JCTA Journal of Computing Theories and Applications ISSN:3024-9104

Research Article

ArchEvolve: A Collaborative and Interactive Search-Based Framework with Preference Learning for Optimizing Software Architectures

Ayobami E. Mesioye [1,*], Adesola M. Falade [2], and Kayode E. Akinola [3]

[1] Department of Cybersecurity, McPherson University, Seriki Sotayo, Ogun State 110106, Nigeria; e-mail : mesioyeae@mcu.edu.ng
[2] Department of Software Engineering, McPherson University, Seriki Sotayo, Ogun State 110106, Nigeria; e-mail : faladeam@mcu.edu.ng
[3] Department of Information Technology, McPherson University, Seriki Sotayo, Ogun State 110106, Nigeria; e-mail : akinolake@mcu.edu.ng
* Corresponding Author : Ayobami E. Mesioye

Abstract: The use of Search-Based Software Engineering (SBSE) for optimizing software architecture has evolved from fully automated to interactive approaches, integrating human expertise. However, current interactive tools face limitations: they typically support only single decision-makers, confine architects to passive roles, and induce significant cognitive fatigue from repetitive evaluations. These issues disconnect them from modern, team-based software development, where collaboration and consensus are crucial. To address these shortcomings, we propose "ArchEvolve," a novel framework designed to facilitate collaborative, multi-architect decision-making. ArchEvolve employs a cooperative coevolutionary model that concurrently evolves a population of candidate architectures and distinct populations representing each architect's unique preferences. This structure guides the search towards high-quality consensus solutions that accommodate diverse, often conflicting, stakeholder viewpoints. An integrated Artificial Neural Network (ANN) serves as a preference learning module, trained on explicit team feedback to act as a surrogate evaluator. This active learning cycle substantially reduces the number of required human interactions and alleviates user fatigue. Empirical evaluation on two industrial case studies (E-Commerce System and Healthcare Management System) compared ArchEvolve to a state-of-the-art interactive baseline. Results indicate that ArchEvolve achieves statistically significant improvements in both solution quality and consensus-building. The preference learning module demonstrated over 90% accuracy in predicting team ratings and reduced human evaluations by up to 46% without compromising final solution quality. ArchEvolve provides a practical, scalable framework supporting collaborative, consensus-driven architectural design, making interactive optimization a more viable and efficient tool for real-world software engineering teams by intelligently integrating cooperative coevolutionary search with a preference learning surrogate.

Keywords: Collaborative Decision-Making; Cooperative Coevolution; Human-in-the-Loop Optimization; Interactive Evolutionary Computation; Multi-Objective Optimization; Preference Learning; Search-Based Software Engineering (SBSE); Software Architecture.

Received: October, 28th 2025
Revised: November, 11th 2025
Accepted: November, 18th 2025
Published: November, 20th 2025

1. Introduction

In the modern era of complex software ecosystems—characterized by microservices, cloud-native deployment, and AI-enabled systems—software architecture constitutes the set of fundamental structural decisions that are critical to achieving quality goals and costly to modify once implemented [1]. These early-stage architectural choices profoundly influence a project's success, long-term maintainability, and ability to satisfy non-functional requirements such as scalability, resilience, and security [2]. Architects must navigate an extensive, multi-dimensional design space, making decisions about components, connectors, and deployment

strategies under uncertainty and often conflicting goals[3]. As a result, architectural decision-making is an inherently challenging and human-centered activity [4].

A major difficulty in this process, especially for large and complex systems, lies in collaborative decision-making. Architectural choices rarely satisfy all stakeholders; developers, project managers, security specialists, and end-users often prioritize different objectives. Achieving consensus among multiple architects—each with distinct expertise and perspectives—can be time-consuming, iterative, and cognitively demanding [5]–[7]. This collaborative nature often leads to debates, compromises, and incremental refinements, making it difficult to converge on an optimal solution efficiently. To support decision-making in such complex and collaborative environments, Search-Based Software Engineering (SBSE) has emerged as a dominant paradigm [8]. SBSE formulates architecture design as a Many-Objective Optimization Problem (MaOP) and employs Many-Objective Evolutionary Algorithms (MaOEAs) to search effectively for Pareto-optimal solutions. These solutions reveal key trade-offs among competing objectives, providing valuable insights for architects during analysis [9]. However, fully automated search often falls short in practice, as architectural knowledge is frequently qualitative, subjective, or too nuanced to encode as static fitness functions. This limitation has motivated the adoption of Interactive Evolutionary Computation (IEC), which integrates experts "in the loop" to provide domain judgments during optimization.

Despite these advances, practical applications of interactive SBSE face several major limitations. Current IEC systems typically restrict architects to a passive role, limiting their input to evaluating pre-generated solutions rather than allowing them to actively guide the search process [10]. Furthermore, most existing approaches assume a single decision-maker, which contradicts the reality of modern software development where architectural decisions are reached collaboratively through team deliberation and consensus-building [11]. This single-user design also increases cognitive load, as frequent evaluations impose substantial fatigue upon users, degrading feedback quality and ultimately compromising search performance [12].

The comparative baseline used in this study, R-NSGA-II, is a widely recognized reference-point–based extension of the Non-dominated Sorting Genetic Algorithm II (NSGA-II). NSGA-II is an influential and efficient multi-objective evolutionary algorithm known for promoting convergence toward the Pareto front while preserving solution diversity [9]. R-NSGA-II enhances this framework by enabling decision-makers to express preferences via reference points in the objective space, guiding the search toward solutions near these desired regions. This capability makes R-NSGA-II suitable for interactive optimization involving a single decision-maker. However, for comparative evaluation in a multi-architect setting, the preferences of three simulated architects were averaged to form a single reference point. While common in single-user tools, this averaging introduces information loss and cannot represent the complexity of conflicting stakeholder priorities—further justifying the need for a coevolutionary approach.

Existing frameworks do not simultaneously support multi-architect collaboration, intelligent preference learning to reduce user interaction, and effective balancing of solution quality with team consensus. A more scalable, collaborative, and cognitively efficient approach is needed to overcome the shortcomings of single-user, high-fatigue interactive methods and better reflect the realities of modern team-based software development. To address these interconnected challenges, this paper proposes ArchEvolve, a collaborative and interactive search-based framework that redefines how architectural design teams engage with optimization. To overcome the limitations of single-user design and support consensus-building, ArchEvolve adopts a cooperative coevolutionary model that explicitly treats each architect's potentially conflicting preferences as distinct, co-evolving populations. This structure guides the search toward solutions that satisfy the entire team. To reduce fatigue and minimize unnecessary human evaluations, an Artificial Neural Network (ANN)–based preference learning module is integrated to learn and generalize the team's decision patterns. In addition, the framework provides a flexible interaction mechanism that allows architects to influence the search beyond simple ratings, including injecting known good solutions or adjusting objective priorities.

The primary contribution of this work is a novel cooperative coevolutionary framework aimed at identifying consensus solutions by navigating the complex and often conflicting preference landscape of multi-architect teams. The effectiveness of the proposed approach is

demonstrated through a comprehensive empirical evaluation involving two industrial case studies, showing improved solution quality, stronger consensus, and reduced interaction effort compared to a state-of-the-art baseline.

2. Literature Review

The design of software architecture is a complex multi-objective challenge in which engineers must balance conflicting quality attributes such as performance, cost, and security. The emergence of SBSE has provided a powerful paradigm for navigating this complexity. This section organizes the relevant literature into key themes to highlight its evolution and to identify the remaining gaps that ArchEvolve aims to address.

2.1. Search-Based Architecture Optimization

This foundational theme focuses on applying evolutionary algorithms and other search heuristics to optimize various aspects of software architecture. Early works such as [13] and [14] applied genetic algorithms to optimize architectures based on non-functional requirements and security considerations. Similarly, [9] demonstrates the use of MaOEAs for exploring Pareto-optimal trade-offs, while [15] applies search-based methods to hardware/software co-design. These automated approaches are highly effective at systematically exploring vast design spaces that are too complex for manual exploration. They can uncover novel solutions and offer insights into trade-offs among conflicting objectives.

However, purely automated SBSE often treats architects as passive recipients of the "optimal" solutions produced. A significant drawback is the difficulty of capturing qualitative, subjective, and tacit architectural knowledge within formalized fitness functions. This mismatch can lead to technically optimal but practically infeasible or undesirable solutions—outcomes that do not align with human intuition or implicit design constraints—effectively turning the optimization process into a "black box."

2.2. Interactive Search-Based Software Engineering (iSBSE)

To overcome this "black box" limitation, iSBSE integrates human expertise directly into the optimization loop. For example, [16] pioneered an interactive metaheuristic approach for service identification that incorporates user preferences into the search. The SQuAT framework introduced by [17] similarly emphasizes collaboration between human architects and the optimization engine to explore design solutions and evaluate trade-offs among quality attributes. Further contributions such as [18] validated interactive ranking operators for algorithms like NSGA-II, showing that human feedback can effectively guide evolutionary search.

iSBSE increases the relevance, interpretability, and acceptability of generated solutions by enabling architects to inject domain knowledge, subjective evaluations, and evolving preferences. It transforms the optimization process into a human-guided exploration, making results more actionable. However, iSBSE also introduces significant challenges—particularly human fatigue. Evolutionary algorithms typically require thousands of evaluations, making frequent human reviews impractical and cognitively taxing [12]. This fatigue degrades feedback quality and limits scalability. In addition, most iSBSE tools are inherently designed for a single decision-maker, misaligning with the collaborative nature of modern software development [11].

2.3. Preference Learning in Optimization

To mitigate human fatigue in iSBSE, research has increasingly focused on incorporating intelligent techniques, particularly preference learning via Machine Learning (ML). Key work by [19] and [20] within the OPLA-Tool ecosystem demonstrates this synergy: architects evaluate only a subset of solutions, and an ML model learns these preferences to automate subsequent evaluations, drastically reducing human effort[21]. Further refinements explore alternative ML models for preference representation [22] and feature selection techniques to enhance model accuracy and efficiency [23]. Analogous efforts in other engineering domains—such as civil architecture—have also combined evolutionary computation and cognitive machine learning to support designer preferences [18].

