds to the original features associated with node v. Different message-passing GNN architectures mainly differ in how the neighboring representations are aggregated and how the resulting representation is updated.

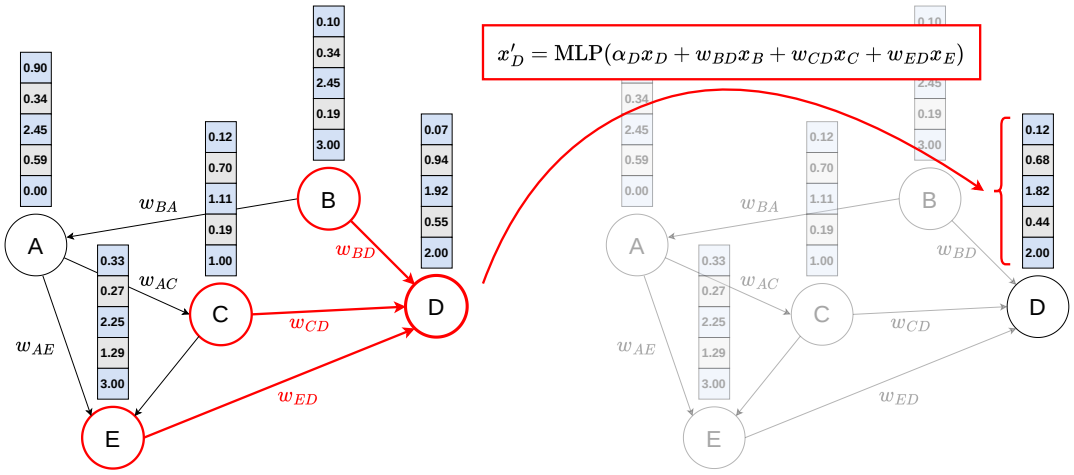


Figure A.7: Caption

An illustrative example of a possible GNN variant is shown in Figure A.7. In this case, node D aggregates weighted representations from its incoming neighbors together with its own representation, followed by a multilayer perceptron (MLP). By stacking several such layers, each with its own learned transformation, information can propagate progressively further through the graph. After L layers, the representation of a node can therefore incorporate information from nodes up to L hops away, assuming each layer aggregates information from its immediate neighbors.

For graph-level prediction, the final node representations are combined through a permutation-invariant readout operation, such as summation or averaging, to obtain a single graph embedding $\mathbf{z}(G)$. To predict objective and constraint values in the SAO setting, one could for instance fit a multi layer perceptron on such an embedding and train it jointly with the rest of the GNN.

Uncertainty in GNNs While GNNs such as the one described above can provide point predictions, they do not directly provide the predictive uncertainty required for Bayesian optimization. Several approaches exist for estimating uncertainty in neural networks, including deep ensembles [20] and Bayesian neural networks [21]. In graph-based BO, however, a common approach is to use the GNN as a learned feature extractor and fit Bayesian linear regression (BLR) to the resulting embeddings [22, 10].

Let $z(G)$ denote the embedding produced by the GNN for graph G. Bayesian linear regression assumes that the objective is a linear function of this embedding,

$$f(G) = w^\top z(G), \qquad w \sim \mathcal{N}(0, \sigma_w^2 I),$$

where w contains the regression weights, which are assumed independent under the prior, each with mean zero and variance σ_w^2. After observing evaluated architectures with their objectives, this prior is updated according to which weight values are consistent with the observed data. For a new graph G_*, this results in the predictive distribution

$$f(G_*) \mid \mathcal{D} \sim \mathcal{N}\left(z_*^\top \mu_w,\ z_*^\top \Sigma_w z_*\right), \qquad \Sigma_w = \left(\frac{1}{\sigma_w^2} I + \frac{1}{\sigma^2} Z^\top Z\right)^{-1}, \qquad \mu_w = \frac{1}{\sigma^2} \Sigma_w Z^\top y.$$

where $z_* = z(G_*)$, Z is the embedding matrix of the previously evaluated graphs, y their observed objective values, and σ^2 is the assumed observation-noise variance. The mean of this distribution gives the predicted objective, while its variance quantifies the uncertainty based on the the point's overlap with previous observations from Z.

