Computers, Materials & Continua

Tech Science Press

https://doi.org/10.32604/cmc.2026.084179

ARTICLE

QIMIG: A Quantum-Inspired Evolutionary Framework for Software Library Migration

Yun Liu[1], Jinghua Zhao[1], Liang Ma[1], Zijie Huang[2,3,*], Lizhi Cai[2,3] and Jianxin Ge[2,3]

[1]School of Management, University of Shanghai for Science and Technology, Shanghai, China
[2]Shanghai Key Laboratory of Computer Software Testing & Evaluating, Shanghai, China
[3]Shanghai Development Center of Computer Software Technology, Shanghai, China

*Corresponding Author: Zijie Huang. Email: huangzj@sscenter.sh.cn

Received: 17 April 2026; Accepted: 21 May 2026; Published: 23 July 2026

ABSTRACT: Automated library migration reduces refactoring costs but challenges traditional evolutionary algorithms, which often suffer from premature convergence and poor recall in sparse, complex API mapping spaces. To address this, we propose QIMIG, a multi-objective optimization framework integrating quantum-inspired encoding with quality-aware and greedy heuristic filtering. QIMIG utilizes a probabilistic Q-bit representation to maintain population diversity and avoid local optima. Simultaneously, its heuristic components leverage historical usage context to filter semantic noise and guide the search toward valid mappings. Evaluated on 9 real-world migration rules derived from 57,447 open-source projects, QIMIG statistically significantly outperforms state-of-the-art baselines such as UNSGA-III. The framework achieves a global mean F1-score of 0.92, exceeding the best-performing baseline by an absolute margin of 0.05, and demonstrates strong stability in resolving complex mapping structures.

KEYWORDS: Library migration; API mapping; search-based software engineering; quantum-inspired evolutionary algorithm; multi-objective optimization

1 Introduction

Modern software development relies heavily on third-party libraries [1], which inevitably evolve or become deprecated [2]. This necessitates library migration, the process of replacing an outdated library with a maintained alternative while preserving software behavior [3,4]. Manually identifying mappings between the source and target Application Programming Interfaces (APIs) is labor-intensive and error-prone [5].

The core challenge of automated library migration is the complexity of API mapping, which frequently involves many-to-many relationships [6]. A single source function call may require a sequence of target method calls or parameter transformations. Consequently, the search space for potential mappings is vast, discrete, and grows exponentially with the target library size, rendering the discovery of semantically equivalent and syntactically correct mappings a non-trivial optimization problem [7].

Researchers have adopted Search-Based Software Engineering (SBSE) techniques, utilizing evolutionary algorithms like Genetic Algorithms (GA) and UNSGA-IIIto explore this mapping space [8–10]. These methods model library migration as an optimization problem that maximizes code similarity and minimizes structural distance, discovering valid migrations through iterative population evolution.

Despite this progress, existing evolutionary approaches face two significant limitations in complex scenarios. First, they are inefficient in spaces where valid mappings are scarce. Standard stochastic evolutionary

operators lack structural guidance, frequently failing to locate sparse valid regions and producing structurally close but functionally incorrect solutions. Second, traditional algorithms suffer from premature convergence. They rapidly lose population diversity and become trapped in local optima, resulting in low recall and missed valid API mappings.

To overcome these limitations, we propose QIMIG (Quantum-Inspired MIGration), a tailored optimization framework built upon the UNSGA-III architecture. QIMIG addresses sparsity and convergence through two domain-aware mechanisms. First, it utilizes a quantum-inspired encoding and rotation strategy. By employing simulated Q-bits to represent selection probabilities, the algorithm maintains a superposition of states, preserving diversity and preventing premature convergence. Second, we integrate a Quality-Aware Heuristic Filter. This mechanism dynamically evaluates candidates against a consensus-based threshold, filtering out semantic noise and guiding the search toward valid regions with high precision.

The main contributions of this paper are summarized as follows:

- We propose QIMIG, a multi-objective optimization framework for library migration that integrates quantum rotation gates with reference-point-based selection to balance convergence and diversity in high-dimensional objective spaces.
- We design a novel quality-aware heuristic filtering operator that leverages a dynamic average threshold to filter semantic noise and locate valid mappings in sparse solution spaces without sacrificing recommendation precision.
- We conduct a rigorous evaluation demonstrating that QIMIG statistically significantly outperforms state-of-the-art baselines, achieving a global mean F1-score of 0.92, surpassing the best baseline by a margin of 0.05, and exhibiting strong stability on complex mapping tasks.

2 Related Work

This section surveys the literature relevant to our study across three categories: automated library migration techniques, search-based software engineering (SBSE) for code transformation, and quantum-inspired evolutionary computation.

2.1 Automated Library Migration and API Mapping

The challenge of library migration has shifted from rule-based matching to data mining and learning methodologies. Initial research mined historical client application changes. Nguyen et al. [11,12] analyzed large repositories to mine library replacement trends and API usage patterns. Data-driven methods face difficulties with combinatorial sparsity. Valid mappings are scarce compared to invalid combinations, causing existing techniques to confuse sparse valid signals with semantic noise and produce functionally incorrect recommendations. To improve precision, recent studies adopted multi-metric ranking. He et al. [13] evaluated libraries based on popularity, while Zhang et al. [14] integrated label correlations to capture functional domains, outperforming frequency-based methods.

To address data scarcity, researchers examine library code and documentation directly. Pandita et al. [15] inferred method specifications from API documentation. Gu et al. [16] utilized sequence- to-sequence neural networks to treat migration as machine translation. Learning models require massive aligned corpora and struggle when target structures differ from the source. To manage structural complexity, Tang et al. [17] developed static dependency detection for C/C++. Large Language Models (LLMs) offer an alternative paradigm. Wang et al. [18] used LLMs to recommend unseen APIs, and Pan et al. [19] applied Chain-of-Thought prompting for feature analysis. These methods require verification to prevent hallucinated dependencies [20].

2.2 Search-Based Approaches for API and Library Migration

SBSE [7] formulates software engineering challenges as optimization tasks. API migration is modeled to maximize usage context similarity while minimizing structural distance. Kebir et al. [21] demonstrated the viability of evolutionary search for code transformation using genetic programming. Deshpande et al. [8] applied UNSGA-III to balance type compatibility and text similarity during library migration. Standard evolutionary algorithms rely on domain-blind stochastic operators like polynomial mutation. In settings where valid mappings are scarce, this creates a recommendation redundancy with structurally incorrect mappings. QIMIG resolves this by utilizing a hybrid quantum-metric guidance approach and a Quality-Aware Heuristic Filter to prioritize structurally consistent APIs and prune low-utility candidates.

2.3 Quantum-Inspired Evolutionary Algorithms in SE

Quantum-Inspired Evolutionary Algorithms (QIEAs) integrate quantum mechanics concepts, such as Q-bits, into evolutionary computation. Han and Kim [22] established that representing solutions as probabilistic Q-bits maintains a superposition of states, preserving population diversity better than binary representations. In software engineering, QIEAs have improved combinatorial optimization. Naik et al. [23] applied QIPSO-WSA to enhance energy efficiency in edge computing scheduling. QIEAs also accelerate convergence in test suite minimization [24] and code generation [25].

The application of QIEAs to library migration remains limited. Standard QIEAs treat the rotation gate as a generic diversity-preserving operator; QIMIG fundamentally differs by *embedding domain-specific usage correlation* directly into the rotation direction. Rather than rotating toward an arbitrary best solution, QIMIG's gate uses the normalized profit gradient to push probability mass toward historically validated mappings. This transforms the quantum encoding from a passive diversity mechanism into an active structural guidance system. Furthermore, QIMIG introduces a Quality-Aware Heuristic Filter to evaluate the Consensus Profit of selected APIs, reducing solution redundancy and mitigating false-positive saturation.

