Multi-Level AI-Driven Analysis of Software Repository Similarities

Honglin Zhang
School of Computer Science
University of St. Andrews
St Andrews, UK
2376817023zhl@gmail.com

Leyu Zhang
School of Computer Science
University of St. Andrews
St Andrews, UK
847805259@qq.com

Lei Fang
School of Computer Science
University of St. Andrews
St Andrews, UK
lf28@st-andrews.ac.uk

Rosa Filgueira
EPCC
University of Edinburgh
Edinburgh, UK
r.filgueira@epcc.ed.ac.uk

2024 IEEE 20th International Conference on e-Science (e-Science) | 979-8-3503-6561-0/24/$31.00 ©2024 IEEE | DOI: 10.1109/e-Science62913.2024.10678701

***Abstract*—This paper introduces significant enhancements to `RepoSim4Py` and `RepoSnipy`, advanced semantic tools for deep analysis of software repositories. `RepoSim4Py` command-line toolbox now supports multi-level embedding, encompassing code, documentation, requirements, README, and comprehensive repository analysis, which enable the understanding of repository dynamics. Concurrently, `RepoSnipy` web-based search engine facilitates sophisticated repository similarity searches and introduces clustering based on both repository tags (*topic_cluster*) and code embeddings (*code_cluster*). We also introduce `SimilarityCal`, a novel binary classification model trained on these clusters, to predict and quantify repository similarities with high accuracy. These developments provide researchers and developers with powerful tools to navigate the complex landscape of software repositories, improving efficiency in software development and fostering innovation through better reuse of existing resources.**

***Index Terms*—repository similarity, semantic analysis, repository clustering, code understanding, multi-level embeddings, pre-trained language models, GitHub, mining software repositories.**

I. Introduction

The domain of software development is rapidly evolving, influenced by the increasing complexity of codebases and the critical importance of efficient software repository management [1]. As scientific research becomes increasingly intertwined with software, the ability to efficiently mine [2], analyze, compare, and leverage repositories, particularly on platforms like GitHub [3] which hosts a vast and diverse array [4], [5], is becoming indispensable. Advanced tools such as `RepoSim4Py` and `RepoSnipy` not only enhance the findability and accessibility of software projects in alignment with the FAIR principles [6], [7] but also foster a more collaborative and innovative research landscape. This strategic alignment with the FAIR principles promotes efficiency and transparency in software development and supports a more interconnected scientific community.

Building on our prior work [8], this paper introduces enhanced versions of these tools, applying machine learning and deep learning techniques to significantly refine how software repositories are analyzed and compared. `RepoSim4Py`[1] which is a command-line toolbox has been enhanced to now implement multi-level embeddings that provide a comprehensive analysis of repositories, enabling more precise comparisons among Python software repositories by analyzing multi-modal input formats including code, documentation, requirements, and README files.

Meanwhile, `RepoSnipy`[2] web-based search engine, has been significantly improved to facilitate the search for semantic similarity across multiple levels and now includes two novel clustering techniques based on code content and GitHub topic similarities. At the core of these enhancements is `SimilarityCal`, an innovative binary classification model that utilizes *repository-level* embeddings from `RepoSim4Py` to predict and quantify similarity probabilities between repositories. This classifier is offered in two variants: *Code_cluster_sim*, trained on code cluster labels, and *Topic_cluster_sim*, trained on GitHub topic cluster labels. These enhancements empower developers and researchers with robust tools to navigate and utilize the extensive landscapes of software repositories more effectively, enhancing productivity, and promoting effective collaboration.

The remainder of the paper is organized as follows: Section II provides the technological background, Section III and Section IV detail the functionalities of `RepoSim4Py` and the classification model `SimilarityCal`, respectively. Section V discusses repository clustering techniques, Section VI covers the enhanced capabilities of `RepoSnipy`, Section VII reviews related work, and Section VIII concludes with a summary of our findings and future research directions.

II. Background

In the following sections, we look into the background work that is relevant for this paper.

A. `inspect4py`

`inspect4py` [9] is a specialized static code analysis framework for Python software repositories. It performs a comprehensive extraction and analysis of critical repository components, enabling an in-depth understanding of a repository's structure and technical characteristics. This includes parsing source code into Abstract Syntax Trees (ASTs) for detailed code structure, documentation and metadata analysis.

[1]https://huggingface.co/Henry65/RepoSim4Py

[2]https://huggingface.co/spaces/Henry65/RepoSnipy

In the context of RepoSim4Py and RepoSnipy, inspect4py is instrumental in extracting essential components from repositories, which are then utilized to generate multi-level embeddings. This includes code and docstrings of functions and methods, READMEs, requirements, as well as other pertinent details such as licenses, GitHub topics, and stars. Note that GitHub topics are labels that create subject-based connections between GitHub repositories and let developers explore projects by type, technology, and more.

B. UnixCoder Language Model

The UnixCoder [10] model tailors the transformer architecture to understand code by translating ASTs into meaningful embeddings. These capabilities allow UnixCoder to excel in tasks such as code search and clone detection, outperforming predecessors like CodeBERT [11] and GraphCodeBERT [12] by aligning code semantics with documented comments across programming languages.

In RepoSim4Py, UnixCoder is integral for generating detailed embeddings (see Section III) at each layer of repository information (code, documentation, requirements, and README files) ensuring comprehensive analysis of both syntactic and semantic aspects. This effectiveness is achieved using the fine-tuned unixcoder-code-search model[3], which was specifically optimized for code search tasks. Our team previously enhanced this model as part of our ongoing research efforts [8], following the fine-tuning protocols described in[4] and utilizing the AdvTest dataset [13]. Furthermore, our team also fine-tuned a second model *unixcoder-clone-detection* [5], which was also tested in this work.

C. Evolution of RepoSim and RepoSnipy

RepoSim4Py and RepoSnipy have significantly evolved since their initial releases. RepoSim, the predecessor of RepoSim4Py, was originally developed as a command-line tool [6] aimed at measuring semantic similarities between software repositories by analyzing source code and documentation (docstrings), while RepoSnipy was developed as a web-based semantic search engine that built upon the capabilities of RepoSim, as depicted in Figure 1. This tool provided a user-friendly interface for querying GitHub repositories, leveraging the code and docstrings embeddings generated by RepoSim to efficiently identify projects with semantic similarities only at the levels of code and docstrings.

Both tools have been enhanced in this work to improve their analytical depth. RepoSim has been upgraded to RepoSim4Py, which now incorporates a five multi-level embedding framework (see Section III for details). Concurrently, RepoSnipy has expanded its functionality to support advanced levels of repository similarity analysis (see Section VI), including two clustering methods and two novel binary classifier models (detailed in Section IV and Section V).

[3]https://huggingface.co/Lazyhope/unixcoder-nine-advtest

[4]https://github.com/microsoft/CodeBERT/tree/master/UniXcoder/downstream-tasks/code-search\#1-advtest-dataset-2

[5]https://huggingface.co/Lazyhope/unixcoder-clone-detection

[6]https://github.com/RepoAnalysis/RepoSim

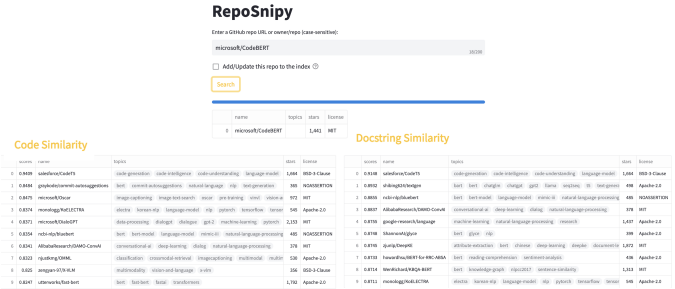


Fig. 1: Previous version of RepoSnipy to find repositories similar to CodeBERT, focusing solely on code and docstrings.