B

Graph Neural Network Surrogates

While this thesis primarily focuses on graph-kernel Gaussian-process surrogates, two graph-neural-network-based surrogate models were also investigated, namely Arch2Vec [22] and DGBO [10]. These approaches offer flexible, learnable graph representations and avoid the $O(n^3)$ scaling of exact GP training with the number of evaluated architectures. However, their larger number of trainable parameters can increase data requirements and make training less reliable in the low-data setting of SAO. This trade-off motivated their investigation, and this appendix presents their adaptation and observed behavior in more detail than the scientific article.

Since architecture graphs can be heterogeneous in terms of their attributes, teh considered GNN approaches required a node feature representation that accommodates this. We first explain how this shared representation is constructed, then introduce the two approaches, their specific motivations, and the observations made during evaluation. Most of this investigation focused on Arch2Vec, as initially it appeared particularly promising, which is explained in more detail later.

B.1. Representing heterogeneous architecture graphs

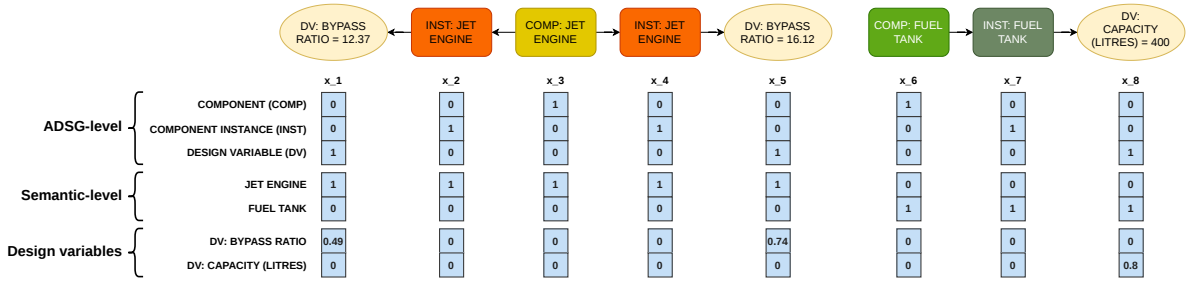


Figure B.1: Example feature encoding for selected nodes of a resolved architecture instance. Each node is mapped to a common feature space containing ADSG-level node-type features, semantic component-type features, and architecture-specific design-variable values. Repeated instances of the same component and design-variable type share the same feature dimensions, while non-applicable dimensions remain zero. Continuous values are normalized to $[0, 1]$.

In many common GNN settings, all nodes are of the same type and are described by the same features. For example, in a social network graph, each node may represent a person described by the same attributes, such as age and gender. This is not naturally the case for SAO architectures, as different node types may carry fundamentally different information. For example, design-variable nodes associated with a jet-engine instance may carry attributes representing bypass ratio and fan-pressure ratio, whereas a design-variable node associated with a fuel tank may carry a capacity feature. Heterogeneous GNNs explicitly accommodate different node and relation types [23]. However, introducing separate trainable transformations for these types can increase the number of parameters, potentially making learning more difficult with few evaluated architectures.

We therefore adopt a shared feature representation that maps these heterogeneous attributes into a common sparse feature space, such that all nodes can be represented by feature vectors of the same

dimensionality while retaining their node type, component semantics, and associated design-variable values. For each node v, it's respective vector x is divided into the three following blocks:

$$x_v = \left[x_v^{\text{ADSG}}, x_v^{\text{semantic}}, x_v^{\text{dv}}\right].$$

A simplified example of such feature space is shown in Figure B.1. The first block one-hot encodes the node's ADSG type, distinguishing functions, components, component instances, ports, attributes, and design-variable nodes, analogous to the ADSG-level labeling used by the graph kernels. The second block encodes the actual semantic component type (i.e. jet engine, fuel tank) associated with the node, allowing the GGN to recognize repeated instances of nodes pertaining to the same component type. The final block contains the attributes and sizing parameters such as bypass ratio or blade count. Continuous design variables are normalized to $[0, 1]$, categorical variables are one-hot encoded, and ordinal variables use a cumulative binary encoding. The feature vector entries that are not applicable to a particular node remain zero.

Both neural surrogate models operate on the resulting attributed architecture graphs, using the node-feature matrix together with the graph connectivity encoded by an adjacency matrix A. Before message passing, the adjacency matrix is symmetrized so that GNN layers can aggregate information in both directions, irrespective of the directed ADSG derivation structure. Parallel connections retain their multiplicity during message passing.