Preference learning substantially improves the feasibility of iSBSE by acting as a surrogate for human evaluation, thus lowering the number of required interactions and mitigating

fatigue. It enables more efficient navigation of complex preference landscapes. However, current preference learning methods primarily focus on modelling the preferences of a single decision-maker or an aggregated preference profile. They rarely provide explicit mechanisms for managing multiple, potentially conflicting preference models from a team of architects. As a result, these approaches may fail to find genuine consensus solutions that are mutually acceptable to all stakeholders. The challenge of effectively combining and evolving distinct preference models in a collaborative setting remains largely unresolved.

2.4. Collaborative Decision Support in Software Architecture

Research in this area recognizes the inherently multi-stakeholder nature of architectural design [11]. While frameworks such as SQuAT [17] promote collaboration between humans and search engines, explicit support for multiple architects simultaneously influencing the search with distinct preferences is still limited. Traditional techniques such as averaging preferences—as used in the R-NSGA-II baseline—attempt to reconcile differences but often produce information loss and suboptimal compromises.

A key gap in current research is the lack of robust, interactive SBSE frameworks designed specifically for managing and evolving the distinct preferences of multiple architects in real time. The challenge is to preserve each architect's individual preferences while guiding the search toward solutions that are collectively acceptable. No existing framework robustly integrates interactive search, preference learning, and explicit management of multiple, co-evolving preference models to achieve true team consensus.

2.5. Synthesis: Positioning the ArchEvolve Contribution

The reviewed literature reveals a clear progression—from automated search-based optimization, to interactive human-guided frameworks, and finally to ML-assisted preference learning that aims to reduce user effort. ArchEvolve is situated at the forefront of this trajectory. It builds directly on the proven concepts introduced by OPLA-Tool and SQuAT, integrating refined machine learning and evolutionary techniques from recent studies.

Critically, ArchEvolve addresses the central gaps identified across the four thematic areas:

- the lack of explicit support for multiple architects with distinct and conflicting preferences;
- the absence of mechanisms for co-evolving these preferences toward consensus;
- and the limited ability of current approaches to mitigate user fatigue in interactive settings.

The ArchEvolve framework introduces a novel cooperative coevolutionary model that explicitly represents and evolves individual preference populations for each architect. This enables the search to move toward design solutions that are acceptable to the entire team. Concurrently, the integrated ANN-based preference learner significantly reduces required human interactions, making collaborative, consensus-oriented architectural design more efficient and practical for real-world software engineering teams[24].

3. Methodology

3.1. The ArchEvolve Framework

ArchEvolve is a semi-automated decision-support framework designed to facilitate collaborative software architecture design. It is composed of three cores, interacting components:

- A Cooperative Coevolutionary Core,
- A Flexible Interactive Mechanism, and
- An ANN-based Preference Learner.

These components work together to model multiple architects with distinct preferences, guide the search toward consensus solutions, and reduce user fatigue through selective, intelligent interaction. An overview of the ArchEvolve architecture is shown in Figure 1. The notation used throughout this section is summarized in Table 1, which defines key symbols for architecture solutions, preference models, quality vectors, and ANN parameters.

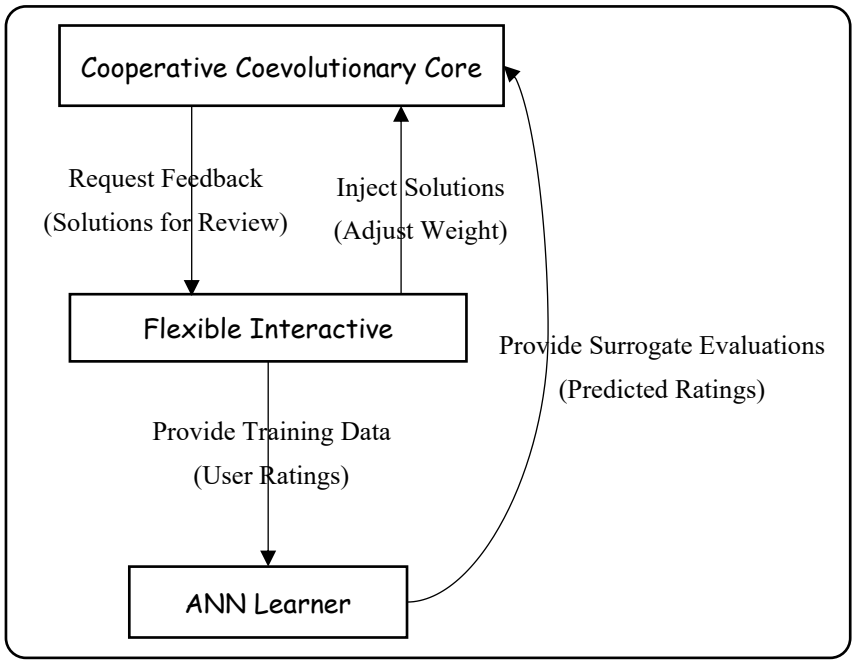

Figure 1. Overview of the ArchEvolve framework

Table 1. Notation used in this work

Symbol	Description
A	A candidate software architecture solution
P_{arch}	Population of candidate architectures
p_i	Preference model for architect i
P_{pref_i}	Population of preference models for architect i
N	Total number of architects in the team
$Q(A)$	Vector of *M* quality-objective values for architecture A
w_i	Weight vector representing architect i's preferences
$U(A, w_i)$	Utility of architecture A for architect i
f_{ANN}	Artificial Neural Network surrogate model
θ	ANN parameters (weights and biases)

ArchEvolve integrates these components into a unified workflow:

- The Cooperative Coevolutionary Core evolves both architecture solutions and multiple architect-specific preference models.
- The Interactive Mechanism allows architects to periodically guide the search via ratings, solution injection, or objective-weight adjustments.
- The ANN-based Preference Learner serves as a surrogate evaluator to reduce interaction load, using active learning to request human input only when necessary.

In the following subsections, we provide a detailed description of each component and its role in the overall optimization process.

3.1.1. Cooperative Coevolutionary Core

The Cooperative Coevolutionary Core is responsible for jointly evolving candidate architecture solutions and the preference models of multiple architects. Unlike traditional MaOEAs, which evolve a single unified population, ArchEvolve adopts a two-level coevolutionary structure consisting of:

- An Architecture Population (P_{arch}): This population contains candidate software architecture designs.
- N Architect-Specific Preference Populations (P_{pref_i}). For each architect $i \in \{1, \ldots, N\}$, a separate population of preference models is maintained.

These populations co-evolve alongside P_{arch}, nfluencing and being influenced by the architectural solutions discovered during the search. This collaborative evolutionary structure is illustrated in Figure 2.

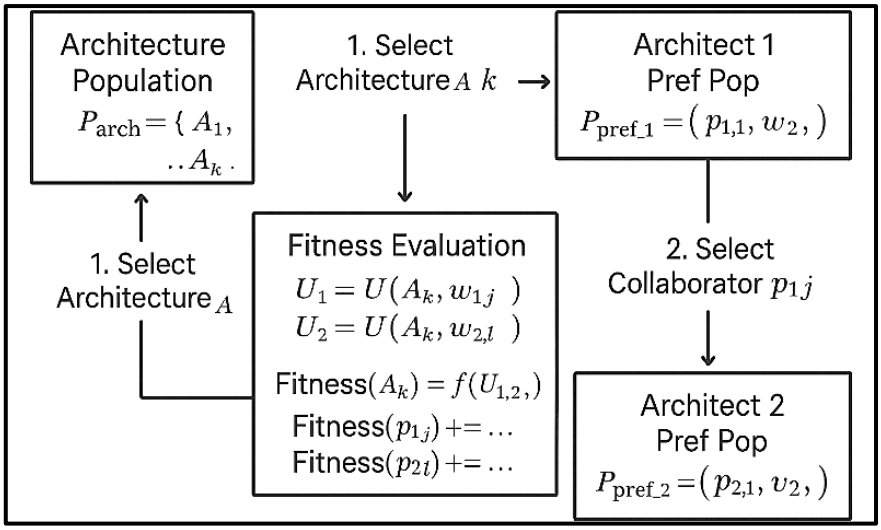

Figure 2. Cooperative coevolutionary evaluation loop

a) Architecture Population

Each architecture solution $A \in P_{\text{arch}}$ is encoded as a vector of design decisions. In the context of a microservice-based system, these may include:

1. Database choice for individual services (e.g., SQL, NoSQL)
2. Communication protocols between services (e.g., REST, Message Queue)
3. Deployment strategy (e.g., containerized, serverless)
4. Security mechanisms (e.g., OAuth, API key)
5. Programming languages for modules (e.g., Java, Python)

Each architecture A is associated with a corresponding vector of M normalized quality-objective values, $Q(A) = [q_1, q_2, \dots, q_M]$. Normalization to the range $[0,1]$ensures consistent interpretation across objectives. For an objective.

$$q_j = \begin{cases} \dfrac{raw_{value_j} - \min\left(raw_{value_j}\right)}{\max\left(raw_{value_j}\right) - \min\left(raw_{value_j}\right)}, & \text{If higher values are better (e.g., performance, security, reliability)} \\ \dfrac{\max\left(raw_{value_j}\right) - raw_{value_j}}{\max\left(raw_{value_j}\right) - \min\left(raw_{value_j}\right)}, & \text{If lower values are better (e.g., cost)} \end{cases} \quad (1)$$

This min–max normalization ensures that a higher normalized score always reflects a more desirable outcome, regardless of whether the original objective is to be maximized or minimized. The normalized vector $Q(A)$ is used consistently in the utility function (see Eq. 3) and the consensus metric (see Eq. 14).

b) Preference Populations

Each architect i is modeled through a preference weight vector:

$$w_i = [w_{i,1}, w_{i,2}, \dots, w_{i,M}], \quad \sum_j w_{i,j} = 1 \quad (2)$$

These vectors describe how strongly architect i prioritizes each quality objective.