III. RepoSim4Py: Advanced Repository Analysis through Multi-Level Embedding

Building upon its predecessor RepoSim, RepoSim4Py introduces a multi-level approach to embedding generation, enhancing our capability to compare and analyze software repositories across different input data formats. RepoSim4Py operates through the following steps:

1) **Information Extraction:** inspect4py extracts functions, docstrings, README files, and requirements, treating each data type independently.
2) **Embedding Generation:** Each data type (also referred as information level) is processed through the unixcoder-code-search model to generate specific embeddings. See Section III-A.
3) **Embedding Aggregation:** Embeddings from each level are averaged to produce mean embeddings, providing a representation for each component, as shown in Figure 2.
4) **Repository-Level Embedding:** Mean embeddings are concatenated to create a comprehensive *repository-level* embedding, offering a holistic view.
5) **Semantic Similarity Calculation:** Cosine similarity, ranging from -1 (high divergence) to 1 (identical), is used to assess semantic relatedness at each level, including *repository-level* embeddings."

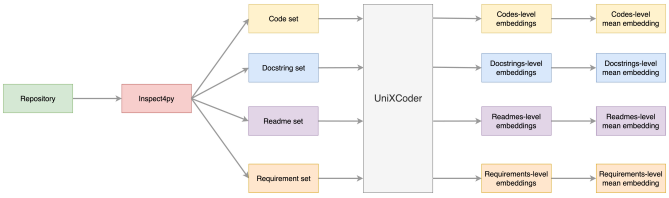


Fig. 2: RepoSim4Py workflow, showing the process from **Information Extraction** to **Embedding Aggregation**.

A. Model Selection Across Information Levels

RepoSim4Py adopts the *Bi-Encoder paradigm* [14] to encode data from multiple repositories independently. This approach facilitates efficient parallel processing while preserving the uniqueness of each information level. In this work, we conducted extensive evaluations of several state-of-the-art language models, including unixcoder-code-search

(introduced in Section II-B), to determine the most effective model for capturing the semantics of software repositories. The selection of models was tailored to the specific nature of the data at each information level:

- **Code:** Models specialized in understanding both syntactic and semantic aspects of programming languages.
- **Docstrings:** Models capable of handling technical documentation.
- **READMEs:** Models suited for longer textual contents.
- **Requirements:** Models that parse and understand lists and structured text.

To evaluate the performance of these models, we utilized the *awesome-python* dataset[7], which comprises 427 unique Python repositories categorized into sectors like *Algorithms, Audio, Authentication, Job Scheduler, etc.* Each repository has been assigned a single category to facilitate meaningful comparisons. If two repositories fall under the same category, their embeddings are expected to show high similarity, with cosine similarity scores approaching one.

We partitioned this dataset into three subsets: 290 repositories for training, 85 for validation, and 52 for testing, generating 90,951 repository pairs for comprehensive performance assessment. We computed cosine similarities between repository pairs (e.g., `lepture/authlib` and `idan/oauthlib`) using different models and subsets.

Each model's performance was measured using ROC (Receiver Operating Characteristic) curves and AUC (Area Under the ROC Curve) metrics [15]. ROC curves plot the true positive rate (TPR) against the false positive rate (FPR) across various thresholds derived from the similarity scores. Repository pairs are classified as positive if the predicted score is above the threshold and negative otherwise. ROC curves are useful for evaluating how well the model can distinguish between repository pairs that belong to the same category and those that do not. The AUC represents the area under the ROC curve; an AUC of 1 indicates perfect classification, while an AUC of 0.5 suggests no discriminative ability.

Table I includes AUC values for each model and information level during both training and testing phases. We use AUC values because they provide a single, easy-to-interpret number that summarizes the model's performance across all thresholds. Notably, only the best-performing models during training were subjected to further testing, ensuring that only the most promising models were evaluated comprehensively.

TABLE I: Models' AUC evaluation across information levels.

Model	Code AUC		Docstrings AUC		READMEs AUC		Requirements AUC	
	Train	*Test*	*Train*	*Test*	*Train*	*Test*	*Train*	*Test*
unixcoder-code-search	**0.84**	**0.90**	**0.80**	**0.93**	**0.77**	**0.88**	**0.64**	**0.81**
Enoch/graphcodebert-py	0.80	–	0.69	–	–	–	–	–
microsoft/unixcoder-base	0.79	–	0.67	–	0.60	–	0.59	–
microsoft/unixcoder-base-nine	0.78	–	0.67	–	0.55	–	0.59	–
unixcoder-clone-detection	0.78	–	0.69	–	0.65	–	0.59	–
Enoch/cocosoda-graphcodebert	0.76	–	0.66	–	–	–	–	–
microsoft/unixcoder-base-unimodal	0.73	–	0.63	–	0.53	–	0.58	–
sentence-transformers/all-mpnet-base-v2	–	–	0.72	–	0.74	–	0.60	–
sentence-transformers/multi-ga-mpnet-base-cos-v1	–	–	0.70	–	0.74	–	0.63	–
sentence-transformers/all-distilroberta-v1	–	–	0.70	–	0.71	–	0.61	–
sentence-transformers/paraphrase-multilingual-mpnet-base-v2	–	–	0.66	–	0.60	–	0.58	–
sentence-transformers/distilbert-base-nli-mean-tokens	–	–	0.59	–	0.53	–	0.56	–
sentence-transformers/bert-base-nli-mean-tokens	–	–	0.59	–	0.53	–	0.56	–
sentence-transformers/distilbert-base-uncased	–	–	–	–	0.52	–	0.57	–

[7]Available at https://github.com/vinta/awesome-python/

As shown in Table I, `unixcoder-code-search` demonstrated the best performance, and therefore it has been selected for creating the embeddings of all levels in `RepoSim4Py`. The summary of `unixcoder-code-search` AUC results is available in Table II, which also includes AUC values for the validation dataset. These results provide a clear view of the model's effectiveness across different types of repository information. These evaluations, along with the instructions to replicate the results are available at [8].

TABLE II: `unixcoder-code-search` AUC values.

Level	Training AUC	Testing AUC	Validation AUC
Code	0.84	0.90	0.85
Docstrings	0.80	0.93	0.84
READMEs	0.77	0.88	0.83
Requirements	0.64	0.81	0.65

B. `RepoSim4Py` Implementation

`RepoSim4Py` offers two implementation options: a HuggingFace pipeline and a command-line tool. The pipeline is available for community use and contributions [9]. Utilizing this pipeline for GitHub Python repositories involves initializing our model `Henry65/RepoSim4Py`, which then generates repository embeddings across all the levels.

Alternatively, the `RepoSim4Py` command-line tool [10] provides a convenient way to analyze repositories. It generates embeddings and computes similarities using cosine measures, ideal for batch tasks.

C. `RepoSim4Py` Evaluation

We conducted an evaluation of `RepoSim4Py` to assess its efficacy in measuring the similarity between software repositories of the *awesome-python* across multiple dimensions. The evaluation focused on both related and unrelated software projects to demonstrate the versatility of our analysis. Our approach involved applying `RepoSim4Py` to all repositories, and comparing the cosine similarity for our five information levels (code, docstrings, READMEs, requirements, overall repository) scores for pairs of repositories.

TABLE III: Cosine similarity scores across multiple levels.