B.2. Arch2Vec

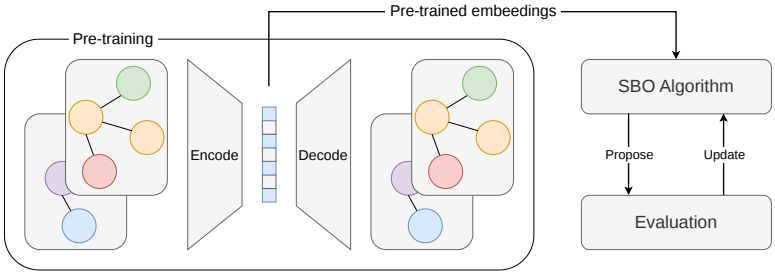


Figure B.2: Overview of the arch2Vec optimization pipeline. Architecture embeddings are first learned through unsupervised autoencoder pretraining on unevaluated architecture instances. The pretrained encoder is then kept fixed and used to map candidate architectures into the latent space, where the surrogate-based optimization algorithm proposes new architectures for expensive evaluation and is updated iteratively with the resulting responses.

Arch2Vec uses unsupervised autoencoder pretraining to learn graph representations, which subsequently serve as inputs to a surrogate for Bayesian optimization [22]. As illustrated in Figure B.2, an encoder maps architecture graphs into a latent space, while a decoder attempts to reconstruct the original graphs from these representations. This encourages the latent representations to retain information about the architectures without requiring their objective values.

This is appealing for SAO because architecture evaluations are expensive, whereas valid instances can be sampled cheaply from the ADSG. If reconstruction produces a useful representation, much of the learning can be performed without requiring the limited evaluated samples, potentially alleviating the data requirements of GNN surrogates. Moreover, the same pretrained representation can support separate surrogate heads for every objective and constraint. The promising NAS results reported for Arch2Vec [22, 24] further motivated its investigation.

Arch2Vec uses a variational graph autoencoder with a Graph Isomorphism Network (GIN) encoder [25, 22]. GIN is a GNN variant that updates each node representation by summing its neighboring representations and combining the result with its own through a learned transformation. The encoder produces a mean vector μ and a variance vector σ^2 for each node, defining a Gaussian latent distribution from which a node embedding z_v is sampled during pretraining. A Kullback–Leibler divergence term regularizes these distributions towards a standard Gaussian prior.

The decoder reconstructs the binary adjacency matrix A, where $A_{ij} = 1$ if a connection exists between nodes i and j, and zero otherwise. A neural network d_A first transforms each sampled node embedding, after which their inner product is passed through a sigmoid function,

$$\hat{A}_{ij} = \sigma\left(d_A(z_i)^\top d_A(z_j)\right).$$

Here, $d_A(z_i)$ and $d_A(z_j)$ are the transformed node representations. Their inner product provides a connection score, which the sigmoid maps to a probability $\hat{A}_{ij} \in (0, 1)$.

For SAO, we adapt feature reconstruction to include both levels of node labels and add a loss term for continuous attributes and sizing parameters, consistently with the featurisation from Appendix B.1. The resulting training objective is

$$\mathcal{L}_{\text{A2V}} = \mathcal{L}_A^{\text{BCE}} + \mathcal{L}_{X,\text{bin}}^{\text{BCE}} + \mathcal{L}_{X,\text{cont}}^{\text{MSE}} + \beta \mathcal{L}_{\text{KL}}.$$

Here, BCE denotes binary cross entropy, used to reconstruct the adjacency A and binary features, including the one-hot-encoded labels at both the ADSG and semantic level. MSE denotes mean-squared error, used to reconstruct normalized sizing variable values. The weight β controls the strength of KL regularization.

After pretraining, the decoder is discarded and the encoder is frozen. Each graph is embedded as the sum of the latent μ vectors of its nodes. Following the Bayesian-optimization variant of Arch2Vec, these embeddings are passed to separate DNGO surrogates [26] for each objective and constraint. Each DNGO head first trains a shallow neural network with a linear output layer to predict the true objectives from the frozen architecture graph embeddings. After training, the final output layer is replaced by a Bayesian linear regressor, while the earlier layers are held fixed [26]. The BLR then provides predictive means and uncertainties, as described in subsection A.4.3.

