Applied Computer Systems
ISSN 2255-8691 (online)
2026, vol. 31, no. 1, pp. 116–126
https://doi.org/10.2478/acss-2026-0010
https://content.sciendo.com

Adaptive Synthesis of Intelligent Monitoring Models for LLM-modified Information Systems: A Genetic Optimisation Approach

Vasyl Lyashkevych*
Ivan Franko National University of Lviv, Lviv, Ukraine

***Abstract* – This study investigates adaptive synthesis of intelligent monitoring (IM) for applications modified by large language models (LLMs). A multi-objective, property-driven genetic approach with a tensor of chromosome representation is proposed to generate monitoring configurations under architectural changes. Using migration from Oracle to PostgreSQL as a controlled case, the study shows that functional equivalence can be maintained while monitoring-related properties drift, supporting adaptive architecture-aware monitoring over static settings.**

***Keywords* – adaptive systems, context-aware services, control systems synthesis, information technology, genetic algorithms, large language models, monitoring, multi-agent systems, optimisation, software systems.**

I. Introduction

LLMs have rapidly evolved from code completion tools to a broader range of software development (SDLC) tools that support requirement analysis, code generation, refactoring, debugging, testing, documentation, and analysis. Recent research describes this evolution as a move toward hybrid SDLC workflows, where LLMs complement rather than completely replace traditional analysis, verification, and optimisation methods [1]. At the same time, these studies show that the outputs of LLM-based SDLC remain probabilistic and may contain incomplete, inconsistent, or weakly justified decisions; therefore, reliable downstream quality assurance is required [2]–[3].

This problem is particularly prominent in the field of software testing. Studies of automated unit tests using LLMs show that while such models can generally generate executable test cases and improve development efficiency, they also exhibit some recurring limitations, such as limited fault-detection capability, incomplete path coverage, sensitivity to the context of test prompts, and inconsistent test results across tasks [4]–[8]. Recent reviews confirm that a robust evaluation of LLM-based test generation requires more advanced metrics than simple coverage, especially estimates of mutation, pass rate, and error detection behaviour [2]–[8].

Research on adaptive systems shows that monitoring is not architecture neutral [9]–[12]. Adaptive decisions depend on the monitored objects, allocation of monitoring points (MPs), signal interpretation, and the priority of quality properties. Therefore, changes to the architecture or runtime configuration may require corresponding adjustments to the monitoring configuration itself [9]–[13].

This problem is especially evident in component-oriented and microservices systems. Practical experience shows that the design, monitoring, and testing of such information systems (ISs) are closely related, and the complexity of the IS creates monitoring and testing challenges that static methods cannot fully solve [14]–[16]. In other words, simply preserving the initial monitoring settings is often not enough when the IS topology or interaction logic changes [2], [13].

Against this background, one of the most useful controlled scenarios for studying IS changes caused by LLM is code migration. The recently published "A Fine-Tuned LLM-Based Code Migration Framework" describes a fine-tuned LLM-centric approach to converting Oracle PL/SQL to PostgreSQL, including syntax mapping, handling of Oracle-PostgreSQL mismatches, and optimisation of stored procedures, triggers, views, and database logic. The scenario is methodologically attractive, because the target IS intends to preserve functional equivalence at the business level while being amenable to changes in implementation, compatibility assumptions, behaviour and available monitoring-related properties [17].

By analogy, intelligent monitoring (IM) for LLM-modified ISs can also be formulated not as a flat configuration problem, but as a synthesis problem with a tensor structure covering architectural layers, controlled properties, and adaptation contexts [18]–[20]. To address this issue, a recent study [20] on the sustainable optimisation of integrated data processing algorithms offers a useful methodological extension. The study uses genetic algorithms (GAs) to synthesize complex multi-stage processes. This study is particularly relevant to this work because it demonstrates that evolutionary search can be represented using structured multi-stage configurations in tensor form. This allows for the parallel evolution of multiple related workflows with structural and coverage constraints.

Thus, this study explores the possibility to synthesize an IM model based on changes in IS's properties and its configuration. This investigation proposes a property-based genetic synthesis method for configuration monitoring and position it as a bridge between LLM-based evolutionary SDLC architecture, adaptive

* Corresponding author's e-mail: vasyl.liashkevych@lnu.edu.ua
Article received 2026-04-06; accepted 2026-05-22

monitoring, architecture optimisation, and tensor-based structured analysis workflows [3], [9], [11]–[13], [18]–[20].

II. Related Work

A. LLMs in Software Engineering

The current literature in SDLC has repeatedly emphasised that LLMs are increasingly used as general-purpose tools in many areas of SDLC, not just code generation. A significant review concludes that LLM-based SDLC should be viewed as a hybrid discipline that combines generative models with traditional SDLC methods to improve reliability and usability. This perspective is directly related to monitoring, as it suggests that the artefacts generated by LLM should be reviewed and managed through additional levels of analysis, rather than accepted as a final version in advance [1]–[3].

The literature on testing presents both promising and cautious approaches. Empirical studies on the use of LLMs for automated unit test generation show significant efficiency, but also reveal limitations regarding semantic accuracy, assertion quality, and behavioural adequacy [4]–[8]. Extension methods based on search and mutation guidance, such as CodaMOSA and later mutation-based prompt methods, demonstrate that LLMs can be even more powerful when embedded in broader optimisation or search processes [2]–[3], [5]–[8].

B. Monitoring and Self-adaptive ISs

The field of adaptive ISs research considers monitoring, analysis, planning, and execution as closely related processes [9]. Recent studies in adaptive programming have shown that the adaptation space is often too large for comprehensive analysis, highlighting the growing importance of optimisation techniques, especially heuristic algorithms, learning methods, and GAs [12]. Recent research on generative artificial intelligence (GAI) for adaptive ISs shows that GAI can support such ISs, but also raises new research questions regarding quality assurance, adaptive control, and integration into architectural considerations. The reliability of performance predictions based on the architecture confirms the assumption that changes to the system architecture affect the quality characteristics that should be monitored [13], [19].

C. Architecture Optimisation and Monitoring Configuration

Systematic analysis of software architecture optimisation shows that the architecture-related decision space is inherently multi-criteria, typically involving trade-offs between performance, cost, reliability, and other quality properties [18]. This is directly reflected in the monitoring configuration, where detection quality, coverage of relevant properties, overhead, and adaptation workload must be balanced. Furthermore, microservices research shows that monitoring decisions are closely related to decomposition and interaction models, further supporting a comprehensive architecture-centric approach to monitoring [3], [13]–[16].

D. Related Contributions to the Problem Domain

Current research on the concept of software functional state is particularly important because it defines software state within the SDLC as a configuration of usability, quality, risk, security, and operational parameters shaped by artefacts, solutions, and environmental factors. This perspective provides a strong conceptual foundation for the synthesis of property-based monitoring because it interprets the state of the IS as a structured configuration rather than as a discrete scalar indicator [21]. Another closely related contribution is the Oracle to PostgreSQL migration framework, which provides a specific transformation scenario based on a fine-tuned LLM suitable for controlled experimental analysis [17].

E. Tensor-based Evolutionary Synthesis

Recent studies on the sustainable optimisation of integrated data processing algorithms have expanded the scope of GAs, extending their application from parameter optimisation to the synthesis of structured analysis workflows. In this model, each decoder encodes the process of creating visualisation components, while the population is represented as a tensor, allowing for the parallel development of multiple workflows [20]. The operations are classified into four types: Extract, Transform, Load (ETL), Machine Learning (ML), LLM, and visualization. The fitness functions evaluate the duration of the workflow, the coverage of operation types, and the structural consistency [20]. This contribution is crucial for this study because it suggests a directly transferable design principle: in ISs modified by LLM, monitoring synthesis can be viewed as an evolutionary design of a multi-step, structurally constrained workflow, rather than as a simple vector of independent monitoring parameters. From this perspective, the chromosome can be naturally represented as a tensor whose axes correspond to the monitored component, MPs, properties, telemetry settings, and adaptive operation, respectively [20].

