https://doi.org/10.31891/csit-2025-3-15
UDC 004.93.1:004.912:004.4'42

Stanislav DANYLENKO, Serhii SMELYAKOV
Kharkiv National University of Radio Electronics

# A META-MODEL FOR LOW-CODE CONFIGURATION AND DEPLOYMENT OF CONTENT-BASED IMAGE RETRIEVAL SYSTEMS

*The object of this study is content-based image retrieval (CBIR) systems with configurable architectures, and the subject is the meta-model for low-code configuration and deployment of CBIR systems. The goal of this work is to develop a meta-model for CBIR systems that enables their low-code configuration and deployment. The proposed meta-model allows users to define a CBIR system by selecting and combining the CBIR components from a predefined catalog, after which deployment artifacts are automatically generated and can be easily deployed. The proposed meta-model formalizes CBIR components: image repository, feature extractor, feature database (logical and physical levels), similarity measure, result aggregator, and user interaction layer; and extends them with two meta-level components: a configuration manager and a deployment engine. The architecture was implemented using Docker for containerization, Spring Boot starters for modularity, and a web-based graphical interface for configuration. A prototype was developed and tested on a dataset of 100 000 images, with systematic variation of component combinations. Experiments confirmed that the meta-model enables rapid reconfiguration and deployment of CBIR systems, allowing the evaluation of performance under different configurations. The examined difference between the best and worst tested configurations highlighting the significant effect of component selection on system performance. Scientific novelty lies in introducing a formalized meta-model that integrates low-code principles into CBIR design, combining modular architecture, containerized deployment, and graphical configuration in a single framework. The practical significance of the solution is in simplifying CBIR experimentation for researchers and practitioners without deep programming expertise, enabling rapid prototyping, testing, and deployment of customized CBIR systems.*

*Keywords: content-based image retrieval, meta-model, low-code development, containerization, modularity, graphical configuration, image processing.*

Станіслав ДАНИЛЕНКО, Сергій СМЕЛЯКОВ
Харківський національний університет радіоелектроніки

# МЕТАМОДЕЛЬ ДЛЯ СПРОЩЕНОЇ КОНФІГУРАЦІЇ ТА РОЗГОРТАННЯ СИСТЕМ ПОШУКУ ЗОБРАЖЕНЬ ЗА ВМІСТОМ

*Об'єктом дослідження є системи пошуку зображень за вмістом (CBIR) з можливістю конфігурування архітектури, а предметом – метамодель для конфігурування та розгортання CBIR-систем з мінімальним написання програмного коду. Метою роботи є розробка та експериментальна перевірка метамоделі, яка забезпечує побудову CBIR-систем шляхом вибору та поєднання компонентів, які можна налаштовувати, із попередньо визначеного каталогу з подальшою автоматичною генерацією артефактів розгортання. Запропонована метамодель формалізує компоненти CBIR: сховище зображень, екстрактор ознак, базу дескрипторів (логічний та фізичний рівні), міру схожості, агрегатор результатів та рівень взаємодії з користувачем; та розширює їх двома метарівневими компонентами: менеджером конфігурації та рушієм розгортання. Архітектура реалізована з використанням Docker для контейнеризації, Spring Boot стартерів для модульності та веб-інтерфейсу для графічної конфігурації. Прототип перевірено експериментально на наборі зі 100 000 зображень із систематичною зміною комбінацій компонентів. Експерименти підтвердили, що метамодель дозволяє швидко вносити зміни до налаштування та розгортати CBIR-системи для оцінки ефективності різних конфігурацій. Досліджена різниця між найкращою та найгіршою перевіреними конфігураціями демонструє значний вплив вибору компонентів на продуктивність системи. Наукова новизна полягає у запропонуванні формалізованої метамоделі, яка інтегрує принципи мінімальним написання програмного коду у проєктування CBIR, поєднуючи модульну архітектуру, розгортання контейнерів та графічну конфігурацію в єдиному середовищі. Практична значимість рішення полягає у спрощенні проведення експериментів із CBIR для дослідників та практиків без глибоких знань програмування, що дає змогу швидко створювати прототипи, тестувати та розгортати CBIR-системи, адаптовані під конкретні потреби.*

*Ключові слова: пошук зображень за вмістом, метамодель, розробка з мінімальним написанням коду, контейнеризація, модульність, графічна конфігурація, обробка зображень.*

Received / Стаття надійшла до редакції 10.08.2025
Accepted / Прийнята до друку 21.09.2025

## Introduction

Data search is a fundamental operation underpinning a wide range of applications, from data warehouses and social networks to large-scale Internet services [1]. Modern search engines such as Google, Bing, and Yahoo employ sophisticated algorithms optimized for general-purpose retrieval, each with its own strengths and limitations [2]. Algorithms for searching numeric and symbolic data have been extensively developed and are effectively applied in areas such as dictionary-based searches or query processing in web browsers [3].

Beyond text, many search engines now support image search. Simpler implementations typically rely on keywords, surrounding text, or image metadata [4]. While these approaches are relatively easy to formalize and computationally efficient, they often provide low accuracy because they do not analyze the actual visual content of the image.