B.2.1. Ablation studies

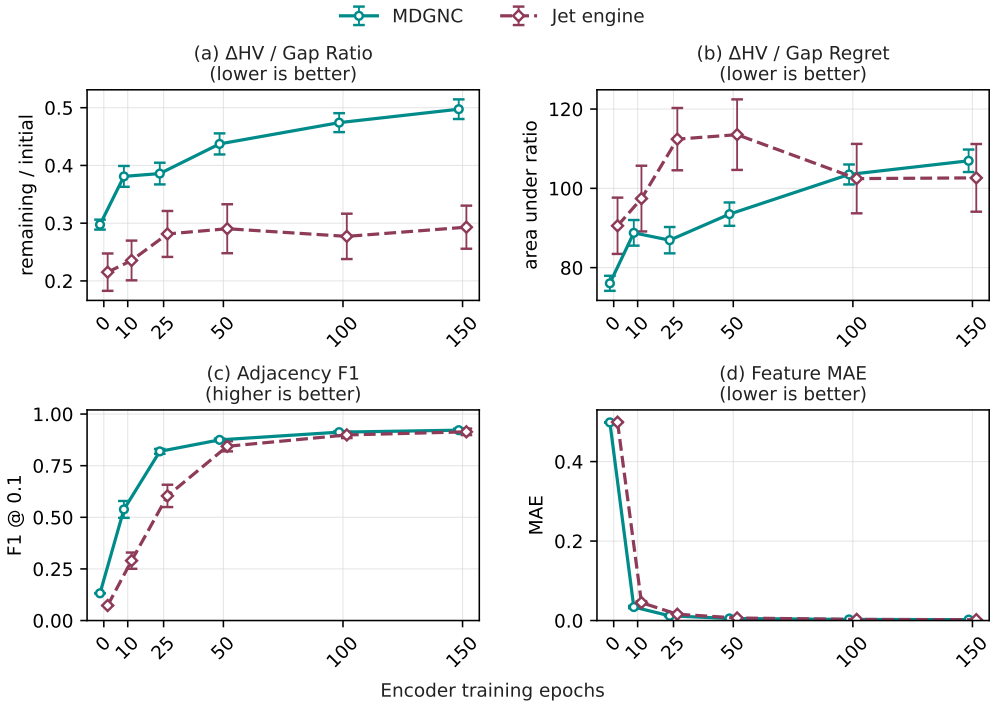


Figure B.3: Effect of Arch2Vec pretraining on optimization and reconstruction performance for MDGNC and ENGINE. Each point aggregates three encoder depths, three KL regularization weights, and 20 independent seeds. Reconstruction is evaluated on 20,000 held-out architectures per problem. Zero epochs denotes a randomly initialized encoder.

To assess the suitability of Arch2Vec for SAO and understand its behavior, I evaluated encoder depths of three, four, and five layers and KL regularization weights $\beta \in \{0, 0.0025, 0.01\}$. The latent dimension was fixed at 32 after preliminary comparisons of $d \in \{16, 32, 64\}$ showed that it provided sufficient reconstruction quality. For each configuration, checkpoints at 0, 10, 25, 50, 100, and 150 pretraining epochs were evaluated in downstream optimization on MDGNC and ENGINE over 20 independent seeds. Zero epochs corresponds to a randomly initialized encoder.

Reconstruction was evaluated on 20,000 held-out architectures per problem. Both adjacency and node features were reconstructed accurately despite using a substantially smaller pretraining dataset,

and increasing its size yielded little additional improvement. However, while reconstruction quality improved with longer pretraining, optimization performance generally deteriorated (Figure B.3). Pretraining therefore offered no consistent advantage over a randomly initialized encoder, suggesting that accurate reconstruction alone does not ensure a latent representation is useful for optimization.

I also examined the downstream DNGO head as a potential source of inaccurate predictions or unreliable uncertainty estimates. On the MDGNC benchmark, models were fitted using 50, 100, and 200 evaluated architectures and varying the number of training epochs. Held-out prediction error stabilized relatively early, while predicted uncertainty continued to decrease (Figure B.4). The model thus became more confident without a corresponding improvement in predictive accuracy.