III. Problem Statement

A common mistake in discussions of LLM-based IS transformations is the assumption that monitoring settings can remain unchanged if the transformed IS retains its target business functionality. This assumption is too strong. For example, migrations from Oracle PL/SQL to PostgreSQL are typically designed to preserve the target functionality. However, the migration process itself must clearly consider aspects such as syntactic differences, logical transformations, stored procedures, triggers, and database-specific behaviour. This means that functional equivalence is a goal, not an automatic guarantee, and that implementation-level and non-functional properties can change even when business behaviour must remain stable [17].

Thus, this paper distinguishes three categories of IS's properties. The first category includes properties of functional goals that describe what the IS should achieve from a user or business perspective. The second category includes architectural and implementation properties, including component relationships, dependency structures, execution paths, integration contracts, and platform-specific constructs. The third category is non-functional properties such as performance, reliability, telemetry overhead, susceptibility to fault propagation, and operational observability [19]. A configuration change can preserve the first category while changing the second and third [17], [21].

Therefore, following [13] and [17], the research question can be formulated as follows: Given an IS that has been

significantly generated, migrated, or modified using LLM, and a set of required system properties and constraints, how can one automatically synthesize a monitoring configuration that remains relevant after architectural or implementation changes, while maintaining a balance between detection quality and operational overhead [9], [11]–[12], [18]–[19]?

Unlike scalar or flat-vector formulas, we view the monitoring synthesis problem as a tensor structure search problem. This is based on the observation that monitoring decisions are not independent of each other: MP, indicator choice, telemetry granularity, structure coverage, and adaptive behaviour form a multidimensional configuration space [18], [19]. According to the principle of tensor growth proposed for the consolidated evolution of the analytical pipeline, the task of monitoring synthesis is reformulated as the search for a structurally consistent tensor chromosome as a decision along multiple axes of the IS [20].

Therefore, the main hypothesis of this study is provided below.

H1. For ISs significantly modified by LLM, a property-based monitoring configuration synthesized by a multi-objective GA will provide better monitoring relevance and error detection quality under IS changes than a static predefined monitoring configuration. The expected benefit of the proposed approach is particularly large when monitoring configurations are represented as tensor chromosomes, as such a representation preserves the interdimensional dependencies between architecture, properties, and telemetry settings more effectively than linear coding. This is consistent with the logic of recent tensor-based evolutionary pipeline synthesis, where structured multi-step solutions outperform random and weakly structured encodings, converging towards logically consistent workflows [13], [18]–[20].

The supporting hypotheses are outlined below.

H1a. Configuration changes caused by LLM can maintain the expected functional equivalence, even though the implementation changes with respect to monitoring and non-functional properties [17].

H1b. As architecture drift increases, the effectiveness of static monitoring decreases because the original mapping between system properties and MPs becomes obsolete [9], [11], [13], [19], [21]–[24].

H1c. GAs and multi-objective evolutionary optimisation algorithms are well suited for monitoring synthesis due to the large search space and inherent conflicts between objectives such as coverage, detection quality, and workload [12], [13], [18]–[20].

H1d. Tensor coding of chromosomes provides more structurally consistent and adaptive monitoring configurations than flat-vector representations because it better captures the interactions between monitored features, components, and workflow steps [18], [20].

IV. Method and Materials

A. Experimental Sample of Dynamic ISs

Let us consider one illustrative example of IS (Fig. 1). Following the recent trend in LLM-driven ISs, the architecture continues to be significantly evolved [1], [25]. At the architecture design stage, we define requirements for the IS and its monitoring subsystem, expected inputs, a nominal component graph, MPs, expected outputs, etc.

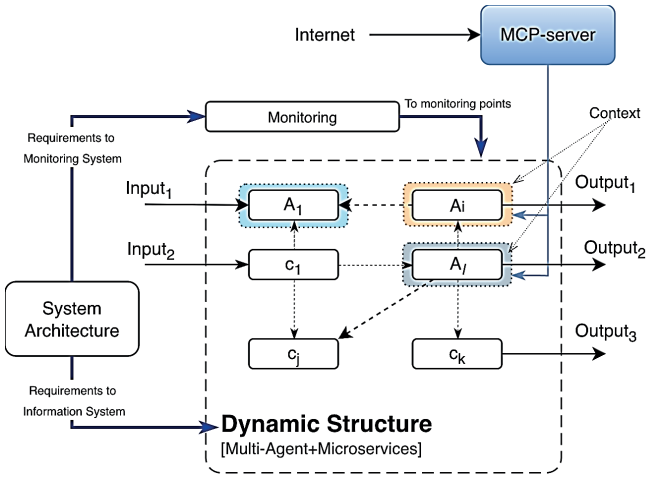

Fig. 1. A high-level example of dynamic structure of ISs.

After deployment, IS may evolve dynamically at runtime. Components such as agents A_1, A_i and A_l appear or change behaviour at runtime, connect to external services such as the Model Context Protocol (MCP) server, and start producing new outputs that were not in the original blueprint. The microservice components are c_1, c_i and c_k respectively. Those outputs are shaped not only by the internal logic of the components but also by contextual factors such as user behaviour, partner APIs, data drift, infrastructure incidents, and regulatory changes. Such conditions cannot be fully enumerated at design time because the monitoring loop that looked straightforward in the static picture becomes under-specified now. The measurements that seemed sufficient in design quickly become blind spots when components scale, reconfigure or pull in third-party logic. The system is no longer a fixed pipeline.

The IS chosen for experiments is the migration scenario from Oracle PL/SQL to PostgreSQL described in [17]. This scenario is suitable because it represents an LLM-oriented transformation of an IS, where the target behaviour must remain unchanged, while the syntax, code organisation, database-specific structure, and execution assumptions can change (Fig. 2). The framework explicitly considers stored procedures, triggers, views, and logical transformations, thus providing an ideal environment for monitoring experiments.

The chosen IS is defined as a multi-component IS. In this interpretation, IS is not only the Oracle or PostgreSQL program itself, but also the entire interacting architecture of software components, data stores, agents, business logic services, and monitoring mechanisms. Figure 2 also shows how monitoring changes are defined. In Stage 1, an Oracle-based IS is supported by a monitoring subsystem version 1.

After Stage 2, which is a migration from Oracle to PostgreSQL, the target IS in Stage 3 retains the general business role and high-level architecture, but its internal configuration changes: the main software platform becomes PostgreSQL-based, the size of the script base changes, and the interaction of components may be affected. Due to this configuration change, the initial monitoring setup is no longer considered fully

adequate. Therefore, monitoring also has to evolve from version 1 to version 2.