To address these limitations, increasing attention has turned to content-based image retrieval (CBIR), where the query itself is an image rather than a textual description. CBIR algorithms aim to identify images that are visually similar to the query by extracting and comparing intrinsic features of the images. However, even CBIR does not guarantee high-quality results. Target images may undergo significant modifications (e.g., scaling, rotation, color changes, partial occlusion), requiring algorithms to determine which visual properties are most significant, select appropriate similarity measures, and efficiently search within large datasets [5]. Effective CBIR systems must therefore account for both image characteristics [6] and the structure of the storage system – for example, whether it is organized as graphs, trees, or hash-based indexes [7].

General-purpose search engines struggle with such tasks. Queries may return no relevant results or, conversely, thousands of visually similar images that only loosely resemble the target [8]. These challenges are particularly pronounced in specialized domains, such as medical imaging (e.g., electrocardiograms), technical drawings, or document images, where subtle differences carry critical meaning. Furthermore, public search engines cannot be used with sensitive or private data, creating a need for domain-specific CBIR systems that operate within controlled environments.

However, designing such systems poses its own challenges. A CBIR system typically involves multiple components: feature extraction (descriptor generation), storage structures and indexing, search algorithms, similarity metrics, and user interaction layers [9]. Configuring these components requires collaboration between domain experts and software engineers. Reconfiguring the system to adapt to new tasks can be costly and time-consuming.

Problem statement. General-purpose search systems offer high retrieval speed but lack the flexibility needed for specialized applications. Domain-specific systems, while more adaptable, face difficulties in balancing accuracy, efficiency, and scalability under big data conditions, where image volumes grow continuously and modified versions of images proliferate. As a result, end users are confronted with fundamental questions: Which search system to choose? Which retrieval model to employ? How to configure it effectively for a given task?

To address these challenges, this work proposes a meta-model for CBIR systems – a conceptual framework that allows configurable assembly of CBIR systems from standardized components.

## **Related Work**s

Research on CBIR has produced a variety of methods aimed at extracting informative features from images in the form of descriptors and efficiently organizing them for search. The OpenCV library provides a robust set of tools widely used in practice [10]. Both local and global feature detectors, such as SIFT, ORB, and FAST, are employed to identify key points and distinctive regions within images [11]. With the advent of deep learning, convolutional and transformer-based neural networks are increasingly utilized to generate high-level image representations, capturing object categories, spatial structures, and other semantic features [12]. These networks, often designed for classification or object detection, perform well when the target images resemble the training data but may degrade on previously unseen or domain-specific content [13].

Once descriptors are obtained, they are typically represented as vectors, and appropriate distance metrics are applied to compare them. Common similarity measures include Euclidean and Manhattan distances, Chi-square dissimilarity, Jaccard index, and Hamming distance [14-15].

For efficient storage, indexing, and retrieval of image descriptors, various data structures are employed to reduce the search space without compromising retrieval quality [16]. Classical multidimensional structures such as KD-trees and R-trees are commonly used [17]. Hashing-based methods, including Locality-Sensitive Hashing (LSH), accelerate similarity search [18], while graph-based approaches, particularly Hierarchical Navigable Small World (HNSW) graphs, have gained popularity for their scalability and fast retrieval performance. Clustering techniques, such as k-means, are often used as building blocks for advanced methods. When combined with Product Quantization (PQ), clustering enables inverted multi-indexing (IMI), one of the most effective techniques for balancing memory efficiency, indexing speed, and retrieval quality [9]. Furthermore, machine learning and deep learning methods increasingly support incremental or fine-tuning training after deployment, allowing systems to adapt and improve performance over time [19].

Several frameworks have attempted to provide modular or flexible solutions. OpenMMLab [20], for instance, enables the combination of different feature extraction models, while Haystack [21] offers a pluggable architecture for text retrieval. In the broader machine learning landscape, platforms such as MLflow facilitate pipeline orchestration, experiment tracking, and deployment [22]. However, these tools are not tailored to CBIR and typically require substantial programming and infrastructure expertise.

Some research has explored low-code or auto-configuration approaches for retrieval systems, including automated feature selection and neural architecture search [23-24]. While promising, these methods usually optimize individual components rather than providing an integrated environment where descriptors, indexing structures, similarity metrics, and user interfaces can be composed and deployed as a cohesive system.

Other solutions attempt to deliver ready-to-use CBIR systems but suffer from different limitations. For example, LIRE offers a convenient interface and a broad set of descriptors, indexing images by multiple characteristics simultaneously [25]. While this facilitates the selection of the best-performing descriptor, it significantly increases indexing time and does not allow evaluating the impact of other system components.

Conversely, libraries such as Faiss provide a highly efficient foundation for similarity search [26], but they require substantial programming expertise to integrate into a complete CBIR pipeline.

Overall, prior research has produced a wide range of methods and tools for feature extraction, similarity measurement, and efficient indexing, as well as flexible machine learning platforms that support modularity and automation. However, these contributions remain fragmented, and their integration into complete CBIR systems still requires substantial expertise and manual effort.

## Problem Statement and Purpose

Building on these observations, the problem addressed in this work is formalized. Modern CBIR solutions typically fall into three categories:

1) general-purpose frameworks, which provide high flexibility but require substantial programming and infrastructure expertise;
2) turnkey tools, which are easier to use but often rigid, offering limited control over system components;
3) technologies not originally designed for CBIR, which provide valuable building blocks but cannot be directly applied to assemble an end-to-end CBIR system without significant customization.