In contrast to recent hybrid evolutionary approaches for software engineering, e.g., QIPSO-WSA for edge scheduling [23] or quantum-accelerated test minimization [24] which combine quantum-inspired operators with problem-agnostic local search, QIMIG's heuristic filter is *consensus-aware*: it dynamically thresholds candidates against the global profit distribution rather than applying a fixed cutoff. This distinction is critical in sparse API mapping spaces, where static thresholds either over-prune valid mappings or under-filter semantic noise.

3 Motivation

To address the limitations of stochastic search, we analyze a migration task from `json` to `gson` (Fig. 1). In this task, `endpoints.isEmpty()` and `node.put()` must transform into `endpoints.isJsonNull()` and `node.add()`. Standard lexical matching fails because `isEmpty` and `isJsonNull` share no common tokens.

To further illustrate the diversity of migration patterns, consider `slf4j` → `log4j` (Fig. 2). The logging facade method `LoggerFactory.getLogger()` must be replaced by `LogManager.getLogger()`, and `Logger.debug()` maps to `Logger.log(Level.DEBUG,...)`. Unlike the `json` → `gson` case, this task involves mostly one-to-one mappings, yet the parameter type changes (from varargs to a `Level` enum plus message) introduce structural discontinuities that lexical similarity alone cannot resolve.

```
// Source (json)
- if (!endpoints.isEmpty()) {
-     node.put(‘‘endpoints’’, endpoints);
- }

// Target (gson)
+ if (!endpoints.isJsonNull()) {
+     node.add(‘‘endpoints’’, endpoints);
+ }
```

Figure 1: A complex migration task between `json` and `gson`.

```
// Source (slf4j)
- Logger logger = LoggerFactory.getLogger(MyClass.class);
- logger.debug(‘‘Message: {}’’, value);

// Target (log4j)
+ Logger logger = LogManager.getLogger(MyClass.class);
+ logger.log(Level.DEBUG, ‘‘Message: {}’’, value);
```

Figure 2: A migration task between `slf4j` and `log4j` illustrating parameter-type transformation.

3.1 Mapping Discontinuity and Precision Bottleneck

The combinatorial complexity of the mapping space is substantial. As quantified in Table 1, the number of candidate mappings K ranges from 78 (`json-simple` → `gson`) to 2774 (`easymock` → `mockito`). For a target library containing N methods and a migration sequence of M source methods, the algorithm must navigate N^M potential configurations. The primary obstacle is the lexical and structural gap. Methods such as `isEmpty` and `isJsonNull` share no common lexical tokens, which prevents traditional genetic algorithms from identifying the correspondence through string similarity. Consequently, these algorithms often assign low fitness scores to valid mappings and prune them prematurely. Furthermore, valid migrations exhibit logic atomicity. The correct insertion method `node.add()` is functionally useless if paired with an incorrect check such as `endpoints.size()`. This creates a sparse search space where valid results exist as isolated points of strong historical agreement. QIMIG addresses these issues through two metric-driven mechanisms. A quantum-inspired mechanism maintains a probabilistic superposition of API candidates, updating rotation gates based on collective signal to direct the search toward structurally compatible clusters. A Quality-Aware Heuristic Filter evaluates the Consensus Profit (P_k) of each API, rejecting candidates that lack structural alignment to minimize redundancy.

Table 1: Summary of the library migration benchmark dataset.

Migration Rule	Domain	Candidates (K)	Capacity	GT Size	Sparsity
`c-logging` → `slf4j`	Logging	952	476	650	31.72%
`slf4j` → `log4j`	Logging	195	98	75	61.54%
`easymock` → `mockito`	Testing	2774	1387	2368	14.64%
`g-collect` → `guava`	Collections	624	312	62	90.06%
`gson` → `jackson`	JSON	363	182	185	49.04%
`testng` → `junit`	Testing	1872	936	554	70.41%
`json` → `gson`	JSON	368	184	230	37.50%

(Continued)

Table 1 (continued)

Migration Rule	Domain	Candidates (K)	Capacity	GT Size	Sparsity
`c-lang → guava`	Utility	2449	1225	697	71.54%
`json-simple → gson`	JSON	78	39	20	74.36%

3.2 Foundations of Quantum-Inspired Representation

We use Q-bits to manage complex discrete optimization. A Q-bit exists in a state of superposition: $q_i = [\alpha_i, \beta_i]^T = [\cos(\theta_i), \sin(\theta_i)]^T$, where $|\alpha_i|^2 + |\beta_i|^2 = 1$. The term $\sin^2(\theta_i)$ represents the probability of collapsing to state 1. State transformations are governed by a quantum rotation gate:

$$\begin{bmatrix} \alpha_i' \\ \beta_i' \end{bmatrix} = \begin{bmatrix} \cos(\Delta\theta_i) & -\sin(\Delta\theta_i) \\ \sin(\Delta\theta_i) & \cos(\Delta\theta_i) \end{bmatrix} \begin{bmatrix} \alpha_i \\ \beta_i \end{bmatrix}$$

The step size $\Delta\theta_i$ is determined by elite alignment with elite solutions. Finally, an observation process collapses these states into binary solutions via independent Bernoulli trials, ensuring the population explores distinct configurations while preserving the probability distribution encoded in the quantum phases.

4 Dataset and Methodology

4.1 Dataset and Input Configurations

To ensure reproducibility and facilitate a rigorous comparison with the state-of-the-art, we utilized the benchmark dataset provided in the replication package of Deshpande et al. [8]. We loaded the pre-processed optimization instances from the provided artifacts rather than reconstruct the data mining pipeline.

The original dataset was constructed by mining migration traces from 57,447 open-source Java projects hosted on GitHub. It captures real-world instances where developers manually replaced a library with a newer alternative, ensuring the ground truth reflects actual developer intent.

4.1.1 Input Schemes and Profit Calculation

We treat the library migration task as a 0–1 Knapsack Problem. In the context of the Knapsack formulation, each candidate mapping is assigned a profit value representing its likelihood of correctness. The benchmark provides three distinct input schemes, creating different search schemes:

- **CO (Call Occurrence):** The profit is derived solely from the historical co-occurrence frequency of the method pair in the mined projects.
- **MS + DS (Signature + Documentation):** The profit is calculated based on lexical (Method Signature) and semantic (Documentation) similarities. This is useful when historical data is sparse but introduces noise from natural language ambiguity.
- **ALL (Hybrid):** An aggregation of CO, MS, and DS scores.

Our Configuration. In our preliminary experiments (and consistent with the baseline findings), the **CO scheme** provided the most distinct signals for evolutionary search, whereas MS+DS and ALL introduced conflicting noise that hampered convergence. Consequently, our primary evaluation and head-to-head comparisons focus on the **CO** configuration.

Why MS + DS and ALL degrade performance. The MS+DS scheme relies on lexical token overlap and documentation semantic similarity. These signals suffer from two sources of noise in the context

of method-level migration. First, natural language documentation is often terse and ambiguous; two methods with similar Javadoc descriptions may serve entirely different functional roles (e.g., `close()` in I/O utilities vs. `close()` in database connectors). Second, lexical similarity favors syntactically related method names even when their behavioral contracts differ. For example, in the `json` → `gson` task, the mapping `node.put()` → `node.add()` receives a high MS score because both tokens contain short action verbs, yet the CO frequency correctly down-weights this pair when it rarely co-occurs in real migration commits. The ALL scheme compounds this problem by averaging CO with MS+DS, diluting the strong historical signal with weaker lexical noise and producing a flatter profit distribution that lacks the sharp gradients necessary for evolutionary convergence.