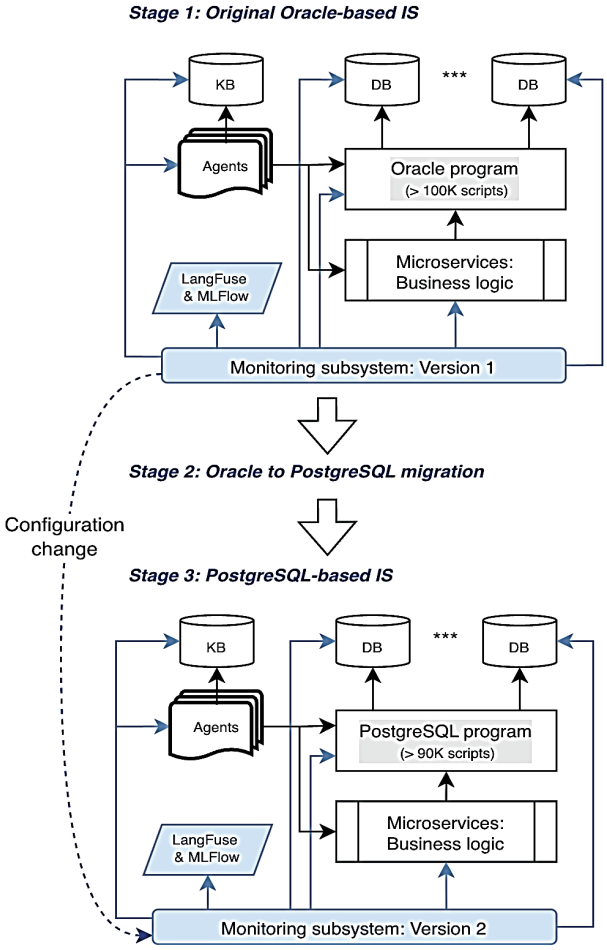

Fig. 2. An experimental IS transformation approach.

According to the chosen scenario, monitoring changes are defined as the adaptation of the monitoring subsystem in response to a structural or platform reconfiguration of the IS, even when the IS retains its general purpose and underlying business logic. This migration scenario is not a case of "configuration changes do not affect properties". Rather, it represents a more precise and scientifically valuable case: the configuration changes are intended to maintain functional equivalence at the business level, while architectural and operational properties may still change. This distinction makes this scenario particularly valuable for this study, as it allows us to analyse whether monitoring needs to be adjusted even if the target functionality remains unchanged.

B. Tensor-based Property Model

The property model contains three levels:

- Functional equivalence properties, such as output consistency for migrated queries and procedural correctness for migrated database logic.
- Structural properties, such as dependency consistency, trigger relationships, stored procedure call paths, and schema interaction constraints.
- Operational properties, such as response latency, resource utilisation, anomaly detection, logging completeness, and telemetry cost.

Such a multi-level representation is compatible with the concept of software functional state, where the state of the software is modelled as a configuration of parameters related to quality, risk, and operability, rather than as a single metric [9], [21].

To support tensor coding, the property model is organised as a multi-axis structure. One axis corresponds to the components or relationships of the IS, another to the functional, structural, and operational properties, another to the mechanisms of observation, and yet another to the contexts or adaptation scenarios. This decomposition is consistent with tensor-based evolutionary design, where several related analytical or decision-making processes evolve in parallel while maintaining internal structural coherence [20].

For the purposes of synthesis, each monitored property $p \in P$ is associated with: semantic definition; a set of permissible observation mechanisms; the required or desired level of confidence; one or more evaluation scenarios; optional weight of value or criticality. This avoids treating all properties as equally important. In practice, some properties are mandatory and must always be considered, while others are desirable but negotiable under cost constraints.

The monitoring synthesis problem operates on a multidimensional space $V \times P \times O \times S$, rather than a flat set of independent parameters, where: V – a set of IS components or component relations; P – a set of monitored properties; O – a set of monitoring operations; S – a set of execution or adaptation scenarios. This formulation is critically important because the same MP may be useful for one property in one scenario and inappropriate or too expensive in another.

C. Tensor-based Chromosome Representation

Each chromosome encodes one candidate monitoring configuration. In the updated formulation, a chromosome is represented as a tensor rather than a flat vector. Let us denote a chromosome:

$$C = (C_{\text{comp}}, C_{\text{rel}}, C_{\text{param}}, C_{\text{adapt}}, C_{\text{trace}}), \quad (1)$$

where: C_{comp} – component-level monitoring tensor, C_{rel} – relation-level monitoring tensor, C_{param} – parameter tensor, C_{adapt} – adaptation tensor, and C_{trace} – property-traceability tensor.

This representation follows the same methodological principle as the evolution of consolidated pipelines based on GA tensors, in which a single structured decision contains multiple coordinated sub-flows rather than a single linear chain of actions [18], [20].

For practical implementation, the tensor chromosome should be interpreted as a mixed discrete-continuous representation. Some entries are binary or categorical, for example, whether the MPs is activated, which metric family is assigned, or which aggregation rule is selected. Other entries are numeric, for example, sampling interval, threshold value, composite indicator weight, or telemetry budget allocation. Therefore, a tensor chromosome can be understood as a structured object consisting of several aligned subtensors: activation subtensor;

parameter subtensor; property mapping subtensor onto MPs and optional adaptation subtensor defining response policies.

According to this approach, crossover and mutation affect not only scalar genes, but also tensor slices, subtensors, or semantically consistent monitoring blocks. This allows the optimizer to extract meaningful structural fragments during evolution, such as component-specific monitoring subgraphs or trait-specific diagnostic processes. Therefore, population-level evolution can be interpreted as a transformation of the tensor population, rather than simply a modification of gene chains.

To reduce the risk of invalid solutions, tensor operators should be constrained by feasibility rules [12], [18]–[19], for example:

- Mandatory properties must remain covered;
- Incompatible MP and property combinations are prohibited;
- Budget constraints must not be violated;
- Adaptation rules must be tied only to valid diagnostic outputs.

This is important because unconstrained genetic search in a multidimensional tensor space can lead to formally encoded but semantically meaningless monitoring structures.

D. Multi-objective Fitness Function

Candidate monitoring configurations should be evaluated based on the following objectives:

- Maximum quality of anomaly and fault detection;
- Maximum coverage of required properties;
- Maximum structural relevance after system changes;
- Minimisation of telemetry overhead;
- Minimisation of false alarms and misdiagnoses;
- Minimisation of reconfiguration effort after subsequent system deviations.

Therefore, a Pareto-based optimizer such as NSGA-II is a natural choice [12], [18]–[19]. The exact optimizer may be varied in future experiments, but the problem structure itself is inherently multi-objective.

The fitness design is extended according to the logic of tensor-based pipeline optimization proposed in [20], where suitability evaluates not only performance but also the constant length of the pipeline, the coverage of key classes of operations, and structural consistency. In this study, this idea is transferred to monitoring synthesis by introducing three additional suitability dimensions:

- Monitoring adequacy;
- Monitoring workflow coverage;
- Structural consistency of the tensor chromosome.

Thus, in addition to detection quality and overhead, the optimizer explicitly rewards monitoring configurations that cover the required classes of monitoring actions and preserve internal logical consistency between tensor dimensions.

A generalized scalarized formulation may be written as:

$$F(C) = \alpha \cdot DQ(C) + \beta \cdot PC(C) + \gamma \cdot SC(C) - \\ -\delta \cdot OH(C) - \varepsilon \cdot AN(C), \quad (2)$$

where: DQ – detection quality, PC – coverage of properties and classes of operations, SC – structural consistency, OH – overhead AN – notification noise, ⟨, ®, ©, ™, $\sum \geqslant 0$, and ⟨ + ® + © + ™ + $\sum = 1$.

However, for the actual study, a vector objective formulation is preferable:

$$\max f(C) = (DQ(C), PC(C), SC(C), ROB(C)) \\ \min g(C) = (OH(C), AN(C), RC(C)), \quad (3)$$

where ROB – robustness under architectural drift and RC – reconfiguration cost.