Existing approaches provide powerful components for CBIR but lack an integrated way to rapidly configure, assemble, and deploy systems tailored to specific tasks. This gap creates a need for a solution that combines the flexibility of modular frameworks with the accessibility of low-code tools.

The goal of this work is to develop a meta-model for CBIR systems that enables their low-code configuration and deployment. The proposed meta-model allows users to define a CBIR system by selecting and combining the main components from a predefined catalog, after which deployment artifacts are automatically generated and can be easily deployed.

To achieve this goal, the following tasks are addressed:

1) design the overall architecture of the meta-model, presented as a high-level component diagram;
2) develop a mathematical model of the meta-model, formalizing all configurable components of a CBIR system;
3) select appropriate software tools and frameworks and define an implementation approach;
4) evaluate the feasibility and performance of the proposed meta-model through experimental validation.

By addressing these tasks, this work lays the foundation for a unified, low-code environment for CBIR. The following sections introduce the proposed approach, including the high-level architecture of the meta-model, its mathematical formulation, an implementation strategy based on state-of-the-art tools, and experimental validation.

## Proposed Approach

In this work, a meta-model for CBIR systems is defined as a conceptual and formalized framework that abstracts the structure of a CBIR system into a set of configurable components. The meta-model specifies the relationships between these components and provides mechanisms for their parametrization, combination, and automated deployment. Unlike a fixed CBIR architecture, the meta-model operates at the meta-level, enabling the creation of diverse CBIR systems without rewriting the source code by the user.

The process of content-based image retrieval has been conceptually established for decades, and while implementation techniques have evolved, its fundamental stages remain consistent. They are depicted in Figure 1.

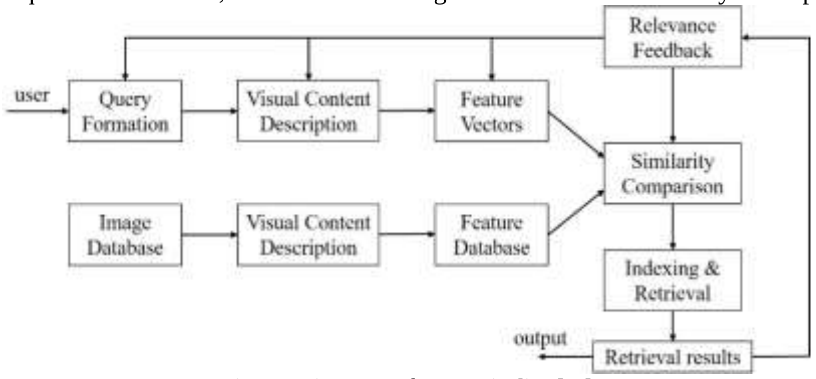


**Fig. 1. Main stages of CBIR pipeline [27]**

These stages can be described as follows:

1) query formation. The user provides a query image. A feature extractor then computes a descriptor of the selected type, representing the image as a feature vector based on visual characteristics;
2) feature extraction for the image collection. Images stored in the repository are processed in the same manner as the query image. Their descriptors are generated once and stored for subsequent retrieval;

3) construction of the feature database. The calculated descriptors are organized into a feature database, which can be described at two levels: logical level – the data structure with its indexing approach and search algorithms, and physical level – the storage strategy of this data structure in the memory;

4) similarity search. The system compares the query descriptor with descriptors in the feature database using a similarity or dissimilarity metric. Navigation through the chosen data structure of the feature database determines how efficiently the most similar candidates are identified.

5) result aggregation and ranking. Retrieved results are ranked according to a predefined function, which orders candidates based on similarity scores. The top-k results are returned to the user according to task-specific criteria;

6) relevance feedback (optional). In some systems, user feedback on retrieved results is used to refine descriptors, metrics, or ranking functions. Since this stage is not mandatory, it is not considered in the meta-model.

Based on these processes, the main components of a CBIR system can be identified:

1) image repository – storage of raw images (e.g., local folder, network storage, cloud object store);

2) feature extractor – module responsible for computing descriptors (e.g., local/global features, SIFT, VLAD);

3) feature database (logical level) – indexing structures and search algorithms (e.g., KD-trees, HNSW graphs, Locality-Sensitive Hashing);

4) feature database (physical level) – mechanisms for storing descriptor data (e.g., in-memory, on-disk, or distributed);

5) similarity measure – metric for comparing feature vectors (e.g., Euclidean, Manhattan);

6) result aggregator – module that ranks and filters retrieved results (ranking and selection strategies, ascending/descending order, setting top-k parameter);

7) user interaction layer – graphical interface (or API, CLI, etc.) for system interaction.

The meta-model encompasses all core CBIR components and introduces two additional meta-level components:

1) configuration manager – allows users to select and combine system components from a predefined catalog;

2) deployment engine – generates deployment artifacts for low-code system assembly.

The main components of a CBIR meta-model are depicted in Figure 2.