Repo 1	Repo 2	Code Sim.	Doc Sim.	Req Sim.	Readme Sim.	Repo Sim.
lepture/authlib	idan/oauthlib	0.94	0.95	0.54	0.83	0.85
idan/oauthlib	evonove/django-oauth-toolkit	0.92	0.85	0.66	0.86	0.84
lepture/authlib	selwin/python-user-agents	0.30	0	0.45	0.32	0.29
idan/oauthlib	Suor/funcy	0.17	0.07	0.39	0.30	0.17

Table III illustrates the efficacy of `RepoSim4Py` in identifying both the commonalities and distinctions between Python

[8]https://github.com/RepoMining/RepoSim4Py/tree/main/Embedding
[9]https://huggingface.co/Henry65/RepoSim4Py
[10]https://github.com/RepoMining/RepoSim4Py

projects using four pairs of repositories. High similarity scores between `lepture/authlib` and `idan/oauthlib` suggest shared attributes, likely due to their focus on authentication. In contrast, the low scores between `idan/oauthlib` and `Suor/funcy` reflect their divergent aims: authentication versus functional programming functions. For extended analyses, further similarity scores are available in the supplementary material on the project's GitHub [11]. For repositories lacking certain information levels (e.g. `selwin/python-user-agent` does not have docstrings), a zero score indicates the absence of comparable data.

IV. SimilarityCal: Binary Classification Model

While `RepoSim4Py` uses cosine similarity to evaluate linear relationships between repositories, it may not fully capture the complex, nonlinear interactions within and between repositories. To address this, we introduce `SimilarityCal`, a binary classification model that leverages *repository-level* mean embeddings from `RepoSim4Py`. This model applies machine learning to predict the likelihood of conceptual similarity between repositories, offering a more dynamic and adaptable approach for assessing subtler and indirect similarities.

`SimilarityCal` depends on labeled datasets like the `awesome-python`, which categorizes repositories into various predefined groups. By training on these categories, where repositories in the same category are labeled as 'similar' (1) and those in different categories as 'dissimilar' (0), `SimilarityCal` learns to discern complex semantic relationships, enhancing its ability to predict similarities across diverse repository attributes.

A. `SimilarityCal` Architecture

The architecture of `SimilarityCal` (see Figure 3) is structured to process paired repositories through several layers:

1) **Input Layer:** Accepts paired repository embeddings, each formed by concatenating vectors from code, documentation, requirements, and README data into a single 3072-element[12] vector.
2) **Non-Linear Activation** Utilizes ReLU activation [16] to introduce non-linearity, transforming the 3072-dimensional input vector to a higher 3600-dimensional space for each repository.
3) **Batch Normalization:** To ensure stable learning and improve efficiency, the outputs from the ReLU layers are normalized.
4) **Linear Transformation:** This step reduces the dimensionality from 3600 to 1200, focusing on the most significant features for similarity prediction.
5) **Concatenation:** The transformed embeddings from the two repositories are concatenated, giving a comprehensive 2400-dimensional vector that captures the features of both repositories.
6) **Further Non-Linear Processing:** Two additional ReLU layers process the concatenated vector, expanding it to 3000 dimensions to capture intricate feature interactions, then reducing it to 1000 dimensions to focus on the most relevant features for similarity assessment.
7) **Sigmoid Classification Layer:** Outputs a 2-dimensional vector from a sigmoid function [17], representing the probability scores for similarity and dissimilarity between the repositories.
8) **Probability Calculation:** Computes mean probabilities for similarity and dissimilarity.

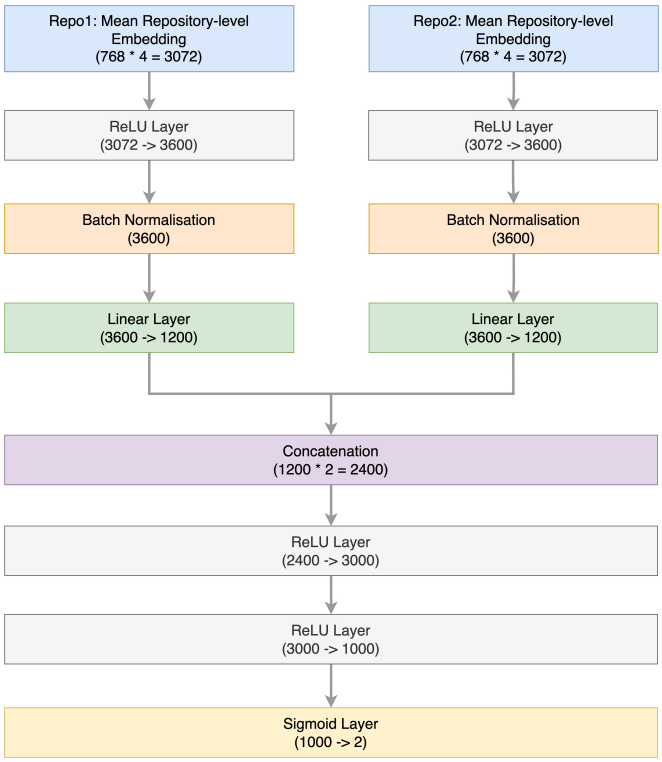


Fig. 3: `SimilarityCal` Architecture.

This architecture is typical of deep learning models [18] used in tasks where high-dimensional data must be compared or related, such as comparing images, texts, or as in our case software repository embeddings. Furthermore, we evaluated two configurations for `SimilarityCal`. **Model No.1**, trained for 300 epochs without weight decay, aimed to maximize the embeddings' predictive power. In contrast, **Model No.2** underwent a shorter 100-epoch training with weight decay to improve generalization and prevent overfitting. Both models shared parameters such as a batch size of 1024, a learning rate of 0.01, the *Adam optimizer*, and *cross entropy* loss. After preliminary evaluations, **Model No.1** outperformed and was chosen for `SimilarityCal`.

B. `SimilarityCal` Evaluation

For the comparative evaluation of `SimilarityCal`, we constructed a dataset from the *awesome-python* collection, forming 207,936 repository pairs, both similar and dissimilar,

[11] https://raw.githubusercontent.com/RepoMining/RepoSim4Py/main/Evaluation/evaluation_results_90951.csv

[12] In the model, the number 768 represents the dimension of embeddings for each data type—code, documentation, etc. The number 4 indicates that four such embeddings are concatenated, resulting in a 3072-dimensional vector.

to mimic real-world conditions. This included self-pairings and randomized cross-pairings to avoid sequence biases. The initial dataset had only 2,242 similar pairs, leading to significant imbalance. We addressed this by downsizing the dissimilar examples, balancing the dataset to 4,484 evenly distributed pairs. We then split this into a training set of 3,586 pairs and a validation set of 898 pairs, each comprising a balanced mix of similarities. This balanced dataset of 4,484 pairs is utilized for all `SimilarityCal` evaluations as detailed in Section IV.

TABLE IV: Comparison of repository similarity using `RepoSim4Py` with and `SimilarityCal` with mean probability.

Repo1	Repo2	Category1	Category2	`RepoSim4Py`	`SimilarityCal`
lepture/authlib	idan/oauthlib	Authentication	Authentication	0.85	0.89
idan/oauthlib	evonove/django-oauth-toolkit	Authentication	Authentication	0.84	0.93
lepture/authlib	evonove/django-oauth-toolkit	Authentication	Authentication	0.80	0.87
lepture/authlib	selwin/python-user-agents	Authentication	Web Crawling	0.29	0.07
idan/oauthlib	Suor/funcy	Authentication	Functional Programming	0.17	0.01

Table IV compares repository similarity for five pairs using two metrics: `RepoSim4Py` (cosine similarity) and `SimilarityCal` (mean probability). Each metric provides different insights; `RepoSim4Py` computes cosine similarity from -1 to 1, indicating degrees of similarity, whereas `SimilarityCal` offers a probability score from a learned model, capturing deeper semantic and structural understanding. Notably, the `RepoSim4Py` column fetches *repository-level* cosine similarity information, while `SimilarityCal` utilizes *repository-level* embeddings. Detailed data on repository pairs and their similarity scores are available at [13].