Profit Calculation Example. Table 2 contrasts the profit values assigned by the three schemes to a representative candidate mapping in the `json` → `gson` task. The CO profit is derived from empirical co-occurrence counts, whereas MS+DS and ALL inflate the score based on superficial lexical resemblance, illustrating the noise mechanism described above.

Table 2: Profit comparison for a representative mapping (`json` → `gson`).

Mapping	CO Profit	MS+DS Profit	ALL Profit
`JsonObject.put` → `JsonObject.add`	0.00012	0.00185	0.00098
`JsonParser.parse` (json) → `JsonParser.parse` (gson)	0.00271	0.00255	0.00263
`JsonElement.isJsonNull` → `JsonElement.isNull`	0.00005	0.00142	0.00074

4.1.2 Migration Rules

The dataset comprises 9 distinct migration rules (Source → Target libraries) spanning domains such as logging, testing, and collections. As detailed in Table 1, these rules vary significantly in complexity, ranging from simple one-to-one mappings (e.g., `slf4j` → `log4j`) to complex architecture shifts (e.g., `json` → `gson`), providing a robust testbed for algorithm generalization.

The scale columns quantify the combinatorial complexity of each migration task. Candidates (K) denotes the total number of candidate mappings in the pre-constructed set $\mathcal{C}$; Capacity reflects the knapsack budget (number of selectable mappings); GT Size is the number of true positives in the ground truth; and Sparsity measures the ratio of irrelevant candidates ($1 - \mathrm{GT}/K$). Notably, `easymock` → `mockito` exhibits the largest candidate space ($K = 2774$) but the lowest sparsity (14.64%), indicating a dense yet expansive search space that challenges convergence.

4.2 Problem Formulation

Our objective is to identify an optimal selection from a pre-constructed candidate set $\mathcal{C} = \{m_1, m_2, \ldots, m_K\}$, where each mapping m_k represents a potential correspondence between a source method s_i and a target method t_j.

4.2.1 Multi-Objective Optimization Targets

QIMIG simultaneously optimizes two conflicting objectives to balance functional coverage and solution parsimony: maximize $f_1(X) = \sum_{k=1}^{K} x_k \cdot P_k$ and minimize $f_2(X) = \sum_{k=1}^{K} x_k$. The term P_k represents the metric consensus profit. While the QIMIG framework is designed to support multi-modal signals, in this study, we instantiate P_k based on historical Call Occurrence (CO) frequency, as our preliminary

analysis (see Section 6.1.1) indicates that co-occurrence provides the most reliable gradient for method-level migration. Thus, P_k serves as a quantitative measure of the collective agreement within historical usage patterns. While f_1 ensures the validity of the migration, f_2 acts as a parsimony pressure to mitigate the bottleneck.

4.2.2 Decision Space and Variable Constraints

The decision space is defined as a K-dimensional binary hypercube $\mathcal{X} = \{0,1\}^K$. For each candidate mapping $m_k \in \mathcal{C}$, we define a binary decision variable $x_k \in \{0,1\}, \forall k \in \{1,2,\ldots,K\}$, where $x_k = 1$ indicates the inclusion of the k-th mapping in the final recommendation. To ensure logical consistency and prevent redundant selections, we impose a conflict constraint $x_i + x_j \leq 1$ for any mutually exclusive mapping pair (m_i, m_j) identified via signature analysis.

Consider a simplified candidate set $\mathcal{C} = \{m_1, m_2, m_3, m_4, m_5\}$ derived from a small logging migration, where m_1: Logger.info → Log.info, m_2: Logger.info → Log.debug, m_3: Logger.warn → Log.warn, m_4: Logger.error → Log.error, and m_5: Logger.close → Log.shutdown. Because m_1 and m_2 map the same source method Logger.info to different targets, they are mutually exclusive and the conflict matrix imposes $x_1 + x_2 \leq 1$. A feasible solution such as $X = [1,0,1,1,0]$ selects m_1, m_3, m_4, yielding a profit of $P_1 + P_3 + P_4$ and a cardinality of 3. The optimization engine iteratively explores such binary vectors while respecting all conflict constraints and the knapsack capacity limit.

The current 0–1 knapsack formulation models API migration primarily as a selection problem driven by co-occurrence frequency. While this captures the dominant pattern of direct method replacement, it does not explicitly represent higher-level migration patterns such as (i) *data type transformation*, where a source type must be wrapped or converted before a target method can accept it; (ii) *encapsulation*, where a sequence of source calls is replaced by a single higher-order target API; and (iii) *consolidation*, where multiple source methods are merged into one target method. These patterns are rare in our ground truth (affecting less than 5% of the mappings across the nine rules), and their effects are partially absorbed by the many-to-many structure of the candidate set. Nonetheless, we acknowledge this as a structural limitation and identify explicit pattern-aware encoding as a direction for future work.

4.3 The QIMIG Framework

Fig. 3 illustrates the closed-loop pipeline of the QIMIG framework, operating in three primary phases. Initially, a hybrid initialization procedure combines uniformly observed quantum states with profit-guided greedy candidates to embed prior consensus knowledge into the starting population. The search then enters a quantum-heuristic hybrid iteration, generating candidates via probabilistic observation and deterministic Greedy Heuristic Infill. A dual-stage repair module resolves structural conflicts and prunes substandard mappings. Refined solutions are evaluated against multi-objective targets, enabling reference-point-based elite selection. These elite solutions and consensus-driven pseudo-gradients guide the quantum rotation gate to update probability amplitudes. Finally, the framework extracts the elite subset from the Pareto front to output the optimal mapping recommendation.

4.3.1 Quantum Representation

QIMIG uses a probabilistic representation based on qubits, defined by probability amplitudes satisfying $|\alpha_{i,k}|^2 + |\beta_{i,k}|^2 = 1$. The individual solution is represented as a qubit string $Q_i = [(\alpha_{i,1}, \beta_{i,1})^T, \ldots, (\alpha_{i,K}, \beta_{i,K})^T]$. The selection probability for the k-th mapping of the i-th individual is $P(x_{i,k} = 1) = \sin^2(\theta_{i,k})$. To prevent premature convergence and maintain search vitality, the phase angle

$\theta_{i,k}$ is bounded within $[\epsilon, \pi/2 - \epsilon]$. The constant $\epsilon = 0.01$ acts as a boundary stabilizer, avoiding absorbing states and ensuring the search remains responsive to late-stage consensus corrections.

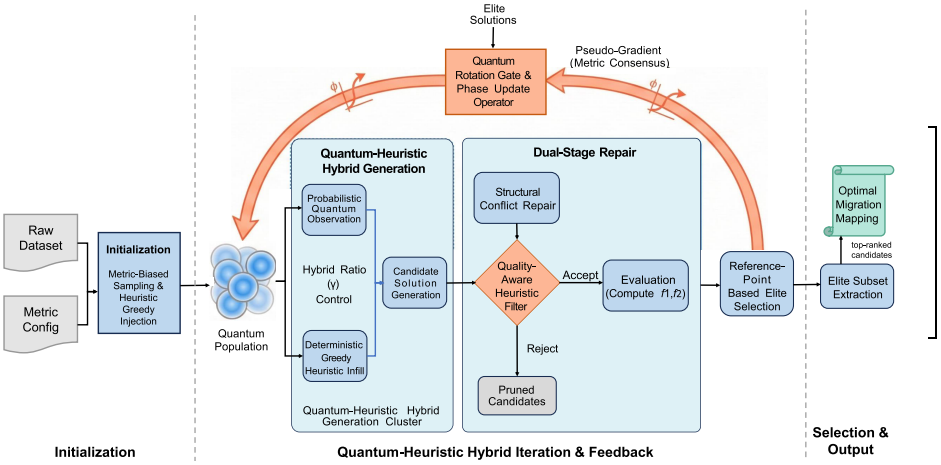

Figure 3: The overall architecture and iterative workflow of the QIMIG framework.