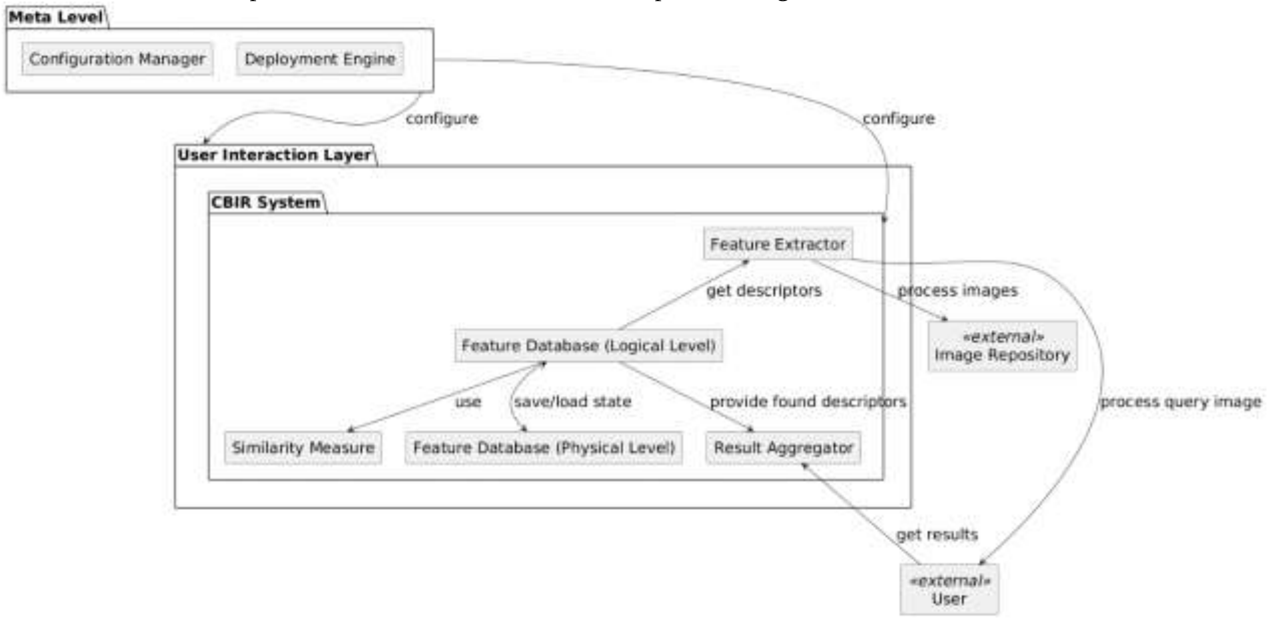


**Fig. 2. Components of the CBIR meta-model**

Since additional meta-components are not part of the CBIR system, but are only used for its configuration and creation, they are located at the meta level. All interaction with the system is performed using the user interaction layer, so in the diagram it includes the CBIR system and limits it from the outside world. An image repository is an external resource from which descriptors are analyzed and entered into the feature database. The image repository and feature database must always be synchronized. For clarity, a user has been added to the diagram. The user uploads an image for search, which is analyzed using the feature extractor and receives results generated by the result aggregator. For the other components, dependencies and the main operations that require these dependencies are shown.

The meta-model for CBIR systems is defined as follows. Let $C$ denote the configuration space:

$$C = E \times S \times I \times M \times A \times U \times R, \tag{1}$$

where $E$ – the set of feature extractors, $S$ – the set of descriptor storage models, $I$ – the set of indexing structures $M$ – the set of similarity metrics, $A$ – the set of result aggregators, $U$ – the set of user interaction interfaces, $R$ – the set of image repositories.

The Cartesian product symbol between the parameters means that each configuration $c$:

$$c = (e, s, i, m, a, u, r), \tag{2}$$

selects one element from each set from the formula (1): $e \in E, s \in S, i \in I, m \in M, a \in A, u \in U, r \in R$.

A CBIR system under configuration $c$ is a function $B$:

$$B : P_q \times C \rightarrow L, \tag{3}$$

where $P_q$ is a set of possible query images, $C$ is from the formula (1), and $L$ is the set of ranked result lists. For a given query image $q \in P_q$ with configuration $c \in C$ the $B_{(q,c)}$ is:

$$B_{(q,c)} = a\left(\left\{\left(x, \sigma_m\left(f_e(q), f_e(x)\right)\right) \middle| x \in D(r)\right\}\right), \tag{4}$$

where $a(\cdot)$ from the formula (2) is a function that ranks and selects results, $x$ – an image from the set $D(r)$ accessible from repository $r$, processed into descriptors stored according to $s$ and indexed by $i$ from the formula (2), $\sigma_m : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}$ computes similarity using metric $m$ from the formula (2), $f_e : P_q \rightarrow \mathbb{R}^n$ maps images to n-dimensional descriptors using the feature extractor $e$ and put it to the $s$ from the formula (2), $P_q$ from the formula (3), $u$ from the formula (2) is just an interaction interface, it doesn't impact the retrieval pipeline and is out of the formula (4).

To evaluate system performance, let $T$ be the function measuring the specific metric important for the CBIR system (e.g., retrieval time, precision, recall):

$$T : P_q \times C \rightarrow \mathbb{R}^+, \tag{5}$$

where $P_q$ from the formula (3), $C$ from the formula (1).

Let's define it as retrieval time (in seconds) required to process one query image under configuration $c$ from the formula (2). For two configurations $c_1$ and $c_2$, their relative performance is expressed as:

$$\Delta T(q; c_1, c_2) = T(q, c_1) - T(q, c_2). \tag{6}$$

where $T$ from the formula (5), $q$ – query. A positive $\Delta T$ indicates that configuration $c_2$ is faster than $c_1$ for query $q$. Over a set of queries $Q \subseteq P_q$ from the formula (3), the average retrieval time difference is:

$$\overline{\Delta T}(c_1, c_2) = \frac{1}{|Q|} \sum_{q \in Q} \Delta T(q; c_1, c_2), \tag{7}$$

where $\Delta T, q, c_1, c_2$ from the formula (6).

