CBRkit: An Intuitive Case-Based Reasoning Toolkit for Python*

Mirko Lenz[1,2][0000−0002−7720−0436], Lukas Malburg[1,2][0000−0002−6866−0799], and Ralph Bergmann[1,2][0000−0002−5515−7158]

[1] Trier University, Universitätsring 15, 54296 Trier, Germany
info@mirko-lenz.de, {malburgl,bergmann}@uni-trier.de
[2] German Research Center for Artificial Intelligence (DFKI)
Branch Trier University, Behringstraße 21, 54296 Trier, Germany
{mirko.lenz,lukas.malburg,ralph.bergmann}@dfki.de

Abstract. Developing Case-Based Reasoning (CBR) applications is a complex and demanding task that requires a lot of experience and a deep understanding of users. Additionally, current CBR frameworks are not as usable as Machine Learning (ML) frameworks that can be deployed with only a few lines of code. To address these problems and allow users to easily build hybrid Artificial Intelligence (AI) systems by combining CBR with techniques such as ML, we present the CBRkit library in this paper. CBRkit is a Python-based framework that provides generic and easily extensible functions to simplify the creation of CBR applications with advanced similarity measures and case representations. The framework is available from GitHub and PyPI under the permissive MIT license. An initial user study indicates that it is easily possible even for non-CBR experts and users who only have limited Python programming skills to develop their own customized CBR application.

Keywords: Case-Based Reasoning · Machine Learning · Hybrid AI · CBR Frameworks · Python Library

1 Introduction

Building Case-Based Reasoning (CBR) [1,17] applications is a demanding and complex task. In addition to the fact that a lot of expertise and knowledge is required to encode cases, similarity measures, possible adaptations, and vocabulary, it is also a challenge to use a CBR framework itself. This is particularly relevant when teaching novices, such as students, the use of CBR with the help of frameworks. With the recent rise in Machine Learning (ML) applications, Python has become an increasingly popular choice due to its simple yet expressive syntax combined with the wide selection of libraries such as NumPy and PyTorch. However, we observe a lack of Python-based CBR frameworks that are easy to use and powerful enough to suit more advanced use cases. This disparity

* The final authenticated publication is available online at https://doi.org/10.1007/978-3-031-63646-2_19

leads to a situation where CBR researchers cannot experiment with up-to-date ML approaches without resorting to complex workarounds, such as using web requests to integrate custom models. At the same time, data scientists and ML engineers are missing out on features that most CBR systems naturally provide, including complex case representations and similarity measures.

In an effort to make ML techniques easier to use by CBR researchers and expand the CBR community to more users, including students and data scientists, we present the CBRkit library. This Python-based toolkit aims to reduce the effort needed to build a case-based application and instead allows the user to focus on solving the problem at hand. CBRkit is available from PyPI and can be easily installed via pip without requiring any further configuration. The library allows for simple import of existing case bases in various formats, provides an extensive set of similarity measures, and allows the definition of custom functionality by a fully typed and generic Application Programming Interface (API). In the first version of CBRkit, we focus on the retrieval step in the CBR cycle—that is, finding the most relevant cases given some query from a user. This paper contains the following contributions: (i) An analysis of existing Python-based CBR libraries/frameworks, (ii) a high-level overview of the CBRkit framework and usage examples, and (iii) a user study comparing the proposed CBRkit with the established Java-based ProCAKE [4] framework.

The remainder of this paper is structured as follows: Section 2 introduces the integration of CBR and ML, followed by a discussion of related work in Section 3. Section 4 introduced the CBRkit framework that is evaluated by a user study in Section 5. Finally, Section 6 concludes the paper and presents future work.

2 Foundations

Recently, ML frameworks are widely used in academia and practice because they are easy to use. Particularly in research, hybrid Artificial Intelligence (AI) systems combining ML with other AI techniques are gaining importance, limiting, for example, the need for a large amount of qualitative data for ML or for increasing the accuracy of the complete hybrid AI system. One possibility of creating a hybrid AI system is to combine ML and CBR. In the following, we describe how CBR can benefit from the integration of ML and vice versa.

Integrating ML in CBR. As CBR is an analogy-based reasoning method [1], it inherently relies on a suitable definition of a vocabulary and the corresponding similarity measures based on the defined vocabulary. Consequently, CBR applications benefit from the knowledge defined in similarity measures during inference, as they are specially tailored to certain attributes and their types [3]. In addition, the similarity assessment is more precise because it is based on the local-global principle, which states that the similarity between cases is calculated based on the similarity of individual attributes (i. e., local similarities), and subsequently the overall similarity is calculated by an aggregation function (i. e., global similarity) [1]. Using this principle, it is possible to individually decide which similarity measures should be used and applied for certain

attributes. ML can be used in this context to automatically learn similarity measures [19,20,8,9] or weights for local similarities during the retain phase. These structure-aware measures can be expensive to compute, which is why the MAC/-FAC *(many are called, but few are chosen)* pattern [7] may be used to discard irrelevant cases early in the retrieval phase. Here, ML could be used as the backbone of the MAC phase—for instance, through a custom classifier that predicts the similarity values of the FAC phase. Another possible integration of ML into the CBR methodology is the use of methods from Natural Language Processing (NLP) [2] in the context of textual CBR—for instance, by computing the similarity of textual attributes with embeddings [5]. With Large Language Models (LLMs) becoming more advanced, it also becomes possible to integrate them in other stages such as the reuse step [10], leading to the development of hybrid CBR applications.