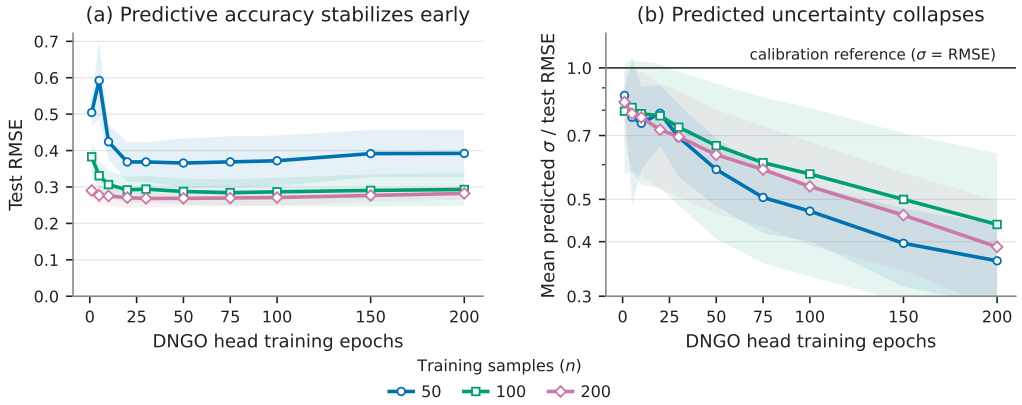


Figure B.4: Effect of training duration on the DNGO head for MDGNC with 50, 100, and 200 training samples. (a) Prediction error on held-out architectures. (b) Mean predicted standard deviation divided by test RMSE. A value of one indicates matching uncertainty and error scales; lower values indicate smaller uncertainty estimates relative to the error.

In the DNGO head, an MLP transforms the pretrained graph embeddings into features on top of which a Bayesian linear regressor is fitted. A possible explanation concerns the noise variance of this Bayesian regression model. This variance represents the assumed scatter of observations around the predictions of the fitted linear function. If the DNGO's last hidden layer learns features from which the true objective values can be recovered perfectly by a linear regressor, likelihood-based estimation can fit a very small noise variance, even when these features generalize poorly to unseen architectures. This in turn can yield a model that is overconfident because it interpolated perfectly on training data.

For the experiments in chapter 2, we selected the configuration offering the best balance between reconstruction quality and optimization performance among the tested settings, preserving Arch2Vec's premise of learning architecture representations through reconstruction. All four problems used five GIN layers with hidden width 64, batch normalization, and latent node dimension 32. For each problem, 5,000 architectures were sampled for pretraining, using Adam with learning rate 10^{-3}, batch size 64, and KL weight $\beta = 0.0025$. The epoch-50 encoder checkpoint was frozen for optimization. Each DNGO head used a wide network with a hidden dimension of 128 and tanh activations, which is consistent with [26]. The head was trained for 10 epochs, based on the observations in Figure B.4.

Although initially promising for SAO, Arch2Vec did not achieve competitive optimization performance (chapter 2). The DNGO analysis also illustrates the potential fragility of neural surrogates. Appropriate training durations, learning rates, and regularization may differ between responses, while reserving enough evaluated architectures for reliable validation reduces the already limited data available for fitting.

B.3. DGBO

The second GNN-based surrogate considered in chapter 2 is Deep Graph Bayesian Optimization (DGBO) [10]. It was included to assess whether a GNN trained directly on the few available evaluated architecture graphs could learn useful representations. Unlike Arch2Vec, which learns its graph representation separately through unsupervised reconstruction, DGBO learns it directly from the objective or constraint values being modeled. The entire network is refitted on the accumulated observations after each new batch of architecture evaluations becomes available during optimization.

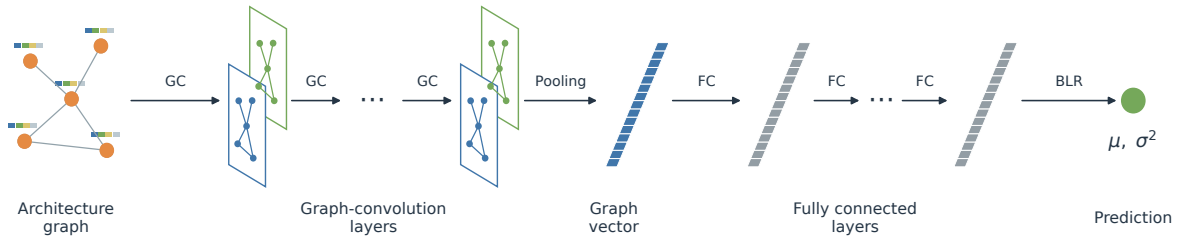


Figure B.5: Overview of the DGBO surrogate, based on Cui et al. [10]. Graph-convolution (GC) layers learn node representations, which are pooled into a graph-level vector and transformed by fully connected (FC) layers. After supervised training, the linear output layer is replaced by Bayesian linear regression (BLR), providing a predictive mean μ and variance σ^2.