This measure links the theoretical model to experimental evaluation by quantifying how changes in configuration (e.g., similarity metric, storage model, indexing structure) affect retrieval efficiency.

The proposed meta-model is designed to support three core principles: containerization, modularity, and graphical configuration. These are enabled through two additional components introduced at the meta-level: the configuration manager and the deployment engine.

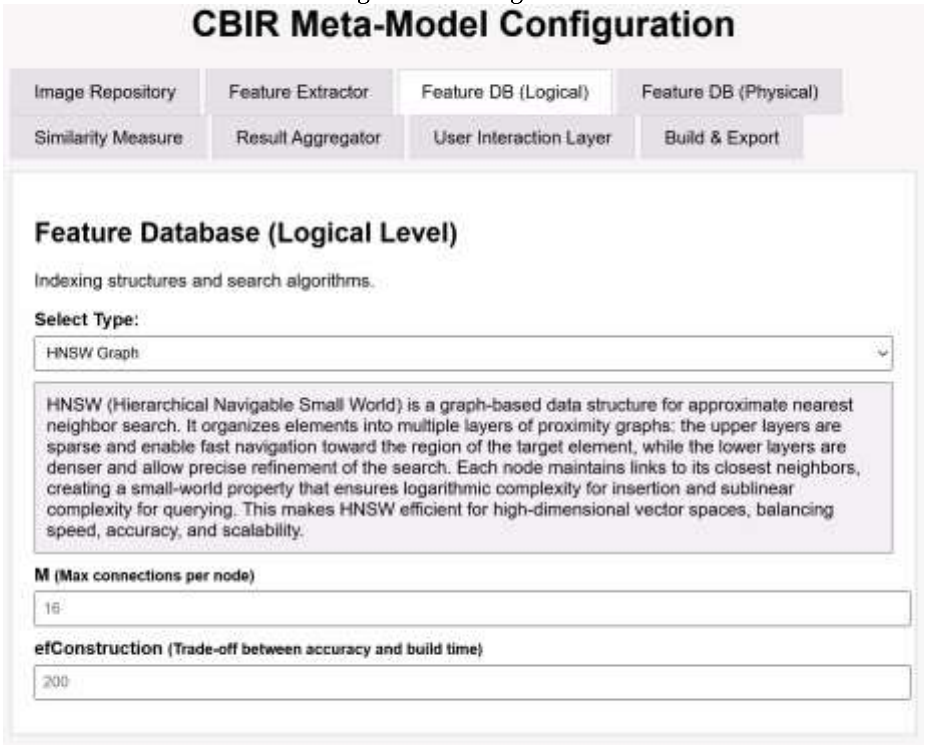
Containerization ensures that each configured CBIR system can be packaged and deployed as an isolated unit, encapsulating all required components and their dependencies. This principle is realized using Docker, which enables the creation of parameterized templates (Docker images) based on the set of user-selected components. Each image can be built and instantiated as a container with user-defined properties, dependencies, and linked to external services such as storage or database engines.

Modularity allows each component of the CBIR system (e.g., feature extraction, descriptor storage, indexing, similarity measures, interaction interfaces) to be independently selected and combined. This principle is achieved using Spring Boot, a framework for Java programming language, where each module is implemented as a configurable starter that adheres to a common interface. Modules can be customized via properties, including those defined during Docker container configuration. This approach enables flexible composition of retrieval systems tailored to specific use cases.

Graphical configuration provides a web-based interface that allows users to configure CBIR systems without writing code. Through this interface, users can select desired modules, which are implementations of components, from a catalog and specify associated parameters. Behind the scenes, the system maps selected modules to corresponding Spring Boot starters, which are automatically included in the project structure. Configuration parameters provided by the user are applied at multiple levels: some are written into Spring Boot configuration files, while others are injected as Docker container arguments or environment variables. This mechanism enables non-technical users to perform complex modifications to the system architecture and behavior through a simple and intuitive UI.

Once configuration is complete, the interface generates a deployment package that includes all required Docker files and instructions. After installing Docker and linking it to a container registry (e.g., Docker Hub), users can build the customized image and instantiate a container in two steps. The resulting system is fully functional and can be accessed through the selected user interface module, allowing immediate interaction with the deployed CBIR instance, including starting image repository indexing and performing searches.

Figure 3 illustrates the GUI of the configuration manager.



CBIR Meta-Model Configuration

Feature Database (Logical Level)

Indexing structures and search algorithms.

Select Type: HNSW Graph

HNSW (Hierarchical Navigable Small World) is a graph-based data structure for approximate nearest neighbor search. It organizes elements into multiple layers of proximity graphs: the upper layers are sparse and enable fast navigation toward the region of the target element, while the lower layers are denser and allow precise refinement of the search. Each node maintains links to its closest neighbors, creating a small-world property that ensures logarithmic complexity for insertion and sublinear complexity for querying. This makes HNSW efficient for high-dimensional vector spaces, balancing speed, accuracy, and scalability.

