UDC 004.89

EVALUATION AND QUALITY ASSURANCE OF MIGRATED ABAP CODE USING AN INTEGRAL METRIC AND GENERATIVE ARTIFICIAL INTELLIGENCE MODELS

Pozdnyakov O. A. – Post-graduate student of the Software Tools Department, National University "Zaporizhzhia Polytechnic", Zaporizhzhia, Ukraine. ROR: https://ror.org/03aph1990. ORCID: https://orcid.org/0009-0006-3955-802X.

Parkhomenko A. V. – PhD, Associate Professor of the Software Tools Department, National University "Zaporizhzhia Polytechnic", Zaporizhzhia, Ukraine. ROR: https://ror.org/03aph1990. ORCID: https://orcid.org/0000-0002-6008-1610.

ABSTRACT

Context. Migration automation of legacy custom code when transitioning to the new version of the SAP S/4HANA system using large language models (LLMs) is a promising option. However, the generated code quality assessment remains an unresolved issue, since existing approaches utilize fragmented metrics which do not allow for a comprehensive software code quality assessment and assurance for further use without additional revision.

Objective. The objective of this work is to improve the efficiency of the process of intelligent reengineering of a computer system based on the method of comprehensive assessment and quality assurance of migrated ABAP custom code.

Method. The developed method is based on two key components. The Integral ABAP Quality Score (IAQS) comprehensively takes into account the syntactic, functional, and semantic characteristics of the code and is based on the provisions of the international software quality standards ISO/IEC 25010, ISO/IEC 25040, as well as the theory of composite indicators. The three-stage approach to LLM fine-tuning (Qwen 2.5 Coder 14B) includes continuous pre-training (CPT), parameter-efficient fine-tuning (PEFT), and alignment based on preferences using the ORPO algorithm. At the same time, the use of the developed IAQS metric to form a set of preference data at the alignment stage creates a mechanism for controlled improvement, namely, it determines the direction of LLM adaptation.

Results. The results of experimental studies demonstrate that the implementation of the developed method allows improving both individual indicators of software code quality and the integral metric of IAQS quality assessment as a whole. The final model, trained on the basis of the proposed three-stage approach, achieved a high IAQS value (0.756), which demonstrates a significant improvement compared to the baseline model (0.117).

Conclusions. The study presents a new problem-oriented approach to automated migration of ABAP code during intelligent reengineering of computer systems. The proposed IAQS integral metric is the basis for creating a formalized and objective system for evaluating the quality of software generated by LLM in the context of legacy custom code migrating. It has been demonstrated that consistent fine-tuning of LLM based on a three-stage approach using IAQS provides a significant improvement in the generated software code integral quality indicator.

KEYWORDS: software quality, integral metrics, large language models, migration of legacy custom code, LLM fine-tuning.

ABBREVIATIONS

ABAP is an advanced business application programming;
AST is an abstract syntax tree;
ATC is an ABAP test cockpit;
CPT is a continued pre-training;
ERP is an enterprise resource planning;
IAQS is an integral ABAP quality score;
ISO is the International organization for standardization;
IEC is the International electro technical commission;
LLM is a large language model;
LoRA is a low-rank adaptation;
ORPO is an Odds ratio preference optimization;
PEFT is a parameter-efficient fine-tuning;
QLoRA is a quantized low-rank adaptation;
SFT is a supervised fine-tuning.

NOMENCLATURE

C_{old} is a code fragment in the outdated version of ABAP;
C_{new} is a code fragment in the modern version of ABAP;
C_{newi} is an i-th candidate migration option;
$F_{quality}$ is an integral quality function;
IAQS_i is an integral assessment value for the i-th code variant;
k is a number of candidate migration variants;
L_{ORPO} is an ORPO loss function;
L_{SFT} is a supervised fine-tuning loss function;
L_{OR} is an odds ratio-based loss function;
M_{func} is a functional correctness metric (pass@1);
$M_{LLM}(\theta)$ is a large language model with parameters θ;
M_{sem} is a semantic-structural similarity metric;
M_{syn} is a syntactic correctness metric;
Q is a quality metrics vector;
q_{func} is a functional correctness;
q_{sem} is a semantic-structural similarity;
q_{syn} is a syntactic correctness;
r is a rank of adaptation matrices in LoRA;
w_{syn}, w_{func}, w_{sem} are the weights of IAQS components;

DOI 10.15588/1607-3274-2026-1-8

y_w is a selected (better) answer in a pair of preferences;

y_l is a rejected (worse) answer in a pair of preferences;

α is a scaling coefficient in LoRA;

θ are the model parameters;

θ^* are the model optimal parameters;

λ is a weight factor (ORPO balance hyper-parameter);

$p(y|x)$ is a probability of y reply generation for x input data;

$\sigma(.)$ is a sigmoid function.

INTRODUCTION

The transition to the modern SAP S/4HANA software platform is currently a strategic initiative for many organisations both in Ukraine and around the world. SAP S/4HANA is based on the SAP HANA in-memory database, which provides significant performance gains but also requires the computer system's software code to be reviewed and adapted in order to fully utilise its capabilities [1].

One of the key challenges along this way is the legacy custom code in the ABAP language, developed over decades of use SAP ERP versions by companies and enterprises [2]. Migrating of custom code is one of the main challenges when transitioning to SAP S/4HANA, accounting for up to 40% of the total project work scope [3]. Manual correction of legacy custom code is not only labour-intensive but also a risky process that can lead to project delays and errors in the production system [1].

Despite the availability of static analysis tools by SAP (ABAP Test Cockpit, Custom Code Migration App) [2] which help identify problematic code fragments, the code transformation process remains largely manual. Automation is limited to simple template replacements, while complex migration scenarios which require an understanding of business logic and context still require significant effort from skilled ABAP developers. This challenge requires the application of modern intelligent reengineering approaches to fully implement the potential of the SAP S/4HANA platform.

In [4], the authors analysed existing problems and developed a methodology for migrating of ABAP custom code based on intelligent reengineering methods and models. A key feature was the justification of the need to use locally deployed open-source Large Language Models (LLMs), which makes it possible to avoid both the significant data security risks and licence agreement violations inherent in cloud-based LLMs.