Integrating CBR in ML. The revival of using ML is mainly driven by the high data availability in the context of Big Data today. Based on that data, a suitable ML model can be trained and used in an application. For this purpose, several ML frameworks such as PyTorch or Scikit-learn are publicly available and can be easily used. However, only a few methods are available for using analogy-based reasoning techniques in these frameworks—for instance, Scikit-learn only provides simple nearest-neighbor algorithms for regression tasks. In this context, the ML community could profit from using CBR in ML for having more sophisticated techniques in analogy-based reasoning. One such technique that goes beyond the use of classic nearest-neighbor algorithms is the application of adaptation methods in CBR by which ML can significantly benefit. Furthermore, the retrieval techniques of CBR can be combined with Retrieval-Augmented Generation (RAG) [11] to improve the capabilities of generative LLMs. Finally, CBR may be part of Explainable Artificial Intelligence (XAI) approaches—for instance, to help explain black-box models like deep neural networks [16,21].

3 Requirements and Related Work

Before presenting our proposed CBRkit library, we first present requirements that must be satisfied by a Python-based CBR framework to be easy to use and applicable in the use cases sketched in Section 2. For this purpose, we analyze existing Python-based CBR frameworks in the context of these requirements.

3.1 Requirements

Due to the lack of literature, we systematically derived requirements that should be satisfied by a Python-based CBR library in brainstorming sessions. We also searched for existing Python libraries and collected available features to complete our requirements list. In the following, we first present two general requirements (GR) in the context of software engineering principles, followed by three requirements (R) for CBR software and its integration with ML.

GR1 (Software Engineering)**.** The software should be available from PyPI to enable easy installation and use type annotations to make the code more maintainable. Its correctness should be ensured by a comprehensive test suite, and documentation to help users get started should be available.

GR2 (Interoperability)**.** The software should be interoperable with other systems by providing some API. In addition to programmatic access—enabling the use in other Python projects—there should be a Representational State Transfer (REST) API—simplifying the integration in existing systems.

R1 (High-Level Abstractions)**.** The software should provide built-in functionality for common CBR tasks, such as predefined similarity measures and case representations. Furthermore, the complete CBR cycle should be available in high-level function calls—for instance, through generic retrieval functions that allow for the combination of sequential retrievals such as MAC/FAC, generic adaptation functions that are applied automatically, or generic retain methods that adapt weights and similarities measures through ML techniques.

R2 (Low-Level Customization)**.** In addition to high-level functions, the software should enable advanced use cases through custom extensions—for instance, through abstract interfaces for similarity measures and the ability to use complex case representations with validation logic. It should be possible to combine such extensions with the built-in ones in the same application.

R3 (Integration with ML Frameworks)**.** The software should allow easy integration with established ML frameworks, such as NumPy, pandas, and PyTorch. It should be possible to combine established CBR measures such as taxonomies with ML-based ones such as embeddings, depending on the case representation. The framework should provide a programmatic interface that allows deep integration with ML libraries—for instance, by accepting data types such as NumPy arrays and using similar function signatures.

3.2 Analysis of Related Work

The selection and evaluation of the CBR frameworks is based on the review of GitHub and PyPI for CBR frameworks.[3]. In addition, we include Python-based CBR frameworks from the study of Schultheis et al. [18] in our analysis.[4] The results are summarized in Table 1 and described in the following. A detailed analysis of CBRkit is provided in Section 4.2.

Clood CBR. The cloud-based CBR framework Clood CBR [15] can be used to develop textual and structural CBR applications. It provides several built-in similarity measures (e. g., for strings, numerics, and categorical values). In addition, it supports the complete CBR cycle with generic methods—for instance,

[3] PyArgCBR (`https://github.com/jaumejordan/pyargcbr`) is not considered as it is not actively maintained, lacks documentation, and specializes in argumentation.

[4] With our goal of integrating CBR and ML, tools in other languages are excluded.

Table 1: Requirement analysis of existing Python-based CBR frameworks with ✓ meaning *fulfilled*, ✓/✗ meaning *partially fulfilled*, and ✗ meaning *not fulfilled*.

Framework	**Engineering** (GR1)	**Interoperability** (GR2)	**Abstraction** (R1)	**Customization** (R2)	**ML** (R3)
Clood CBR	✓/✗	✓/✗	✓	✓/✗	✓/✗
pycbr	✗	✓/✗	✓/✗	✓	✓/✗
cbrlib	✓/✗	✓/✗	✓/✗	✓	✓/✗
CBRkit	✓	✓	✓/✗	✓	✓

a MAC/FAC retriever is available (R1). Creating customized case representations is possible, but support for custom similarity measures and generic retrieval pipelines is lacking (R2). Some ML frameworks like sentence-transformers are natively integrated with the application, but adding new ones is not straightforward (R3). Documentation is available on a dedicated website, but the code does not use type annotations, and tests are only available for the frontend part of the application (GR1). For accessing the framework, a REST API is available. Being optimized for the cloud use case, the library does not include support for programmatic use and is also not available from PyPI (GR2).