M (Max connections per node): 16

efConstruction (Trade-off between accuracy and build time): 200

**Fig. 3. GUI configuration manager**

In summary, the proposed meta-model provides a unified framework for designing CBIR systems, combining a component-based architecture, mathematical formalization, and a practical implementation based on containerization, modularity and graphical configuration. This approach allows users to flexibly select and combine system components while avoiding the need for extensive coding or complex infrastructure setup. The next section experimentally evaluates the proposed solution, demonstrating how different configurations impact retrieval efficiency.

## Experiments and Discussions

To evaluate the practical applicability of the proposed meta-model, a prototype was developed according to the conceptual and implementation principles described in the previous section.

The prototype was implemented in Java 17 (Amazon Corretto build) using Spring Boot (3.1.5) and Docker (28.3.2). Experiments were conducted on a workstation Apple MacBook Pro 2021 equipped with M1 Pro processor on ARM architecture, 10-cores up to 3.2 GHz, 16 GB of LPDDR5 SDRAM up to 200 Gb/s, 512 GB SSD, integrated GPU with 16 cores. The experiments were conducted using the COCO2017 dataset [28], consisting of more than 100 000 images. However, for experiments, 100 000 images were randomly selected and used.

Table 1 summarizes the configurable components implemented in the prototype, corresponding to the categories defined by the meta-model.

Even for the prototype, at least two different implementations were developed for each component, demonstrating the ease of creating pluggable modules and the potential for third-party developers to contribute their own.

To isolate the impact of key configurable components, several modules of the CBIR system were held constant during experimentation:

1) image repository: local directory;
2) feature database (physical level): local file;
3) result aggregator: top-10 ranking with ascending difference order (including image ID and similarity value),
4) user interaction layer: GUI.

Table 1

**Available implementations for the components in the experimental prototype**

| **Component** | **Available implementations** |
|---|---|
| Image repository | local directory, S3 |
| Feature extractor | Invariant Brightness Histogram, Wavelet Hash |
| Feature database (logical level) | Locality-Sensitive Hashing, Hierarchical Navigable Small World, Multidimensional Cube, Inverted Multi-Index, KD-tree |
| Feature database (physical level) | local file, database |
| Similarity measure | Manhattan distance, Euclidean distance, Squared Euclidean distance |
| Result aggregator | top-10 with ascending difference order (including image ID and similarity value), top-25 with ascending difference order (including image ID, recall and precision) |
| User interface | GUI, REST API |

Three components were varied systematically to assess their influence on retrieval performance:

1) feature extractor: Invariant Brightness Histogram (IBH) and Wavelet Hash (WH),
2) feature database (logical level): Locality-Sensitive Hashing (LSH) and KD-Tree (KDT),
3) similarity measure: Manhattan Distance (MD) and Squared Euclidean Distance (SED).

These parameters are components $e,\ i, m$ of the configuration $c$ from the formula (2). This configuration yields eight unique system variants, formed by combining the possible values of these three components.

The performance was evaluated by retrieval (search) time per query (in seconds), as it was introduced in the formula (5). However, the meta-model allows the use of alternative or composite metrics depending on application requirements. A set of 100 images from the repository was randomly selected and used as query images. The same selected images were used for searching across all configurations.

The experimental results are summarized in Table 2. The first columns show the specific implementations of the components, followed by the average search time, calculated by formula (7), for the search of the 100 query images, and the last column shows the difference in search time between each configuration $c$ and the best-performing one calculated by the formula (6). The best result is highlighted with bold text style in the table. The abbreviations are described above.

Table 2

**Comparison of the performance of the CBIR system with different configurations**

| **Feature extractor** | **Feature database** | **Similarity measure** | **Retrieval time (s)** | **Difference vs the best configuration (s)** |
|---|---|---|---|---|
| IBH | LSH | MD | 0.00558 | 0.00555 |
| IBH | LSH | SED | 0.00538 | 0.00535 |
| IBH | KDT | MD | 0.03674 | 0.03671 |
| IBH | KDT | SED | 0.00032 | 0.00029 |
| WH | LSH | MD | 0.00005 | 0.00002 |
| WH | LSH | SED | **0.00003** | - |
| WH | KDT | MD | 0.03745 | 0.03742 |
| WH | KDT | SED | 0.02606 | 0.02603 |

The best result was 0.00003 seconds, achieved with the combination of Wavelet Hash + LSH + Squared Euclidean Distance, while the worst result was 0.03745 seconds using Wavelet Hash + KD-Tree + Manhattan Distance. This indicates that Wavelet Hash descriptors perform better when paired with LSH-based feature databases (logical level). The choice of similarity metric had a limited impact on the performance of LSH, whereas the descriptor type was a more influential factor. In contrast, for KD-Tree, the descriptor choice had little effect, but the similarity metric significantly affected performance.

These results also demonstrate the potential performance penalty of selecting a suboptimal configuration: the time difference between the best and worst setups was 0.03742 seconds, which can become a critical factor under high query loads in a real-world search system.

The time required to build and deploy containers depends on workstation specifications and internet bandwidth. In our experiments, this process did not exceed 5 minutes. This means that deploying the 8 configurations used in the experiment took approximately 40 minutes. System startup typically takes a few seconds, plus the time required to read images from the repository and generate descriptors and indexes if it's the container's first launch. Subsequent startups are faster because the indexes have already been built. Since created containers can be quickly restarted, for example, to rerun an experiment, this significantly accelerates the process and allows for rapid switching between CBIR systems when needed.