This formulation is more scientifically convincing because it avoids hiding trade-offs through arbitrary scalar weights [18]. The scalarized version can still be used for sensitivity analysis, but the main optimisation study should report Pareto-front results. To prevent circularity, it is also important to distinguish between:

- Optimisation time objectives used by the GA during synthesis;
- Evaluation time metrics used later for the experimental comparison of monitoring strategies.

The same conceptual dimensions may appear in both places, but they should not be methodologically mixed.

E. Experimental Basis

The experimental setup should additionally include a tensor-aware configuration synthesis mode, in which each candidate monitoring setup is encoded as a multidimensional chromosome and evolved in parallel over multiple migration scenarios. Methodologically, this builds on previous work on tensor-growing populations of GAs for consolidated data processing algorithms, where parallel workflow evolution enabled the automated construction of multicomponent analytical solutions. In this study, the same principle is adapted to multicomponent monitoring synthesis.

This is necessary because LLM-driven transformations are stochastic, and the results of individual instances may not generalize. Thus, for the experimental evaluation, five monitoring strategies were considered:

- M0: static monitoring defined before migration and left unchanged after;
- M1: monitoring manually configured after migration;
- M2: adaptive rule-based monitoring;
- M3: GA with vector representation of multi-objective evolutionary synthesis of monitoring;
- M4: GA with tensor representation of multi-objective evolutionary synthesis of monitoring.

The difference between M4 and M3 is not only in the GA used, but also in the structure of the chromosome. M3 uses traditional planar or vector coding, while M4 uses tensor chromosomes, the dimensions of which explicitly model the relationships among components, properties, telemetry operations, and adaptation stages. This allows for direct empirical comparisons between planar and tensor evolutionary coding.

F. Evaluation Metrics

Evaluation of the migration process itself lies out of this study as it has already been analysed in [17]. Therefore, the evaluation focuses on the monitoring subsystem version 2, to have a vision if the monitoring configuration can be adequately re-synthesized and applied after the transition to the target IS.

Let $P = \{p_1, \dots, p_J\}$ – a set of monitored properties, $M = \{m_1, \dots, m_K\}$ – a set of monitoring points, $O = \{o_1, \dots, o_L\}$ – a set of monitoring operations, and C – a candidate monitoring configuration. Let $w_j \geq 0$ – the criticality weight of property p_j, $\mathrm{cov}(p_j, C) = 1[\exists\, \pi \in \Pi(C, p_j)]$ – a binary coverage indicator.

On a par with conventional metrics such as Precision, Recall, and F1-score let us consider Mean Time to Detection (MTTD), Localization Accuracy (LA) and False Alarm Rate (FAR):

$$MTTD = \frac{1}{N}\sum_{n=1}^{N}\left(t_n^{\mathrm{det}} - t_n^{\mathrm{start}}\right), \quad (4)$$

where N – total number of detected anomaly cases included in evaluation; n – index of one anomaly case, where $n = 1 \dots N$; t_n^{start} – time when anomaly n starts; t_n^{det} – time when anomaly n is detected by the monitoring subsystem.

$$LA = \frac{N_{\mathrm{loc}}^{\mathrm{correct}}}{N_{\mathrm{anomaly}}}, \quad (5)$$

where $N_{\mathrm{loc}}^{\mathrm{correct}}$ – number of anomaly cases for which the faulty location was identified correctly; N_{anomaly} – total number of anomaly cases evaluated.

$$FAR = \frac{FP}{TP+FP}, \quad (6)$$

where *TP*, *FP*, and *FN* are true positives, false positives, and false negatives, respectively.

To test the main hypothesis, the study also employs property-aware metrics. The Property Coverage Ratio (*PCR*):

$$PCR(C) = \frac{1}{J}\sum_{j=1}^{J}\mathrm{cov}\left(p_j, C\right), \quad (7)$$

or, in weighted form:

$$PCR_w(C) = \frac{\sum_{j=1}^{J} w_j\, \mathrm{cov}(p_j, C)}{\sum_{j=1}^{J} w_j}, \quad (8)$$

Let $C_{\mathrm{trace}}(j,k,l) \in \{0,1\}$ whether property p_j – traceably covered through monitoring point m_k using operation O_l. The Traceability of Properties to Probes (*TPP*):

$$TPP(C) = \frac{1}{J}\sum_{j=1}^{J} tr\left(p_j, C\right), \quad (9)$$

where

$$tr\left(p_j, C\right) = 1 \; if \; \sum_{k=1}^{K}\sum_{l=1}^{L} C_{\mathrm{trace}}(j,k,l) > 0 \; else \; 0. \quad (10)$$

The Monitoring Reconfiguration Success Rate (*MRSR*):

$$MRSR = \frac{1}{R}\sum_{r=1}^{R} s_r, \quad (11)$$

where $s_r = 1$ if reconfiguration r preserves mandatory properties and feasibility constraints, else *O*. The Architecture Change Sensitivity (*ACS*):

$$ACS = \frac{Q_{\mathrm{base}} - Q_{\mathrm{drift}}}{Q_{\mathrm{base}}}, \quad (12)$$

where Q_{base} and Q_{drift} – monitoring-quality scores before and after architectural drift.

To reflect the tensor formulation, three tensor-aware metrics are introduced. The Tensor Structural Consistency (*TSC*):

$$TSC(C) = \frac{N_{\mathrm{valid}}}{N_{\mathrm{all}}}, \quad (13)$$

where N_{valid} – number of semantically valid tensor assignments and N_{all} – total number of active assignments. The Operation-Class Coverage (*OCC*):

$$OCC(C) = \frac{|O^{\mathrm{req}} \cap O^{\mathrm{used}}(C)|}{|O^{\mathrm{req}}|}, \quad (14)$$

where O^{req} – set of required operation classes and $O^{\mathrm{used}}(C)$ – set used by chromosome *C*. The Tensor Workflow Completeness (*TWC*):

$$TWC(C) = \frac{|W^{\mathrm{valid}}(C)|}{|W^{\mathrm{req}}|}, \quad (15)$$

where W^{req} – set of required monitoring subtasks, $W^{\mathrm{valid}}(C)$ – subset realized by valid end-to-end monitoring paths.

For the optimizer itself, the study evaluates Pareto Front Quality (*PFQ*), Hypervolume (*HV*), Convergence Stability (*CS*), and Cross-run Robustness (*CRR*). *HV* is computed in the standard form with respect to a reference point *r*:

$$HV(F) = \lambda\left(\cup_{X \in F}[f_1(C), r_1] \times \cdots \times [f_n(C), r_n]\right), \quad (16)$$

where F – obtained Pareto set. For the tensor-based variant, two additional optimisation metrics are used: Tensor Slice Stability (*TSS*) and Subtensor Reuse Ratio (*SRR*), which characterise the preservation of semantically meaningful tensor structures across repeated runs and evolutionary operators.

Although the main optimisation problem is multi-objective, the experiments additionally use a scalarized fitness function for comparative analysis:

$$F(C) = 0.35 \cdot DQ(C) + 0.25 \cdot PC(C) + 0.2 \cdot SC(C) - \\ -0.10 \cdot AN(C) - 0.05 \cdot ACS(C) - 0.05 \cdot OH(C), \quad (17)$$

where $DQ(C) = F1(C)$, $PC(C) = PCR(C)$, $SC(C) = TSC(C)$, $AN(C) = FAR(C)$, and $OH(C)$ – normalized monitoring overhead. These metrics allow us to assess both the ultimate quality of monitoring and the internal search dynamics of the proposed tensor-based evolutionary synthesis approach.