pycbr. The library pycbr[5] describes itself as a microframework for CBR applications and, as such, is built for microservice architectures. It provides a set of predefined similarity measures for common data types such as strings and numbers and includes ways to import data, but does not support retrieval pipelines and is limited to the retrieval step (R1). The pycbr library exposes an abstract method for custom similarity measures / attribute types (R2). The framework integrates with some ML libraries (e. g., nltk and Scikit-learn), but more advanced libraries such as transformers are missing, and its programmatic interface is based mainly on classes (R3). The source code does not use type annotations, the documentation must be built manually, and end-to-end tests are missing (GR1). Developed as a microframework, pycbr offers a REST API for interoperability, but programmatic usage is not a focus (GR2).

cbrlib. There exist two versions with a major difference in their overall design: While cbrlib v1 uses a class-based interface with a relatively verbose syntax, cbrlib v2[6] has a functional interface with a more concise syntax. The release of v2 was in March 2024—three months *after* the first public version of CBRkit. With v1 not receiving updates since 2021, we will focus on v2 in our analysis.

The framework includes some similarity measures, but they are limited in scope (e. g., no specialized functions for strings), and the user is responsible for loading the case base. The library is focused on the retrieval step and provides no straightforward way to implement the MAC/FAC pattern (R1). However, it provides an interface for defining custom similarity measures (R2). Native integration with ML frameworks is not provided, but the functional approach leads to a programmatic interface that should be familiar to data scientists (R3).

[5] `https://github.com/dih5/pycbr`
[6] `https://github.com/cdein/cbrlib`

The code uses type annotations and is tested, but no documentation is available except for a few examples. The framework makes extensive use of the Python method `functools.partial` to parameterize the built-in measures, which may be less intuitive for inexperienced Python users (GR1). A REST API is not available, but the library can be used programmatically (GR2).

4 CBRkit Framework

Based on our requirements and the analysis of existing Python-based CBR systems, we developed the CBRkit framework. It has been engineered from the ground up for modern Python features and is designed to be easy to use, yet flexible enough to support advanced use cases. CBRkit uses a *compositional* and *functional* approach of declaring similarity measures, which means that complex ones can be built from simple building blocks. The source code is publicly available on GitHub under the MIT license with releases published to PyPI.[7] In the following section, we first provide an overview of the main components of CBRkit followed by an analysis of the requirements defined in Section 3.

4.1 Conceptual Overview

The overall goal of CBRkit is to provide a Python-based framework that simplifies the development of CBR applications. In the first version of the framework, we focus on the retrieval step and, thus, address three main aspects: (i) The representation of cases and queries, (ii) the definition of similarity measures, and (iii) the retrieval of similar cases. These fundamental building blocks will eventually allow for the implementation of more advanced features, such as a generic interface for adaptation techniques.

In the following, we demonstrate how CBRkit can be used to retrieve similar cases based on a query. As an application scenario, we use the cars domain from previous work [13]. The case base consists of millions of cars and their attributes, such as paint color, mileage, and manufacturer. Due to the large number of cases, the MAC/FAC pattern [7] is used. In the following, we provide a fully working example in Listing 1, containing both simple and advanced patterns, which are discussed in the following section. It also shows that CBRkit makes it possible to implement a two-stage retrieval with a custom similarity measure in approximately 50 lines of code, regardless of whether the user wants to use simple knowledge-poor similarity measures or knowledge-intensive ones based on taxonomies or ontologies [19].

Case Representation. In CBRkit, a case base is modeled as a dictionary—that is, a collection of key-value pairs. The keys represent the name (or unique identifier) of a case, while the value may have an arbitrary structure. A fundamental assumption of CBRkit is that the query has the same type as the

[7] `https://github.com/wi2trier/cbrkit` and `https://pypi.org/project/cbrkit`

```python
import cbrkit
import pandas as pd

df = pd.read_csv("casebase.csv")
casebase = cbrkit.loaders.dataframe(df)
query = pd.Series({
  "miles": 20000,
  "year": 2010,
  "model": {"make": "a4", "manufacturer": "audi"},
  "engine": {"fuel": "gas"},
  "paint_color": "red",
})

def miles_similarity(x: int, y: int) -> float:
  if abs(x - y) < 10000:
    return 1.0
  elif abs(x - y) < 25000:
    return 0.7

  return 0.0

mac_sim = cbrkit.sim.attribute_value(
  attributes={
    "paint_color": cbrkit.sim.generic.equality(),
  },
)

fac_sim = cbrkit.sim.attribute_value(
  types={
    str: cbrkit.sim.strings.spacy("en_core_web_lg"),
    int: cbrkit.sim.numbers.linear(max=9999999),
  },
  attributes={
    "miles": miles_similarity,
    "model": cbrkit.sim.attribute_value(
      attributes={
        "make": cbrkit.sim.strings.levenshtein(),
        "manufacturer": cbrkit.sim.strings.taxonomy.load(
          "cars-taxonomy.yaml",
          measure=cbrkit.sim.strings.taxonomy.wu_palmer(),
        ),
      }
    ),
    "engine": cbrkit.sim.attribute_value(
      types={str: cbrkit.sim.strings.openai("text-embedding-3-large")}
    ),
  },
  aggregator=cbrkit.sim.aggregator(
    pooling="mean",
    pooling_weights={"miles": 0.5, "model": 0.3},
    default_pooling_weight=1.0,
  ),
)

mac_retriever = cbrkit.retrieval.build(mac_sim, min_similarity=1.0)
fac_retriever = cbrkit.retrieval.build(fac_sim, limit=10)

result = cbrkit.retrieval.apply(casebase, query, [mac_retriever,
    fac_retriever])
```