The process of selecting components and specifying the necessary parameters may vary for each expert and directly depends on their knowledge and experience with CBIR systems. It also depends on the completeness and quality of the module description provided by its developer. However, after an initial familiarization, the process should be straightforward and proceed quickly.

The experiments can be considered successful, as they have practically confirmed the feasibility of using the proposed meta-model for constructing and deploying CBIR systems with varying parameters, allowing for the evaluation of their effectiveness under different conditions. The use of the meta-model, based on the developed prototype, is straightforward and enables efficient attainment of the desired outcomes.

Limitations. As no directly comparable low-code solutions for CBIR or alternative meta-models are currently available, the experiments do not include a direct benchmark against existing systems. The focus was instead on demonstrating the ability of the proposed meta-model to configure and evaluate diverse CBIR systems through systematic variation of key components. The chosen metric for evaluating the meta-model is quite simple, as its primary purpose is to demonstrate the capabilities of the meta-model rather than to conduct a detailed analysis of its application to a specific task.

## Conclusions

This work proposed and validated a meta-model for content-based image retrieval (CBIR) systems that enables low-code configuration and deployment of these systems from modular components. The developed solution addresses the identified gap between flexible but technically demanding frameworks and user-friendly but rigid turnkey tools, by providing a unified approach that combines modularity, containerization, and graphical configuration.

All objectives set in this study have been achieved. The architecture was designed as a high-level component diagram. A mathematical model of the meta-model was formulated, formalizing the configurable components of a CBIR system. An implementation strategy was developed based on Docker and Spring Boot starters. A prototype was implemented according to this strategy, and its feasibility was experimentally confirmed.

The experiments demonstrated that the proposed meta-model makes it possible to rapidly assemble, deploy, and evaluate CBIR systems with different configurations, significantly reducing the effort required to test alternative designs. Performance tests showed that the choice of component combinations can have a measurable impact on retrieval efficiency, with differences of up to 0.037 seconds per query between the best and worst configurations. The results also confirmed that the graphical configuration interface and containerized deployment process allow even non-technical users to work with complex CBIR architectures.

Future work will focus on three main directions:

1) expanding the catalog of available modules for each type of CBIR component;
2) conducting more experiments with clearly defined retrieval tasks and appropriate evaluation metrics;
3) comparing the proposed meta-model with analogous solutions as they appear, to position it within the evolving landscape of CBIR technologies.

Overall, the proposed meta-model has proven to be an effective and practical solution for rapid configuration and deployment of domain-specific CBIR systems, and it establishes a foundation for further research and development in this area.

## References

1. Tucker V.M., Edwards S.L. Search evolution for ease and speed: A call to action for what's been lost. *Journal of Librarianship and Information Science*. 2021. Vol. 53. P 668–685. DOI: https://doi.org/10.1177/0961000620980827.
2. Palanisamy R. Evaluation of Search Engines: A Conceptual Model and Research Issues. *International Journal of Business and Management*. 2013. Vol. 8. №6. DOI: https://doi.org/10.5539/ijbm.v8n6p1.
3. Raj M., Mishra N. Exploring new Approaches for Information Retrieval through Natural Language Processing. *arXiv*. 2025. DOI: https://doi.org/10.48550/arXiv.2505.02199.
4. Jony R.I., Woodley A., Perrin D. Flood Detection in Social Media Images using Visual Features and Metadata. *2019 Digital Image Computing: Techniques and Applications (DICTA)*. 2019: P. 1–8. DOI: https://doi.org/10.1109/dicta47822.2019.8946007.
5. Chen L., Luo C., Li X., Xiao J. Rotating target detection algorithm in remote sensing images based on improved YOLOv5s. *2023 4th International Conference on Computer Vision, Image and Deep Learning (CVIDL).* 2023. P. 180–184. DOI: https://doi.org/10.1109/cvidl58838.2023.10167055.
6. Meng K., Wo Y. An image compression and encryption scheme for similarity retrieval. *Signal Processing: Image Communication*. 2023. Vol. 119. DOI: https://doi.org/10.1016/j.image.2023.117044.
7. Dimopoulou M., Antonini M. Efficient Storage of Images onto DNA using Vector Quantization. *2020 Data Compression Conference (DCC)*. 2020. P. 363–363. DOI: https://doi.org/10.1109/dcc47342.2020.00085.
8. Bitirim Y., Bitirim S., Celik Ertugrul D., Toygar O. An Evaluation of Reverse Image Search Performance of Google. *2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC).* 2020. P. 1368–1372. DOI: https://doi.org/10.1109/compsac48688.2020.00-65.
9. Li X., Yang J., Ma J. Recent developments of content-based image retrieval (CBIR). *Neurocomputing.* 2021. Vol. 452. P. 675–689. DOI: https://doi.org/10.1016/j.neucom.2020.07.139.
10. Nazir A., Ashraf R., Hamdani T., Ali N. Content based image retrieval system by using HSV color histogram, discrete wavelet transform and edge histogram descriptor. *2018 International Conference on Computing, Mathematics and Engineering Technologies (iCoMET).* 2018. P. 1–6. DOI: https://doi.org/10.1109/icomet.2018.8346343.