Fitness of an Architecture: The utility of an architecture A with respect to architect i is computed using the dot product of its quality vector and the architect's weight vector:

$$U(A, w_i) = \sum_{j=1}^{M} q_j \cdot w_{i,j} \quad (3)$$

To promote strong consensus, the overall fitness of architecture A is defined as:

$$\text{Fitness}(A) = \min\big(U(A, w_1), U(A, w_2), \dots, U(A, w_N)\big) \quad (4)$$

Why Use the Minimum Utility? The minimum aggregator enforces team-level acceptability, ensuring no architect is significantly dissatisfied with the proposed design.
Average utility could mask dissatisfaction by allowing high-satisfaction architects to overshadow those with conflicting priorities; using min avoids masking and promotes robust, consensus-oriented solutions.

c) Fitness of a Preference Model

Each preference model p_i is evaluated based on how well its weight vector aligns with high-quality architectures located on the current Pareto front:

$$\text{Fitness}(p_i) = \sum_{A \in \text{PF}_{\text{current}}} U(A, w_i) \quad (5)$$

A higher value means the preference model identifies solutions consistent with the evolutionary search trends.

d) Coevolutionary Loop Dynamics

At each generation:

1. A representative preference model is sampled from each P_{pref_i} (typically the best-performing model).
2. Each architecture in P_{arch} is evaluated using these representative preferences (see Eq. 3 and Eq. 4).
3. Each preference model receives feedback from how well its weights align with high-performing architectures (see Eq. 5).

This iterative process forms a closed coevolutionary loop, visually represented in Figure 2.

e) Algorithm Parameters

The cooperative coevolutionary algorithm uses the following settings for all case studies:

1. Population Sizes: P_{arch}: 100 individuals, each P_{pref_i}: 20 individuals
2. Generations: 500 (or until user-evaluation cap for ArchEvolve-NoLearn)
3. Crossover Rate: 0.9 (Simulated Binary Crossover for P_{arch}, and uniform crossover for P_{pref_i} which are weight vectors).
4. Mutation Rate: 0.1 (Polynomial Mutation for P_{arch}, and Gaussian mutation for P_{pref_i}).

3.1.2. Flexible Interactive Mechanism

ArchEvolve empowers the architect team with multiple ways to interact with the search process at any point during evolution:

- Guided Evaluation: This is the core interaction mode. At specific intervals, the algorithm presents a small, diverse subset of solutions to the team. Each architect provides a feedback rating from 1–5, which is then used to update the fitness of individuals in their corresponding P_{pref_i}. Algorithm 1 provides the pseudo-code for this process.
- Solution Injection: Architects may pause the search and manually create or modify an architecture, injecting it directly into P_{arch}. This allows domain expertise and creativity to seed the population with promising solutions.
- Objective Weighting: Architects can dynamically adjust the importance of different quality objectives. These adjustments directly influence the fitness calculation for individuals in P_{arch}.

Each architect provides a rating $R \in [1,5]$. The rating is normalized to a $[0,1]$, scale as $R_{\text{norm}} = R - R_{\text{min}}\ / R_{\text{max}} - R_{\text{min}}$, where $R_{min} = 1$ and $R_{max} = 5$. The normalized rating R_{norm} is then used to update the fitness of individuals in their respective P_{pref_i}. The fitness of a preference model p is directly linked to how accurately its calculated utility $U(A, p.weights)$ predicts the normalized human rating R_{norm} for the reviewed architectures. Below is the pseudo-code illustrating the user interaction logic:

Algorithm 1. Guided Evaluation and Feedback Integration

INPUT: Architecture population P_{arch}; reference populations $\{P_{pref_1}, \dots, P_{pref_N}\}$
OUTPUT: Updated preference populations; Set of normalized feedback ratings

function GuidedEvaluation$(P_{arch}, \{P_{pref_1}, \dots, P_{pref_N}\})$:
// 1. Select a diverse subset of solutions for review
$S_{review} \leftarrow$ SelectDiverseSubset$(P_{arch}, \mathrm{k} = 5)$
// 2. Present solutions and gather feedback from each architect
for i from 1 to N: // For each architect
    $feedback_i = \{\}$
    for A in S_{review}:
        $rating \leftarrow$ PresentToArchitect$(A, architect_i)$ // e.g., rating from 1-5
        $R_{\text{norm}} \leftarrow (R-1)\ /4$//Normalize rating to $[0,1]$
        feedback$_i$.add$((A, R_{\text{norm}}))$
// 3. Update the preference population for architect i
UpdatePreferencePopulation$(P_{pref_1}, feedback_i)$
function UpdatePreferencePopulation$(P_{pref}, feedback)$:
// Fitness is based on accuracy: lower error means higher fitness
for p in P_{pref}:
    $p.error_{sum} = 0$
    for (A, R_{norm}) in feedback:
        $predicted_{utility} = U(A, p.weights)$
// Calculate Absolute Error
$Error = |R_{norm} - predicted_{utility}|$
$p.error_{sum} += Error$
// Fitness is maximized when error is minimized
$p.fitness = 1 / (1 + p.error_{sum})$

3.1.3. ANN-Based Preference Learner

To reduce user fatigue, ArchEvolve integrates an ANN that functions as a surrogate evaluator for the architect team. The ANN takes the feature vector of a candidate architecture as input and outputs a predicted rating for that architecture. The implemented ANN is a fully connected feedforward network with the following configuration:

- Input Layer: M neurons, corresponding to the number of quality objectives in $Q(A)$.
- Hidden Layers: two hidden layers, each containing 32 neurons.
- Activation Function: Rectified Linear Unit (ReLU) for both hidden layers.
- Output Layer: A single neuron with a sigmoid activation function to produce a normalized predicted rating in $[0,1]$, which is later rescaled to the 1–5 user rating range.
- Loss Function: Mean Squared Error (MSE), as shown in Equation (7).
- Optimizer: Adam optimizer with a learning rate of 0.001.

An ANN is chosen due to its scalability, computational efficiency, and fast inference time—features particularly important for an interactive system. Although other models such as Gaussian Processes could offer more principled uncertainty estimation, they typically scale poorly with large datasets, making them less suitable for continuous online updates.

A sigmoid output is used to constrain predictions to $[0,1]$. This design choice supports stability during training and simplifies integration with rating normalization. While a linear output layer would be more expressive for continuous regression, the sigmoid-based normalization is preferred because the human-provided ratings themselves are discrete. The use of 0–1 normalization helps maintain consistency with the interactive rating input and avoids unstable outputs.

Mathematically, the ANN surrogate model maps an architecture's quality vector to a predicted team rating:

$$\hat{y} = f_{ANN}(Q(A); \theta) \tag{6}$$

Where θ denotes the trainable parameters of the ANN (weights and biases).

The ANN is trained using a dataset of K human-provided ratings, minimizing the following MSE loss:

$$L(\theta) = \frac{1}{K}\sum_{j=1}^{K}\left(y_j - \hat{y}_j\right)^2 \tag{7}$$

Here y_j is the true rating provided by the architect team for architecture A_j, $\hat{y}_j = f_{ANN}\left(Q(A_j);\theta\right)$ is the predicted rating.

To estimate prediction confidence and decide when human feedback is needed, we employ Monte Carlo Dropout. During inference, dropout layers remain active, effectively sampling E different stochastic variants of the ANN. Applying dropout masks multiple times produces slightly different predictions for the same architecture.

For a new architecture A_{new} the prediction variance across the E samples serves as the uncertainty measure:

$$\text{TriggerHumanFeedback} = \left(Var\{f_{ANN,1}(Q(A_{\text{new}})), \ldots, f_{ANN,E}(Q(A_{\text{new}}))\} > \tau_{\text{conf}}\right) \tag{8}$$

Where $E = 10$ Monte Carlo samples, $\tau_{\text{conf}} = 0.05$

The ANN is initially trained on a small batch of human-provided ratings from the first few GuidedEvaluation rounds. For subsequent generations, the ANN continuously makes predictions. If the variance of these predictions for a new candidate architecture A_{new} exceeds τ_{conf}, indicating high uncertainty, the system explicitly requests human feedback via Guided Evaluation. Additionally, to ensure a regular flow of fresh data and prevent model drift, a human interaction is always triggered every 10 generations, regardless of the τ_{conf} threshold. When new human ratings are collected, the ANN is immediately retrained (online) using the accumulated dataset of all explicit human feedback provided so far, strengthening its predictive capability.

Figure 3 summarizes the active learning cycle as follows: The system reduces user fatigue by employing an ANN surrogate to predict solution utility. When the surrogate's confidence is low, it triggers an active learning loop, querying the user via Guided Evaluation to gather data for re-training. Also with high value confidence, the system engages the surrogate's predictions directly by automating the search process and thereby reducing the need for user interaction.

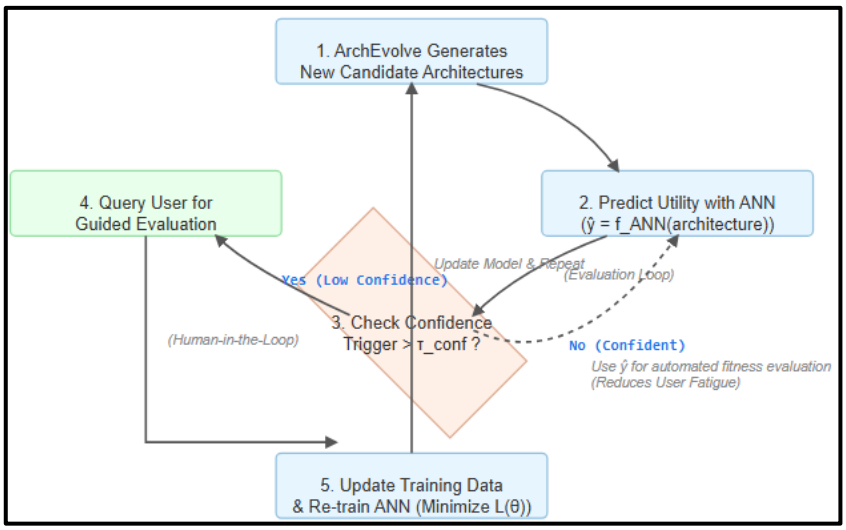

Figure 3. The ANN-driven active learning cycle

3.2. Experimental Setup

3.2.1. Simulated Case Studies: E-Commerce System (ECS) and Healthcare Management System (HMS)

To evaluate the effectiveness of ArchEvolve, two simulated case studies were used: an ECS and an HMS. Both systems are modeled programmatically rather than based on real

deployed systems. This simulation-based approach allows controlled experimentation, reproducibility, and the ability to embed realistic architectural trade-offs directly into the environment.

a) E-Commerce System (ECS)

The ECS is modeled as a component-based architecture with modules such as:

- Frontend/User Interface: Components for user interaction (browsing products, carts, checkout).
- Product Catalog Service: Manages product information, inventory.
- Order Management Service: Handles order creation, processing, and status.
- Payment Gateway Service: Integrates with payment providers.
- User Account Service: Manages user profiles, authentication, and authorization.
- Database: Stores system data (products, users, orders).
- Recommendation Engine (Optional): Suggests products to users.