Listing 1: Full example of a retrieval with CBRkit. The case base uses an object-oriented representation with nested attributes. The code contains two advanced patterns: A custom similarity function for the attribute miles and the use of multiple retrievers to apply the MAC/FAC pattern.

```
def threshold_sim(threshold: float):
  def wrapped_func(x: int, y: int) -> float:
    return 1.0 if abs(x - y) <= threshold else 0.0

  return wrapped_func
```

Listing 2: Example definition of a similarity function for integers showcasing the functional pattern used in CBRkit.

cases—otherwise, the similarity measure would not be applicable. In the application scenario used, an object-oriented case representation with nested attributes is used, such as the model and engine of a car. The cases are stored in a CSV file that is first loaded into a pandas DataFrame (Line 4) and then converted (Line 5). The query could also be loaded from a file, but is defined directly in the code for simplicity (Line 6). The structure of cases can be validated through an integration with the popular Pydantic library (omitted for brevity).

Similarity Measures. The library provides a set of built-in similarity measures for common data types, such as strings and numbers, but also allows the definition of custom ones. In CBRkit, a similarity measure is defined as a mathematical function f that receives two arguments $x, y \in T$ of the same type T and returns a similarity value between 0 and 1—that is, $f\colon T \times T \to [0, 1]$. To improve efficiency when working with ML models, CBRkit allows for batch processing of multiple values at once. In this case, the function f' expects a list with tuples of values and returns an ordered list of similarity values—that is, $f'\colon \{T \times T, \dots\} \to \{[0, 1], \dots\}$. Implementation-wise, similarity measures are implemented as nested functions, allowing for the parameterization of the measure. An example of a relatively simple numeric measure is shown in Listing 2. Here, the outer function expects a threshold value and returns an inner function that implements the actual similarity measure and thus conforms to the previously defined signature $f\colon T \times T \to [0, 1]$. As a result, the function `threshold_sim` needs to be called twice: (i) to create a similarity measure with a specific threshold and (ii) to actually calculate the similarity between two values.

The same pattern is used in CBRkit to implement seemingly more complex similarity measures, especially the one for attribute-value data types. Since the cars domain uses an object-oriented case representation, this function is used both for the MAC phase (Line 22) and for the FAC phase (Line 28). It allows the user to apply similarity functions based on an attribute's type (Line 29) and/or name (Line 33). Local similarities can be aggregated into a global one using an aggregation function (Line 48). In the given example, the unweighted arithmetic mean is used as an aggregation function in the MAC phase, while a weighted version is applied in the FAC phase. The result of the attribute-value similarity function is a method that complies with the previously defined signature, which means that it can even be used for nested attributes (Line 35). This definition style has the additional advantage of providing visual cues for the structure of the similarity measure, making it easier to understand and maintain.

CBRkit comes with numerous built-in similarity measures, including generic objects, collections, numbers, and strings. With the rise of language models, the latter category has seen numerous changes. Consequently, we include not only *traditional* measures such as Levenshtein distance-based or taxonomy-based ones, but also more recent approaches such as text embeddings from popular NLP libraries like spaCy (Line 30) and OpenAI (Line 44). At the same time, we are aware that it is never possible to cover all available similarity measures, so we provide an easy way to define custom ones: Users simply need to define a function that conforms to the previously defined signature and can be used in the same way as the built-in ones. In the example used, the car dealer wanted to use a non-standard measure to compare the attribute *miles* and thus defined a custom function (Line 14). Note that instead of calling the function in Line 34, it is passed as a reference since it does not contain an inner function.

Retrieval. With both the cases loaded and the similarity measures defined, the final step is to perform the actual retrieval. In our two-stage scenario, we first apply the MAC phase with a minimum similarity of 1.0 (Line 55). In this pre-filter, we only consider cars that have exactly the same paint color that the user is searching for (i. e., red) and enforce this by setting the minimum similarity to 1.0. The FAC retriever is set to return at most ten cars (Line 56) so that the user is not overwhelmed with too many results. The result of the retrieval (Line 58) is an object containing the global similarity scores and the imposed ranking. For advanced use cases, it is also possible to inspect the result of each retriever individually and even obtain the local similarity values for each attribute.