V. Results and Discussions

A. Property-driven Adaptive Monitoring Model Synthesis

The synthesized monitoring model is no longer viewed as a static configuration file, but rather as a structured monitoring workflow with phases such as data collection, transformation, aggregation, evaluation, diagnosis, and adaptation. This interpretation is critical because systems optimised by LLM rarely involve a simple selection of a few metrics; instead, they

typically require the establishment of a continuous monitoring process that remains effective even when the system deviates.

The proposed method generates and optimises monitoring configuration that clearly defines the location of monitoring devices and MPs, the content to be monitored, the frequency of telemetry collection, and the rules or composite metrics to be used. The method is in line with research directions in adaptive systems and architecture-based decision making, where monitoring is part of an adaptive control loop rather than a fixed external component [9], [11], [13]. Each candidate solution defines not only a set of MPs and thresholds, but also an ordered and structurally constrained monitoring workflow that can include data collection, aggregation, anomaly assessment, property-level interpretation, and adaptation triggering. This makes the monitoring configuration an executable analytical pipeline rather than a static set of settings [18], [20].

B. Encoding of Monitoring System and IS Configurations

All input data is encoded in a composite tensor chromosome. The configuration of the IS is represented by sets of components and relationships that restrict the permissible monitoring locations. Figure 3 shows the experimental sample of IS used in our study – the sets of components, relationships, monitored properties, MPs, monitoring operations, scenarios, and adaptation actions.

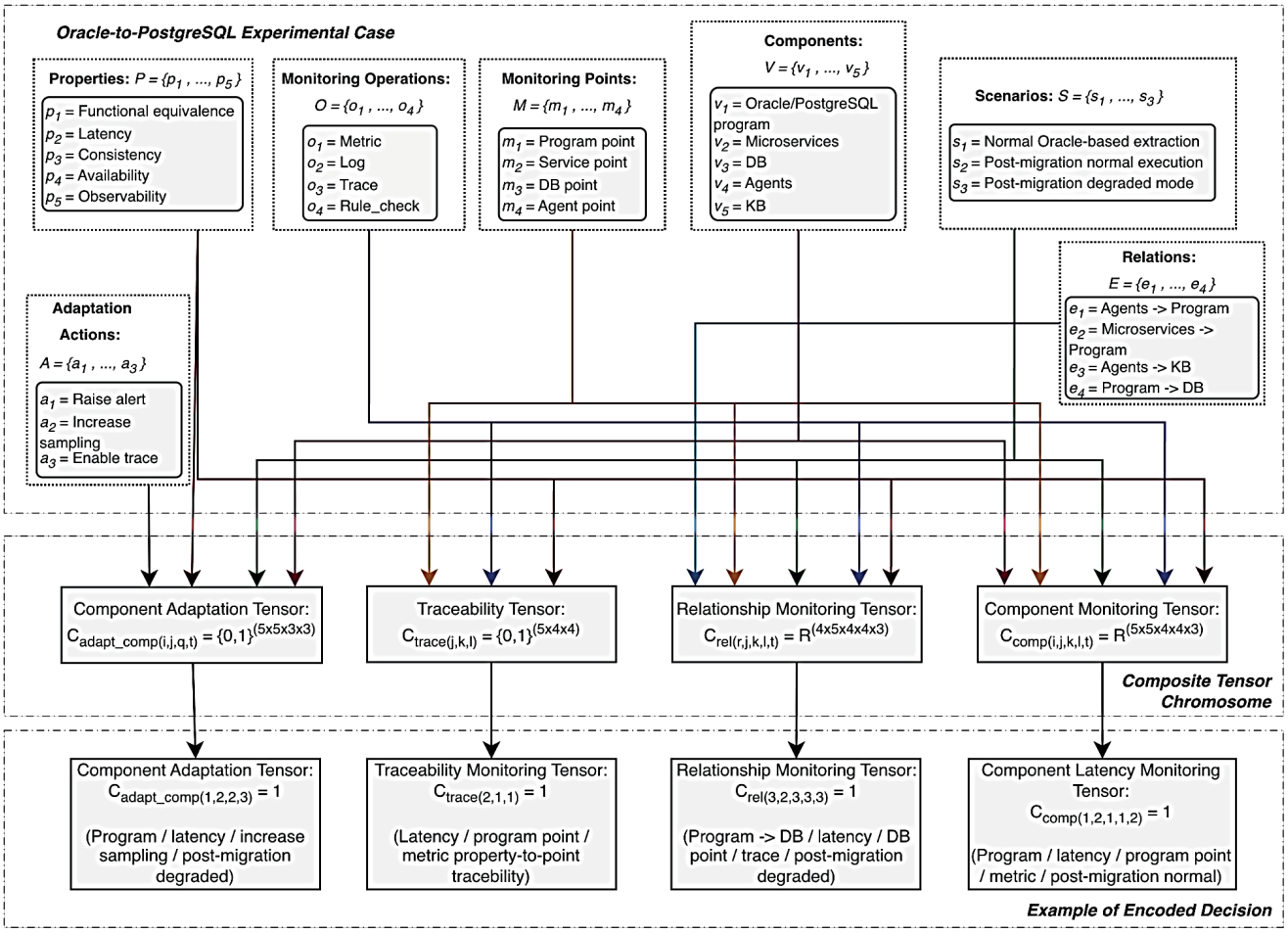

Fig. 3. Example of monitoring solution to experimental IS in post-migration Oracle to PostgreSQL stage.

These elements are encoded in a composite tensor chromosome, which forms the basis for the evolutionary synthesis of the updated monitoring subsystem. Thus, the figure visualizes how the experimental IS for adaptive monitoring optimisation is formally represented.

C. Crossover Operator

The crossover operator operates on semantically coherent tensor blocks, not on isolated scalar genes. Separate crossover points are defined for component-level monitoring, relation-level monitoring, parameter subtensors, adaptation subtensors, and traceability subtensors. After recombination, a correction step is applied to restore appropriateness with respect to mandatory property coverage, valid probe-property-operation mappings, resource constraints, and adaptation logic. There are some strategies for the crossover operator implementation, depending on the type of structural integrity that needs to be preserved during recombination. Slice-based crossover, where the exchange is carried out along one of the tensor axes, for example, along the *i* axis, which corresponds to the IS components, or along the *j* axis, which corresponds to the monitored properties. In this case, the offspring inherits entire

slices of solutions associated with a specific component or property, which allows for locally consistent monitoring blocks to be preserved.

Scenario-block crossover, where all decisions corresponding to a single scenario t are exchanged. This means that for a particular execution context, such as post-migration degraded mode, the offspring receives a complete consistent block of monitoring decisions from one of the parents. This type of crossover is especially useful when the monitoring structure is highly scenario-dependent and the integrity of the scenario-specific monitoring workflow is critical.

Property-block crossover, where all solutions related to a certain property p_j are recombined. In this case, the offspring inherits a full subtensor that describes how one particular property, such as latency or consistency, is monitored across different components, through different MPs, operations, and scenarios. Such a mechanism preserves the integrity of the property-specific monitoring logic and is particularly useful for problems in which the monitored properties have different criticalities and different permissible observation mechanisms.

Component-relation hybrid crossover, where the subtensors C_{comp} and C_{rel} are crossed independently or semi-independently. This allows us to maintain a balance between component-oriented and relation-oriented monitoring synthesis. For example, the offspring can inherit component-level monitoring blocks from one parent and relation-level monitoring blocks from the other, after which it can go through a repair stage to restore the overall consistency of the chromosome.