C. Exploration of Alternative Models

To enhance `SimilarityCal` , we explored a variety of classification models, aiming to assess their effectiveness in analyzing repository similarity.

- **Multi-Class Models**: These models categorize repositories into various classes based on data levels like code, docstrings, READMEs, requirements, and structural information, showcasing the challenges of capturing semantic relationships at these levels.
- **Binary Classification at Different Levels**: Simplifying the task to a binary decision (similar or not), these models focus on individual data levels and improve accuracy over multi-class models though they still struggle to fully capture repository similarities.
- **Recurrent Neural Networks (RNNs) at Different Levels**: Utilizing RNNs [19] for their sequential data processing capabilities, we applied these to single-level embeddings to enhance our understanding of similarity in elements such as code and documentation.
- **RNN at *repository-level***: This approach applies RNNs to combined embeddings of each repository, analyzing them in a sequential context.

[13]https://github.com/RepoMining/SimilarityCal/blob/main/scripts/evaluation_comparison_result.csv and https://github.com/RepoMining/SimilarityCal/blob/main/scripts/SimilarityCal_Evaluation_4484_dataset.csv

TABLE V: Summary of models' evaluations.

Model	Level	Train Acc.	Val. Acc.
Multi-Class	Codes	86%	18.4%
Multi-Class	Docs	78%	13.3%
Multi-Class	READMEs	99%	14.6%
Multi-Class	Reqs	54%	3.2%
Multi-Class	Struct	11%	1.9%
Binary (Single)	Codes	92.1%	71.05%
Binary (Single)	Docs	85.6%	67.04%
Binary (Single)	READMEs	89.7%	57.46%
Binary (Single)	Reqs	81.1%	58.8%
Binary (Single)	Struct	61.2%	53.56%
Binary (Repo-Level)	**Combined**	**99.7%**	**94.21%**
RNN (Single)	Codes	87.8%	81.7%
RNN (Single)	Docs	69.9%	52.34%
RNN (Single)	READMEs	64.8%	55.13%
RNN (Single)	Reqs	61%	54.02%
RNN (Repo-Level)	Combined	99.8%	90.4%

The summary of our model evaluations is presented in Table V. While both the Binary (Repo-Level) and RNN (Repo-Level) models exhibited high training accuracies, the Binary (Repo-Level) model, demonstrated superior validation accuracy at 94.21%, compared to 90.4% for the RNN model. Thereby justifying its selection for integration into `SimilarityCal`. Note that in this evaluation we used the the balanced dataset introduced in Section IV-B.

V. Repository Clustering

The *awesome-python* dataset, while comprehensive, includes only 75 categories for 427 repositories, as we can see in Figure 4. This limited number of labels poses a significant challenge for training classifiers that require a dense and varied label space to achieve high generalization on unseen data. Therefore, in this work, we explored two clustering techniques to organize repositories into unified category labels, which are essential for training the `SimilarityCal` classifier introduced in Section IV.

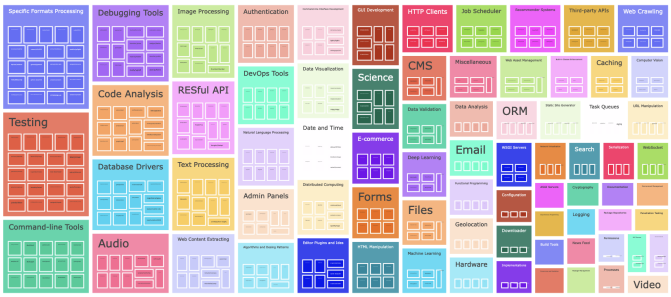


Fig. 4: Representation of the *awesome-python* repositories into 75 categories, with each colored box representing a category and each small box a repository within.

A. Clustering Criteria

Our methodology employs two clustering criteria: GitHub topics and code content. GitHub topics cluster based on repository tags, while code content uses *code-mean* embeddings generated by `RepoSim4Py`. We curated and refined the list of GitHub topics for each repository using

`inspect4py`, removing generic terms such as python' and python3' to ensure specificity. Each topic list was then processed through lemmatization and tokenization to normalize the data, and the cleaned topics were merged into a single string per repository, with terms separated by spaces. This aggregated string served as the input for generating semantic embeddings. For the embedding process, we utilized `SciBERT` (*allenai/scibert_scivocab_uncased*), which is particularly adept at capturing the semantic essence of scientific texts. This method allowed us to create embeddings that reflect the thematic similarities among repositories, thus enhancing our clustering analysis. We evaluated several language models, including `GloVe` (*glove-wiki-gigaword-100*) and `unixcoder-code-search`, to ensure the best fit for our data. `SciBERT` proved to be the most effective, as detailed in our comparative analysis available here [14].

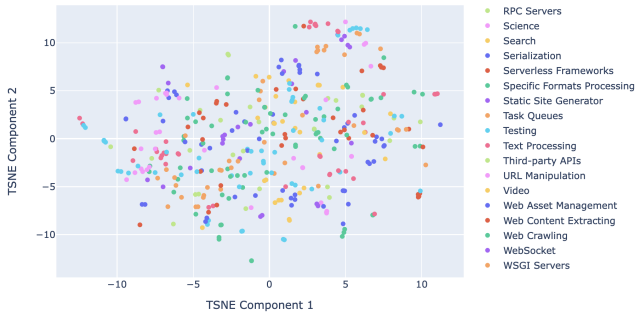


Fig. 5: Visualization of repositories' `RepoSim4Py` code-mean embeddings using a 2D t-SNE projection. Markers represent repositories, while colors indicate categories.

Figure 5 illustrates how repositories classified under the same category (see Figure 4) exhibit proximity in the code's embedding space, indicating similar code characteristics and justifying the clustering approach based on *code content*. This visualization demonstrates the effectiveness of our clustering criteria, revealing cohesive groupings of repositories with similar code characteristics. Similar results were observed when using GitHub topic embeddings, reinforcing the validity of our clustering methodology across different criteria.

B. Clustering Techniques

The Spherical *KMeans* algorithm [20] was selected for clustering due to its effectiveness in handling large datasets. *KMeans* operates by partitioning data into a predefined number of clusters, optimizing the position of centroids iteratively to minimize intra-cluster variance. Considering the high-dimensional nature of our repository data, we explored two distance metrics [21] and vector normalization: 1) Euclidean distance for original vector data, which measures the straight-line distance between points; 2) Cosine distance for normalized data, ideal for emphasizing direction over magnitude, thereby focusing on the angular differences between vectors.

[14]https://tinyurl.com/35eya6uh

Our experimental setup [15] involved testing various *KMeans* configurations with these metrics to identify the optimal clustering strategy. Notably, using cosine distance with normalized vectors significantly enhanced clustering performance, as demonstrated by key metrics including Normalized Mutual Information(NMI) [22], Silhouette Score, and Consistency Rate. Note that, NMI measures the quality of clustering by comparing the predicted clusters with the true data categories.

TABLE VI: Clustering outcomes using cosine distance with normalized vectors, showing effectiveness in analyzing both code content and GitHub topics

Configuration	NMI	Silhouette	Consistency	K
Code (Cosine dist, Norm)	0.71	0.08	0.97	75
Topics (Cosine dist, Norm)	0.67	0.06	0.96	63

We selected $K = 75$ for code content clustering to align with the 75 pre-existing categories in the *awesome-python* dataset, which encompasses all 427 repositories (see Figure 4). For GitHub topics clustering, we applied $K = 63$, as only 273 of the 427 repositories had complete GitHub topic metadata, which further categorized these repositories into 63 distinct categories. The clustering results, summarized in Table VI, support our choice: utilizing cosine similarity with normalized vectors effectively captures the directional nature of our data, essential for semantic analysis. Consequently, *KMeans* clustering using cosine distance and normalized vectors was chosen for repository categorization.