4.2 Requirements Analysis

Having introduced the central concepts of CBRkit through an end-to-end example, we now analyze the framework based on the requirements defined in Section 3.1. The library offers a suite of built-in similarity measures that may be parameterized to fit the user's needs, includes abstractions to implement the MAC/FAC pattern, and also offers methods to load case bases in common data formats. Our library focuses on the retrieval step, which means that the remaining phases need to be implemented manually (R1). Custom similarity functions are first-class citizens in CBRkit and can be easily defined and used in conjunction with built-in ones. By providing multiple retrieval functions, it is possible to build arbitrarily complex pipelines that go beyond the two-stage MAC/FAC pattern (R2). CBRkit is designed to be used in conjunction with established ML frameworks and provides native integrations with pandas, spaCy, sentence-transformers, and other popular libraries. Its functional design resembles the programmatic interface of these frameworks to make it easier for data scientists to start (R3). The code is fully typed and makes extensive use of generics for best-in-class type safety and completion support in modern Integrated Development Environments (IDEs). The library contains a comprehensive test suite—including end-to-end ones—and contains documentation for its public interface. Together with a detailed tutorial, the documentation is available as a static website. CBRkit uses Nix [6] to ensure reproducibility and utilizes a Continuous

Integration (CI) pipeline to automatically publish new releases based on conventional commit messages (GR1). The framework can be consumed through a REST API, a Command Line Interface (CLI), or directly imported into other Python projects, making it interoperable with third-party systems. To simplify deployment, a ready-to-use Docker image is provided (GR2).

5 User Study

While in theory, CBRkit meets the requirements of data scientists, CBR researchers, and students, it is essential to evaluate the usability of the framework in practice. As part of our investigation of related work, we identified a lack of evaluations: In many cases, new tools are only introduced but not tested in a real-world scenario. To address this, we conducted a user study comparing the proposed CBRkit with the established Java-based ProCAKE framework, which is also developed in our department. While a complete evaluation would only be possible once CBRkit has matured and has been used in a variety of projects, this first study tries to assess whether our library is even considered to be a suitable CBR framework by our target audience. After analyzing what requirements are met by ProCAKE, in the following we will describe the experimental setup, present the results, and discuss the implications of the findings.

Among the frameworks discussed by Schultheis et al. [18], ProCAKE has been selected because it has extensive documentation and tests (GR1), and provides a programmatic interface as well as a REST API (GR2). In addition, it includes a suite of similarity measures, but does not provide generic functions for the complete CBR cycle (R1). The MAC/FAC pattern is supported, but support for generic pipelines is missing, and even though complex case representations can be used, custom similarity measures are difficult to integrate, as they must be registered in proprietary classes (R2). Being a Java-based framework, integration with ML frameworks is not straightforward (R3). Choosing our internal framework enables us to recruit people with three different backgrounds: Core developers of ProCAKE, computer science students familiar with ProCAKE, and data science students having (almost) no prior experience with CBR.

5.1 Experimental Setup

The study has been conducted in a controlled environment through a reproducible Virtual Machine (VM) built using Nix. The image contained all the necessary software: A desktop environment, the JetBrains IDEs PyCharm and IntelliJ, and the CBRkit and the ProCAKE libraries with their respective documentation. The study was carried out with v0.6.1 of CBRkit and v5.0 of ProCAKE. The code to generate the image is publicly available on GitHub[8] in an effort to simplify the setup of future user studies in the CBR community. It can be used to create VMs that are compatible with VirtualBox, QEMU/KVM,

[8] `https://github.com/wi2trier/cbr-vm`

and Proxmox. In the study, we used Proxmox to host the VM on a server in our department, which was accessible via Remote Desktop Protocol (RDP).

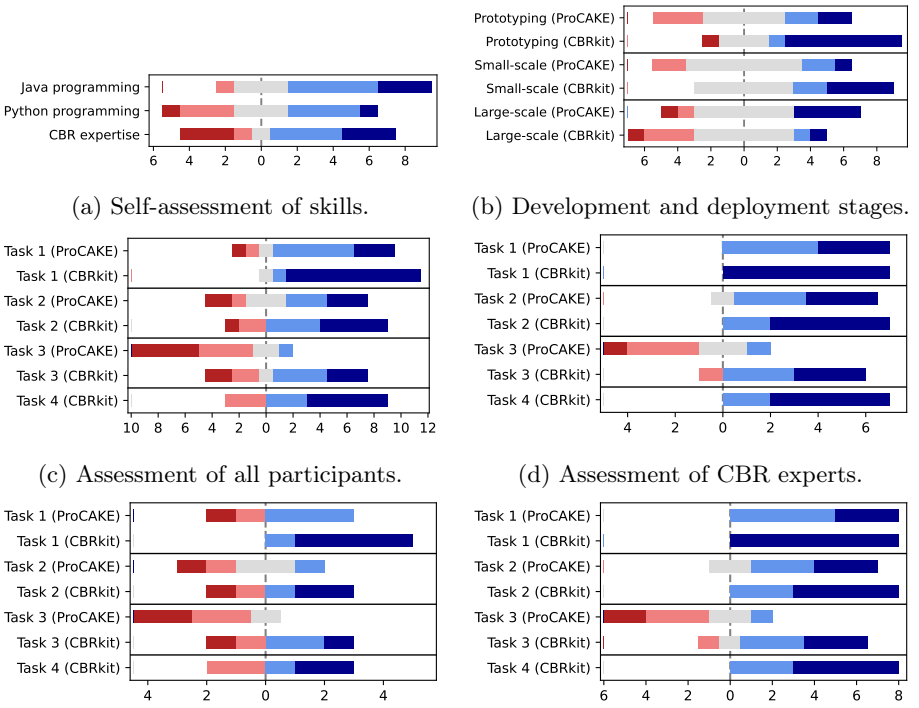
In total, we recruited twelve participants, each receiving a voucher worth 10€ as compensation. The survey was conducted on-site for two days with six participants each. Based on the car domain application scenario, we have specified a set of tasks designed to test the features that both CBRkit and ProCAKE provide. Since CBRkit and ProCAKE are implemented in different programming languages, we provided code templates for both of them in an effort to abstract them from the underlying implementation.[9] There have been three tasks for both frameworks and one additional task that should only be solved with CBRkit: (i) Create a structured query for a car with specific attributes, (ii) implement missing similarity measures for two attributes, (iii) create a retrieval pipeline with a MAC and a FAC retriever, and (iv) implement a custom similarity measure for one attribute (only CBRkit). Since we are dealing with a quite heterogeneous group of participants, we set strict time limits for each task to ensure that all participants had the opportunity to get an impression of both frameworks. Except for the first task (i. e., defining the query), all assignments were designed to be independent of each other so that participants could skip those they had been struggling with. We provided help to those who struggled with the query definition so that they could proceed with the other tasks.