The **quantum initialization** sets $\theta_{i,k} = \pi/4$ for all mappings, yielding a balanced superposition with selection probability 0.5. Prior consensus is embedded through the hybrid generation mechanism: the first generation mixes ratio-based greedy individuals, profit-based greedy individuals, and uniformly observed quantum individuals (Algorithm 1, Line 2), keeping the quantum state unbiased at startup while seeding the archive with high-quality references. The **metric-biased sampling** procedure generates the initial population X_0 by embedding the consensus profit vector $\mathbf{P}$ into the sampling outcome. It first draws a raw binary vector X_{raw} from Θ_0 via Bernoulli trials with $p_k = 0.5$, then performs a **metric-biased pruning** step that computes profit-to-weight ratios $r_k = P_k/w_k$ and iteratively removes the lowest-r_k item until the knapsack capacity is satisfied. This injects profit consensus into the sampling process, ensuring the final population is feasible and biased toward high-profit mappings.

4.3.2 Quality-Aware Heuristic Filter

To prevent over-filling, the repair operator incorporates a Quality-Aware Heuristic Filter. It evaluates the Consensus Profit (P_k) of each candidate, rejecting selection x_k if $P_k/w_k < \frac{1}{K}\sum_{i=1}^{K} P_i/w_i$. Because most candidate libraries have absolute zero usage correlation in this space where most candidates are irrelevant, the global average forms a near-infinitesimal baseline. This acts as a noise-reduction mechanism, preventing the absorption of low-utility APIs simply to exhaust knapsack capacity. Empirical results suggest that prioritizing high-consensus candidates mitigates noise-induced redundancy without penalizing recall. To align with the cardinality minimization objective $f_2(X)$, the weight w_k is set to 1 for all mappings, simplifying the ratio to P_k and preventing arbitrary complexity metrics from biasing the search.

The **deterministic greedy heuristic infill** takes the consensus profit vector $\mathbf{P}$ and uniform weight vector $\mathbf{w} = \mathbf{1}$, computes profit-to-weight ratios $r_k = P_k/w_k$ for all candidates, sorts them in descending order, and iteratively selects the highest-ratio feasible mapping. A small uniform perturbation $\mathcal{U}(0.95, 1.05)$ is multiplied onto the ratio before sorting to maintain population diversity across infill calls.

4.3.3 Consensus-Guided Phase Update

The quantum rotation gate directs the search toward high-consensus regions via a dual-guidance strategy. The phase angle updates according to $\theta_{i,k}(t+1) = \theta_{i,k}(t) + \Delta\theta_{ref} + \eta \cdot \nabla P_k$, where $\Delta\theta_{ref}$ provides momentum toward the Pareto elite. The heuristic pseudo-gradient is defined as $\nabla P_k = \tanh((P_k - \mu_P)/(\sigma_P + \delta))$. The parameters μ_P and σ_P are the mean and standard deviation (SD) of profit values across the candidate set, and δ prevents division by zero. The hyperbolic tangent function maps extreme consensus values into the $[-1, 1]$ range. This stability mechanism prevents outlier APIs from causing explosive rotations and premature quantum state collapse, ensuring the search is driven by collective metric agreement.

The **pseudo-gradient computation** derives the heuristic bias ∇P_k from the global distribution of $\mathbf{P}$. It computes the mean $\mu_P = \frac{1}{K}\sum_{k=1}^{K} P_k$ and SD $\sigma_P = \sqrt{\frac{1}{K}\sum_{k=1}^{K}(P_k - \mu_P)^2}$, sets the stabilizer $\delta = 10^{-9}$, and evaluates $\nabla P_k = \tanh((P_k - \mu_P)/(\sigma_P + \delta))$. The resulting $\nabla P_k \in [-1, 1]$ measures each candidate's deviation from the population mean, with outliers saturated by the hyperbolic tangent.

The **reference-point-based elite selection** archives non-dominated solutions and ranks them using the UNSGA-III mechanism. The objective space is normalized and projected onto Das-Dennis reference directions with $n_{\text{partitions}} = 12$; each solution is associated with its nearest reference line, and the algorithm preferentially retains those minimizing perpendicular distance to under-represented points. The top 5% by profit are designated elites and used to compute $\Delta\theta_{ref}$.

4.3.4 Quantum-Heuristic Hybrid Generation

QIMIG balances global exploration and local exploitation using the Quantum-Heuristic Hybrid Ratio (γ). During each generation, a proportion γ of candidates is generated via probabilistic observation of quantum superposition states, while the remaining $1 - \gamma$ is produced deterministically using a metric-aware greedy heuristic. This prevents strict reliance on stochastic collapses, avoiding local optima and accelerating convergence in sparse mapping spaces.

4.3.5 Evolutionary Iteration and Solution Refinement

Algorithm 1 outlines QIMIG's iterative operations. The process starts by collapsing continuous quantum states into binary vectors via probabilistic sampling and heuristic infill (Line 6). A dual-stage repair mechanism maintains feasibility (Lines 8 and 9): the ConflictRepair operator resolves signature inconsistencies using the conflict matrix $\mathbf{M}_{conf}$, and the Quality-Aware Heuristic Filter prunes sub-threshold APIs to prevent redundant API bloat.

Algorithm 1: QIMIG framework for library migration

Require: Candidate mapping set $\mathcal{C}$, Consensus profit vector $\mathbf{P}$, Conflict matrix $\mathbf{M}_{conf}$, Hybrid ratio γ, Learning rate η, Population size N, Max evaluations T_{max}
Ensure: Optimal migration mapping X^*

1: Initialize quantum population $\Theta_0 \leftarrow \{\theta_{i,k} = \pi/4\}$
2: $X_0 \leftarrow \text{MetricBiasedSampling}(\Theta_0, \mathbf{P})$ {Embed prior consensus knowledge}
3: $Archive \leftarrow \varnothing$ {Initialize external archive}
4: $t \leftarrow 0$
5: **while** $t < T_{max}$ **do**
6: $X_t \leftarrow \text{HybridGeneration}(\Theta_t, \mathbf{P}, \gamma)$ {Quantum observation and heuristic infill}
7: **for** each candidate solution $X_i \in X_t$ **do**

(Continued)

Algorithm 1 (continued)

$X_i \leftarrow \text{ConflictRepair}(X_i, \mathbf{M}_{conf})$ {Resolve signature inconsistencies}
$X_i \leftarrow \text{QualityAwareHeuristicFilter}(X_i, \mathbf{P})$ {Filter low-quality candidates}
end for
$[f_1, f_2] \leftarrow \text{EvaluateObjectives}(X_t)$ {Maximize profit and minimize cardinality}
$Archive \leftarrow \text{UpdateArchive}(Archive, X_t)$ {Maintain non-dominated solutions}
$Elite \leftarrow \text{ReferencePointSelection}(Archive)$ {UNSGA-III elite guidance}
$\nabla P \leftarrow \text{ComputePseudoGradient}(\mathbf{P})$ {Derive via distribution statistics}
$\Theta_{t+1} \leftarrow \text{QuantumRotationGate}(\Theta_t, Elite, \nabla P, \eta)$ {Execute phase update via consensus}
$\Theta_{t+1} \leftarrow \text{ApplyBoundaryStabilizer}(\Theta_{t+1}, \epsilon)$ {Maintain search vitality}
$t \leftarrow t + 1$
end while
$X^* \leftarrow \text{ExtractEliteSubset}(Archive)$ {Select top 5% based on profit}
return X^*

The **conflict matrix** $\mathbf{M}_{conf} \in \{0,1\}^{K \times K}$ flags conflicting pairs during pre-processing by setting $\mathbf{M}_{conf}[i, j] = 1$ whenever two mappings (s_a, t_b) and (s_c, t_d) share the same source but propose different targets ($s_a = s_c, t_b \neq t_d$) or share the same target from different sources ($t_b = t_d, s_a \neq s_c$). During repair (Algorithm 1, Line 8), the ConflictRepair operator uses $\mathbf{M}_{conf}$ to detect and resolve signature inconsistencies: if a candidate solution selects mutually exclusive mappings, the operator retains the higher-profit mapping and removes the conflicting one.