The design space ("approximately 10×12 architectural choices") reflects ten decision variables, each with multiple discrete options. Quality objectives and their simulation methods are summarized in Table 2, and normalized using Equation (1).

Table 2. Quality objectives and their simulated calculation methods.

Quality Objective	Description	Simulated Calculation Method / Reference
Performance	How quickly the system responds to user requests and processes transactions.	Simulated: Typically, this would be a calculated value based on the chosen architectural components (for example, microservices vs. monolithic, database type, caching mechanisms). In a simulation, a function would assign a performance score based on the combination of selected architectural choices. For example, a design with more distributed components and efficient database access might receive a higher (better) performance score. The paper normalizes this such that higher values are better, see Equation (1).
Security	The system's ability to protect data and prevent unauthorized access or attacks.	Simulated: Scores are assigned based on security-related architectural decisions. For instance, designs incorporating robust authentication protocols, encrypted data transmission, and secure API gateways would yield higher security scores. The specific simulation function would map architectural configurations to security levels. The paper normalizes this such that higher values are better, see Equation (1).
Cost	The monetary expenses associated with developing, deploying, and maintaining the architecture (example, infrastructure, licensing, development effort).	Simulated: Costs are derived from the resources and complexity implied by architectural choices. A design relying on expensive proprietary software, large server farms, or complex integration patterns would have a higher raw cost. The paper normalizes this such that lower raw values are better, by transforming it, see Equation (1).
Reliability	The probability that the system will perform its intended functions without failure for a specified period.	Simulated: Scores are generated based on architectural patterns that enhance fault tolerance, redundancy, and error handling. For example, architectures with built-in replication, load balancing, and circuit breakers would likely receive higher reliability scores in the simulation. The paper normalizes this such that higher values are better, see Equation (1).
Modifiability	The ease with which the system can be changed or adapted to new requirements.	Simulated: This objective reflects how easily new features can be added or existing ones modified without significant effort or risk. Architectures promoting loose coupling, high cohesion (e.g., microservices), and clear module interfaces would be assigned higher modifiability scores. The paper normalizes this such that higher values are better, see Equation (1).

Simulated Architectural Model and Objective Mapping

To ensure reproducible experimentation while still capturing realistic architectural trade-offs, the transformation from a candidate architecture A (a vector of design decisions) into its quality vector $Q(A)$ is modeled using a deterministic multi-criteria simulation function, f_{sim}. Each architecture A consists of K design choices (e.g., $K = 10$ for ECS). Each design choice C_k is represented as an integer corresponding to one of its discrete configuration options.

For example, $C_1 = 3$ might denote "Microservices architecture using PostgreSQL." The raw score for objective j denoted as raw_{value_j} is calculated using a weighted linear combination of design choices. Here, $W_{(k,j)}$represents the influence of decision C_k on objective j, and a small controlled noise term ϵ is added to simulate uncertainty:

$$raw_{value_j} = \sum_{k=1}^{K} W_{(k,j)} \cdot C_k + \epsilon \tag{9}$$

Illustrative Example (ECS Performance)

For the ECS, the performance objective $(j = 1)$ may be primarily influenced by the database choice (C_{DB}) and the communication protocol (C_{Proto}):

$$raw_{performance} = W_{(\text{DB,Perf})} \cdot C_{\text{DB}} + W_{(\text{Proto,Perf})} \cdot C_{\text{Proto}} + \cdots \tag{10}$$

In this simulation setup, the W matrices are predefined to model realistic trade-offs. For example, selecting a highly scalable NoSQL option (high C_{DB}) yields a strong positive weight for performance but may introduce negative weight contributions to Modifiability or Cost. This helps embed inherent conflicts essential for multi-objective optimization. The noise term ϵ reflects real-world uncertainty, preventing the optimization landscape from becoming overly deterministic.

Once computed, each raw_{value_j} is normalized using Equation (1) (the min–max formulas) to produce the final normalized score $q_j \in Q(A)$.

Simulated Quality Scores and Normalization

As described earlier, each architecture A produces a quality vector $Q(A)$ consisting of M objective values. These values are:

- Generated programmatically using the simulation function.
- Normalized to $[0,1]$, where 1 represents the most desirable outcome.
- Transformed using the formulas in Equation (1), depending on whether the objective is to be maximized or minimized.

Here, $\min\left(raw_{value_j}\right)$ and $\max\left(raw_{value_j}\right)$ are the minimum and maximum possible scores for objective j within the simulated design space, as defined by the environment.

Simulated Architects and Preference Weights (ECS)

The ECS scenario includes a team of three simulated architects, each with predefined and intentionally conflicting preferences:

- Architect 1 (Prioritizes Costs & Performance): $w_1 = [0.3, 0.1, 0.4, 0.1, 0.1]$ (Performance, Security, Cost, Reliability, Modifiability)
- Architect 2 (Prioritizes Security & Reliability): $w_2 = [0.1, 0.4, 0.1, 0.3, 0.1]$ (Performance, Security, Cost, Reliability, Modifiability)
- Architect 3 (Prioritizes Modifiability & Performance): $w_3 = [0.2, 0.1, 0.1, 0.1, 0.5]$ (Performance, Security, Cost, Reliability, Modifiability)

Simple Architecture Diagram (Illustrative for ECS)

A conceptual high-level representation of the ECS simulation model is shown in Figure 4.

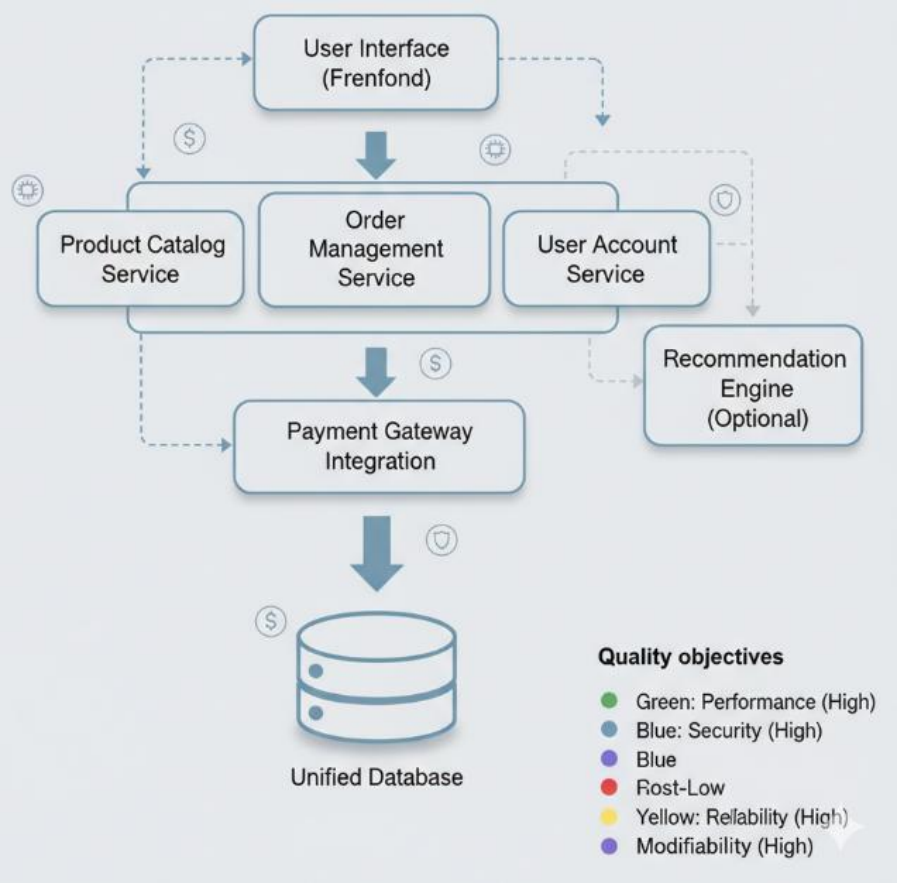

Figure 4. Simple Architecture Diagram for Simulated ECS

b) Healthcare Management System (HMS)
System Structure and Main Components (Simulated)

The Healthcare Management System (HMS) is also modeled as a component-based system but features a larger design space and more quality objectives than the ECS. This reflects the inherently higher complexity of healthcare systems. A typical HMS may include the following components:

- Patient Registration & Admissions – Manages patient demographics and admissions.
- Electronic Health Records (EHR) Management – Stores and retrieves patient health information.
- Appointment Scheduling – Coordinates appointments between doctors and patients.
- Billing & Insurance Processing – Handles financial workflows and insurance claims.
- Laboratory Information System (LIS) Integration – Interfaces with laboratory equipment and test results.
- Pharmacy Management Integration – Manages prescription orders and medication inventory.
- Reporting & Analytics – Generates analytical reports from healthcare data.
- Security & Compliance Module – Ensures adherence to regulations such as HIPAA.

The larger design space indicates that the HMS simulation uses more architectural variables or more configuration options per variable compared to the ECS, producing a more complex and challenging optimization scenario. Each quality objective for the HMS is generated programmatically, using simulation functions designed to reflect realistic architectural trade-offs. A summary is shown in Table 3.

Simulated Quality Scores and Normalization

Similar to the ECS setup, the HMS simulation generates raw quality scores for each architecture A, which are then normalized to the $[0,1]$ range using the formulas in Equation

(1). The values $\min\left(raw_{value_j}\right)$ and $\max\left(raw_{value_j}\right)$ for each objective j are determined from within the simulated HMS design environment.

Table 3. Quality objectives and their simulated calculation methods.