After completing all assignments, the participants were asked to complete a questionnaire. Among others, we asked whether they have been able to solve the tasks, what issues they encountered, and what aspects they liked or disliked about the frameworks. Furthermore, we collected some information about their prior knowledge of Java/Python/CBR. These rating-based questions must be answered on a 5-point Likert scale [12]. Finally, participants could provide free-text feedback on the libraries and the study itself.

5.2 Results and Discussion

The participants' ratings are shown in Figure 1. In general, the participants rated themselves as more proficient in Java than in Python, while the self-assessment of CBR skills tended to be more balanced (Figure 1a). CBRkit has been considered the more viable option for the initial development/prototyping stage and small-scale deployments for individual users or small teams, while ProCAKE has been considered the better choice for large-scale deployments and possibly cloud use cases (Figure 1b). The responses to the four tasks are mixed (Section 5.2): Although the first task has been considered to be easy by most participants, the third has been rated as the most difficult. For all tasks, the scores given to CBRkit are slightly higher than those given to ProCAKE, for task three, the difference is even more pronounced. Here, no user assigned the highest 5 rating to ProCAKE, meaning the distribution of the scores ended up skewed towards the left, while the distribution of the scores for CBRkit has been distributed more

[9] `https://github.com/wi2trier/cbrkit-demo` and `https://gitlab.rlp.net/procake/publications/procake-demos-for-cbrkit-eval-iccbr-2024`



(a) Self-assessment of skills.

(b) Development and deployment stages.

(c) Assessment of all participants.

(d) Assessment of CBR experts.

(e) Assessment of Python developers.

(f) Assessment of Java developers.

Fig. 1: Survey results of Likert scale questions as diverging stacked bar charts centered on neutral responses. Red tones towards the left represent negative scores, whereas blue tones towards the right represent positive scores.

evenly. In addition to the difficulty of the tasks, we also asked the participants about the usefulness of the documentation. We found that the scores are almost identical to the difficulty assessed, indicating that documentation plays a central role in making the frameworks accessible to users and even enabling them to solve more complex tasks. With ProCAKE, five participants needed help with the query definition, while only one participant was unable to solve the task on their own with CBRkit.

To gain additional insights into the results, we also analyzed the responses of the participants based on their self-assessed skills. In the context of this analysis, a participant is deemed an expert if they rated themselves with a 4 or 5 in the respective category. Consequently, there may be some overlap between the groups—for instance, a participant may be equally proficient in Java and Python. When analyzing the ratings of CBR experts (Section 5.2), we still see a slight preference for CBRkit over ProCAKE with a notable difference for task three: Although there is only a single score below 4 for CBRkit, only one participant rated ProCAKE with a 4 or higher. The Java developers' ratings (Section 5.2)

are very similar to those of the CBR experts, with the only difference being that the ratings are slightly lower overall. Python developers (Section 5.2) tend to favor CBRkit even more, with the differences between the systems being more pronounced and no participant rating ProCAKE with a 5 for any task. We also see that familiarity with Python has no direct influence on the ability to implement custom measures (task 4): No CBR expert or Java developer rated the task as difficult, while two Python experts did.

In addition to the Likert scale questions, we also asked the participants about the best and worst aspects of the frameworks and gave them the opportunity to provide free-text feedback. The most common positive aspects of ProCAKE include the Java programming language (namely the type system and the IDE support), the built-in similarity measures, and its modularity. The most common negative aspects have been the large amount of boilerplate code, its overall complexity (e. g., the need for custom classes to declare similarities) and the issues of starting with the framework. The response to the documentation has been mixed, with some participants praising the detailed explanations and others criticizing the lack of examples. Feedback regarding the XML-based configuration has also been twofold: Some liked the clear modeling, but others found it cumbersome to work with.

For CBRkit, the most common strengths include ease of use, the few lines of code required to implement tasks, and its intuitive design. The most common weaknesses have been Python-related issues (e. g., potential performance issues compared to Java and its general syntax) and the smaller scope compared to ProCAKE (e. g., no adaptation support). Similarly to ProCAKE, the documentation received mixed reviews, with praise from some participants and criticism from others. One user without prior CBR experience reported problems with the use of built-in similarity measures and the implementation of custom ones. Consistent with previous findings, one respondent mentioned that they are more proficient in Java, but still found CBRkit easy to handle.