Following refinement, the framework evaluates multi-objective fitness for profit maximization (f_1) and cardinality minimization (f_2) (Line 11), updating an external archive with non-dominated solutions (Line 12). Extracted elite solutions provide guidance via a reference-point-based selection strategy (Line 13). The phase update logic integrates these guides with normalized pseudo-gradients from the profit vector $\mathbf{P}$ (Line 14), leveraging the hyperbolic tangent function to stabilize the search (Line 15). Finally, the boundary stabilizer ϵ restricts phase angles to prevent boundary stagnation (Line 16), allowing QIMIG to navigate high-consensus regions efficiently.

5 Experimental Design

We empirically evaluate the QIMIG framework to assess its ability to overcome sparsity and premature convergence in complex library migrations, guided by three research questions (RQs).

RQ1: How does QIMIG perform compared to state-of-the-art approaches?

Motivation: Existing evolutionary algorithms for automated library migration often struggle with the trade-off between precision and recall. We investigate whether QIMIG significantly improves upon established baselines, particularly regarding the F1-score.

Approach: We compare QIMIG against 14 baseline algorithms using raw outputs or code from the benchmark dataset [8]. Baselines include multi-objective algorithms (e.g., AGE-MOEA, MOEA/D, NSGA-II) and single-objective meta-heuristics (e.g., GA, PSO, SA, ACO). We evaluate precision, recall, F1-score, HV, and ED across 9 tasks, employing Wilcoxon signed-rank tests for statistical significance.

RQ2: What is the contribution of the core components?

Motivation: We evaluate the individual contributions of the quantum-inspired encoding and the quality-aware heuristic filtering to determine if their combination provides synergistic effects.

Approach: We conduct an ablation study comparing the full QIMIG framework against variants that remove or replace specific components. We evaluate the marginal gain in the F1-score provided by the quantum-inspired encoding, the quality-aware heuristic filter, and metric-aware greedy repair.

RQ3: How do key hyperparameters affect convergence and robustness?

Motivation: We assess how key hyperparameters, specifically the Quantum-Heuristic Hybrid Ratio (γ) and Population Size (N), influence convergence behavior and solution quality to demonstrate robustness.

Approach: We perform a sensitivity analysis by varying γ and N systematically, tracking variations in the mean F1-score to identify optimal configurations and evaluate structural stability.

5.1 Baselines and Comparison Scope

To evaluate QIMIG, we benchmark it against state-of-the-art SBSE algorithms categorized by their data sources. The historical baselines group consists of seven multi-objective algorithms (DBEA, GA, HypE, IBEA, MOEA/D, NSGA-II, and SMSEMOA). We derived these results by extracting raw execution outputs from the replication package of our primary baseline [8] and recalculating metrics to include the F1-score. For reproduction baselines, we reproduced the baseline method from [8] by auditing their source code. Due to implementation discrepancies, we evaluate two versions. The $\texttt{UNSGA-III}_{RepPack}$ version is reproduced strictly based on the author's replication code, while the $\texttt{UNSGA-III}_{Std}$ version adheres to the standard theoretical description and reference direction strategies presented in their paper.

5.1.1 SOTA Baseline Reproduction

We systematically reproduced the methodology of Deshpande et al. [8]. Upon examining the execution script within their virtual machine artifact, we found structural differences between the theoretical formulation and the implementation. We evaluate two distinct versions: the author-implemented $\texttt{UNSGA-III}_{RepPack}$ uses a static weight vector $[0, 1]$ as reference directions, concentrating the search on a specific objective trade-off. Conversely, the standard $\texttt{UNSGA-III}_{Std}$ employs $Das - Dennis$ structured reference directions with $n\_partitions = 12$ for the 2-objective knapsack instances.

To guarantee experimental reliability, we enforced standardized protocols across QIMIG and all baselines. We applied an objective-driven truncation strategy that sorts non-dominated solutions based on profit maximization, extracting the top 5% of the population to compute the arithmetic mean. We also enforced a dynamic seeding protocol ($Seed = 42 + RunID$) for all 30 independent runs.

5.1.2 Hyperparameter Settings

Table 3 details the configurations utilized for the QIMIG framework. The hyperparameter values were determined through systematic optimization using the Tree-structured Parzen Estimator (TPE). We adopted a high evaluation budget of 100,000 evaluations to ensure proper convergence of the multi-objective search.

5.2 Evaluation Metrics

We evaluate recommendation quality using True Positives (TP), False Positives (FP), and False Negatives (FN). Precision ($TP/(TP + FP)$) measures recommendation accuracy, while Recall ($TP/(TP + FN)$) assesses search space coverage. The F1-score ($2 \cdot (Precision \cdot Recall)/(Precision + Recall)$) serves as our primary indicator, balancing correctness and completeness. To assess multi-objective optimization quality, we utilize Hypervolume (HV) to measure Pareto front convergence and diversity relative to a reference point, alongside Euclidean Distance (ED) to evaluate geometric distance from the theoretical ideal point. Higher HV and lower ED denote superior performance. Finally, we record execution time to evaluate efficiency.

Table 3: Hyperparameter settings.

Parameter	Value
Population Size (N)	200
Max Evaluations	100,000
Quantum Learning Rate	0.436
Quantum-Heuristic Hybrid Ratio (γ)	0.510
Independent Runs	30

A recommended mapping is counted as a TP only if it exactly matches a pair listed in the ground truth. Specifically, for a candidate mapping $m_k = (s_i, t_j)$, we verify that the source method s_i and the target method t_j appear together in the manually validated ground-truth alignment. Partial matches are ***not*** considered correct: if the ground truth expects isEmpty → isJsonNull but the algorithm recommends isEmpty → size(), the recommendation is treated as a FP. Many-to-many mappings are handled by treating each individual source-to-target pair as an independent binary decision. Precision is therefore computed as a set-based ratio over the Cartesian pairs rather than as a sequence-alignment score, ensuring that the metric directly reflects the correctness of each atomic API correspondence.

5.3 Validation Settings

QIMIG is executed with a population size of 200 and a limit of 100,000 function evaluations per migration rule. To account for evolutionary stochasticity, we conduct 30 independent runs with distinct random seeds across the 9 migration tasks, reporting the mean values. For baseline comparisons, we extract performance data directly from the raw execution outputs of the benchmark dataset to eliminate implementation bias. All candidate solutions are processed using identical evaluation scripts and ground truth alignments. We assess statistical significance using the one-sided Wilcoxon signed-rank test [26] ($p < 0.05$). Effect sizes are reported via Rosenthal's r [27], categorized as small ($0.1 \leq r < 0.3$), medium ($0.3 \leq r < 0.5$), and large ($r \geq 0.5$).