11. Wu S., Oerlemans A., Bakker E.M., Lew M.S. A comprehensive evaluation of local detectors and descriptors. *Signal Processing: Image Communication*. 2017. Vol. 59. P. 150–167. DOI: https://doi.org/10.1016/j.image.2017.06.010.

12. Singh P.N., Gowdar T.P. Reverse Image Search Improved by Deep Learning. *2021 IEEE Mysore Sub Section International Conference (MysuruCon)*. 2021. P.596–600. DOI: https://doi.org/10.1109/mysurucon52639.2021.9641572.

13. Alruwaili M., Atta M.N., Siddiqi M.H., Khan A., Khan A., Alhwaiti Y., Alanazi S. Deep Learning-Based YOLO Models for the Detection of People With Disabilities. *IEEE Access*. 2024. Vol. 12. P. 2543–2566. DOI: https://doi.org/10.1109/access.2023.3347169.

14. Marinov M., Valova I., Kalmukov Y. Comparative Analysis of Existing Similarity Measures used for Content-based Image Retrieval. *2019 X National Conference with International Participation (ELECTRONICA)*. 2019. P. 1–4. DOI: https://doi.org/10.1109/electronica.2019.8825645.

15. Shi M., Zhu H., Helleseth T. The Connections Among Hamming Metric, b-Symbol Metric, and r-th Generalized Hamming Metric. *IEEE Transactions on Information Theory*. 2023. Vol. 69. P. 2485–2493. DOI: https://doi.org/10.1109/tit.2023.3239543.

16. Ahmad P.H., Rai M. Analysis of Optimization Strategies for Big Data Storage Management: A Study. *2023 4th International Conference on Electronics and Sustainable Communication Systems (ICESC)*. 2023. P. 1747–1753. DOI: https://doi.org/10.1109/icesc57686.2023.10193738.

17. Gosain A., Singh J. Conceptual Multidimensional Modeling for Data Warehouses: A Surve. *Proceedings of the 3rd International Conference on Frontiers of Intelligent Computing: Theory and Applications (FICTA)*. 2015. P. 305–316. DOI: https://doi.org/10.1007/978-3-319-11933-5_33.

18. Pandey D. Pandey K. Improvement of Searching Methodology for Network Applications using Bloom Filters and Hashing. *2020 Sixth International Conference on Parallel, Distributed and Grid Computing (PDGC)*. 2020. P. 511–514. DOI: https://doi.org/10.1109/pdgc50313.2020.9315753.

19. Zhang Y. Similarity Image Retrieval Model based on Local Feature Fusion and Deep Metric Learning. *2020 IEEE 5th Information Technology and Mechatronics Engineering Conference (ITOEC)*. 2020. P. 563–566. DOI: https://doi.org/10.1109/itoec49072.2020.9141871.

20. OpenMMLab. URL: https://github.com/open-mmlab (accessed 03.08.2025).

21. Haystack Introduction. URL: https://docs.haystack.deepset.ai/docs/intro (accessed 03.08.2025).

22. MLflow: A Tool for Managing the Machine Learning Lifecycle. URL: https://mlflow.org/docs/latest/ml/ (accessed 03.08.2025).

23. Wang Y., Zhao X., Xu T., Wu X. AutoField: Automating Feature Selection in Deep Recommender Systems. *arXiv*. 2022. DOI: https://doi.org/10.48550/arXiv.2204.09078.

24. Salmani Pour Avval S., Eskue N.D., Groves R.M., Yaghoubi V. Systematic review on neural architecture search. *Artificial Intelligence Review*. 2025. Vol. 58. DOI: https://doi.org/10.1007/s10462-024-11058-w.

25. Lux M., Chatzichristofis S.A.,Lire: lucene image retrieval. *Proceedings of the 16th ACM International Conference on Multimedia*. 2008. P. 1085–1088. DOI: https://doi.org/10.1145/1459359.1459577.

26. facebookresearch/faiss: A library for efficient similarity search and clustering of dense vectors. URL: https://github.com/facebookresearch/faiss (accessed 03.08.2025).

27. Long F., Zhang H., Feng D.D. Fundamentals of Content-Based Image Retrieval. *Signals and Communication Technology*. 2003. P. 1–26. DOI: https://doi.org/10.1007/978-3-662-05300-3_1.

28. COCO, Common Objects in Context. URL: https://cocodataset.org/#home (accessed 05.08.2025).

| | | |
|---|---|---|
| **Stanislav Danylenko**<br>**Станіслав Даниленко** | PhD student, Software Engineering Department, Kharkiv National University of Radio Electronics, Kharkiv, Ukraine.<br>e-mail: stanislav.danylenko@nure.ua,<br>https://orcid.org/0000-0002-8142-3018 | аспірант кафедри Програмної інженерії, Харківський національний університет радіоелектроніки, Харків, Україна. |
| **Serhii Smelyakov**<br>**Сергій Смеляков** | Doctor of Mathematics, Professor of Software Engineering Department, Kharkiv National University of Radio Electronics, Kharkiv, Ukraine,<br>e-mail: serhii.smeliakov@nure.ua<br>https://orcid.org/0000-0002-5791-2479 | доктор фізико-математичних наук, професор кафедри Програмної інженерії, Харківський національний університет радіоелектроніки, Харків, Україна. |