The formalised method for selecting the optimal LLM [5] developed by the authors is based on a hybrid AHP-TOPSIS approach and allows the identification of leading candidates (in particular, the Qwen, DeepSeek and Llama models) for solving of the ABAP code migration problem.

Although LLMs have already demonstrated significant potential for generating software code, the problem of objective comprehensive assessment and quality assurance of the generated code remains. Therefore, the research topic is relevant.

The object of study is the process of intelligent reengineering of a computer system when migrating to a new version of SAP S/4HANA.

The subject of study is methods for evaluating and ensuring the quality of migrated ABAP code inherited from outdated versions of SAP ERP.

The purpose of the work is to improve the efficiency of the process of intelligent reengineering of a computer system based on the method of comprehensive assessment and quality assurance of migrated ABAP custom code.

To achieve this goal, the following tasks must be solved:

– formally define and justify the IAQS integral metric based on the international software quality standards ISO/IEC 25010 [6], ISO/IEC 25040 [7];

– develop a three-stage LLM fine-tuning methodology (continuous pre-training, parameter-efficient fine-tuning, advantage-based alignment), where IAQS is used as an integral quality assessment criterion for forming data on advantages;

– experimentally verify the effectiveness of the proposed approach on real ABAP code migration tasks;

– conduct a statistical analysis of the results and investigate the impact of each fine-tuning stage on the components of the integral metric.

1 PROBLEM STATEMENT

The research task is formalised as follows.

Let C_{old} be a code fragment in an outdated version of the ABAP language. $M_{LLM}(\theta)$ is a large language model with parameters θ, which performs code conversion: $C_{new}=M_{LLM}(\theta, C_{old})$.

The quality of the generated code fragment C_{new} is evaluated using a vector Q with n metrics that reflect various aspects of quality and is defined as:

$$Q=[q_{syn}, q_{func}, q_{sem}]. \tag{1}$$

The problem is that evaluating the quality of program code using individual metrics is ineffective when migrating to new versions of computer systems.

Therefore, an integral quality function $F_{quality}$ is introduced, which maps the vector of metrics Q to a single scalar value representing the overall quality of the code. According to the ISO/IEC 25040 [7] software quality assessment methodology and the theory of composite indicators [8], such a function should aggregate a set of quality metrics into a single numerical value. For the specific task of ABAP code migration, this function can be formally defined as an integral metric IAQS for a comprehensive assessment of the quality of the ABAP code generated during the migration process. In this case, its range of values is defined as a normalised interval:

$$F_{quality}: Q \rightarrow [0,1]. \tag{2}$$

© Pozdnyakov O.A., Parkhomenko A. V., 2026
DOI 10.15588/1607-3274-2026-1-8

It is necessary to define and justify the function $F_{quality}$ (IAQS) and develop a methodology for adjusting the parameters θ of the model M_{LLM} in order to find the optimal set of parameters θ^* that maximises the expected value of the value $F_{quality}$ on the distribution of fragments of the legacy custom code:

$$\theta^* = \arg\max_{\theta} E\left[F_{quality}\left(Q\left(M_{LLM}\left(\theta, C_{old}\right)\right)\right)\right]. \qquad (3)$$

Thus, this formulation defines:

– the object of optimisation are the parameters of the model θ^*;

– success criterion is a maximisation of the integral quality function $F_{quality}$;

– constraint – the quality function must be justified in accordance with international standards (ISO/IEC 25010) [6] and reflect the syntactic, functional and semantic aspects of the code.

2 REVIEW OF THE LITERATURE

The concept of software quality is fundamental to building reliable and effective computer information systems.

In [9], it is proven that in critical industries (which undoubtedly include enterprise-scale ERP systems), the concept of quality goes beyond purely technical characteristics and acquires economic significance.

Reliability and maintainability are considered not just as desirable properties, but as economic categories which directly affect the cost of ownership of the system and business risks [9].

The author emphasises the need to apply the philosophy of "Cleanroom Software Engineering", which shifts the focus from correcting errors post factum to preventing them at the design and development stage [9].

This thesis is critically important for automated migration tasks, where the cost of an error replicated across thousands of lines of code can be catastrophic.

Standardised models have been developed to formalise these requirements, the most widely recognised of which is the ISO/IEC 25010 (SQuaRE) quality model [6].

It defines a hierarchical structure of eight product characteristics (e.g., Functional Suitability, Reliability, Compatibility, Maintainability, etc.) and their sub characteristics. This standard provides a theoretical basis for selecting and justifying software quality metrics.

Modern approaches to the quantitative measurement of software code quality indicators are analysed in detail in [10]. The effectiveness of using machine learning methods, in particular the Random Forest algorithm, for aggregating classical metrics (such as Holsted metrics, McCabe cyclomatic complexity) and code quality predicting has been proven. It has been shown that quality is a measurable entity that can be objectively assessed. The author also emphasises the importance of comprehensive analysis of code quality based on the selection and justification of a system of weight factors which reflect the relative importance of each metric [10]. However, the approaches proposed by the author allow for the effective identification of problematic code, but do not provide tools for its improvement.

Researches in the field of software engineering confirm that a combination of metrics covering different aspects of quality provides a more accurate and complete assessment of quality than any single metric. This approach allows for a more comprehensive representation of code quality, which is especially important for complex tasks such as automated migration.

While ISO/IEC 25010 defines the quality metrics that need to be measured [6], the theory of integral indicators explains how to combine individual metrics into a single indicator [7, 8].

An integral metric is a function that aggregates several individual indicators into a single numerical value, most often using a weighted arithmetic mean value of normalised components [7].

In recent years, LLMs [4] have demonstrated significant capabilities in solving a wide range of tasks related to software code. Models specifically trained on code (Code LLMs), such as the Qwen Coder series [11], Code Llama [12], and others, achieve state-of-the-art results on common coding benchmarks such as HumanEval and MBPP. These models, based on transformer architecture, are pre-trained on trillions of tokens of code and text from publicly available sources, giving them a deep fundamental understanding of programming syntax, semantics, and logic.