TABLE VII: Comparison of Manual vs *Code Content KMeans* Clustering for *Recommender Systems* Repositories

Repository	Manual Category	*KMeans* Category
NicolasHug/Surprise	Recommender Systems	Recommender Systems
ibayer/fastFM	Recommender Systems	Recommender Systems
maciejkula/spotlight	Recommender Systems	Recommender Systems
benfred/implicit	Recommender Systems	Recommender Systems
jfkirk/tensorrec	Recommender Systems	Recommender Systems
spotify/annoy	Recommender Systems	–
dmlc/xgboost	Machine Learning	Recommender Systems

Table VII provides as an example, a side-by-side comparison of a manual categorization and the automated *KMeans* clustering based on code content. The manual approach originally identified six repositories within the *Recommender Systems* category. Our automated method mirrors this classification closely, accurately grouping five out of the six repositories, thus underscoring the precision of our *KMeans* clustering technique in capturing thematic consistencies. The sixth repository, `spotify/annoy`, was not categorized as a *Recommender System* by our method, which could be due to its broader applicability beyond recommendation tasks, reflected in its codebase's features. Conversely, `dmlc/xgboost` was manually classified as a *Machine Learning* tool but was

[15]*Code content* KMeans experiments are available https://tinyurl.com/ycxhpt6s, while *GitHub topics* KMeans experiments are available here https://tinyurl.com/mr32yzyh

assigned to *Recommender Systems* by our technique. This categorization may stem from `xgboost's` algorithmic features and its use in recommendation systems, where gradient boosting is a common technique to enhance ranking efficacy.

VI. RepoSnipy: Enhanced Semantic Search and Clustering

RepoSnipy[16], now enhanced, supports advanced semantic search and clustering of GitHub repositories, as shown in Figure 6, searching for repositories similar to `obspy/obspy` [17]. This version expands capabilities to perform similarity searches across multiple levels including code, documentation, READMEs, and requirements. It also introduces clustering based on GitHub topics (*topic_cluster*) and code content (*code_cluster*), as explained in Section V. Additionally, it integrates two `SimilarityCal` models (see Section IV), *Code_cluster_sim* and *Topic_cluster_sim*, which are trained on the outcomes of these clustering methods to predict and assess repository similarities with enhanced precision. This represents a substantial evolution from its predecessor, which primarily utilized cosine similarity for analyzing code and docstring.

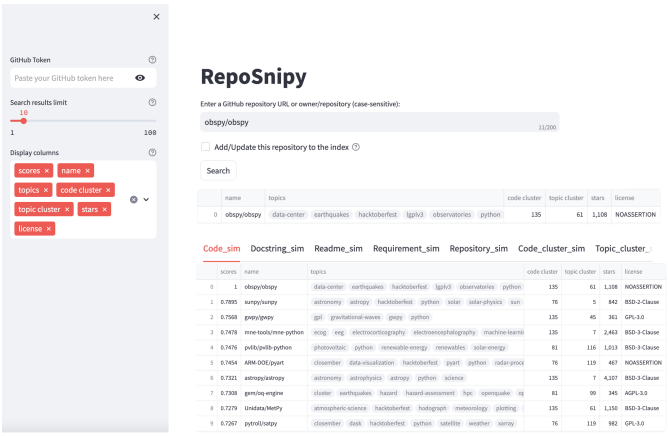


Fig. 6: New RepoSnipy interface showing a query for `obspy/obspy`, showing multi-level similarity capacities.

A. Architecture, Dataset and User Interface

RepoSnipy[18] uses the *streamlit* library [19], which provides an intuitive user interface for interactive querying and visually appealing results display. The backend, powered by the *docarray* framework [20], efficiently retrieves and processes data using the `RepoSim4Py` pipeline, which generates multi-level embeddings from repository code, documentation, READMEs and requirements, as well as from overall repository level. Additionally, it incorporates two `SimilarityCal` models, trained for binary classification of repositories based on *topic_cluster* and *code_cluster*.

[16]https://huggingface.co/spaces/Henry65/RepoSnipy
[17]Python Toolbox for seismology/seismological observatories.
[18]https://github.com/RepoMining/RepoSnipy
[19]https://streamlit.io
[20]https://docs.docarray.org

The dataset includes around 9678 GitHub Python repositories, each with over 300 stars and valid licenses, spanning a wide range of domains and topics. An additional 10 repositories that the authors are familiar with, were added specifically for tool testing, but those were not included in our evaluations. This dataset available at [21], allows for comprehensive comparisons across a varied repository landscape. Using `inspect4py`, we downloaded all metadata, and with `RepoSim4Py`, we generated embeddings at different levels. `SciBERT` was also used to create embeddings for GitHub topics. All metadata and embeddings for each repository are stored in a *docarray* index, facilitating rapid data retrieval and sophisticated similarity calculations using methods like cosine similarity or the `SimilarityCal` classifier. This robust dataset and architecture empower `RepoSnipy` to deliver relevant search results and manage complex clustering tasks based on repository features.

`RepoSnipy` offers a user-friendly interface that initiates searches by entering the target repository in the format <owner>/<repo>. Users can specify the number of search results (default is 10) and select the columns to display. Upon submission, `RepoSnipy` analyzes data to evaluate semantic similarities and perform repository comparisons within code-based and GitHub topic-based clusters. Results are then displayed as a ranked list based on relevance across various metrics, facilitating comparison and categorization.

B. Multi-Level Similarity Analysis Using `RepoSim4Py`

`RepoSnipy` leverages the `RepoSim4Py` pipeline to enable multi-level semantic search across code, documentation, and other key aspects of repositories. Upon querying a repository like `numpy/numpy` [22], `RepoSnipy` generates a repository card (see Figure 7) displaying its name, similarity scores, topics, and classification into code and topic clusters (refer to Section VI-C), along with star ratings and licensing information obtained with `inspect4py`.

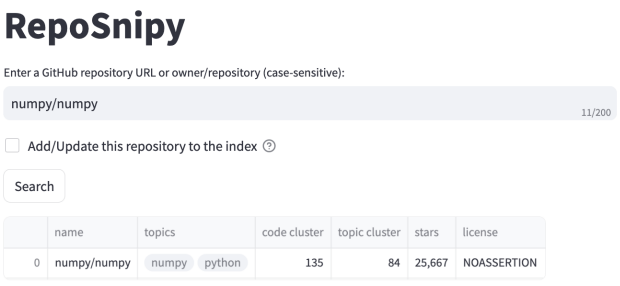


Fig. 7: Repository card for `numpy/numpy`.

Additionally, `RepoSnipy` produces ranked lists for different content levels, ordered by cosine similarity scores where values near 1 indicate stronger relationships. These findings are presented in repository cards, facilitating quick and intuitive comparison of relevance across repositories (Figure 8).