In free-text feedback, one participant noted that custom classes have been used for the domain model in ProCAKE, but not for CBRkit. This is a direct consequence of the way similarities are defined in the two frameworks: In ProCAKE, similarity measures are linked directly to classes, meaning that custom classes are required if one wants to use different measures for objects of the same type. In CBRkit, similarities can also be assigned based on an attribute's name, meaning that custom classes are not necessary. As mentioned in Section 4, CBRkit can be integrated with specialized tools such as Pydantic to add schema validation in the case representation if desired.

The same participant also noted that the simple interfaces for common CBR tasks in CBRkit could in principle be implemented in Java and, thus, ProCAKE as well. We fully agree with this assessment and believe that the functional approach used in CBRkit could serve as a blueprint for future versions of established CBR frameworks. Although there may be certain limitations in Java that make it harder to implement a similar design—mostly due to Java being a compiled language that is centered around classes—such abstract interfaces have

already been successfully implemented in the past (e. g., Keras as a high-level API for TensorFlow). At the same time, such a redesign would not solve the lack of support for Python-based ML frameworks, which is a major reason for the development of CBRkit.

5.3 Limitations

The study has been conducted with a limited number of participants who have been recruited from a single university. Although participants have been selected to represent a wide range of backgrounds, the results may not be generalizable to the broader CBR and ML communities. Furthermore, the study has been based on solving a set of predefined CBR tasks, which may not fully capture the complexity of real-world CBR applications, especially when combined with other ML techniques. Consequently, the results should not be taken as a definitive assessment of the usability of CBRkit and ProCAKE, but rather as a first indication of how the frameworks are perceived by potential users. To gain a more comprehensive understanding of the frameworks, more studies with a larger and more diverse group of participants are needed.

6 Conclusion and Future Work

The CBRkit library proposed in this paper is a simple to use Python-based framework, which enables users to easily combine CBR and ML applications. Current Python-based CBR frameworks are partly cumbersome to use and do not provide full support for users—for instance, missing documentation and interoperability difficulties. CBRkit is based on modern software engineering principles and comes with a comprehensive testing suite and documentation. In an initial user study, we compare CBRkit with the ProCAKE framework, which is also developed in our department. The study indicates that CBRkit enables users to easily build their own CBR applications, even if they only have limited knowledge of CBR or the Python programming language. CBRkit is available from GitHub and PyPI under the permissive MIT license. Moreover, we are open to any kind of contribution to further develop CBRkit.

In future work, we want to investigate how well CBRkit can be used to develop hybrid AI systems. For this purpose, we plan to acquire ML experts and give them CBRkit as an additional tool. In this context, we examine how CBRkit is used in the future and what feedback we receive from the community and users. Another starting point for future work is the implementation of additional functions for the library. We aim to provide functions for the remaining CBR phases—for instance, a reuse method that enables users to easily adapt retrieved cases. In addition, further similarity measures could be added to the library, including those from popular ML libraries like Scikit-learn. This generic adaptation function could be based on our recent work of using adaptation rules in conjunction with rule engines [14]. Finally, a web interface could be provided to allow novices to develop CBR applications.

Acknowledgments. This work was partly funded by the Deutsche Forschungsgemeinschaft (DFG) within the projects *ReCAP* and *ReCAP-II* (№ 375342983, 2018–2024) as part of the priority program *RATIO* (Robust Argumentation Machines, SPP-1999), the *Studienstiftung*, and the Federal Ministry for Economic Affairs and Climate Action within the project *EASY* (№ 01MD22002C).