However, the effectiveness of universal LLMs is significantly reduced when working with proprietary languages such as ABAP due to a lack of representation in the training data [13].

To overcome these limitations, fine-tuning methods are used. Continuous pre-training (CPT) adapts the model to a new domain by continuing self-supervised learning on a large corpus of unlabelled data [14].

After adapting to the domain, the model must be trained to perform a specific task, for which parameter-efficient methods (PEFT) have been developed [15].

The most popular PEFT method is LoRA (Low-Rank Adaptation) [16], which freezes the main weights of the model and trains only small adapter matrices.

A further development is the QLoRA (Quantized LoRA) method [17], which applies 4-bit quantisation to frozen weights.

Alignment methods are used to improve the model output. The latest approach is Odds Ratio Preference Optimisation (ORPO) [18], which combines supervised fine-tuning (SFT) and preference-based alignment into a single step.

Therefore, there is a pressing need to create an integral quality assessment metric that combines diagnostic accuracy (based on metrics [10]) with compliance to a regulatory framework of standards, while being adapted to manage the quality improvement process when using LLM for migrating legacy ABAP custom code.

DOI 10.15588/1607-3274-2026-1-8

3 MATERIALS AND METHODS

Based on the research conducted and the principles of software quality assessment described in ISO/IEC 25040 [7], a domain-specific integral quality assessment metric IAQS was developed, which formally combines three key code quality metrics into a single indicator for the ABAP code migration task.

The components of IAQS are justified within the framework of ISO/IEC 25010 [6] and presented in Table 1.

Table 1 – Compliance of IAQS components with the ISO/IEC 25010 standard

IAQS component	ISO 25010 characteristic	Sub-characteristic	Justification
M_{syn}	Functional suitability, maintainability	Functional correctness, analysability	Syntactically incorrect code cannot be executed or analysed
M_{func}	Functional suitability	Functional correctness	A direct measure of whether the code performs the task at hand
M_{sem}	Maintainability	Modifiability, reusability	Code that is structurally close to the standard is easier to analyse and modify

Unlike general frameworks of quality assessment, IAQS is an integral metric specifically designed for comprehensive assessment of ABAP code quality, which simultaneously takes into account the syntactic, functional, and semantic aspects of the generated code.

Syntactic correctness (M_{syn}). The percentage of generated fragments that pass static analysis without errors. This metric is directly related to the characteristics of Maintainability (sub-characteristic Analysability) and Functional Suitability (sub-characteristic Functional Correctness), since syntactically incorrect code cannot be executed.

Functional correctness (M_{func}). The pass@1 metric is based on dynamic code execution and is defined as the percentage of cases where the first generated response is successfully compiled followed by successful passing of all functional tests [12] in the target environment. It corresponds to the Functional Suitability characteristic and its key sub-characteristic, Functional Correctness. This is the most direct measurement of whether the code performs the task at hand.

Semantic-structural similarity (M_{sem}). The CodeBLEU metric [19] is used – a composite indicator developed to evaluate both lexical and structural code similarity by taking into account syntactic structures and keywords of the programming language. This metric correlates with Maintainability, since code that is structurally and semantically close to a quality benchmark is easier to understand, analyse, and modify (sub-characteristics Modifiability and Reusability).

IAQS is defined as the weighted arithmetic mean value of its normalised components [7]:

$$IAQS = w_{syn} \cdot M_{syn} + w_{func} \cdot M_{func} + w_{sem} \cdot M_{sem}, \quad (4)$$

where $w_{syn}+w_{func}+w_{sem}=1$ and all weights $w_i \geq 0$.

For this study, an equal weighting approach ($w_{syn}=w_{func}=w_{sem}=1/3$) was adopted, which is a common and reasonable starting point in the absence of prior data on priorities [10]. Equal weighting reflects the assumption that all three aspects of quality are equally important for the overall evaluation of the generated code.

The proposed approach to LLM fine-tuning is a sequential process aiming at improving of the integral IAQS assessment.

Stage 1. Continuous pre-training (CPT). The model adapts to the syntax and semantics of ABAP by continuing self-supervised learning on a large corpus of unlabelled data [4]. This stage improves the model's internal representations, making it more "familiar" with the idioms of the target language.

Stage 2. Parameter-efficient fine-tuning (PEFT with QLoRA). The model learns a specific task of translating obsolete ABAP constructs into modern equivalents using supervised learning on pairs (instruction response) [15,16]. QLoRA allows the model to be trained efficiently by updating only a small portion of the parameters [17]. Instead of fine-tuning all model parameters, LoRA [16] freezes the initial weights and trains only small low-rank adapter matrices A and B. Weight updates are approximated by their product: $\Delta W=BA$. A direct pass through the modified layer is described by the formula:

$$h = W_0x + BAx, \quad (5)$$

where $x \in R^k$ is the input vector, $W_0 \in R^{(d \times k)}$ is the output weight matrix, $B \in R^{(d \times r)}$, $A \in R^{(r \times k)}$, and rank $r \ll \min(d,k)$ is a hyperparameter.

Stage 3. Alignment based on preferences (Alignment with ORPO). The final stage uses the ORPO method to improve the quality of model output based on preference pairs [18]. Its key advantage is that it combines supervised fine-tuning and preference alignment into a single monolithic step, eliminating the need for a reference model π_{ref}, which simplifies the training process.

Direct optimisation of formula (3) via gradient descent is technically challenging because $F_{quality}$ includes non-differentiable components (syntactic correctness M_{syn}, functional correctness M_{func} (pass@1)). Therefore, we propose to implement an indirect optimisation strategy through three-stage fine-tuning, where IAQS is used as a control signal at the alignment stage. The metric ranks candidate code versions, forming preference data for the ORPO algorithm, which in its turn optimises the differentiable loss function L_{ORPO}. This creates an indirect link between $F_{quality}$ and the learning process.