[21]https://tinyurl.com/2s47vkwb
[22]Python package for scientific computing.

	scores	name	topics	code cluster	topic cluster	stars	license
0	1	numpy/numpy	numpy python	135	84	25,667	NOASSERTION
1	0.8805	cupy/cupy	cublas cuda cudnn cupy curand cusolver cusparse cusparselt	23	144	7,532	MIT
2	0.8678	astropy/astropy	astronomy astrophysics astropy python science	135	7	4,107	BSD-3-Clause
3	0.8594	numba/numba	compiler cuda llvm numpy parallel python	132	68	9,223	BSD-2-Clause
4	0.8399	ryfeus/lambda-packs	aws aws-lambda hdf keras lightgbm numpy opencv pandas	23	144	1,105	MIT
5	0.8309	rtosholdings/riptable	analytics dataframes numpy	39	45	342	NOASSERTION
6	0.8192	scipy/scipy	algorithms closember python scientific-computing scipy	7	45	12,177	BSD-3-Clause
7	0.8138	pydata/sparse	numpy python sparse-arrays sparse-data sparse-matrices sparse-ma	23	91	541	BSD-3-Clause
8	0.8117	dask/dask	dask numpy pandas pydata python scikit-learn scipy	39	8	11,941	BSD-3-Clause
9	0.8097	pandas-dev/pandas	alignment data-analysis data-science flexible pandas python	39	9	41,167	BSD-3-Clause

(a) Code-level similarity (Code_sim).

	scores	name	topics	code cluster	topic cluster	stars	license
0	1	numpy/numpy	numpy python	135	84	25,667	NOASSERTION
1	0.8483	cupy/cupy	cublas cuda cudnn cupy curand cusolver cusparse cusparselt	23	144	7,532	MIT
2	0.8465	inducer/pycuda	array cuda gpu gpu-computing multidimensional-arrays pycuda p	23	45	1,700	NOASSERTION
3	0.7979	pydata/sparse	numpy python sparse-arrays sparse-data sparse-matrices sparse-ma	23	91	541	BSD-3-Clause
4	0.791	arrayfire/arrayfire-python	arrayfire cuda gpgpu gpu hpc opencl python python-bindings	23	68	412	BSD-3-Clause
5	0.7659	google/jax	jax	23	89	26,547	Apache-2.0
6	0.7559	unifyai/ivy	abstraction autograd deep-learning gpu ivy jax machine-learning	23	70	13,937	NOASSERTION
7	0.7543	inducer/pyopencl	amd array cuda gpu heterogeneous-parallel-programming multidim	23	72	1,012	NOASSERTION
8	0.7493	aesara-devs/aesara	aesara automatic-differentiation optimizing-compiler optimizing-compiler	23	96	1,141	NOASSERTION
9	0.7388	rtosholdings/riptable	analytics dataframes numpy	39	45	342	NOASSERTION

(b) Docstring-level similarity (Docstring_sim).

	scores	name	topics	code cluster	topic cluster	stars	license
0	1	numpy/numpy	numpy python	135	84	25,667	NOASSERTION
1	0.8019	scipy/scipy	algorithms closember python scientific-computing scipy	7	45	12,177	BSD-3-Clause
2	0.7803	luispedro/mahotas	c-plus-plus computer-vision numpy python python-2 python-3	135	127	818	NOASSERTION
3	0.7769	pydata/bottleneck	c c-extension fast numpy python	23	143	979	BSD-2-Clause
4	0.7546	scikit-hep/uproot3	analysis big-data bigdata file-format hep hep-ex numpy pythor	122	99	315	BSD-3-Clause
5	0.7541	pymatting/pymatting	alpha-matting foreground image-processing python3	117	110	1,663	MIT
6	0.7534	nschloe/perfplot	performance-analysis python	104	60	1,268	GPL-3.0
7	0.7421	spcl/dace	cuda fpga high-level-synthesis high-performance-computing programm	46	45	445	BSD-3-Clause
8	0.7322	catboost/catboost	big-data catboost categorical-features coreml cuda data-mining d	132	8	7,582	Apache-2.0
9	0.7311	fbcotter/pytorch_wavelets	dtcwt filterbank pytorch wavelet wavelet-transform	117	45	817	NOASSERTION

(c) README-level similarity (README_sim).

	scores	name	topics	code cluster	topic cluster	stars	license
0	1	numpy/numpy	numpy python	135	84	25,667	NOASSERTION
1	0.9089	CamDavidsonPilon/lifelines	cox-regression data-science maximum-likelihood python reliability-ana	21	112	2,236	MIT
2	0.8915	oracle/oci-cli	bare-metal cli cloud infrastructure	85	23	389	NOASSERTION
3	0.8881	python/mypy	linter python typechecker types typing	69	96	17,127	NOASSERTION
4	0.8872	ansible/ansible	ansible hacktoberfest python	55	78	60,180	GPL-3.0
5	0.8843	streamlink/streamlink	cli livestream python streaming streaming-services streamlink twi	26	47	9,380	BSD-2-Clause
6	0.883	StackStorm/st2	auto-remediation automation chatops cicd deployment devops if	66	34	5,827	Apache-2.0
7	0.8805	CamDavidsonPilon/lifetimes	data-science python statistics	21	9	1,423	MIT
8	0.8803	pantsbuild/pants	aws-lambda build build-system build-tool build-tools docker gola	16	144	2,999	Apache-2.0
9	0.8801	intel/cve-bin-tool	cve cvss devsecops hacktoberfest python sbom sbom-tool secu	114	31	1,010	GPL-3.0

(d) Requirements-level similarity (Requirements_sim).

	scores	name	topics	code cluster	topic cluster	stars	license
0	1	numpy/numpy	numpy python	135	84	25,667	NOASSERTION
1	0.7723	cupy/cupy	cublas cuda cudnn cupy curand cusolver cusparse cusparselt	23	144	7,532	MIT
2	0.7488	google/jax	jax	23	89	26,547	Apache-2.0
3	0.7479	rtosholdings/riptable	analytics dataframes numpy	39	45	342	NOASSERTION
4	0.7441	aesara-devs/aesara	aesara automatic-differentiation optimizing-compiler optimizing-compiler	23	96	1,141	NOASSERTION
5	0.7226	ryfeus/lambda-packs	aws aws-lambda hdf keras lightgbm numpy opencv pandas	23	144	1,105	MIT
6	0.7059	luispedro/mahotas	c-plus-plus computer-vision numpy python python-2 python-3	135	127	818	NOASSERTION
7	0.6993	arrayfire/arrayfire-python	arrayfire cuda gpgpu gpu hpc opencl python python-bindings	23	68	412	BSD-3-Clause
8	0.695	unifyai/ivy	abstraction autograd deep-learning gpu ivy jax machine-learning	23	70	13,937	NOASSERTION
9	0.6935	ramonhagenaars/nptyping	numpy pandas python3 typehints	80	90	518	MIT

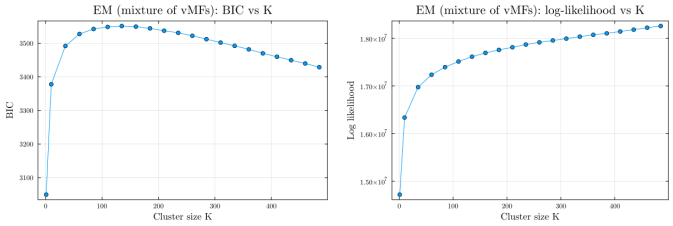
(e) Overall repository-level similarity (Repository_sim).

Fig. 8: `RepoSnipy` similarity analysis for `numpy/numpy`.

Using the `numpy/numpy` repository as an example, `RepoSnipy` showcases its comprehensive similarity analysis across multiple content levels. At the code level, it identifies repositories with similar focuses on numerical operations and Python bindings for C/C++ libraries (Figure 8a). The docstring level reveals correlations with projects emphasizing comprehensive documentation, indicative of wide usage and extensive APIs (Figure 8b). Analysis of README content uncovers repositories presenting their work similarly to `numpy`, potentially indicating shared audiences or related domains of application (Figure 8c). Similarly, requirements similarity highlights repositories sharing common dependencies, reflective of similar environments or technical stacks (Figure 8d). Finally, at the repository level, an aggregate of these aspects provides a holistic view of similarity (Figure 8e). This layered approach allows users to discover repositories aligned with their interests or requirements across different content dimensions, facilitated by `RepoSnipy`'s comprehensive similarity framework.