In the experiments, the crossover operator was implemented as a combined scenario-block and property-block tensor crossover. Such a combined operator allows us to simultaneously preserve two critically important forms of structural consistency:

- First, the integrity of monitoring workflows specific to a particular execution scenario;
- Second, the integrity of logic blocks responsible for monitoring individual IS properties.

This option best meets the task of adaptive synthesis of monitoring after Oracle-to-PostgreSQL migration, where a change in IS configuration affects both execution scenarios and methods of monitoring individual functional, structural, and operational properties.

D. Experiment Settings

The experimental procedure (Fig. 4) consisted of deploying an Oracle-based program. Totally, 2K Oracle scripts – with multiple workloads, complexity, and failure scenarios – were used as Oracle programs in Azure, implementing baseline monitoring, and recording the initial architectural and observability state.

The Oracle-based subsystem was then migrated to PostgreSQL, and the post-migration monitoring synthesis problem was encoded as a composite tensor chromosome. An initial population of 100 feasible chromosomes was generated and evolved using selection, crossover, mutation, repair, and repeated evaluation. The best synthesized monitoring configuration was implemented in Azure as Monitoring Subsystem Version 2 and assessed under normal, degraded, and fault-injection scenarios. The resulting metrics were recorded and compared across monitoring strategies M0–M4.

The population size was set at 100 chromosomes, with 150 iterations, a crossover rate of 0.85, and a mutation rate of 0.10. Parent selection employed a binary tournament selection method based on non-dominated ranking and crowding distance, following the standard NSGA-II algorithm. The termination condition was set at reaching a pre-set number of iterations; if the improvement in population oversize remained below a certain threshold for 15 consecutive generations, an early termination condition was activated. After crossover and mutation, each offspring chromosome was processed by a repair strategy algorithm designed to restore the population's feasibility in terms of necessary coverage property, effective MP-property-operation mapping, resource constraints, and fitness consistency.

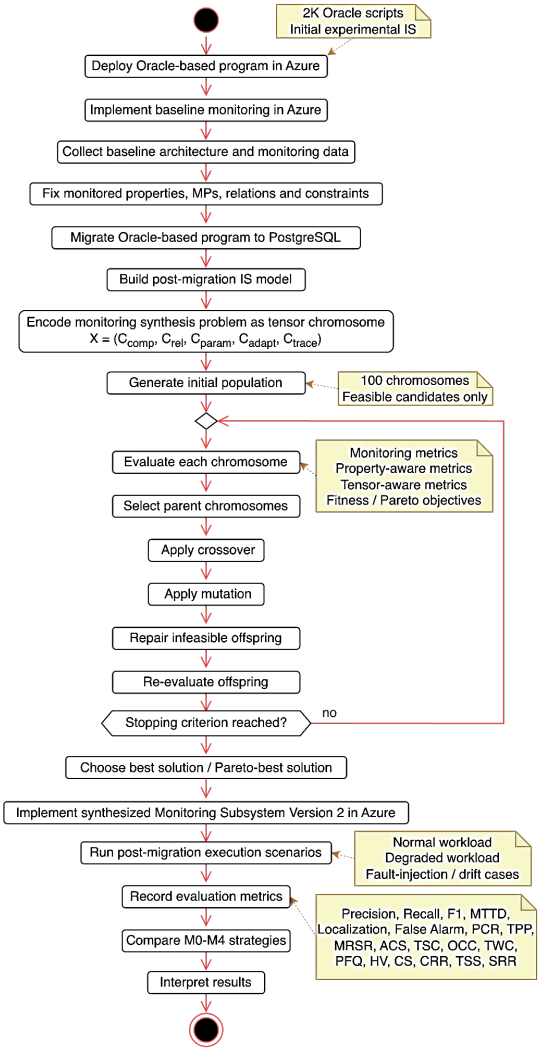

Fig. 4. Experimental algorithm for adaptive monitoring synthesis.

To evaluate the proposed monitoring configuration method, we executed controlled experiments under conditions that reflected the structure of the target problem, had a controlled computational complexity, and were statistically interpretable. The experiments included five monitoring strategies (M0 to M4), three execution scenarios (baseline, normal after migration, and degraded after migration), and 40 independent runs of each strategy with scenario combination. Thus, the total

experiments workload was 600. This experimental framework is sufficient to reveal stable comparative tendencies.

Firstly, the integration of the five monitoring strategies allows the proposed tensor evolution method to be compared not only with static benchmarks but also with intermediate methods such as manual and rule-based adjustment, as well as vector evolutionary synthesis. This avoids reducing the comparison to a binary opposition ("proposed method" vs. "not proposed method"), focusing instead on observing the contributions that arise from continuous improvements in adaptability and structural expressiveness.

Secondly, the use of three scenarios ensures that the method is evaluated not only in the nominal case, but also under post-migration conditions and under degraded post-migration behaviour. This is important because the aim of the paper is not only to prove the average quality of monitoring, but also to assess whether the synthesized monitoring remains relevant after architectural and operational drift. In this sense, the set of scenarios is sufficient to reveal whether the proposed method is robust to configuration changes and whether it maintains monitoring adequacy beyond the initial Oracle-based setup.

Thirdly, 40 independent runs per strategy-scenario pair provide sufficient replication to reduce the impact of random fluctuations caused by stochastic evolutionary search and stochastic scenario generation. While this experiment volume does not replace full industrial-scale empirical validation, it is scientifically adequate to demonstrate the advantages of the proposed tensor-based monitoring synthesis method and to reveal whether those advantages are systematic rather than accidental.

E. Results of Experiments

An example of a high-quality tensor chromosome obtained in experiments is a configuration in which the PostgreSQL application is monitored for latency by collecting metrics in a normal post-migration scenario, while trace-based latency monitoring is additionally activated in a degradation scenario. At the same time, the database component is monitored for consistency by rule-based checks and for availability by collecting metrics, while observability is covered by log-based monitoring of microservices and agents. The adaptation subtensor allows for sample escalation and trace activation for latency degradation, as well as generating consistency violation notifications. Such a chromosome (Fig. 4) achieves high monitoring relevance because it jointly preserves functional equivalence checks, property traceability, structural consistency, and scenario-specific adaptive behaviour.

In the binary chromosome representation, each gene encodes whether a particular monitoring assignment is selected. A value of 1 denotes that the corresponding component-, relation-, adaptation-, traceability-, or parameter-selection decision is active in the chromosome, whereas a value of 0 denotes that the corresponding option is not selected. The post-migration results, aggregated under two target scenarios: normal post-migration and degraded post-migration, are presented in Table I for conventional metrics and Table II for property-aware metrics.

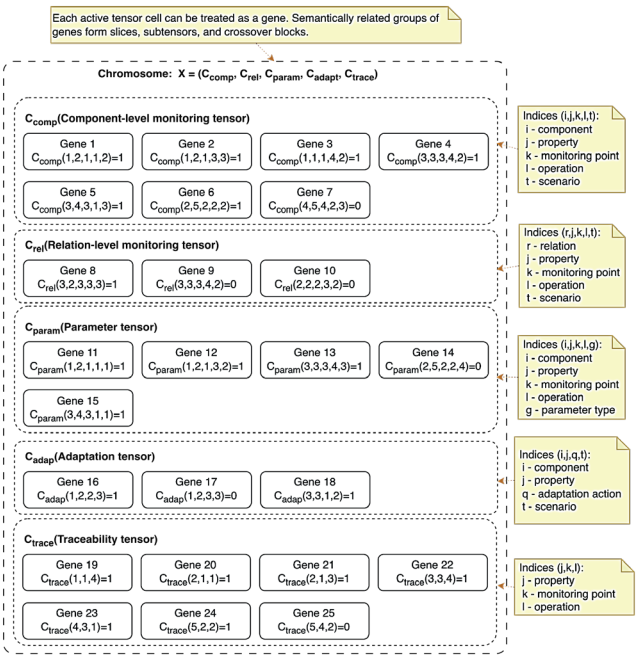

Fig. 5. Example of a high-quality tensor chromosome obtained for the Oracle-to-PostgreSQL monitoring synthesis case.