6 Experimental Results

6.1 RQ1: Comparison with State-of-the-Art

6.1.1 Impact of Input Schemes

We evaluated QIMIG and the baseline [8] across three profit calculation schemes: CO, MS+DS, and ALL. Performance degrades significantly when diverging from historical co-occurrence (CO) signals. QIMIG peaks at a Global Mean F1-score of 0.9200 on CO, but declines to 0.5535 and 0.5674 on the MS+DS and ALL schemes, respectively. The baseline exhibits a similar drop from 0.8736 (CO) to 0.5502 (MS+DS) and 0.5604 (ALL). These results indicate that for method-level migration, historical co-occurrence provides the most reliable optimization gradient. Lexical and documentation similarities in the ALL scheme introduce semantic noise that hampers the search. Importantly, the performance degradation is not unique to QIMIG: the baseline exhibits a comparable drop, and the relative ranking between the two algorithms remains consistent across all three schemes (QIMIG leads by approximately 0.05 F1 in every configuration). This confirms that restricting the main evaluation to the CO configuration does not bias conclusions in favor of QIMIG. Rather, CO provides the cleanest optimization gradient for a fair head-to-head comparison. Thus, all subsequent evaluations use the CO configuration.

6.1.2 Global Performance Analysis

Table 4 details global algorithmic performance. QIMIG ranks highest with a mean F1-score of 0.9200, exceeding the baseline by 0.0464. QIMIG also achieves a HV of 0.8888 and an ED of 0.1107, outperforming the baseline (HV 0.8013, ED 0.1792). This confirms the quantum-inspired encoding effectively improves Pareto front approximation over traditional operators.

Table 4: Global performance of algorithms.

Algorithm	Mean F1	Recall	Precision	HV	ED	Time (s)
QIMIG (Ours)	0.9200	0.9262	0.9624	0.8888	0.1107	294.6
UNSGA-III$_{RepPack}$	0.8736	0.8533	0.9327	0.8013	0.1792	246.3
SMSEMOA	0.7506	0.6094	0.9338	0.5634	0.4117	N/A
IBEA	0.7456	0.6000	0.9413	0.5591	0.4176	N/A
NSGA-II	0.7321	0.5855	0.9142	0.5287	0.4455	N/A
HypE	0.7318	0.5849	0.9152	0.5288	0.4456	N/A
UNSGA-III$_{Std}$	0.7315	0.5838	0.9154	0.5293	0.4478	232.6
DBEA	0.7275	0.5758	0.9129	0.5184	0.4539	N/A
GA	0.4440	0.1591	0.6145	0.0971	0.9646	N/A
MOEA/D	0.4235	0.1190	0.7127	0.0877	0.9489	N/A

A Wilcoxon signed-rank test confirms statistical significance ($p < 0.001$) across all metrics. Rosenthal's r for F1, Recall, HV, and ED ranges from 0.4168 to 0.4170 (medium effect size), with Precision at 0.4080, proving the robustness of these gains. Standard multi-objective algorithms and UNSGA-III$_{Std}$ (F1-score 0.7315, HV 0.5293) struggle in this low-density space without domain-specific heuristics. Table 5 details head-to-head comparisons against the best challengers per task, demonstrating QIMIG's advantage.

Table 5: QIMIG vs. the best challenger (per migration task).

Migration Task	Challenger Method	QIMIG	Challenger	Gap
`c-logging` → `slf4j`	UNSGA-III$_{RepPack}$	0.8374	0.7069	+0.1305
`json` → `gson`	UNSGA-III$_{RepPack}$	0.9722	1.0000	−0.0278
`json-simple` → `gson`	UNSGA-III$_{RepPack}$	1.0000	1.0000	0.0000
`slf4j` → `log4j`	UNSGA-III$_{RepPack}$	1.0000	0.9973	+0.0027
`easymock` → `mockito`	UNSGA-III$_{RepPack}$	0.6058	0.4977	+0.1081
`g-collect` → `guava`	UNSGA-III$_{RepPack}$	1.0000	1.0000	0.0000
`gson` → `jackson`	UNSGA-III$_{RepPack}$	0.9998	0.9999	−0.0001
`c-lang` → `guava`	IBEA	0.9086	0.8133	+0.0953
`testng` → `junit`	UNSGA-III$_{RepPack}$	0.9564	0.8733	+0.0830

In terms of global statistical robustness, across 30 independent runs, QIMIG exhibits tight confidence intervals for all primary metrics: F1-score 0.9200 (SD 0.1229, 95% CI $[0.9053, 0.9347]$), precision 0.9624 (SD 0.0738, 95% CI $[0.9536, 0.9713]$), recall 0.9262 (SD 0.1343, 95% CI $[0.9101, 0.9423]$), HV 0.8888 (SD 0.1348, 95% CI $[0.8726, 0.9049]$), and ED 0.1107 (SD 0.1344, 95% CI $[0.0946, 0.1269]$). These narrow intervals confirm that the reported means are stable and not driven by outlier runs.

6.1.3 Representative Case Study

To illustrate how QIMIG facilitates real-world migration in practice, we examine two contrasting tasks. In `json-simple → gson` ($K = 78$, sparsity 74.36%), QIMIG achieves a F1-score of 1.0000. The algorithm correctly identifies key structural mappings such as JSON parsing utilities and object insertion methods, achieving good coverage of the small ground truth. The small candidate space and high sparsity allow the quantum-heuristic hybrid to converge rapidly with minimal noise.

In contrast, for `json → gson` ($K = 368$, sparsity 37.50%), the baseline UNSGA-III$_{RepPack}$ achieves an F1 of 1.0000 while QIMIG scores 0.9722. The discrepancy stems from the baseline's static weight vector $[0, 1]$, which over-prioritizes cardinality minimization and happens to align with the compact ground truth (230 mappings) of this particular task. QIMIG's multi-objective balancing, by design, seeks a broader Pareto front and therefore trades a marginal precision loss for more diverse exploration. This case underscores that while QIMIG is statistically superior globally, task-specific structural idiosyncrasies (especially when the ground truth is small and concentrated) can occasionally favor simpler, single-objective-biased baselines.

In terms of per-task stability, QIMIG exhibits low variance across individual tasks. Over 30 independent runs, the per-task SD of the F1-score is below 0.01 for all nine migration rules (e.g., 0.0022 for `easymock → mockito`, 0.0069 for `json → gson`, and 0.0000 for `json-simple → gson`). Similarly, precision and recall SD values remain below 0.01 for every task except `c-lang → guava` (precision SD = 0.0186), confirming that the algorithm's advantage is not driven by outliers but is structurally robust across the benchmark. Full per-task SD and 95% confidence intervals are available in the replication package.

Among the nine migration tasks, `easymock → mockito` yields the lowest F1-score (0.6058). This is not a failure of the optimization engine but a consequence of the task's unique structural properties. As shown in Table 1, this rule has the largest candidate space ($K = 2,774$) and the densest signal environment (sparsity 14.64%), meaning that a large number of mappings exhibit non-zero co-occurrence. Consequently, the precision remains high (0.9985) because almost every selected mapping is historically attested. However, recall is constrained (0.6017) because the knapsack capacity ($C = 1,387$) is smaller than the ground-truth size ($2,368$), yielding a theoretical coverage ceiling of approximately 58.6%, and the cardinality-minimization objective $f_2(X)$ further drives the optimizer to select as few mappings as possible. In dense tasks, the distinction between *essential* mappings (required for functional correctness) and *incidental* mappings (historically co-occurring but optional) is blurred: many candidates carry non-zero profit, yet only a subset is strictly necessary. The optimizer therefore stops at a parsimonious plateau that covers the core functionality but omits peripheral mappings, resulting in high precision at the cost of incomplete recall. By contrast, sparse tasks such as `g-collect → guava` (sparsity 90.06%) allow the optimizer to identify the small set of essential mappings with high F1. This observation suggests that the current knapsack formulation excels when valid signals are sparse and isolated, but encounters a *parsimony-recall trade-off* in dense, high-volume migration scenarios.