The ORPO loss function consists of two components: a standard loss of negative log-likelihood (NLL loss) for the desired response y_w and a term based on the odds

DOI 10.15588/1607-3274-2026-1-8

ratio, which penalises the undesired response y_l and is defined as:

$$L_{ORPO} = E_{(x,y_w,y_l)\sim D}\left[-\log P_\theta(y_w \mid x) - \lambda \log\sigma\left(\log\frac{odds_\theta(y_w \mid x)}{odds_\theta(y_l \mid x)}\right)\right], \quad (6)$$

where $odds_\theta(y \mid x) = \dfrac{1 - P_\theta(y \mid x)}{P_\theta(y \mid x)}$.

The first term $-\log P_\theta(y_w \mid x)$ is responsible for training the correct response (as in SFT), and the second is responsible for ensuring that the model "distinguishes" a good response from a bad one.

The final stage of alignment with ORPO is key to the proposed approach. Unlike standard approaches, where preference data is collected manually, IAQS is used instead as a preference control signal. This creates a powerful, self-consistent cycle, as the proposed integral metric directly controls the training of the model.

For the input code C_{old} several k=5 candidate migration options are generated using the model after the PEFT stage with sampling temperature variation.

For each candidate C_{newi} its score IAQS$_i$ is calculated according to formula (4).

Pairs of preferences (C_{newi},C_{newj}) are formed such that $IAQS_i$>$IAQS_j$. In this case, C_{newi} becomes the "selected" response y_w, and C_{newj} – becomes the "rejected" response y_l.

This data set is then used to fine-tune the model using the ORPO loss function [18], which now implicitly aligns (or corrects) the model policy by minimising L_{ORPO} to achieve a high IAQS value.

4 EXPERIMENTS

A specialised code generation model, Qwen 2.5 Coder 14B [11], was selected as the base LLM. This model was chosen due to its advanced results on coding benchmarks and its open licence, which is consistent with the results of [5].

The experiments were conducted on an NVIDIA A100 GPU (40 GB) using PyTorch 2.1, Hugging Face Transformers 4.36, and TRL (Transformer Reinforcement Learning) 0.7 libraries to implement ORPO.

Three datasets were prepared for fine-tuning.

The corpus for CPT is an unstructured corpus of modern ABAP code with a volume of 500 million tokens, collected from publicly available GitHub repositories (150+ repositories), SAP Press documentation with code examples, and anonymised fragments from open SAP Community archives.

Parallel corpus for PEFT – a set of 50,000 "instruction-response" pairs in the format "legacy code – modern equivalent". The data was prepared by experts who included code for migration to SAP S/4HANA and supplemented with synthetically generated examples using migration rules from official SAP documentation.

A set of advantage data for Alignment – 8,000 triplets (prompt-chosen-rejected), where "chosen" is the option with high IAQS, and "rejected" is the option with lower IAQS. The data was generated automatically using the above-described procedure for forming preference pairs.

For independent evaluation of the models, a test sample was prepared containing 1,000 pairs (obsolete code – reference_modern_code) that were not used during any stage of training. The test examples cover typical migration tasks: improving the efficiency of SELECT queries, replacing outdated functional modules with modern classes, refactoring of outdated constructs.

The IAQS component calculation methodology involves the following steps:

– syntax checking using ABAP Parser (a component of ABAP Development Tools). The M_{syn} metric is calculated as the percentage of fragments without syntax errors;

– validation of functional equivalence by dynamically executing code in the SAP S/4HANA environment. Sets of unit tests were prepared for each pair (old code – new code). Thanks to access to the SAP S/4HANA system, the verification was performed not on mock-ups, but on a real ABAP processor. The metric (pass@1) is calculated as the percentage of generated fragments which are compiled successfully and successfully passed all functional tests on the first attempt [12];

– calculation *of* M_{sem} (CodeBLEU) relative to the reference code using the codebleu library [19, 20]. Given the specifics of the proprietary ABAP language and the lack of publicly available stable grammars (Tree-sitter grammars) for building AST in Python, the calculation was performed in an adapted mode. N-gram match and weighted N-gram match components (with customised weights for ABAP keywords) were used, which made it possible to evaluate the lexical and structural correspondence of the code without the need to export syntax trees from SAP systems.

Each model was evaluated on a full test sample with a fixed seed (random seed=42) to ensure reproducibility of results. The hyperparameters for fine-tuning are presented in Table 2.

Table 2 – Hyperparameters for fine-tuning

Parameter	CPT stage	PEFT stage (QLoRA)	Alignment stage (ORPO)
Learning Rate	1e–5	2e–4	5e–6
Batch Size	8	16	8
Gradient Accumulation Steps	4	2	4
Epochs	1	3	2
LoRA Rank (r)	–	16	16
LoRA Alpha	–	32	32
LoRA Dropout	–	0.05	0.05
Quantization Type	–	4-bit NF4	4-bit NF4
Optimizer	AdamW	AdamW	AdamW
Weight Decay	0.01	0.01	0.01
Max Sequence Length	2048	2048	2048
Warmup Steps	500	100	50
ORPO λ	–	–	0.1

DOI 10.15588/1607-3274-2026-1-8

For QLoRA, 4-bit NF4 (Normal Float 4) quantisation with double quantisation was used, which reduced the model's memory footprint.

5 RESULTS

To evaluate the effectiveness of the proposed approach, not only individual metrics were calculated, but also an integral IAQS assessment at each stage of fine-tuning. The results are presented in Table 3.

Table 3 – Results of model evaluation on the test sample

Model	Syntactic correctness (M_{syn}), %	Semantic similarity (M_{sem})	Functional correctness (M_{func}), %	IAQS (w=1/3)
Base Model (zero-shot)	12.4	0.215	3.1	0.117
After CPT	28.7	0.365	11.2	0.255
After PEFT (QLoRA)	78.3	0.724	64.8	0.718
After Alignment (ORPO)	82.1	0.761	68.5	0.756

Figure 1 shows the visualisation of the dynamics of improvement in the IAQS integral assessment and its components at each stage of fine-tuning.