C. Code and GitHub Topic Clustering

`RepoSnipy` clusters 9678 repositories based on GitHub topics (*topic_cluster*) and code content (*code_cluster*), without using predefined categories. To identify the optimal number of clusters (K), we used a finite mixture of von Mises-Fisher model to fit the embeddings. The model can be viewed as the probabilistic generative model for the spherical K-means algorithm; therefore, metrics such as Bayesian Information Criterion (BIC) can be used to do model selection in a statistically principled way [23]. An Expectation-Maximization (EM) algorithm is used to fit the mixture model and the results are shown in Figure 9. The log-likelihood plot in Figure 9b demonstrates that improvements in model fit diminish beyond K=150. Similarly, BIC scores, displayed in Figure 9a, indicate the most efficient model complexity is achieved around K=150-160. The BIC scores help to choose the best model by penalizing models with more parameters, thus preventing overfitting. Consequently, K=150 is selected as the optimal cluster size, balancing model fit and complexity.



(a) BIC scores vs cluster sizes (b) Log-likelihood vs cluster sizes

Fig. 9: Determining optimal K for repository segmentation.

With K=150 established, we applied clustering methods focused on GitHub topics and code content. The GitHub topics clustering utilized `SciBERT` embeddings as was introduced in Section V-A, while code content clustering employed `RepoSim4Py` mean-code embeddings. Both models used *KMeans* enhanced with cosine distance and normalized vectors to segment the data, resulting in *topic_cluster* and *code_cluster* indices. These indices form the basis for `SimilarityCal` models within `RepoSnipy`, enhancing the system's ability to predict repository similarities.

The clustering outcomes shown in Figure 8 include *code_cluster* and *topic_cluster* classifications, grouping repositories by codebase similarities and thematic content. For example, repositories in the same *code cluster* as `numpy/numpy` (Cluster 135) share parallel computational frameworks or a focus on numeric computation. Among

them, `astropy/astropy` joins `numpy/numpy` due to its array-intensive astronomical computations, demonstrating `RepoSnipy`'s ability to align repositories with diverse applications but shared computational methodologies.

D. Cluster-Based Similarity with `SimilarityCal`

`RepoSnipy` enhances semantic search capabilities with two `SimilarityCal` models: *Code_cluster_sim* and *Topic_cluster_sim*, trained on *code_cluster* and *topic_cluster* frameworks respectively. These models refine repository similarity evaluations by utilizing *repository-level* embeddings from the `RepoSim4Py` pipeline, enabling them to discern both computational and thematic connections among repositories. By classifying repositories within the same cluster as similar (label 1) and those in different clusters as dissimilar (label 0), these models provide quantifiable similarity scores for any repository pair. Once trained, they can predict the likelihood of new repository pairs sharing a cluster, enhancing the process of repository discovery.

Note that the *Code_cluster_sim* model is adept at identifying coding patterns, while the *Topic_cluster_sim* model focuses on thematic resemblances. Together, these models offer a comprehensive view of repository relationships, empowering users to find repositories that match their specific needs.

	name	topics	code cluster	topic cluster	stars	license
0	spotify/luigi	hadoop luigi orchestration-framework python scheduling	4	34	17,092	Apache-2.0

(a) Luigi repository card displaying key metadata.

	rankings	name	topics	code cluster	topic cluster	stars	license
0	1	apache/airflow	airflow apache apache-airflow automation dag data-engineering data-integration data-orchestrator data-pi	4	72	34,197	Apache-2.0
1	2	tensorflow/tfx	apache-beam machine-learning tensorflow	4	13	2,053	Apache-2.0
2	3	logspace-ai/langflow	chatgpt langchain large-language-models react-flow	4	122	14,491	MIT
3	4	databrickslabs/dbx	ci cicd databricks databricks-api databricks-cli mlops	4	5	417	NOASSERTION
4	5	haidaraM/ansible-playbook	ansible ansible-playbook ansible-role grapher graphviz playbook python	4	127	521	GPL-3.0
5	6	theupdateframework/pyth	cncf compromise key python repository revocation security software update	4	21	1,573	Apache-2.0
6	7	studiolm/studio	hacktoberfest	4	78	378	Apache-2.0
7	8	intelowlproject/IntelOwl	cyber-security cyber-threat-intelligence cybersecurity dfir enrichment hacktoberfest honeynet incident-respon	4	74	3,023	AGPL-3.0
8	9	DataBiosphere/toil	aws common-workflow-language cwl gridengine kubernetes mesos pipeline python slurm wdl workfl	4	34	863	Apache-2.0
9	10	spotify/spiff-workflow	bpmn bpmn-engine python spiff-workflow workflow workflow-specification workflowpatterns	4	34	1,565	LGPL-3.0
10	11	argoproj-labs/hera	argo-events argo-workflows cloud-native kubernetes library machine-learning pypi python workflow-autom	4	34	427	Apache-2.0

(b) *Code_cluster_sim:* Similarity ranking based on **code content**

	rankings	name	topics	code cluster	topic cluster	stars	license
0	1	bsc-wdc/compss	c distributed-computing docker hpc java pipeline-framework python singularity slurm workfl	3	34	41	Apache-2.0
1	2	StreamingFlow/d4py	dag data-pipeline dataflow pipeline-framework python3 streaming workflow-engine workflow-man	68	34	1	BSD-2-Claus
2	3	viewflow/viewflow	bpmn django process-engine python workflow workflow-engine	34	34	2,507	AGPL-3.0
3	4	rsalmei/clearly	asynchronous celery distributed flower monitor monitoring-tool python queue real-time realti	124	34	387	MIT
4	5	hjacobs/kube-janitor	cleanup garbage-collector kubernetes kubernetes-operator resource-management ttl	66	34	471	GPL-3.0
5	6	mher/flower	administration asynchronous celery monitoring python rabbitmq redis task-queue workers	26	34	6,047	NOASSERTIO
6	7	Miksus/rocketry	automation automation-framework framework pydantic python scheduler scheduling	139	34	3,135	MIT
7	8	ActiDoo/gamification-engine	engine gamification gamification-engine gamification-framework gamification-platform python rules-e	129	34	409	MIT
8	9	nsacyber/WALKOFF	administration analytics automation automation-framework cybersecurity devops framework integ	11	34	1,183	NOASSERTIO
9	10	celery/celery	amqp python python-library python3 queue-tasks queue-workers queued-jobs redis redis-queue	139	34	22,997	NOASSERTIO
10	11	mpco/AlfredWorkflow-Recent-Documents	alfred3 alfred3-workflow macos recent-files	122	34	349	MIT

(c) *Topic_cluster_sim:* Similarity ranking based on **GitHub topics**

Fig. 10: Similarity analysis for `spotify/Luigi`: (a) repository card; (b) *Code_cluster_sim*; (c) *Topic_cluster_sim.*

Figure 10 illustrates the advanced assessments for the `spotify/luigi` repository[23]. The repository card (10a) provides essential metadata, including classification into specific code (*code cluster*: 4) and topic (*topic cluster*: 34) clusters. In the *Code_cluster_sim* model (10b), `spotify/luigi` aligns closely with repositories emphasizing orchestration and machine learning pipelines, such as `apache/airflow` and `tensorflow/tfx`. Conversely, *Topic_cluster_sim* (10c) draws broader thematic parallels, aligning `spotify/luigi` with diverse workflow-centric repositories.