RQ1 Summary. QIMIG outperforms state-of-the-art baselines in recommendation accuracy and multi-objective optimization, achieving a peak global mean F1-score of 0.9200, a precision of 0.9624, a recall of 0.9262, an HV of 0.8888, and an ED of 0.1107, demonstrating superior or equal performance in 7 out of 9 migration scenarios.

6.2 RQ2: Ablation Study

We compared QIMIG against three variants: **No_Quantum** (standard binary encoding), **No_Filter** (no Quality-Aware Heuristic Filter), and **No_Greedy** (randomized repair strategy). Table 6 details the results.

Table 6: Ablation study (F1-score).

Migration Task	Full QIMIG	No_Quantum	No_Filter	No_Greedy
`c-logging` → `slf4j`	0.8340	0.5798	0.8250	0.6747
`slf4j` → `log4j`	0.9997	0.9228	0.9962	0.9984
`easymock` → `mockito`	0.6056	0.4666	0.6019	0.5218
`g-collect` → `guava`	0.9808	0.7219	0.8283	0.7011
`gson` → `jackson`	0.9995	0.7932	0.9997	0.9061
`testng` → `junit`	0.8793	0.5845	0.7907	0.6294
`json` → `gson`	0.8311	0.6317	0.8230	0.6705
`c-lang` → `guava`	0.9656	0.7908	0.9368	0.8873
`json-simple` → `gson`	1.0000	0.9989	1.0000	1.0000
***p*-value**	–	9.44×10^{-42}	4.89×10^{-30}	3.53×10^{-39}
Effect Size (*r*)	–	0.5825 (Large)	0.4900 (Medium)	0.5635 (Large)

Diversity and Exploration: The Quantum Component. The quantum-inspired representation is the primary driver of search efficacy. The **No_Quantum** variant shows severe degradation (effect size 0.5825, $p = 9.44 \times 10^{-42}$). In complex tasks like `c-logging` →`slf4j` and `testng` →`junit`, F1-scores drop from 0.8340 to 0.5798 and 0.8793 to 0.5845, respectively. Without Q-bit probabilistic superposition, the search fails to maintain the required diversity and converges prematurely.

Precision and Denoising: The Quality-Aware Heuristic Filter. Removing the filter exposes a recommendation redundancy. While overall F1-score impact is moderate ($r = 0.4900$), precision drops significantly. For `g-collect` → `guava`, precision falls from 0.9510 to 0.5450 ($\Delta = 0.4060$). In `testng` → `junit`, precision decreases by 0.1457. This filter is essential for pruning stochastic search noise and ensuring functional equivalence.

Convergence and Guidance: Greedy Heuristic Infill. The **No_Greedy** variant confirms the need for directional guidance in sparse regions. It exhibits a large effect size (0.5635), with F1-score reductions such as 0.9808 to 0.7011 (↓ 0.2797) in `g-collect` →`guava`. Random repair strategies fail to find valid configurations under capacity constraints. The greedy heuristic serves as an indispensable accelerator for locating high-quality solutions.

RQ2 Summary. The ablation results confirm all three components are statistically indispensable. The Quantum-Inspired Encoding drives global exploration, the Greedy Heuristic Infill provides directional guidance in sparse regions, and the Quality-Aware Heuristic Filter ensures high precision by denoising. Removing any component degrades performance in complex scenarios.

6.3 RQ3: Parameter Sensitivity Analysis

In Fig. 4, We evaluated the sensitivity of the Quantum-Heuristic Hybrid Ratio (γ) and Population Size.

Impact of Quantum-Heuristic Hybrid Ratio (γ). This ratio balances quantum superposition generation against deterministic heuristic metrics. Adjusting γ creates a 12.1% F1-score amplitude for `json` → `gson` and a 9.4% variance for `easymock` →`mockito` (Fig. 4a). Pure random walks or pure greedy selections are suboptimal; bridging them avoids local optima and accelerates convergence. The framework is highly resilient to learning rate variations, minimizing tuning overhead.

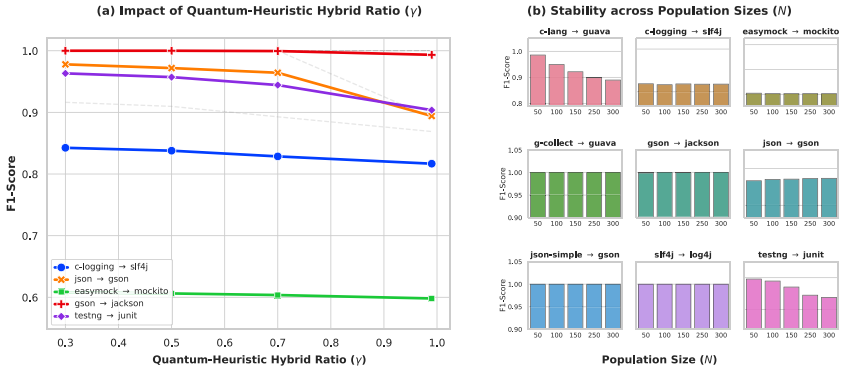

Figure 4: Sensitivity analysis of the QIMIG framework. (**a**) The impact of the quantum-heuristic hybrid ratio (γ) on mapping performance. (**b**) The stability of the algorithm across varying population sizes (N).

Impact of Population Size (N). The framework shows consistent stability across population sizes. Expanding beyond $N = 50$ yields negligible performance variation (Fig. 4b). The quantum-inspired encoding preserves genetic diversity even in restricted populations. QIMIG maintains Pareto convergence without requiring the massive populations ($N \geq 250$) typical of traditional evolutionary metaheuristics.

RQ3 Summary. While the Quantum-Heuristic Hybrid Ratio (γ) requires task-specific tuning to maximize structural exploration, QIMIG remains resilient to restrictive population constraints. Optimal mapping performance is maintained even at miniature population scales ($N = 50$), reducing computational overhead.

7 Threats to Validity

The QIMIG framework exhibits three primary limitations. First, its reliance on historical co-occurrence data creates a cold-start problem for newly released libraries, forcing a fallback to noisy lexical similarities. This dependency introduces a *historical bias*: the mined ground truth reflects patterns prevalent in the 57,447 repositories analyzed, which may include outdated coding practices or even erroneous migrations that were never corrected. Consequently, QIMIG may inherit noise from historical patterns when co-occurrence signals are strong but semantically misleading. Future work will integrate confidence scoring based on repository quality metrics to mitigate this bias. Second, static analysis cannot capture dynamic runtime behaviors or complex dependency injections, necessitating manual verification of recommended sequences. Third, the algorithm requires preliminary empirical tuning due to its sensitivity to the hybrid ratio γ.

Construct validity may be affected by omissions in the mined ground truth, which could penalize valid alternative mappings as false positives. Internal validity is safeguarded by executing 30 independent trials with controlled random seeds, applying Wilcoxon signed-rank tests, and extracting baseline metrics directly from raw artifacts to prevent implementation bias. Finally, external validity is constrained to nine Java ecosystem migration rules; The framework's reliance on static type signatures and method-level call graphs limits its direct applicability to dynamically typed languages (e.g., Python or JavaScript) where runtime duck typing blurs API boundaries. Moreover, all evaluated migrations involve imperative, object-oriented libraries; functional or reactive paradigms may introduce mapping structures (e.g., monadic transformations) that violate the current knapsack assumptions. Extending QIMIG to these ecosystems would require adapting the conflict matrix and profit definitions to accommodate language-specific abstraction mechanisms.