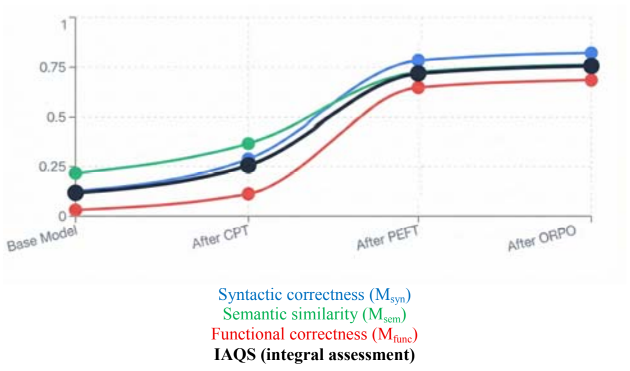

Figure 1 – Dynamics of IAQS growth and its components at the stages of fine-tuning

The graph shows the non-linear nature of quality improvement.

The base LLM shows very low results across all metrics, which is explained by the practical absence of ABAP code in its baseline training data.

The CPT stage provides basic adaptation to the domain, increasing semantic similarity (M_{sem}) from 0.215 to 0.365 (+69.8%).

The largest increase is observed at the PEFT stage, from 0.255 to 0.718 (+181.6%), confirming the critical importance of supervised fine-tuning for achieving functional correctness. The ORPO stage improves (+5.3%) all components simultaneously.

Analysis of the results shows that each stage contributes to a significant and consistent increase in the overall integral quality assessment of ABAP code.

The baseline model demonstrates very low code quality (IAQS = 0.117), which is explained by the lack of ABAP representation in its baseline training data.
The CPT stage increases it more than twice (IAQS = 0.255, an increase of +117.9%).

The PEFT stage provides the largest increase, after which the integral quality assessment metric rises to 0.718 (an increase of +181.6%).

The final alignment stage, controlled by IAQS, brings the integral assessment to 0.756, which is a 546% improvement over the baseline model.

6 DISCUSSION

The results confirm the central thesis of the work: formalising quality assessment in the form of an integral metric and using it as a target function allows for systematic improvement of the results of the ABAP code automated migration process.

A detailed analysis shows that different stages of fine-tuning have different effects on IAQS components.

The continuous pre-training (CPT) stage mainly improves the semantic component M_{sem} (from 0.215 to 0.365, an increase of +69.8%), familiarising the model with idioms and constructs of the ABAP language. This is because CPT adapts the tokenizer and internal representations of the model to the specific syntax of ABAP, making the generated code more "natural" in terms of structure. However, functional correctness increases insignificantly (from 3.1% to 11.2%) because the model has not yet learned the specific task of migration.

The parameter-efficient fine-tuning (PEFT) stage provides the biggest leap in functional correctness M_{func} (from 11.2% to 64.8%, an increase of +478.6%), as at this stage the model learns the specific task of converting obsolete constructs into modern equivalents based on 50,000 examples of pairs (legacy code, modern code). This stage also significantly improves syntactic correctness M_{syn} (from 28.7% to 78.3%, an increase of +172.8%), indicating successful assimilation of the rules for generating valid code. Semantic similarity also increases (from 0.365 to 0.724, +98.4%) as the model learns to generate code that is structurally similar to the reference samples.

The final stage of preference-based alignment (ORPO) provides "fine-tuning" of all components simultaneously. Although the absolute increase is smaller (+5.3% for IAQS, +4.9% for M_{syn}, +5.1% for M_{sem}, +5.7% for M_{func}, this stage is critical for achieving high quality, as it trains the model to distinguish between "good" and "acceptable" code versions. The improvement in semantic-structural similarity (CodeBLEU from 0.724 to 0.761) is particularly noticeable, confirming that the model has learned to generate code that is not only functionally correct but also structurally close to idiomatic ABAP.

It is important to note the synergistic effect between the stages: each subsequent stage builds on the results of the previous one, creating a cumulative improvement in

DOI 10.15588/1607-3274-2026-1-8

quality. CPT lays the foundation for domain knowledge, PEFT trains for a specific task, and ORPO performs targeted model alignment to achieve the maximum value of the IAQS comprehensive quality indicator.

It is necessary to note the limitations of the proposed approach and ways to overcome them.

Firstly, the choice of weights for IAQS is subjective [10]. Although equal weighting (w=1/3) is a reasonable starting point [10], in industrial projects these weights can be calibrated to reflect specific business priorities. For example, in critical financial systems, functional correctness may have a higher weight (w_{func}=0.5, w_{syn}=0.3, w_{sem}=0.2), while in projects with an emphasis on maintainability, semantic similarity (w_{sem}=0.5, w_{func}= 0.3, w_{syn}=0.2). Future research could use data-driven methods (e.g., regression analysis or analytical hierarchy process) to determine optimal weights based on historical migration project data.

Secondly, benchmark-based metrics such as CodeBLEU may be prone to "surface bias", favouring textual similarity over true functional equivalence [20]. Research [21] has shown that CodeBLEU can give high scores to code that looks similar to the benchmark but has subtle semantic differences. This risk is deliberately mitigated in IAQS by including the pass@1 functional correctness metric, which evaluates the actual behaviour of the code through the execution of unit tests [12]. This combination makes IAQS more robust to evaluation errors than any single metric, as the code must simultaneously pass functional tests (pass@1) and be structurally similar to a high-quality benchmark (CodeBLEU).

Thirdly, the scalability of the approach to very large code bases (>100,000 lines) still needs empirical confirmation. The current study focused on the migration of individual functions and methods (average length 50 lines), whereas industrial projects often require the migration of inter-module dependencies, global variables, and complex integration scenarios. Future work should investigate how IAQS scales up to the level of entire applications and whether additional metric components are needed (e.g., inter-module compatibility assessment).

Fourthly, the current implementation of IAQS does not take into account such important aspects of code quality as performance, security, and energy efficiency. Although the ISO/IEC 25010 [6] standard includes the characteristic "Performance Efficiency", its integration into IAQS requires the development of automated methods for measuring the performance of generated ABAP code, which is a non-trivial task due to the need to execute the code in a real SAP environment.

The proposed approach paves the way for the development of more reliable and predictable automated code migration systems. The IAQS integral metric can be integrated into SAP projects as an objective indicator of code quality during migration, allowing teams to track progress and automatically identify problematic code fragments. The proposed fine-tuning methodology can be used as a template for adapting LLM to other domain-specific languages and code migration tasks.