An overview of the surrogate is shown in Figure B.5. DGBO combines five graph-convolution (GC) layers, which aggregate information between neighboring nodes, with a sum pooling operation that produces a graph-level representation. This representation is passed through fully connected layers to predict the modeled response. The network is first trained with a linear output layer, which is subsequently replaced by a Bayesian linear regressor fitted to the final hidden-layer features, which again follows the DNGO method that is also adopted in Arch2Vec. This provides predictive means and uncertainties for Bayesian optimization. A separate surrogate is fitted for each objective and constraint.

We retained the network architecture proposed by Cui et al. [10], comprising five graph-convolution layers with 48 hidden units each, a pooling width of 50, and five fully connected layers with 45 hidden units each. We used tanh activations, no dropout, and the node features described in Appendix B.1. Each refit used 200 training epochs, a batch size of 10, and weight decay of 10^{-5}. The learning rate was increased from 10^{-4} to 10^{-3} following preliminary observations of improved convergence on MDGNC.

Despite the limited training data, DGBO performed well on the structurally driven MDGNC and MDGNC-EF problems, substantially outperforming Arch2Vec, as reported in chapter 2. However, its performance was considerably weaker on ENGINE and ROCKET. I anticipated that the flexibility of a learned graph representation might allow DGBO to adapt effectively to a broad range of objectives, but this was not observed across the tested problems. The strong performance on the structurally driven problems could mean that useful graph representations may be learned even from the limited observations available in SAO. However, the weaker performance on the remaining problems could reflect sensitivity to choices such as the learning rate, network width, and training duration. Further studies would be required to validate these possible explanations.

References

[1] Jasper H. Bussemaker. "System Architecture Optimization: Function-Based Modeling, Optimization Algorithms, and Multidisciplinary Evaluation". PhD thesis. Delft University of Technology and ISAE-SUPAERO, 2025. ISBN: 978-94-6518-083-0. DOI: 10.4233/uuid:246b18f9-1f8c-4ff7-b824-2b1786cf9d14.

[2] Frederico Afonso et al. "Strategies towards a more sustainable aviation: A systematic review". In: *Progress in Aerospace Sciences* 137 (2023), p. 100878. DOI: 10.1016/j.paerosci.2022.100878.

[3] Eytan J. Adler and Joaquim R. R. A. Martins. "Hydrogen-powered aircraft: Fundamental concepts, key technologies, and environmental impacts". In: *Progress in Aerospace Sciences* 141, 100922 (2023). DOI: 10.1016/j.paerosci.2023.100922.

[4] Edward F. Crawley, Bruce G. Cameron, and Daniel Selva. *System Architecture: Strategy and Product Development for Complex Systems*. Harlow, UK: Pearson, 2015.

[5] Jasper H. Bussemaker et al. "System Architecture Optimization Strategies: Dealing with Expensive Hierarchical Problems". In: *Journal of Global Optimization* 91.4 (2025), pp. 851–895. DOI: 10.1007/s10898-024-01443-8.

[6] Donald R. Jones, Matthias Schonlau, and William J. Welch. "Efficient Global Optimization of Expensive Black-Box Functions". In: *Journal of Global Optimization* 13.4 (1998), pp. 455–492. DOI: 10.1023/A:1008306431147.

[7] Paul Saves et al. "Modeling Hierarchical Spaces: A Review and Unified Framework for Surrogate-Based Architecture Design". In: *Structural and Multidisciplinary Optimization* 69.3 (2026), p. 65. DOI: 10.1007/s00158-026-04249-2.

[8] Binxin Ru et al. "Interpretable Neural Architecture Search via Bayesian Optimisation with Weisfeiler-Lehman Kernels". In: *International Conference on Learning Representations*. 2021.

[9] Lizheng Ma, Jiaxu Cui, and Bo Yang. "Deep Neural Architecture Search with Deep Graph Bayesian Optimization". In: *Proceedings of the 2019 IEEE/WIC/ACM International Conference on Web Intelligence*. WI '19. 2019, pp. 500–507. DOI: 10.1145/3350546.3360740.