References

1. Aamodt, A., Plaza, E.: Case-Based Reasoning - Foundational Issues, Methodological Variations, and System Approaches. AI Commun. (Jan 1994). https://doi.org/10.3233/AIC-1994-7104
2. Allen, J.F.: Natural language processing. In: Encyclopedia of Computer Science, pp. 1218–1222. John Wiley and Sons Ltd., GBR (Jan 2003)
3. Bergmann, R., Goos, G., Hartmanis, J., Van Leeuwen, J., Carbonell, J.G., Siekmann, J. (eds.): Experience Management, Lecture Notes in Computer Science, vol. 2432. Springer, Berlin, Heidelberg (2002). https://doi.org/10.1007/3-540-45759-3
4. Bergmann, R., Grumbach, L., Malburg, L., Zeyen, C.: ProCAKE: A Process-Oriented Case-Based Reasoning Framework. In: Kapetanakis, S., Borck, H. (eds.) Workshops Proceedings for the Twenty-seventh International Conference on Case-Based Reasoning. CEUR Workshop Proceedings, vol. 2567, pp. 156–161. CEUR, Otzenhausen, Germany (Sep 2019)
5. Bergmann, R., Lenz, M., Ollinger, S., Pfister, M.: Similarity Measures for Case-Based Retrieval of Natural Language Argument Graphs in Argumentation Machines. In: Barták, R., Brawner, K.W. (eds.) Proceedings of the Thirty-Second International Florida Artificial Intelligence Research Society Conference. pp. 329–334. AAAI Press, Sarasota, Florida, USA (2019-05-19/2019-05-22)
6. Dolstra, E.: The Purely Functional Software Deployment Model. Ph.D. thesis, Utrecht University, Utrecht, The Netherlands (2006)
7. Forbus, K.D., Gentner, D., Law, K.: MAC/FAC - A Model of Similarity-Based Retrieval. Cognitive Science **19**(2), 141–205 (Jan 1995). https://doi.org/10.1207/s15516709cog1902_1
8. Hoffmann, M., Malburg, L., Klein, P., Bergmann, R.: Using Siamese Graph Neural Networks for Similarity-Based Retrieval in Process-Oriented Case-Based Reasoning. In: Watson, I., Weber, R. (eds.) Case-Based Reasoning Research and Development. pp. 229–244. Lecture Notes in Computer Science, Springer International Publishing, Cham (2020). https://doi.org/10.1007/978-3-030-58342-2_15
9. Klein, P., Malburg, L., Bergmann, R.: Learning Workflow Embeddings to Improve the Performance of Similarity-Based Retrieval for Process-Oriented Case-Based Reasoning. In: Bach, K., Marling, C. (eds.) Case-Based Reasoning Research and Development. pp. 188–203. Lecture Notes in Computer Science, Springer International Publishing (2019)
10. Lenz, M., Bergmann, R.: Case-Based Adaptation of Argument Graphs with WordNet and Large Language Models. In: Massie, S., Chakraborti, S. (eds.) Case-Based Reasoning Research and Development. pp. 263–278. Lecture Notes in Computer Science, Springer Nature Switzerland, Cham (2023). https://doi.org/10.1007/978-3-031-40177-0_17

11. Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.t., Rocktäschel, T., Riedel, S., Kiela, D.: Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In: Advances in Neural Information Processing Systems. vol. 33, pp. 9459–9474. Curran Associates, Inc. (2020)
12. Likert, R.: A technique for the measurement of attitudes. Archives of Psychology **22 140**, 55–55 (1932)
13. Malburg, L., Hoffmann, M., Trumm, S., Bergmann, R.: Improving Similarity-Based Retrieval Efficiency by Using Graphic Processing Units in Case-Based Reasoning. In: The International FLAIRS Conference Proceedings. vol. 34. Florida (Apr 2021). https://doi.org/10.32473/flairs.v34i1.128345
14. Malburg, L., Hotz, M., Bergmann, R.: Improving Complex Adaptations in Process-Oriented Case-Based Reasoning by Applying Rule-Based Adaptation. In: Case-Based Reasoning Research and Development - 32nd International Conference, ICCBR 2024, Merida, Mexico, July 1-4, 2024, Proceedings. Lecture Notes in Computer Science, vol. 14775, pp. 50—-66. Springer (2024)
15. Nkisi-Orji, I., Wiratunga, N., Palihawadana, C., Recio-García, J.A., Corsar, D.: Clood CBR: Towards Microservices Oriented Case-Based Reasoning. In: Watson, I., Weber, R. (eds.) Case-Based Reasoning Research and Development. pp. 129–143. Springer International Publishing, Cham (2020). https://doi.org/10.1007/978-3-030-58342-2_9
16. Recio-García, J.A., Parejas-Llanovarced, H., Orozco-del-Castillo, M.G., Brito-Borges, E.E.: A Case-Based Approach for the Selection of Explanation Algorithms in Image Classification. In: Sánchez-Ruiz, A.A., Floyd, M.W. (eds.) Case-Based Reasoning Research and Development. pp. 186–200. Springer International Publishing, Cham (2021). https://doi.org/10.1007/978-3-030-86957-1_13
17. Richter, M.M., Weber, R.: Case-Based Reasoning: A Textbook. Springer-Verlag, Berlin Heidelberg (2013). https://doi.org/10.1007/978-3-642-40167-1
18. Schultheis, A., Zeyen, C., Bergmann, R.: An Overview and Comparison of Case-Based Reasoning Frameworks. In: Massie, S., Chakraborti, S. (eds.) Case-Based Reasoning Research and Development. pp. 327–343. Springer Nature Switzerland, Cham (2023). https://doi.org/10.1007/978-3-031-40177-0_21
19. Stahl, A.: Learning of Knowledge-Intensive Similarity Measures in Case-Based Reasoning. Ph.D. thesis, University of Kaiserslautern, Kaiserslautern (2004)
20. Stahl, A.: Learning Similarity Measures: A Formal View Based on a Generalized CBR Model. In: Muñoz-Ávila, H., Ricci, F. (eds.) Case-Based Reasoning Research and Development. pp. 507–521. Springer, Berlin, Heidelberg (2005). https://doi.org/10.1007/11536406_39
21. Wijekoon, A., Wiratunga, N., Martin, K., Corsar, D., Nkisi-Orji, I., Palihawadana, C., Bridge, D., Pradeep, P., Agudo, B.D., Caro-Martínez, M.: CBR Driven Interactive Explainable AI. In: Massie, S., Chakraborti, S. (eds.) Case-Based Reasoning Research and Development. pp. 169–184. Springer Nature Switzerland, Cham (2023). https://doi.org/10.1007/978-3-031-40177-0_11