The improvement in economic performance is expected to be significant. If automated migration with IAQS 0.756 can replace even 50% of manual work, this potentially reduces the total cost of an SAP S/4HANA migration project by 20–30%, considering that code migration accounts for up to 40% of the total workload [3].

CONCLUSIONS

This paper presented an approach to the automated migration of legacy ABAP custom code based on the principles of intelligent reengineering and objective code quality assessment. An integral quality assessment metric, IAQS, was proposed, which provides a comprehensive view of the quality of the generated code by combining syntactic, functional, and semantic characteristics based on the ISO/IEC 25010 and ISO/IEC 25040 standards.

A three-stage approach to LLM training (CPT → PEFT → ORPO) was also developed and tested, which purposefully aligns the model to achieve the maximum value of the integral quality assessment metric. A key feature of the methodology is the use of the IAQS metric itself to automatically generate preference data at the alignment stage, creating a self-consistent tuning cycle. The research results confirm that this approach allows for significant and controlled improvement in the quality of migrated code. The final model achieved a high IAQS score (0.756), which is a 546% improvement over the baseline LLM.

The scientific novelty lies in the development of the method for ensuring the quality of legacy ABAP custom code, based on the IAQS code quality assessment metric and a three-stage approach to LLM fine-tuning. Unlike existing approaches that use disjointed code quality assessment metrics, IAQS is an integral metric specifically designed for comprehensive assessment of ABAP code migration quality based on ISO/IEC 25040 principles and composite indicator theory.

A distinctive feature of the proposed three-stage approach to LLM alignment is the use of the IAQS integral metric in the automatic generation of preference data for the ORPO algorithm. This creates a self-consistent cycle of target alignment "metric-model", where the IAQS integral metric directly controls the learning process, ensuring target alignment with respect to the integral quality indicator.

The **practical significance** is that the proposed approach paves the way for the creation of more reliable and predictable automated code migration systems, which will reduce costs in computer system intelligent reengineering projects. The IAQS integral metric can be implemented in SAP projects as an objective indicator of the effectiveness of the migration process of legacy custom code.

Prospects for further research include studying various weighting schemes for IAQS based on expert assessments and real project data, expanding metrics to account for performance, security, and energy efficiency aspects, and comparative testing of the proposed approach with existing methods in real SAP S/4HANA migration projects. A separate promising area of research is the deepening of semantic similarity metrics. We plan to develop a mechanism for integration with open static analysis tools such as abaplint or internal SAP parsers. This will allow the generation and export of complete abstract syntax trees (AST) in JSON format directly to the evaluation pipeline, ensuring that deep structural relationships and data flow are taken into account in the CodeBLEU metric.

ACKNOWLEDGEMENTS

This work was performed as part of the research project "Information technologies of intelligent computing" (state registration number 0124U004188) of the Software Tools Department of Zaporizhzhia Polytechnic National University.

DECLARATIONS

Conflict of interest: The authors declare that they have no conflict of interest in relation to this research, whether financial, personal, authorship, or otherwise, that could affect the research and its results presented in this paper.

Authors contributions: Oleg Pozdnyakov: data curation, formal analysis, investigation, methodology; validation, visualization, writing – original draft. Anzhelika Parkhomenko: conceptualization, supervision, writing – review & editing.

Data availability: The data will be provided upon reasonable request to the authors by email to oleg.pozdnyakov@zp.edu.ua.

Software availability: The manuscript has no associated software.

Use of artificial intelligence tools: The authors used the Qwen 2.5 Coder 14B model in the "Experiments" section to generate ABAP code. The model was sequentially fine-tuned in three stages, and the results of each stage were controlled based on the calculation of component metrics and the integral IAQS metric. Comparative analysis confirmed the improvement of the integral quality indicator of the generated ABAP code. Additionally, during the preparation of this paper, the authors used DeepL in order to: grammar and spelling check, paraphrase and reword. After using this service, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