The results in Tables I and II show a clear gradual improvement from M0 to M4. Static monitoring (M0) exhibits the weakest performance after migration, confirming that the initial monitoring configuration before migration becomes insufficient after the IS experiences configuration drift. Manual reconfiguration (M1) improves all the main metrics, but still remains limited in property tracking, workflow completeness, and reconfiguration success.

TABLE I

POST-MIGRATION SUMMARY OF MONITORING STRATEGIES: CONVENTIONAL METRICS

Strategy	Precision	Recall	F1	*MTTD*	Localization	False alarm
M0	0.700	0.621	0.658	8.421	0.571	0.211
M1	0.754	0.700	0.725	7.171	0.642	0.159
M2	0.796	0.746	0.769	6.675	0.700	0.136
M3	0.854	0.826	0.839	5.610	0.771	0.099
M4	0.886	0.853	0.869	5.200	0.805	0.092

TABLE II

POST-MIGRATION SUMMARY OF MONITORING STRATEGIES: PROPERTY-AWARE METRICS

Strategy	*PCR*	*TPP*	*MRSR*	*ACS*	*TSC*	*OCC*	*TWC*
M0	0.518	0.453	0.091	0.468	0.419	0.446	0.400
M1	0.648	0.596	0.340	0.335	0.557	0.588	0.555
M2	0.711	0.663	0.461	0.284	0.623	0.648	0.628
M3	0.801	0.766	0.619	0.211	0.722	0.750	0.715
M4	0.844	0.815	0.723	0.168	0.812	0.828	0.810

Rule-based adaptation (M2) further improves the quality of diagnostics and coverage, although it still falls short of the quality of evolutionary approaches. The strongest performance is observed for M3 and M4. M4 is achieving the best values for F1, PCR, TPP, MRSR, TSC, OCC, and TWC, and also providing the lowest ACS and shortest MTTD. These results support the main hypothesis of the study that chromosome tensor encoding preserves cross-dimensional dependencies more effectively than flat vector encoding.

The distribution of monitoring quality across scenarios is shown in Fig. 6.

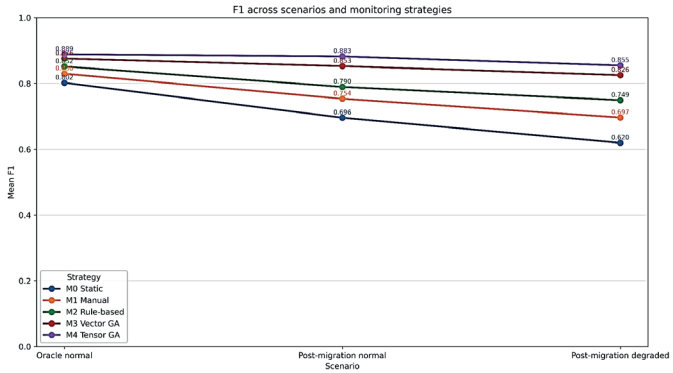

Fig. 6. F1 distribution for all scenarios.

The box plots show that the degradation caused by migration is most severe for M0, while M3 and especially M4 remain much more stable across all scenarios. Since the methodological novelty of the study lies in tensor evolutionary synthesis, the comparison at the optimizer level between the vector genetic algorithm M3 and the tensor genetic algorithm M4 is also important. The corresponding results are presented in Table III. According to Table III, the tensor method improves not only the final monitoring configuration but also the optimisation process itself. The higher quality of the Pareto front and the M4 HV indicate that the tensor search finds stronger trade-off solutions for conflicting objectives. The improvements in CS and CRR show that the results are more reproducible in repeated runs. Finally, the higher TSS and SRR values indicate that semantically meaningful tensor structures are preserved more effectively during crossover and mutation, which is consistent with the design assumptions of the proposed chromosome representation.

TABLE III

OPTIMIZER-QUALITY SUMMARY FOR M3 AND M4

Strategy	***PFQ***	***HV***	***CS***	***CRR***	***TSS***	***SRR***
M3	0.755	0.728	0.799	0.757	0.685	0.649
M4	0.840	0.815	0.861	0.831	0.800	0.765

The scenario-specific and distributional scores of fitness functions are presented Fig. 7, which confirms the same ranking observed in Tables I, II, and III: M4 (Tensor GA) remains the strongest strategy, followed by M3 (Vector GA), while M0 static remains the weakest.

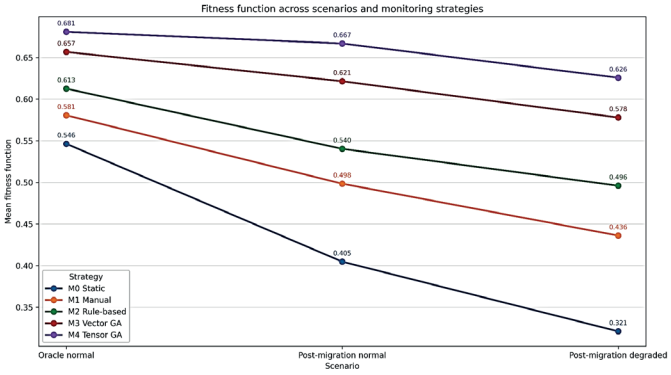

Fig. 7. Distribution of fitness function across repeated runs.

Overall, the experimental results demonstrate that adaptive monitoring synthesis is more effective than static and manually adjusted monitoring under post-migration drift conditions, and that tensor-based evolutionary coding provides the strongest and most structurally consistent monitoring performance.

VI. CONCLUSION

Using Oracle PL/SQL to PostgreSQL migration as a controlled case, the study shows that functional equivalence can be maintained while architectural and operational properties relevant to monitoring change. Based on this finding, a property-driven, multi-objective genetic approach to monitoring synthesis is proposed, which integrates three perspectives: LLM-based IS transformation, self-adaptive monitoring, and architecture-aware optimisation.

The key novelty of the study is the introduction of a tensor chromosome model for the synthesis of adaptive monitoring in ISs modified by LLM. In contrast to traditional vector formulations, the proposed representation jointly encodes components, relations, controlled properties, telemetry operations, adaptation actions, and traceability mapping within a single structured evolutionary object. This allows us to preserve interdimensional dependencies during genetic search and to consider monitoring not as a static configuration artefact, but as an evolving, architecture-dependent workflow.

The key methodological contribution is the tensor interpretation of chromosomes. This takes the research beyond traditional parameter tuning and considers monitoring synthesis as an evolutionary construct of a structured monitoring workflow. By transferring the principles of tensor optimisation from integrative analytical workflows to IM, the paper strengthens the theoretical foundation of the approach and supports scalable monitoring synthesis for complex IMs modified by LLM.

For future work, the study will be extended to larger and more realistic ISs by increasing the number of components, relationships, controlled properties, MPs, telemetry operations, adaptation actions, and execution scenarios. In parallel, future experiments will compare test generation strategies – such as regression tests prepared manually, tests generated by LLM, tests generated by LLM with mutation-based refinement – to study the relationship between test adequacy and monitoring adequacy in LLM-modified ISs.