[10] Jiaxu Cui, Bo Yang, and Xia Hu. "Deep Bayesian Optimization on Attributed Graphs". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 33.01 (2019), pp. 1377–1384. DOI: 10.1609/aaai.v33i01.33011377.

[11] Ksenia Korovina et al. "ChemBO: Bayesian Optimization of Small Organic Molecules with Synthesizable Recommendations". In: *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*. Vol. 108. Proceedings of Machine Learning Research. 2020, pp. 3393–3403.

[12] Miles Wang-Henderson et al. "Graph Neural Bayesian Optimization for Virtual Screening". In: *NeurIPS 2023 Workshop on New Frontiers of AI for Drug Discovery and Development*. 2023. URL: https://openreview.net/forum?id=3OW9rqKGKy.

[13] Jasper H. Bussemaker et al. "System Architecture Optimization: An Open Source Multidisciplinary Aircraft Jet Engine Architecting Problem". In: *AIAA AVIATION 2021 Forum*. 2021. DOI: 10.2514/6.2021-3078.

[14] Jasper H. Bussemaker et al. "Effectiveness of Surrogate-Based Optimization Algorithms for System Architecture Optimization". In: *AIAA AVIATION 2021 Forum*. 2021. DOI: 10.2514/6.2021-3095.

[15] Nestor V. Queipo et al. "Surrogate-Based Analysis and Optimization". In: *Progress in Aerospace Sciences* 41.1 (2005), pp. 1–28. DOI: 10.1016/j.paerosci.2005.02.001.

[16] Bobak Shahriari et al. "Taking the Human Out of the Loop: A Review of Bayesian Optimization". In: *Proceedings of the IEEE* 104.1 (2016), pp. 148–175. DOI: 10.1109/JPROC.2015.2494218.

[17] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. Cambridge, MA: MIT Press, 2006. ISBN: 9780262182539. DOI: 10.7551/mitpress/3206.001.0001.

[18] Alain Berlinet and Christine Thomas-Agnan. *Reproducing Kernel Hilbert Spaces in Probability and Statistics*. 1st ed. New York, NY: Springer, 2004. ISBN: 978-1-4020-7679-4. DOI: 10.1007/978-1-4419-9096-9.

[19] Giannis Nikolentzos, Giannis Siglidis, and Michalis Vazirgiannis. "Graph Kernels: A Survey". In: *Journal of Artificial Intelligence Research* 72 (2021), pp. 943–1027. DOI: 10.1613/jair.1.13225.

[20] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. "Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles". In: *Advances in Neural Information Processing Systems*. Vol. 30. 2017.

[21] Arman Hasanzadeh et al. "Bayesian Graph Neural Networks with Adaptive Connection Sampling". In: *Proceedings of the 37th International Conference on Machine Learning*. Vol. 119. Proceedings of Machine Learning Research. 2020, pp. 4094–4104.

[22] Shen Yan et al. "Does Unsupervised Architecture Representation Learning Help Neural Architecture Search?" In: *Advances in Neural Information Processing Systems*. Vol. 33. 2020, pp. 12486–12498. URL: https://proceedings.neurips.cc/paper/2020/hash/937936029af671cf479fa893db91cbdd-Abstract.html.

[23] Rui Bing et al. "Heterogeneous graph neural networks analysis: a survey of techniques, evaluations and applications". In: *Artificial Intelligence Review* 56 (2023), pp. 8003–8042. DOI: 10.1007/s10462-022-10375-2.

[24] Yize Sun et al. "Quantum Architecture Search with Unsupervised Representation Learning". In: *Quantum* 10 (Feb. 2026), p. 1994. DOI: 10.22331/q-2026-02-03-1994.

[25] Keyulu Xu et al. "How Powerful Are Graph Neural Networks?" In: *International Conference on Learning Representations*. 2019. URL: https://openreview.net/forum?id=ryGs6iA5Km.

[26] Jasper Snoek et al. "Scalable Bayesian Optimization Using Deep Neural Networks". In: *Proceedings of the 32nd International Conference on Machine Learning*. Vol. 37. Proceedings of Machine Learning Research. PMLR, 2015, pp. 2171–2180. URL: https://proceedings.mlr.press/v37/snoek15.html.