Quality Objective	Description	Simulated Calculation Method / Reference
Performance	System responsiveness and throughput for critical operations (e.g., fetching records, processing lab results).	Scores reflect architectural factors such as server capacity, database optimizations, and network configuration. Higher scores indicate better performance (normalized using, see Equation (1).
Security	Protection of sensitive patient data from breaches and cyber threats.	Generated based on architectural inclusion of RBAC, encryption, intrusion detection, and secure APIs. Higher values are better, see Equation (1).
Cost	Development, deployment, and operational costs under healthcare constraints.	Costs arise from specialized hardware, compliance tools, and complex integrations. Higher raw values indicate higher cost and are transformed such that lower cost yields higher normalized utility, see Equation (1).
Reliability	Continuity and correctness of system operations, crucial for patient safety.	Scores reflect high availability, redundancy, fault tolerance, and error-handling mechanisms, see Equation (1).
Interoperability	Ability to exchange information with external systems (labs, pharmacies, other hospitals).	Architectures adopting standards such as HL7 FHIR or robust APIs receive higher scores, see Equation (1).
Privacy	Protection of sensitive patient information and compliance with privacy regulations.	Based on support for data minimization, anonymization, access control, and audit mechanisms, see Equation (1).

Simulated Architects and Preference Weights (HMS)

A team of three simulated architects is used, each with predefined and intentionally conflicting preferences to create a realistic multi-stakeholder decision environment:

- Architect 1 (Prioritizes Costs & Performance): $w_1 = [0.3, 0.1, 0.3, 0.1, 0.1, 0.1]$ (Performance, Security, Cost, Reliability, Interoperability, Privacy)
- Architect 2 (Prioritizes Security & Reliability): $w_2 = [0.1, 0.3, 0.1, 0.3, 0.1, 0.1]$ (Performance, Security, Cost, Reliability, Interoperability, Privacy)
- Architect 3 (Prioritizes Interoperability & Privacy): $w_3 = [0.1, 0.1, 0.1, 0.1, 0.3, 0.3]$ (Performance, Security, Cost, Reliability, Interoperability, Privacy)

These conflicting weight vectors are deliberately structured to introduce complex preference interactions, making HMS a robust testing ground for consensus-building.

Simple Architecture Diagram (Illustrative for HMS)

A high-level conceptual architecture for the simulated HMS is shown in Figure 5, highlighting the major components and interactions modeled in the simulation. These conflicting architectural preferences and the larger design space create a significantly challenging multi-objective optimization problem, reinforcing the importance of consensus-oriented frameworks such as ArchEvolve.

3.2.2 Baseline for Comparison

ArchEvolve is compared against an interactive Many-Objective Evolutionary Algorithm (MaOEA), R-NSGA-II, which is a reference-point-based extension of NSGA-II. For this baseline, the preferences of the three simulated architects were averaged to form a single reference point for each interaction. The averaging was performed using the arithmetic mean of their individual weight vectors. Specifically, if w_1, w_2, w_3 denote the weight vectors of the

three architects, the averaged vector was computed as $w_{avg} = (w_1 + w_1 + w_1)/3$. This averaged preference vector served as the single reference point that guided R-NSGA-II during the search, effectively representing a "compromise" preference intended to approximate a single decision-maker.

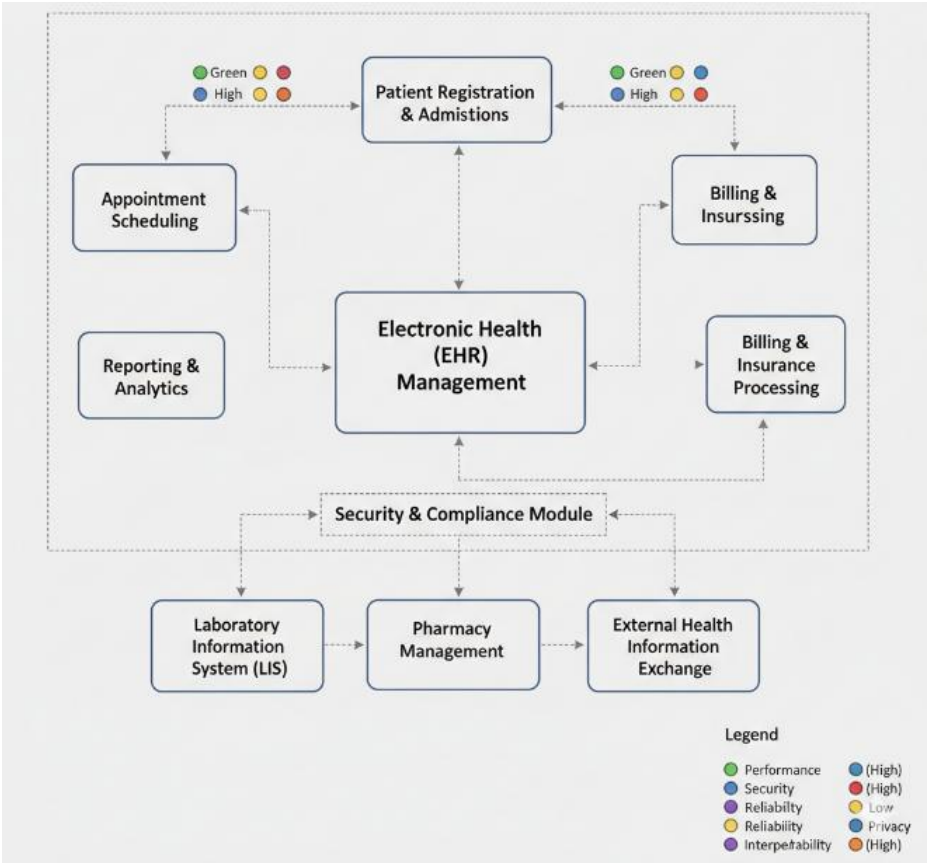

Figure 5. Simple architecture diagram for simulated HMS

3.2.3. Experimental Protocol

For each case study, two variants of the proposed framework were executed:

- ArchEvolve-Full: The complete framework with the ANN-based preference learner enabled.
- ArchEvolve-NoLearn: A variant without the ANN learner, requiring explicit human feedback at every interaction step.

A total of 30 independent runs were conducted for each algorithm on both case studies. To simulate realistic user fatigue constraints, the number of allowed human evaluations was capped.

- For the E-Commerce System (ECS), the cap was set to 200 evaluations.
- For the Healthcare Management System (HMS), the cap was set to 260 evaluations.

These thresholds were chosen based on preliminary experiments to reflect a practical limit for prolonged human interaction while still allowing ArchEvolve-NoLearn to converge before reaching its evaluation limit, as reflected in the capped values reported in Table 3.

3.2.4. Metrics

To evaluate the performance of ArchEvolve and answer the research objectives, several quantitative metrics were employed.

a) Metrics for first objective: Consensus

Two metrics were used to assess solution quality and team consensus: Hypervolume (HV) and the Consensus Metric (CM). HV measures both the convergence and diversity of the final Pareto front [25]. Higher values indicate better coverage of the objective space. Given a set of solution points $S = \{s_1, s_2, \dots, s_n\}$ and a reference point r dominated by all solutions, HV is defined as:

$$HV(S,r) = \text{volume}\left(\bigcup_{i=1}^{n}[s_i, r]\right) \tag{11}$$

The CM evaluates how close the final solution set is to the ideal preference points of each architect. Lower values indicate stronger consensus[26]. For each architect i, an Aspirational Individual Ideal Point I_i is defined. Unlike the standard Utopia point, I_i assigns: 1.0 to objectives with the highest preference weight in w_i 0.0 to all other objectives. Formally:

$$I_{i,j} = \begin{cases} 1.0 & \text{, if } w_{i,j} = \max(w_i) \\ 0.0 & \text{, otherwise} \end{cases} \tag{12}$$

For each final solution A_k with quality vector $Q(A_k) = [q_{k,1}, \ldots, q_{k,M}]$, the Euclidean distance to architect i's ideal point is:

$$D(A_k, I_i) = \sqrt{\sum_{j=1}^{M}\left(q_{k,j} - I_{i,j}\right)^2} \tag{13}$$

CM is the average distance across all solutions and all architects:

$$CM = \frac{1}{nN}\sum_{k=1}^{n}\sum_{i=1}^{N} D(A_k, I_i) \tag{14}$$

This formulation captures how close the final solution set is to the diverse, and often conflicting, aspirations of all architects, making CM a robust indicator of team-wide consensus.

b) Metrics for second objective: Fatigue Reduction

Two additional measures were used to evaluate ArchEvolve's ability to reduce the burden of human interaction:

- Number of Interactions (NI): NI counts the total number of times the system required explicit human feedback before reaching a stable and high-quality solution set. Lower NI indicates reduced user fatigue.
- ANN Prediction Accuracy: This metric measures the ANN learner's ability to accurately predict architect ratings on a holdout test set. Higher accuracy indicates better surrogate performance and fewer required human interactions.

3.3. Implementation Details and Parameters

The ArchEvolve framework was fully implemented in Python 3.9. Its system architecture is composed of a backend, frontend, and analysis environment that work together to support interactive evolutionary search and real-time preference learning.

On the backend, the framework uses FastAPI to expose lightweight API endpoints for communication between components. Evolutionary search operations— including the Cooperative Coevolution design and the NSGA-II baseline—were implemented using the Pymoo library, whose extensibility allows seamless integration of custom evolutionary operators. The preference learning module was developed using PyTorch, enabling efficient training and inference of the Artificial Neural Network (ANN) model.

The frontend was developed in React, providing a responsive and modular interface for interactive evaluations. Visualization components—used for inspecting architectures, reviewing solution sets, and guiding user evaluations—were implemented using D3.js, which supports dynamic and interactive charting essential for the guided evaluation workflow.

All experimental logs, analyses, and plots were generated in Jupyter Notebooks using Pandas, NumPy, and Matplotlib, ensuring reproducibility and transparent interpretation of the results. Experiments were executed on a server equipped with an 8-core Intel i7 CPU, 32 GB RAM, and an NVIDIA GeForce RTX 3070 GPU, the latter accelerating training of the ANN surrogate model.

The evolutionary algorithm settings were kept consistent across both case studies. The architecture population consisted of 100 candidate solutions per generation, while each architect's preference population contained 10 individuals. The evolutionary process was run for up to 200 generations for the ECS case study and 260 generations for HMS, or until the human evaluation cap described in Section 3.2.3 was reached. For statistical reliability, 30 independent runs were executed per algorithm and per case study.