REFERENCES

1. Hardy P. Migrating custom code to SAP S/4HANA. Boston, Rheinwerk Publishing Inc., 2020, 333 p.
2. Ktern AI. SAP S/4HANA 2022: The ultimate custom code migration guide [Electronic resource]. 2025. Access mode: https://ktern.com/article/sap-custom-code-migration-guide-2024/.
3. SAPinsider. Technical guide: using ABAP Test Cockpit for SAP S/4HANA Transition [Electronic resource], 2017. Access mode: https://sapinsider.org/articles/technical-guide-using-abap-test-cockpit-for-sap-s-4hana-transition/.
4. Pozdnyakov O. A., Parkhomenko A. V. Migration of custom code to new versions of complex computer systems using methods and models of intelligent reengineering, *Scientific Works of DonNTU. Series "Informatics, Cybernetics and Computer Engineering"*, 2025, Vol. 2 (41), pp. 86–98 (in Ukranian).
5. Pozdnyakov O. A., Parkhomenko A. V. Research and selection of large learning modes for automation of ABAP-code migration, *Management of the development of complex systems*, 2025, Vol. 63, pp. 191–200 (in Ukrainian).
6. ISO/IEC 25010:2023. Systems and software engineering – systems and software quality requirements and evaluation (SQuaRE). Product quality model. [Electronic resource]. Access mode: https://www.iso.org/ru/standard/78176.html.
7. ISO/IEC 25040:2024. Systems and software engineering – systems and software quality requirements and evaluation (SQuaRE). Evaluation process. [Electronic resource]. Access mode: https://www.iso.org/standard/83467.html.
8. Nardo M., Saisana M., Saltelli A., Tarantola S., Hoffman A., Giovannini E. Handbook on constructing composite indicators: Methodology and user guide. Paris, OECD Publishing, 2008, 162 p.
9. Yurchyshyn V. M. Methodological Approaches to Assessing Software Quality for Oil and Gas Industry Facilities, *Methods and Instruments of Quality Control*, 2020, Vol. 2 (45), pp. 40–57. DOI:10.31471/1993-9981-2020-2(45)-40-57 (in Ukranian).
10. Prokofiev I. Method of Static Analysis of Code Quality Using Machine Learning, *Measuring and Computing Technology in Technological Processes*, 2025, Vol. 3, pp. 126–133. DOI:10.31891/2219-9365-2025-83-17 (in Ukrainian).
11. Qwen Team. Qwen2.5-Coder series: Powerful, Diverse, Practical [Electronic resource]. 2024. Access mode: https://qwenlm.github.io/blog/qwen2.5-coder-family/.
12. Rozière B. , Gehring J., Gloeckle F. et al. Code Llama: Open foundation models for code, *Meta AI Technical Report*, 2023, 38 p. DOI:10.48550/arXiv.2308.12950.
13. Weyssow M., Zhou X., Kim K. et al. Exploring parameter-efficient fine-tuning techniques for code generation with large language models, *ACM Transactions on Software Engineering and Methodology*, 2024, Vol. 33(6), pp. 1–25. DOI:10.1145/3714461.
14. Ray J. Teach an old dog new tricks: LLM continual pre-training (CPT) [Electronic resource], 2025. Access mode: https://medium.com/better-ml/teach-an-old-dog-new-tricks-llm-continual-pre-training-cpt-684cfb931247.
15. Han Z., Gao C., Liu J. et al. Parameter-efficient fine-tuning for large models: a comprehensive survey, *Transactions on Machine Learning Research*, 2024, pp. 1–44. DOI:10.48550/arXiv.2403.14608.
16. Hu E. J., Shen Y., Wallis P. et al. LoRA: Low-rank adaptation of large language models, *International Conference on Learning Representations (ICLR 2022)*, pp. 1–13. DOI:10.48550/arXiv.2106.09685.
17. Dettmers T., Pagnoni A., Holtzman A., Zettlemoyer L. QLoRA: Efficient finetuning of quantized LLMs, *37th Conference on Neural Information Processing Systems (NeurIPS 2023)*, 2023, pp. 1–28. DOI:10.48550/arXiv.2305.14314.
18. Hong J., Lee N., Thorne J. ORPO: Monolithic preference optimization without reference model, *2024 Conference on*

Empirical Methods in Natural Language Processing, 2024, pp. 11170–11189. DOI:10.18653/v1/2024.emnlp-main.626.

19. Ren S., Guo D., Lu S. et al. CodeBLEU: a method for automatic evaluation of code synthesis, *Computer Science. Software Engineering*, 2020, pp. 1–8. DOI:10.48550/arXiv.2009.10297.
20. Microsoft. CodeXGLUE: CodeBLEU evaluation metric [Electronic resource]. 2025. Access mode: https://github.com/microsoft/CodeXGLUE/tree/main/Code-Code/code-to-code-trans/evaluator/CodeBLEU.
21. Bhattacharjee A., Dwyer C. Analyzing and mitigating surface bias in code evaluation metrics, *Computer Science. Software Engineering*, 2025, pp. 1–22. DOI:10.48550/arXiv.2509.15397.

Received 09.12.2025.
Accepted 09.01.2026.
Published 27.03.2026.

УДК 004.89

ОЦІНКА ТА ЗАБЕЗПЕЧЕННЯ ЯКОСТІ МІГРОВАНОГО ABAP-КОДУ ЗА ДОПОМОГОЮ ІНТЕГРАЛЬНОЇ МЕТРИКИ ТА МОДЕЛЕЙ ГЕНЕРАТИВНОГО ШТУЧНОГО ІНТЕЛЕКТУ

Поздняков О. А. – аспірант кафедри програмних засобів, Національний університет «Запорізька політехніка», Запоріжжя, Україна. ROR: https://ror.org/03aph1990. ORCID: https://orcid.org/0009-0006-3955-802X.

Пархоменко А. В. – канд. техн. наук, доцент, доцент кафедри програмних засобів, Національний університет «Запорізька політехніка», Запоріжжя, Україна. ROR: https://ror.org/03aph1990. ORCID: https://orcid.org/0000-0002-6008-1610.

АНОТАЦІЯ

Актуальність. Автоматизація процесу міграції успадкованого користувацького коду при переході на нову версію системи S/4HANA за допомогою великих мовних моделей (LLM) є перспективним напрямом. Проте оцінка якості згенерованого коду залишається невирішеною проблемою, оскільки існуючі підходи використовують розрізнені метрики, що не дозволяють комплексно оцінити та забезпечити якість програмного коду для подальшого використання без додаткового доопрацювання.

Мета роботи – підвищення ефективності процесу інтелектуального реінжинірингу комп’ютерної системи на основі методу комплексного оцінювання та забезпечення якості мігрованого користувацького ABAP-коду.

Метод. Розроблений метод базується на двох ключових компонентах. Інтегральна метрика оцінки якості IAQS (Integral ABAP Quality Score) комплексно враховує синтаксичні, функціональні та семантичні характеристики коду та ґрунтується на положеннях міжнародних стандартів якості програмного забезпечення ISO/IEC 25010, ISO/IEC 25040, а також теорії композитних індикаторів. Триетапний підхід до донавчання LLM (Qwen 2.5 Coder 14B) включає безперервне попереднє навчання (CPT), параметро-ефективне донавчання (PEFT) та вирівнювання на основі переваг (Alignment) на основі алгоритму ORPO. При цьому використання розробленої метрики IAQS для формування набору даних переваг на етапі вирівнювання створює механізм керованого вдосконалення, а саме визначає напрямок адаптації LLM.

Результати. Результати експериментальних досліджень демонструють, що реалізація розробленого методу дозволяє покращити як окремі показники якості програмного коду, так і інтегральну метрику оцінку якості IAQS в цілому. Фінальна модель, донавчена на основі запропонованого триетапного підходу, дозволила досягти високого значення IAQS (0.756), що демонструє суттєве підвищення у порівнянні з базовою моделлю (0.117).