8 Conclusion and Future Work

This study presents QIMIG, a multi-objective optimization framework that formulates method-level API mapping as a knapsack problem to navigate sparse combinatorial spaces. The integration of quantum-inspired encoding ensures population diversity, while heuristic filters provide directional guidance, allowing the framework to achieve a superior global mean F1-score of 0.9200. Empirical evaluations confirm its operational stability and significant advantage over established search-based baselines. Future research will integrate LLMs to extract deeper syntactic dependencies, expand support to dynamic language ecosystems, and identify explicit pattern-aware encoding for better objective comprehensibility.

Acknowledgement: None.

Funding Statement: This work was partially supported by the Shanghai Yangfan Special Project, 24YF2719900; Shanghai Soft Science Research Youth Program (25692112700); China Postdoctoral Science Foundation General Program (2024M761927); Shanghai Key Technology R&D Program "Technical Standards" Project (25DZ2201200).

Author Contributions: Methodology, Yun Liu; conceptualization, Zijie Huang; software implementation, Jinghua Zhao; validation, Jianxin Ge; formal analysis, Liang Ma; supervision, Lizhi Cai. All authors reviewed and approved the final version of the manuscript.

Availability of Data and Materials: We have made our source code, datasets, baseline implementations, and full experimental results publicly available at https://github.com/escapar/QIMIG-Replication-Package.

Ethics Approval: Not applicable.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Kula RG, German DM, Ouni A, Ishio T, Inoue K. Do developers update their library dependencies? An empirical study on the impact of security advisories on library migration. Empir Softw Eng. 2018;23(1):384–417. doi:10.1007/s10664-017-9521-5.
2. Hora A, Valente MT. Apiwave: keeping track of API popularity and migration. In: Proceedings of the 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME); 2015 Sep 29–Oct 1; Bremen, Germany. p. 321–3.
3. Alrubaye H, Alshoaibi D, Alomar E, Mkaouer MW, Ouni A. How does library migration impact software quality and comprehension? An empirical study. In: Proceedings of the 2020 International Conference on Software and Software Reuse (ICSR); 2020 Dec 2–4; Hammamet, Tunisia. p. 245–60.
4. Tanaka H, Yamasaki K, Hirose M, Nakano T, Fan Y, Shimari K, et al. Mining for lags in updating critical security threats: a case study of Log4j library. In: Proceedings of the 2025 IEEE/ACM 22nd International Conference on Mining Software Repositories (MSR); 2025 Apr 28–29; Ottawa, ON, Canada. p. 319–23.
5. Dig D, Johnson R. The role of refactorings in API evolution. In: Proceedings of the 21st IEEE International Conference on Software Maintenance (ICSM); 2005 Sep 26–29; Budapest, Hungary. p. 389–98.
6. Zhong H, Thummalapenta S, Xie T, Zhang L, Wang Q. Mining API mapping for language migration. In: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering (ICSE); 2010 May 1–8; Cape Town, South Africa. p. 195–204.
7. Harman M, Jones BF. Search-based software engineering. Inf Softw Technol. 2001;43(14):833–9. doi:10.1016/s0950-5849(01)00189-6.
8. Deshpande N, Mkaouer MW, Ouni A, Sharma N. Third-party software library migration at the method-level using multi-objective evolutionary search. Swarm Evol Comput. 2024;84(11):101444. doi:10.1016/j.swevo.2023.101444.

9. Deshpande N, Mkaouer MW, Ouni A, Sharma N. Search-based third-party library migration at the method-level. In: Proceedings of the 2022 International Conference on the Applications of Evolutionary Computation (EvoApplications); 2022 Apr 20–22; Madrid, Spain. p. 173–90.
10. Alrubaye H, Mkaouer MW, Khokhlov I, Reznik L, Ouni A, Mcgoff J. Learning to recommend third-party library migration opportunities at the API level. Appl Soft Comput. 2020;90:106140.
11. Nguyen TD, Nguyen AT, Phan HD, Nguyen TN. Exploring API embedding for API usages and applications. In: Proceedings of the 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE); 2017 May 20–28; Buenos Aires, Argentina. p. 438–49.
12. Nguyen AT, Nguyen TT, Nguyen HA, Tamrawi A, Nguyen HV, Al-Kofahi J, et al. Graph-based pattern-oriented, context-sensitive source code completion. In: Proceedings of the 2012 34th International Conference on Software Engineering (ICSE); 2012 Jun 2–9; Zurich, Switzerland. p. 69–79.
13. He H, Xu Y, Ma Y, Xu Y, Liang G, Zhou M. A multi-metric ranking approach for library migration recommendations. In: Proceedings of the 2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER); 2021 Mar 9–12; Honolulu, HI, USA. p. 72–83.
14. Zhang J, Luo Q, Wu P. A multi-metric ranking with label correlations approach for library migration recommendations. In: Proceedings of the 2024 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER); 2024 Mar 12–15; Rovaniemi, Finland. p. 183–91.
15. Pandita R, Xiao X, Zhong H, Xie T, Oney S, Paradkar A. Inferring method specifications from natural language API descriptions. In: Proceedings of the 34th International Conference on Software Engineering (ICSE); 2012 Jun 2–9; Zurich, Switzerland. p. 815–25.
16. Gu X, Zhang H, Zhang D, Kim S. DeepAM: migrate APIs with multi-modal sequence to sequence learning. In: Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI); 2017 Aug 19–25; Melbourne, Australia. p. 3675–81.
17. Tang W, Xu Z, Liu C, Wu J, Yang S, Li Y, et al. Towards understanding third-party library dependency in c/c++ ecosystem. In: Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering (ASE); 2022 Oct 10–14; Oakland County, MI, USA. p. 1–12.
18. Wang Y, Zhang Y, Qin Z, Zhi C, Li B, Huang F, et al. ExploraCoder: advancing code generation for multiple unseen APIs via planning and chained exploration. In: Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (ACL); 2025 Jul 27–Aug 1; Vienna, Austria. p. 18124–45.
19. Pan R, Ibrahimzada AR, Krishna R, Sankar D, Wassi LP, Merler M, et al. Lost in translation: a study of bugs introduced by large language models while translating code. In: Proceedings of the IEEE/ACM 46th International Conference on Software Engineering (AsE); 2024 Oct 27–Nov 1; Sacramento, CA, USA. p. 1–13.
20. Tambon F, Moradi-Dakhel A, Nikanjam A, Khomh F, Desmarais MC, Antoniol G. Bugs in large language models generated code: an empirical study. Empir Softw Eng. 2025;30(3):65.
21. Kebir S, Borne I, Meslati D. A genetic algorithm-based approach for automated refactoring of component-based software. Inf Softw Technol. 2017;88(11):17–36. doi:10.1016/j.infsof.2017.03.009.
22. Han KH, Kim JH. Quantum-inspired evolutionary algorithm for a class of combinatorial optimization. IEEE Trans Evol Comput. 2002;6(6):580–93. doi:10.1109/tevc.2002.804320.
23. Naik BB, Priyanka B, Ansari MSA. Energy-efficient task offloading and efficient resource allocation for edge computing: a quantum inspired particle swarm optimization approach. Clust Comput. 2025;28:155.
24. Hussein H, Younes A, Abdelmoez W. Quantum algorithm for solving the test suite minimization problem. Cogent Eng. 2021;8(1):1882116. doi:10.1080/23311916.2021.1882116.
25. Nehzati M. A quantum-inspired, biomimetic, and fractal framework for self-healing AI code generation: bridging responsible automation and emergent intelligence. Front Artif Intell. 2025;8:1662220. doi:10.3389/frai.2025.1662220.
26. Wilcoxon F. Individual comparisons by ranking methods. Biom Bull. 1945;1(6):80–3. doi:10.2307/3001968.
27. Rosenthal R, Cooper H, Hedges L. Parametric measures of effect size. Handb Res Synth. 1994;621(2):231–44.