Evolutionary variation operators followed standard Pymoo defaults with appropriate modifications for this problem. Simulated Binary Crossover (SBX) was applied with a probability of 0.9, and Polynomial Mutation was applied with probability $1/M$, where M denotes the number of objectives, using a distribution index of 20.

The ANN preference learner was configured as a fully connected feedforward network with two hidden layers of 32 neurons, each using a ReLU activation function. The network was trained using MSE loss and the Adam optimizer with a learning rate of 0.001, typically for 50 epochs each time new human feedback was added. For uncertainty estimation, Monte Carlo Dropout was applied with $E = 10$ stochastic forward passes. A confidence threshold of $\tau_{\text{conf}} = 0.05$ was adopted; values above this threshold triggered additional human evaluations. This threshold provided a practical balance between reducing user fatigue and ensuring adequate human guidance when ANN predictions were uncertain.

4. Results and Discussion

4.1. Results

This section presents the experimental results obtained from the two case studies. Statistical significance was evaluated using the Wilcoxon Rank-Sum test with a significance threshold of $p < 0.05$. The comparison of Hypervolume (HV) and Consensus Metric (CM) across algorithms is summarized in Table 4. Across both case studies, ArchEvolve-Full consistently outperforms R-NSGA-II, achieving higher HV values—indicating better convergence and diversity—and lower CM values, demonstrating stronger agreement with the preference profiles of all architects.

Table 4. Comparison of Hypervolume and Consensus Metric (averaged over 30 runs).

Algorithm	Case Study	Hypervolume (HV)	Consensus Metric (CM)
ArchEvolve-Full	ECS	0.89	0.15
R-NSGA-II	ECS	0.76	0.32
ArchEvolve-Full	HMS	0.82	0.21
R-NSGA-II	HMS	0.69	0.45

The improvements in CM across both ECS and HMS indicate that ArchEvolve-Full is more effective at navigating the conflicting preferences of multiple architects, producing solutions that lie closer to their aspirational ideal points (Eq. 12–14).

The evaluation of fatigue reduction is reported in Table 5. The ArchEvolve-Full variant required significantly fewer human interactions compared to ArchEvolve-NoLearn, staying well below the evaluation caps discussed in Section 3.2.3. The ANN learner achieved prediction accuracies above 90% for both case studies, enabling it to reliably automate evaluations and reduce user burden.

Table 5. Comparison of Fatigue Reduction (averaged over 30 runs).

Algorithm	Case Study	Avg. No. of Interactions (NI)	ANN Accuracy
ArchEvolve-Full	ECS	112	92.5%
ArchEvolve-NoLearn	ECS	200 (capped)	N/A
ArchEvolve-Full	HMS	135	90.1%
ArchEvolve-NoLearn	HMS	260 (capped)	N/A

These results highlight the effectiveness of the ANN-based surrogate model in reducing human workload while preserving solution quality.

Although the core contribution of ArchEvolve lies in its cooperative coevolutionary algorithm and ANN-based preference learning module, its practical usefulness largely depends on the design of an intuitive and collaborative interface. To support real-world applicability, ArchEvolve was implemented as a web-based decision support system using React for the frontend and D3.js for dynamic visualizations. This interface-centered design helps reduce cognitive load and makes human-in-the-loop guidance efficient and accessible to architect teams.

The ArchEvolve dashboard (Figure 6) serves as the main workspace where architects monitor the search progress, examine architectural trade-offs, and observe team consensus development.

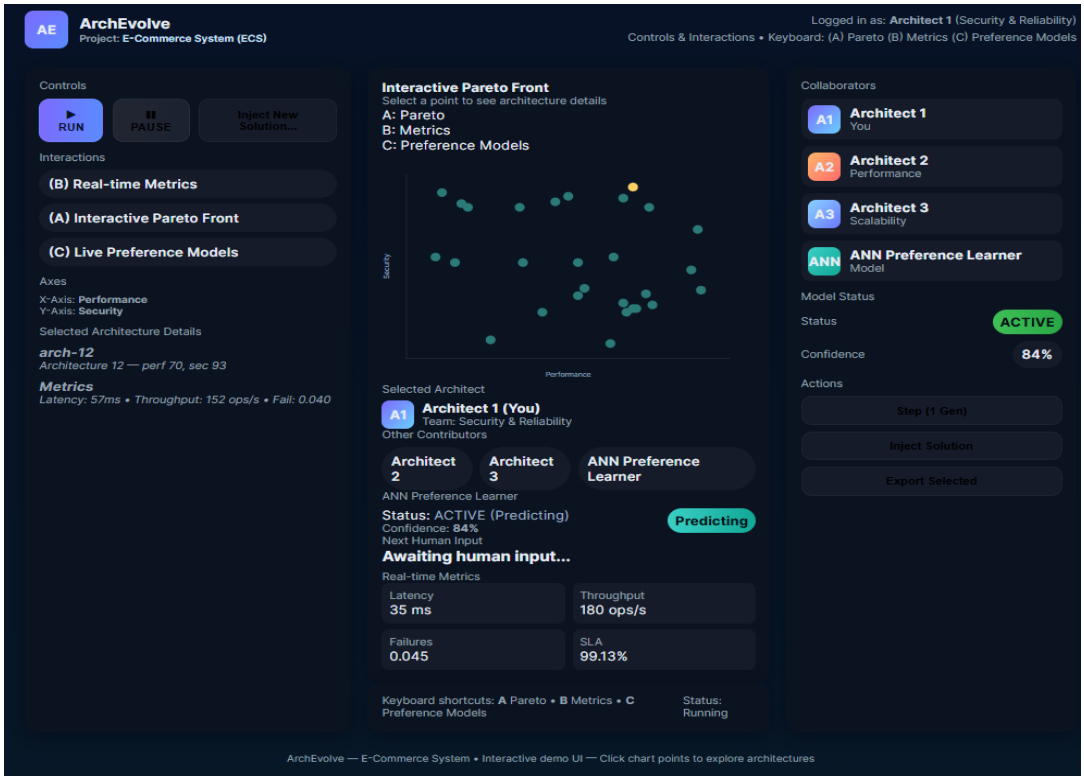

Figure 6. The ArchEvolve dashboard

The dashboard provides a clear visualization of the current population of candidate architectures in the normalized objective space, including the evolving Pareto front and highlighted high-consensus solutions. It also displays historical metrics—such as the progression of Hypervolume and Consensus Metric—allowing the team to observe convergence patterns over time. Additionally, the dashboard shows the learned preference weights (w_i) for each architect, increasing transparency and enabling teams to better understand how individual preferences influence the search.

The Guided Evaluation Interface is the primary mechanism through which architects provide explicit feedback for training the preference learner and influencing the coevolutionary process. The Guided Evaluation interface (Figure 7) is triggered either by the Active Learning Protocol (due to low ANN confidence) or at predefined generation intervals. When activated, the algorithm presents a small, diverse subset of solutions, S_{review}. Architects utilize this interface to provide their subjective assessment for each presented solution A using a rating scale (1–5). This explicit team feedback—which is normalized to R_{norm}—is immediately used to retrain the f_{ANN} surrogate model and update the fitness of the individuals in their respective P_{pref_i} preference populations (as detailed in Algorithm 1).

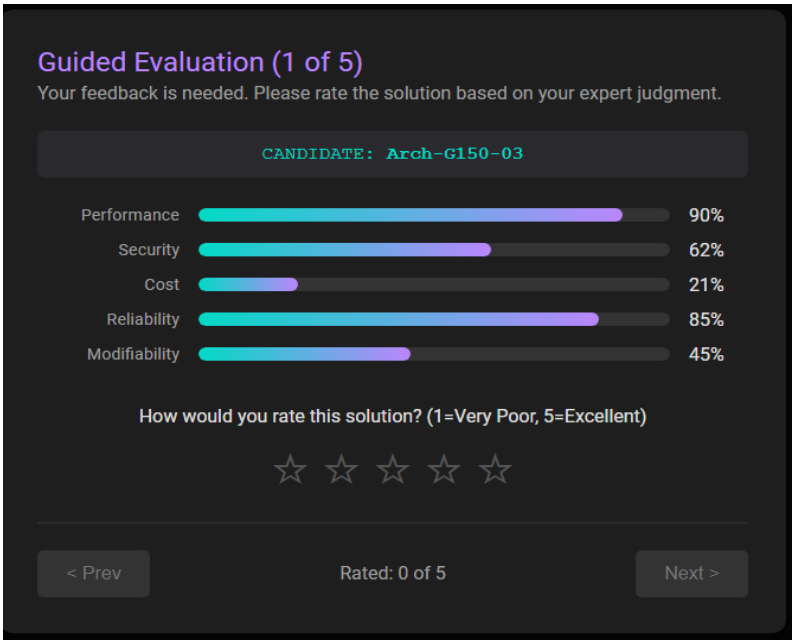

Figure 7. The guided evaluation interface

To complement automated search and guided evaluation, ArchEvolve also provides mechanisms for direct human intervention, enabling architects to proactively steer the optimization process.

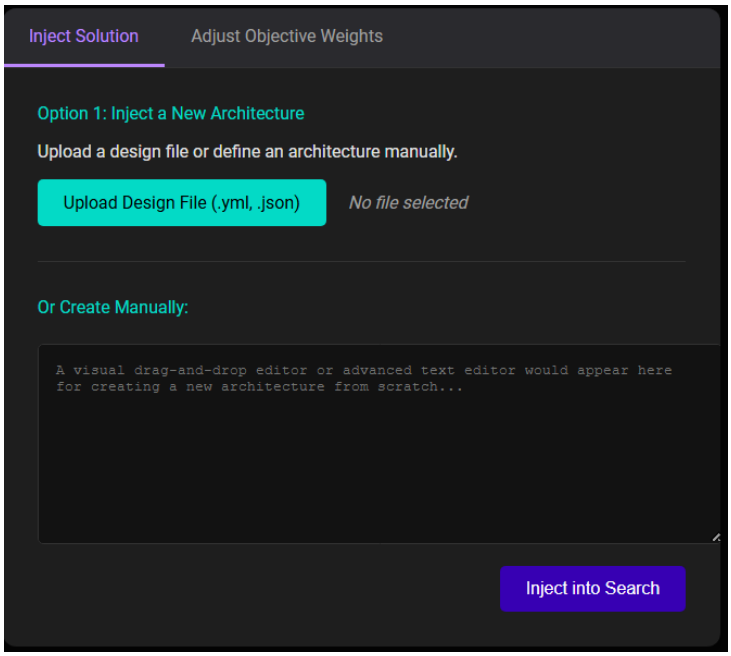

Figure 8. The manual intervention interface

The interface supports two key forms of manual control:

- Solution Injection: Architects can pause the evolutionary search at any point and manually design a new architecture or modify an existing candidate solution based on their domain expertise and tacit knowledge. This promising solution is then injected into the P_{arch} population, effectively seeding the search with human creativity.
- Objective Weighting: This mechanism allows architects to dynamically adjust the relative importance of the quality objectives. By changing their preference weight vectors (w_i), they directly influence the utility calculation (Equation 3) and thus immediately affect

the fitness landscape for P_{arch} in subsequent generations. This dynamic capability enables the team to adapt quickly to changing project requirements or newly discovered trade-offs.