REFERENCES

[1] A. Fan, Gokkaya, M. Harman, M. Lyubarskiy, S. Sengupta, S. Yoo, and J. M. Zhang, "Large language models for software engineering: Survey and open problems," in *2023 IEEE/ACM International Conference on Software Engineering: Future of Software Engineering (ICSE-FoSE)*, Melbourne, Australia, May 2024, pp. 31–53. https://doi.org/10.1109/ICSE-FoSE59343.2023.00008

[2] V. Terragni, A. Vella, P. Roop, and K. Blincoe, "The future of AI-driven software engineering," *ACM Transactions on Software Engineering and Methodology*, vol. 34, no. 5, May 2025, Art. no. 120. https://doi.org/10.1145/3715003

[3] X. Hou *et al*., "Large language models for software engineering: A systematic literature review," *ACM Transactions on Software Engineering and Methodology*, vol. 33, no. 8, Dec. 2024, Art. no. 220. https://doi.org/10.1145/3695988

[4] M. Schäfer, S. Nadi, A. Eghbali, and F. Tip, "An empirical evaluation of using large language models for automated unit test generation," *IEEE Transactions on Software Engineering*, vol. 51, no. 1, pp. 85–105, Jan. 2024. https://doi.org/10.1109/TSE.2023.3334955

[5] L. Yang *et al*., "On the evaluation of large language models in unit test generation," in *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, Oct. 2024, pp. 1607–1619. https://doi.org/10.1145/3691620.3695529

[6] Y. Chen, Z. Hu, C. Zhi, J. Han, S. Deng, and J. Yin, "ChatUniTest: A framework for LLM-based test generation," in *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*, July 2024, pp. 572–576. https://doi.org/10.1145/3663529.3663801

[7] A. M. Dakhel, A. Nikanjam, V. Majdinasab, F. Khomh, and M. Desmarais, "Effective test generation using pre-trained large language models and mutation testing," *Information and Software Technology*, vol. 171, July 2024, Art. No. 107468. https://doi.org/10.1016/j.infsof.2024.107468

[8] N. Alshahwan "Automated unit test improvement using large language models at Meta," in *Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*, July 2024, pp. 185–196. https://doi.org/10.1145/3663529.3663839

[9] R. de Lemos *al*., "Software engineering for self-adaptive systems: A second research roadmap," in *Software Engineering for Self-Adaptive Systems II. Lecture Notes in Computer Science,* R. de Lemos, H. Giese, H. A. Müller, and M. Shaw, Eds., vol 7475. Springer, Berlin, Heidelberg, 2013, pp. 1–32. https://doi.org/10.1007/978-3-642-35813-5_1

[10] V. E. S. Souza, A. Lapouchnian, and J. Mylopoulos, "Awareness requirements for adaptive systems," in *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, May 2011, pp. 60–69. https://doi.org/10.1145/1988008.1988018

[11] N. M. Villegas, G. Tamura, H. A. Müller, L. Duchien, and R. Casallas, "Architecting software systems for runtime self-adaptation: Concepts, models, and challenges," in *Managing Trade-Offs in Adaptable Software Architectures*. Elsevier, 2017, pp. 17–43. https://doi.org/10.1016/B978-0-12-802855-1.00002-2

[12] E. Henrichs, V. Lesch, M. Strasser, S. Kounev, and C. Krupitzer, "A literature review on optimization techniques for adaptation planning in adaptive systems: State of the art and research directions," *Information and Software Technology*, vol. 149, Sep. 2022, Art. no. 106940. https://doi.org/10.1016/j.infsof.2022.106940

[13] J. Li, M. Zhang, N. Li, D. Weyns, Z. Jin, and K. Tei, "Generative AI for self-adaptive systems: State of the art and research roadmap," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 19, no. 3, Sep. 2024, Art. no. 13. https://doi.org/10.1145/3686803

[14] M. Waseem, P. Liang, M. Shahin, A. Di Salle, and G. Márquez, "Design, monitoring, and testing of microservices systems: The practitioners' perspective," *Journal of Systems and Software*, vol. 182, Dec. 2021, Art. no. 111061. https://doi.org/10.1016/j.jss.2021.111061

[15] P. Di Francesco, P. Lago, and I. Malavolta, "Architecting with microservices: A systematic mapping study," *Journal of Systems and Software*, vol. 150, pp. 77–97, Apr. 2019. https://doi.org/10.1016/j.jss.2019.01.001

[16] M. Waseem, P. Liang, and M. Shahin, "A systematic mapping study on microservices architecture in DevOps," *Journal of Systems and Software*, vol. 170, Dec. 2020, Art. no. 110798. https://doi.org/10.1016/j.jss.2020.110798

[17] O. Grynets, V. Lyashkevych, D. Baran, M. Orliansky, T. Zelenyy, and M. Leshchyshyn, "Fine-tuned LLM-based code migration framework," *arXiv:2512.13515*, Dec. 2025. https://arxiv.org/abs/2512.13515

[18] A. Aleti, B. Buhnova, L. Grunske, A. Koziolek, and I. Meedeniya, "Software architecture optimization methods: A systematic literature review," *IEEE Transactions on Software Engineering*, vol. 39, no. 5, pp. 658–683, May 2013. https://doi.org/10.1109/TSE.2012.64

[19] Q. Li, M. Lu, T. Gu, and Y. Wu, "Runtime software architecture-based reliability prediction for self-adaptive systems," *Symmetry*, vol. 14, no. 3, Mar. 2022, Art. no. 589. https://doi.org/10.3390/sym14030589

[20] V. Y. Lyashkevych, "Sustainable optimization of consolidated data processing algorithms based on machine learning and genetic algorithms," *Electronics and Information Technologies*, vol. 32, pp. 39–54, 2025. https://doi.org/10.30970/eli.32.3

[21] M. Y. Lyashkevych, V. Y. Lyashkevych, and R. Y. Shuvar, "Definition and formalization of the software functional state concept throughout the development life cycle," *Electronics and Information Technologies*, vol. 32, pp. 151–170, 2025. https://doi.org/10.30970/eli.32.11

[22] S. S. da Silva, P, Pelliccione, and A. Bertolino, "Self-adaptive testing in the field," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 19, no. 1, Feb. 2024, Art. no. 4. https://doi.org/10.1145/3627163

[23] E. M. Fredericks, B. DeVries, and B. H. C. Cheng, "Towards run-time adaptation of test cases for self-adaptive systems in the face of uncertainty," in *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, June 2014, pp. 17–26. https://doi.org/10.1145/2593929.2593937

[24] C. E. da Silva and R. de Lemos, "Dynamic plans for integration testing of self-adaptive software systems," in *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, May 2011, pp. 148–157. https://doi.org/10.1145/1988008.1988029

[25] O. Grynets and V. Lyashkevych, "Unified architecture metamodel of information systems developed by generative AI," *arXiv:2604.00171*, Mar. 2026. https://doi.org/10.48550/arXiv.2604.00171

Vasyl Lyashkevych received the Candidate of Science degree in 2007, and the PhD degree in Computer Science in 2008. He is an Assistant Professor of the System Design Department, Ivan Franko National University of Lviv, Lviv, Ukraine. He also worked at Accenture (Riga, Latvia), GlobalLogic (Lviv, Ukraine), and EPAM Systems (Lviv, Ukraine) in R&D, AI architecture, innovation, and data analytics roles. His research interests include artificial intelligence, genetic algorithms, knowledge management, intelligent monitoring, multi-agent systems, and software engineering technologies.
E-mail: vasyl.liashkevych@lnu.edu.ua
ORCID iD: https://orcid.org/0000-0003-2810-6061