Висновки. Дослідження представляє новий проблемно-орієнтований підхід до автоматизованої міграції ABAP-коду при інтелектуальному реінжинірингу комп’ютерних систем. Запропонована інтегральна метрика IAQS є основою для створення формалізованої та об’єктивної системи оцінки якості програмного забезпечення, згенерованого LLM у контексті міграції успадкованого користувацького коду. Продемонстровано, що послідовне донавчання LLM на основі триетапного підходу з використанням IAQS забезпечує суттєве підвищення інтегрованого показника якості згенерованого програмного коду.

КЛЮЧОВІ СЛОВА: якість програмного забезпечення, інтегральна метрика, великі мовні моделі, міграція успадкованого користувацького коду, донавчання LLM.

ЛІТЕРАТУРА

1. Hardy P. Migrating custom code to SAP S/4HANA / P. Hardy. – Boston : Rheinwerk Publishing Inc., 2020. – 333 p.
2. Ktern AI. SAP S/4HANA 2022: The ultimate custom code migration guide [Electronic resource] / AI. Ktern. – 2025. – Access mode: https://ktern.com/article/sap-custom-code-migration-guide-2024/.
3. SAPinsider. Technical guide: using ABAP Test Cockpit for SAP S/4HANA Transition [Electronic resource] / SAPinsider. – 2017. – Access mode: https://sapinsider.org/articles/technical-guide-using-abap-test-cockpit-for-sap-s-4hana-transition/.
4. Поздняков О. А. Міграція користувацького коду на нові версії складних комп’ютерних систем з використанням методів і моделей інтелектуального реінжинірингу / О. А. Поздняков, А. В. Пархоменко // Наукові праці ДонНТУ. Серія «Інформатика, кібернетика та обчислювальна техніка». – 2025. – № 2 (41). – С. 86–98.
5. Поздняков О. А. Дослідження та вибір великих мовних моделей для автоматизації міграції ABAP-коду/ О. А. Поздняков, А. В. Пархоменко // Збірник наукових праць «Управління розвитком складних систем». – 2025. – № 63. – С. 191–200.
6. ISO/IEC 25010:2023. Systems and software engineering – systems and software quality requirements and evaluation (SQuaRE) – Product quality model. [Electronic resource]. – Access mode: https://www.iso.org/ru/standard/78176.html.
7. ISO/IEC 25040:2024. Systems and software engineering – systems and software quality requirements and evaluation (SQuaRE) – Evaluation process. [Electronic resource]. – Access mode: https://www.iso.org/standard/83467.html.
8. Handbook on constructing composite indicators: Methodology and user guide / [M. Nardo, M. Saisana, A. Saltelli et al.]. – Paris : OECD Publishing, 2008. – 162 p.

DOI 10.15588/1607-3274-2026-1-8

9. Юрчишин В. М. Методологічні підходи щодо оцінки якості програмного забезпечення для об'єктів нафтогазового комплексу / В. М. Юрчишин // Методи та прилади контролю якості. – 2020. – № 2 (45). – С. 40–57. DOI:10.31471/1993-9981-2020-2(45)-40-57.
10. Прокоф'єв І. Метод статичного аналізу якості коду з допомогою машинного навчання / І. Прокоф'єв // Вимірювальна та обчислювальна техніка в технологічних процесах. – 2025. – № 3. – С. 126–133. DOI:10.31891/2219-9365-2025-83-17.
11. Qwen Team. Qwen2.5-Coder series: Powerful, Diverse, Practical [Electronic resource] / Qwen Team // QwenLM Blog. – 2024. – Access mode: https://qwenlm.github.io/blog/qwen2.5-coder-family/.
12. Code Llama: Open foundation models for code / [B. Rozière, J. Gehring, F. Gloeckle et al.] // Meta AI Technical Report. – 2023. – 38 p. DOI:10.48550/arXiv.2308.12950.
13. Exploring parameter-efficient fine-tuning techniques for code generation with large language models / [M. Weyssow, X. Zhou, K. Kim et al.] // ACM Transactions on Software Engineering and Methodology. – 2024. – Vol. 33, № 6. – P. 1–25. DOI: 10.1145/3714461.
14. Ray J. Teach an old dog new tricks: LLM continual pre-training (CPT) [Electronic resource] / J. Ray // Medium. – 2025. – Access mode: https://medium.com/better-ml/teach-an-old-dog-new-tricks-llm-continual-pre-training-cpt-684cfb931247.
15. Parameter-efficient fine-tuning for large models: A comprehensive survey / [Z. Han, C. Gao, J. Liu et al.] // Transactions on Machine Learning Research. – 2024. – P. 1–44. DOI:10.48550/arXiv.2403.14608.
16. LoRA: Low-rank adaptation of large language models / [E. J. Hu, Y. Shen, P. Wallis et al.] // International Conference on Learning Representations (ICLR 2022). – P. 1–13. DOI:10.48550/arXiv.2106.09685.
17. QLoRA: Efficient finetuning of quantized LLMs / [T. Dettmers, A. Pagnoni, A. Holtzman, L. Zettlemoyer] // 37th Conference on Neural Information Processing Systems (NeurIPS 2023). – 2023. – P. 1–28. DOI:10.48550/arXiv.2305.14314.
18. Hong J. ORPO: Monolithic preference optimization without reference model / J. Hong, N. Lee, J. Thorne // 2024 Conference on Empirical Methods in Natural Language Processing. – 2024. – P. 11170–11189. DOI:10.18653/v1/2024.emnlp-main.626.
19. CodeBLEU: A method for automatic evaluation of code synthesis / [S. Ren, D. Guo, S. Lu et al.] // Computer Science. Software Engineering. – 2020. – P. 1–8. DOI:10.48550/arXiv.2009.10297.
20. Microsoft. CodeXGLUE: CodeBLEU evaluation metric [Electronic resource] / Microsoft. – 2025. – Access mode: https://github.com/microsoft/CodeXGLUE/tree/main/Code-Code/code-to-code-trans/evaluator/CodeBLEU.
21. Bhattacharjee A. Analyzing and mitigating surface bias in code evaluation metrics / A. Bhattacharjee, C. Dwyer // Computer Science. Software Engineering. – 2025. – P. 1–22. DOI:10.48550/arXiv.2509.15397.

DOI 10.15588/1607-3274-2026-1-8