4.2. Discussion

4.2.1. Addressing the Research Objectives

With respect to the first objective, the results clearly demonstrate that the cooperative coevolutionary model is more effective at identifying consensus-oriented architectural solutions than the baseline. As shown in Table 4 and Figure 9, treating each architect's preference model as a dedicated co-evolving population enables the search to more faithfully preserve individual preference structures. This approach avoids the information loss caused by averaging preferences into a single reference vector, as done in R-NSGA-II, and leads to solutions that better balance the needs of multiple stakeholders.

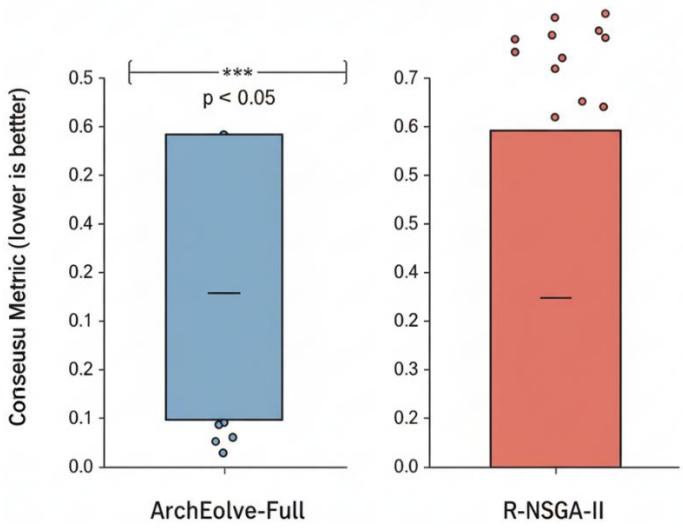

Figure 9. Box plots showing the distribution of the Consensus Metric for ArchEvolve vs. R-NSGA-II on the HMS case study.

Regarding the second objective, the integration of the ANN-based preference learner proves highly successful in reducing user fatigue. ArchEvolve-Full achieved prediction accuracies above 90%, allowing it to reach high-quality solutions with substantially fewer required human interactions—approximately 44% fewer for ECS and 46% fewer for HMS, as shown in Table 5.

The reduction in interaction frequency arises from the stability and predictability of architect preference patterns. Although individual evaluations may seem subjective or complex, architects tend to exhibit consistent tendencies when assessing trade-offs. The ANN effectively captures these patterns from the guided evaluation rounds. Once trained, it functions as a surrogate evaluator capable of predicting the team's consensus rating for new architectural solutions. Additional human input is requested only when the ANN's confidence is low—as determined by the Monte Carlo Dropout variance threshold (τ_{conf})—or at predefined active learning intervals. This targeted use of human feedback significantly reduces cognitive load and eliminates repetitive evaluation tasks.

Interactive evolutionary computation traditionally requires extensive human participation, which can lead to fatigue and diminishing feedback quality over time. Without preference learning (as in ArchEvolve-NoLearn or traditional IEC), achieving strong convergence or consensus typically demands a large number of evaluations. ArchEvolve's preference learner minimizes this burden by enabling the algorithm to continue searching effectively even when human evaluations are infrequent. The ANN's ability to preserve and propagate learned preferences ensures that solution quality remains high while the interaction load remains manageable. The active learning mechanism ensures that human effort is reserved for cases where it provides the most value, making the framework far more practical for real-world software engineering teams.

4.2.2. Practical Significance of the Metrics

A higher HV value indicates a stronger, more diverse Pareto front. Practically, this means that ArchEvolve provides architects with a broader set of well-balanced design alternatives, offering superior flexibility when addressing evolving project constraints. Higher HV implies that the search has effectively explored the design space and delivered a richer set of high-quality architectural solutions.

A lower CM value directly reflects better agreement among architects. In practical terms, it indicates that the final solution set is closer to the individual aspirational ideal points of all team members. Achieving lower CM reduces internal conflict, accelerates decision-making, and increases the likelihood of solution acceptance—factors that are critical for maintaining project momentum and long-term architectural stability.

NI represents the human effort required during the interactive optimization process. A lower NI is particularly meaningful in professional environments where architects must balance decision-making tasks with numerous other responsibilities. Reducing NI improves the sustainability of the process, minimizes fatigue, and allows architects to focus their attention on strategic decisions rather than repetitive evaluations. ArchEvolve's ability to dramatically reduce NI without compromising solution quality highlights its practical viability for industry use.

4.2.3. Implications for Practice

For practitioners, ArchEvolve offers a structured, semi-automated decision-support framework that enhances collaborative architectural design. The system transforms the design process into an iterative dialogue between architects and the optimization engine. The ArchEvolve Dashboard (Figure 6) and Guided Evaluation Interface (Figure 7), built with React and D3.js, prioritize usability and visual clarity, minimizing cognitive load. Features such as solution injection and dynamic objective weighting (Figure 8) empower architects to directly steer the search, helping maintain engagement and supporting more informed decision-making. This emphasis on usability is critical for adoption in real-world team settings.

4.2.4. Trade-off Visualization and Exploration

Although ArchEvolve focuses on identifying consensus-based solutions, it also facilitates deeper understanding of trade-offs among quality objectives. The Guided Evaluation interface (Figure 7) presents architects with diverse solution subsets, implicitly exposing the trade-offs within the current Pareto front. Visualizations of the final consensus Pareto front allow teams to collaboratively examine how architectural choices influence multiple quality attributes. The dynamic objective weighting feature further enables architects to interactively adjust priorities and observe how the evolutionary search responds in subsequent generations. This iterative, visually supported exploration helps teams internalize the trade-offs inherent in architectural design and converge on mutually acceptable solutions. A comparative visualization of Consensus Metric performance is shown in Figure 10, which illustrates the superiority of ArchEvolve relative to the baseline approach.

4.3. Threats to Validity

A rigorous assessment of our findings requires consideration of several potential threats to validity:

- Internal Validity.

The experimental setup relied on simulated architects with predefined preference vectors. While this was necessary to ensure reproducibility and maintain control over experimental variables, it introduces limitations when compared to the inherently unpredictable behavior of real human decision-makers. Real architects often exhibit dynamic, evolving, and sometimes inconsistent preferences, whereas our simulated preferences remain static throughout the experiments. Consequently, a full human-subject study would be required to fully validate the framework's performance under authentic human-in-the-loop conditions.

- External Validity.

ArchEvolve was empirically evaluated on two simulated case studies—an ECS and an HMS. Although these domains are representative of complex, multi-objective software systems, the findings may not generalize to all types of architectural design problems. Other

domains may involve different design spaces, objective structures, or team interaction patterns that could influence the framework's behavior. A wider range of case studies, including real-world deployments, would help strengthen the generalizability of the results.

- Construct Validity.

The CM, although formally defined and justified, represents only one approach to quantifying consensus through average Euclidean distance to ideal preference points. Alternative consensus formulations or aggregation strategies might produce different insights. Similarly, the NI is a practical and direct proxy for user fatigue, but it does not capture deeper cognitive load factors. More nuanced measures, such as physiological indicators or detailed subjective workload assessments—could provide a richer understanding of user fatigue. These represent deliberate construct choices, and future work could examine alternative or complementary metrics.

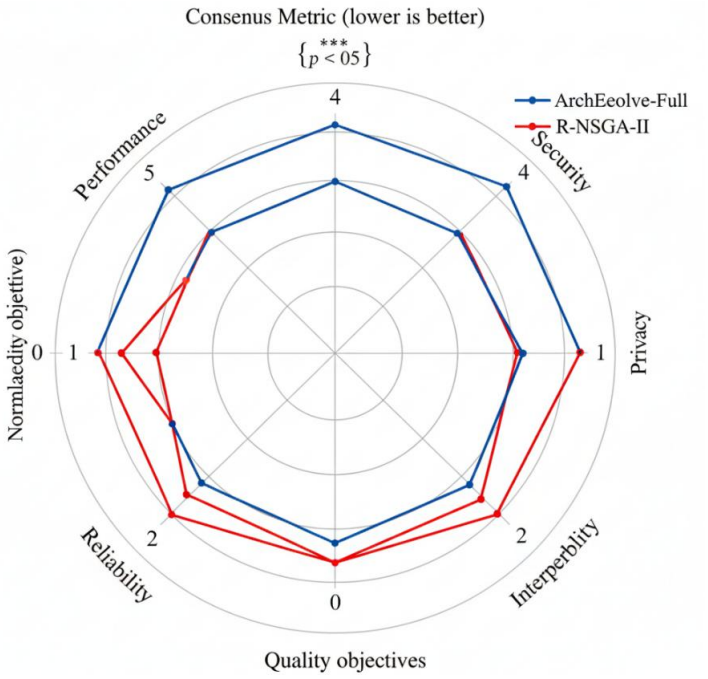

Figure 10. Radar chart illustrating the superiority of ArchEvolve in consensus metric.

5. Conclusions

This paper presented ArchEvolve, a collaborative and interactive framework for software architecture optimization. By integrating a cooperative coevolutionary algorithm with an ANN-based preference learner, ArchEvolve effectively addresses two persistent challenges in interactive SBSE: multi-architect collaboration and user fatigue. The empirical results demonstrate that the framework is capable of producing higher-quality, consensus-oriented architectural solutions while requiring significantly less human effort compared to a state-of-the-art baseline.

The primary contribution of this work is the development and validation of a cooperative coevolutionary framework that can navigate the complex and often conflicting preference landscape within multi-architect teams. ArchEvolve bridges the gap between automated optimization and collaborative human expertise by ensuring that team preferences remain actively represented in the search process. At the same time, its ANN-based preference learner intelligently reduces interaction load, making interactive optimization more feasible and efficient for real-world software engineering teams.

From a practical perspective, ArchEvolve offers clear value for industrial adoption. The framework can be integrated into existing architectural modeling tools (such as Enterprise Architect or ArchiMate-based platforms) or deployed as a standalone web system. By structuring the design process as a semi-automated dialogue between architects and the optimization engine, ArchEvolve enhances decision-making efficiency while promoting transparency

and collaboration. Features such as solution injection and dynamic objective weighting (Figure 5) further empower architects to steer the search, increasing engagement and ownership—both essential factors for successful adoption in industry.

Several limitations should be acknowledged. The use of simulated architects, although necessary for ensuring reproducibility, may not fully capture the nuanced and evolving behavior of real human decision-makers. The findings may also not generalize to all software domains, as only two simulated case studies (ECS and HMS) were evaluated. Additionally, while the Consensus Metric (CM) provides a well-defined measure of agreement among architects, alternative formulations could lead to different interpretations. The Number of Interactions (NI) is a meaningful proxy for fatigue, but a comprehensive human-subject study would be required to directly measure cognitive load and overall user experience.

Future work may explore several promising directions, including: (1) Conducting a large-scale human-subject study with professional software architects to validate the framework under realistic conditions. (2) Investigating more advanced preference learning models (e.g., Gaussian Processes) for better uncertainty estimation. (3) Extending ArchEvolve to support additional architectural activities, such as architecture evolution, refactoring, or runtime adaptation. (4) Performing a direct empirical comparison with advanced interactive SBSE tools such as the OPLA-Tool, with a focus on multi-architect decision-making scenarios.

Author Contributions: Conceptualization: A.E.M. and K.E.A.; Methodology: A.M.F.; Software: A.E.M.; Validation: A.E.M., A.M.F. and K.E.A.; Formal analysis: A.E.M.; Investigation: A.M.F.; Resources: K.E.A.; Data curation: A.E.M.; Writing—original draft preparation: A.E.M.; Writing—review and editing: A.M.F.; Visualization: K.E.A.; Supervision: K.E.A.; Project administration: A.E.M. All authors have read and agreed to the published version of the manuscript"

Funding: This research received no external funding.

Data Availability Statement: Data used will be made available on request.

Conflicts of Interest: There is no conflict of interest.

References

[1] P. Clements, D. Garlan, R. Little, R. Nord, and J. Stafford, *Documenting software architectures: views and beyond*, 2nd ed. Boston: Pearson Education, 2010.

[2] A. Mohamed and M. Zulkernine, "Architectural Design Decisions for Achieving Reliable Software Systems," in *Architecting Critical Systems*, 2010, pp. 19–32. doi: 10.1007/978-3-642-13556-9_2.

[3] A. El Murabet and A. Abtoy, "Methodologies of the Validation of Software Architectures," *J. Comput. Theor. Appl.*, vol. 1, no. 2, pp. 29–36, 2023, doi: 10.33633/jcta.v1i2.9332.

[4] L. Dobrica, "Exploring Approaches of Integration Software Architecture Modeling with Quality Analysis Models," in *2011 Ninth Working IEEE/IFIP Conference on Software Architecture*, Jun. 2011, pp. 113–122. doi: 10.1109/WICSA.2011.23.

[5] O. Sievi-Korte, I. Richardson, and S. Beecham, "Software architecture design in global software development: An empirical study," *J. Syst. Softw.*, vol. 158, p. 110400, Dec. 2019, doi: 10.1016/j.jss.2019.110400.

[6] S. R. V and H. Muccini, "Group decision-making in software architecture: A study on industrial practices," *Inf. Softw. Technol.*, vol. 101, pp. 51–63, Sep. 2018, doi: 10.1016/j.infsof.2018.04.009.

[7] M. Ö. Demir, O. Chouseinoglou, and A. K. Tarhan, "Factors affecting architectural decision-making process and challenges in software projects: An industrial survey," *J. Softw. Evol. Process*, vol. 36, no. 10, Oct. 2024, doi: 10.1002/smr.2703.

[8] M. Harman, P. McMinn, J. T. de Souza, and S. Yoo, "Search Based Software Engineering: Techniques, Taxonomy, Tutorial," in *Empirical Software Engineering and Verification*, 2012, pp. 1–59. doi: 10.1007/978-3-642-25231-0_1.

[9] J. Zhao, H. Zhang, H. Yu, H. Fei, X. Huang, and Q. Yang, "A many-objective evolutionary algorithm based on three states for solving many-objective optimization problem," *Sci. Rep.*, vol. 14, no. 1, p. 19140, Aug. 2024, doi: 10.1038/s41598-024-70145-8.

[10] M. Harman, "The Current State and Future of Search Based Software Engineering," in *Future of Software Engineering (FOSE '07)*, May 2007, pp. 342–357. doi: 10.1109/FOSE.2007.29.

[11] S. Dasanayake, J. Markkula, S. Aaramaa, and M. Oivo, "An Empirical Study on Collaborative Architecture Decision Making in Software Teams," in *Software Architecture*, 2016, pp. 238–246. doi: 10.1007/978-3-319-48992-6_18.

[12] A. Ramirez, J. R. Romero, and C. L. Simons, "A Systematic Review of Interaction in Search-Based Software Engineering," *IEEE Trans. Softw. Eng.*, vol. 45, no. 8, pp. 760–781, Aug. 2019, doi: 10.1109/TSE.2018.2803055.

[13] T. Houichime and Y. El Amrani, "Optimized design refactoring (ODR): a generic framework for automated search-based refactoring to optimize object-oriented software architectures," *Autom. Softw. Eng.*, vol. 31, no. 2, p. 48, Nov. 2024, doi: 10.1007/s10515-024-00446-9.

[14] M. Babar, S. Azeem, and F. Arif, "Integration of Security Non-Functional Requirements and Architectural Design: A Comparative Analysis," *Int. J. Secur. Its Appl.*, vol. 11, no. 10, pp. 45–54, Oct. 2017, doi: 10.14257/ijsia.2017.11.10.05.
[15] W. Lou *et al.*, "Unleashing Network/Accelerator Co-Exploration Potential on FPGAs: A Deeper Joint Search," *IEEE Trans. Comput. Des. Integr. Circuits Syst.*, vol. 43, no. 10, pp. 3041–3054, Oct. 2024, doi: 10.1109/TCAD.2024.3391688.
[16] H. Aljawawdeh, "An interactive metaheuristic search framework for software service identification from business process models," The University of the West of England, 2019. [Online]. Available: https://uwe-repository.worktribe.com/output/853842
[17] A. Rago, S. Vidal, J. A. Diaz-Pace, S. Frank, and A. van Hoorn, "Distributed quality-attribute optimization of software architectures," in *Proceedings of the 11th Brazilian Symposium on Software Components, Architectures, and Reuse*, Sep. 2017, pp. 1–10. doi: 10.1145/3132498.3132509.
[18] W. Freire, C. Rosa, A. Amaral, and T. Colanzi, "Validating an Interactive Ranking Operator for NSGA-II to Support the Optimization of Software Engineering Problems," in *Proceedings of the XXXVI Brazilian Symposium on Software Engineering*, Oct. 2022, pp. 337–346. doi: 10.1145/3555228.3555232.
[19] C. V. Bindewald, W. M. Freire, A. M. M. M. Amaral, and T. E. Colanzi, "Towards the support of user preferences in search-based product line architecture design," in *Proceedings of the XXXIII Brazilian Symposium on Software Engineering*, Sep. 2019, pp. 387–396. doi: 10.1145/3350768.3351993.
[20] C. V. Bindewald, W. M. Freire, A. M. M. M. Amaral, and T. E. Colanzi, "Supporting user preferences in search-based product line architecture design using Machine Learning," in *Proceedings of the 14th Brazilian Symposium on Software Components, Architectures, and Reuse*, Oct. 2020, pp. 11–20. doi: 10.1145/3425269.3425278.
[21] A. Hamza, W. Hussain, H. Iftikhar, A. Ahmad, and A. M. Shamim, "Evaluating Open-Source Machine Learning Project Quality Using SMOTE-Enhanced and Explainable ML/DL Models," *J. Comput. Theor. Appl.*, vol. 3, no. 2, pp. 206–222, Nov. 2025, doi: 10.62411/jcta.14793.
[22] F. H. Kuviatkovski, W. M. Freire, A. M. M. M. Amaral, T. E. Colanzi, and V. D. Feltrim, "Evaluating Machine Learning Algorithms in Representing Decision Makers in search-based PLA," in *2022 IEEE 19th International Conference on Software Architecture Companion (ICSA-C)*, Mar. 2022, pp. 68–75. doi: 10.1109/ICSA-C54293.2022.00057.
[23] C. Arasaki, L. Wolschick, W. Freire, and A. Amaral, "Feature selection in an interactive search-based PLA design approach," in *Proceedings of the 17th Brazilian Symposium on Software Components, Architectures, and Reuse*, Sep. 2023, pp. 11–20. doi: 10.1145/3622748.3622750.
[24] C. A. Akiotu *et al.*, "A Predictive Nutritional Assessment System for Vegetarians using Artificial Neural Networks," *J. Futur. Artif. Intell. Technol.*, vol. 2, no. 2, pp. 294–312, Aug. 2025, doi: 10.62411/faith.3048-3719-117.
[25] K. Yang, M. Emmerich, A. Deutz, and T. Bäck, "Multi-Objective Bayesian Global Optimization using expected hypervolume improvement gradient," *Swarm Evol. Comput.*, vol. 44, pp. 945–956, Feb. 2019, doi: 10.1016/j.swevo.2018.10.007.
[26] J. Goers and G. Horton, "On the Combinatorial Acceptability Entropy Consensus Metric for Multi-Criteria Group Decisions," *Gr. Decis. Negot.*, vol. 33, no. 5, pp. 1247–1268, Oct. 2024, doi: 10.1007/s10726-024-09891-z.