E. `RepoSnipy` Evaluation

We evaluated `RepoSnipy`'s performance using a similar approach as in the evaluation of `RepoSim4Py`. To assess similarity, we considered shared topics between repositories as a proxy, assuming that repositories with shared topics are more likely to be similar. We conducted evaluations for code, docstring, README, requirements and overall repository similarities, pairing our dataset with each other. For each pair of distinct repositories, we calculated cosine similarity scores based on their code, docstring, READMEs, requirements and overall repository embeddings, yielding 46,836,681 similarity scores across all levels. Additionally, we identified shared topics between repositories (excluding 'python' and 'python3') and used this information as binary similarity labels, resulting in 46,836,681 similarity labels. We then assessed performance using ROC and AUC scores [24] from these labels and scores, summarized in Table VIII. The results indicate that the embeddings from `RepoSnipy`, effectively differentiate between similar and dissimilar repositories.

TABLE VIII: ROC-AUC Scores for different embeddings.

Embedding Type	**ROC Curve AUC**
Code	0.84
Docstring	0.81
README	0.78
Requirements	0.64
Repository	0.83

To evaluate *Code_cluster_sim* and *Topic_cluster_sim*, we partitioned our repository pairs into training and validation sets in a 7:3 ratio. Subsequently, we computed the accuracy and loss metrics[25] for each model using the validation set. The *Topic_cluster_sim* model demonstrated an overall accuracy of 96.40% and an AUC of 0.99, showcasing its robust capability in identifying thematic relationships effectively. Furthermore, with a precision of 0.98, recall of 0.95, and an F1 score of 0.96, the model underscored its precision and reliability in classifying thematic similarities. In contrast, the *Code_cluster_sim* model, which focuses on code-based similarities, exhibited even higher accuracy, recording 99.20%, and an AUC of 1.00. With a precision of 1.00, recall of 0.99, and an F1 score of 0.99, the model demonstrated its performance in identifying code similarities across diverse repositories.

VII. Related Work

This section reviews key advancements in repository analysis. *Repo2vec* [24], which uses the Skip-gram model to create embeddings for Java software repositories, lacks user-friendly tools such as command-line or web interfaces. In contrast,

[23]Python module that helps users to build complex pipelines of batch jobs.

[24]https://github.com/RepoMining/RepoSnipy/tree/main/evaluation

[25]https://github.com/RepoMining/RepoSnipy/blob/main/similarityCal

RepoSim4Py and RepoSnipy offer comprehensive interfaces, including web-based neural search engines, for detailed repository analysis. *RepoGraph* [25] visualizes repository interactions via graph-based methods, focusing on structure rather than content, while RepoSnipy adds value by providing semantic embeddings analysis. *GHTorrent* [26] offers a scalable GitHub data access platform without deep learning analysis tools, a gap filled by RepoSim4Py and RepoSnipy through semantic analysis and clustering. *Ast2Vec* [27] generates code embeddings using Abstract Syntax Trees, primarily focusing on syntactic elements like code duplication, whereas RepoSim4Py and RepoSnipy provide a more nuanced multi-level analysis. Additionally, *SourcererCC* [28], identifies similar code through token-based indexing, while our tools add semantic analysis for deeper insights into code similarity.

VIII. Conclusions and Future Work

This paper presented enhancements to RepoSim4Py and RepoSnipy, advanced tools for analyzing and comparing Python software repositories. By employing multi-level embeddings these tools facilitate a detailed examination of software project dynamics. The SimilarityCal model within RepoSnipy further refines the assessment of repository similarities, providing a platform for developers and researchers to explore and discover valuable software resources, thereby fostering code reuse and efficient software management.

In future works, we aim to improve these tools by integrating more detailed semantic features, such as method-level analysis, to gain deeper insights into coding practices. Expanding the scope to include multiple programming languages would broaden their applicability across various technical ecosystems. Additionally, we plan to explore advanced clustering algorithms that adapt cluster numbers based on repository characteristics for more accurate similarity assessments. Finally, developing a larger, categorized dataset of repositories will help validate and refine our models, ensuring their effectiveness in real-world scenarios.

References

[1] M. A. Heroux, Bernholdt, et al., Basic research needs in the science of scientific software development and use: Investment in software is investment in science (8 2023). doi:10.2172/1846009.

[2] F. Sattler, S. Böhm, P. D. Schubert, N. Siegmund, S. Apel, Seal: Integrating program analysis and repository mining, ACM Trans. Softw. Eng. Methodol. 32 (5) (jul 2023). doi:10.1145/3585008.

[3] E. Escamilla, M. Klein, T. Cooper, et al., The rise of github in scholarly publications, in: Linking Theory and Practice of Digital Libraries, Springer International Publishing, Cham, 2022, pp. 187–200.

[4] G. Robles, M. R. Chaudron, R. Jolak, R. Hebig, A reflection on the impact of model mining from github, Information and Software Technology 164 (2023) 107317.

[5] D. Kang, T. Kang, J. Jang, Papers with code or without code? impact of github repository usability on the diffusion of machine learning research, Information Processing Management 60 (6) (2023) 103477. doi:https://doi.org/10.1016/j.ipm.2023.103477.

[6] M. D. Wilkinson, M. Dumontier, I. J. Aalbersberg, G. Appleton, M. Axton, A. Baak, N. Blomberg, J.-W. Boiten, L. B. da Silva Santos, P. E. Bourne, et al., The fair guiding principles for scientific data management and stewardship, Scientific data 3 (2016).

[7] M. Barker, N. P. Chue Hong, Katz, et al., Introducing the fair principles for research software, Scientific Data 9 (2022). doi:10.1038/s41597-022-01710-x.

[8] Z. Li, R. Filgueira, Mapping the repository landscape: Harnessing similarity with reposim and reposnipy, in: 2023 IEEE 19th International Conference on e-Science (e-Science), 2023, pp. 1–10. doi:10.1109/e-Science58273.2023.10254873.

[9] R. Filgueira, D. Garijo, Inspect4py: A Knowledge Extraction Framework for Python Code Repositories, in: IEEE/ACM 19th International Conference on Mining Software Repositories, MSR 2022, Pittsburgh, PA, USA, May 23-24, 2022, IEEE, 2022, pp. 232–236. doi:10.1145/3524842.3528497.

[10] D. Guo, S. Lu, N. Duan, Y. Wang, M. Zhou, J. Yin, Unixcoder: Unified cross-modal pre-training for code representation, in: S. Muresan, P. Nakov, A. Villavicencio (Eds.), Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022, Association for Computational Linguistics, 2022, pp. 7212–7225. doi:10.18653/v1/2022.acl-long.499.

[11] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, M. Zhou, CodeBERT: A Pre-Trained Model for Programming and Natural Languages (2020). arXiv:2002.08155.

[12] D. Guo, S. Ren, et al., Graphcodebert: Pre-training code representations with data flow (2020). doi:10.48550/ARXIV.2009.08366.

[13] S. Lu, D. Guo, S. Ren, et al., CodeXGLUE: A Machine Learning Benchmark Dataset for Code Understanding and Generation, CoRR abs/2102.04664 (2021).

[14] X. Gu, H. Zhang, S. Kim, Deep code search, in: Proceedings of the 40th International Conference on Software Engineering, ICSE '18, Association for Computing Machinery, New York, NY, USA, 2018, p. 933–944. doi:10.1145/3180155.3180167.

[15] Z. H. Hoo, J. Candlish, D. Teare, What is an roc curve? (2017).

[16] C. Banerjee, T. Mukherjee, E. Pasiliao, An empirical study on generalizations of the relu activation function, in: Proceedings of the 2019 ACM Southeast Conference, ACM SE '19, Association for Computing Machinery, New York, NY, USA, 2019, p. 164–167. doi:10.1145/3299815.3314450.

[17] S. Narayan, The generalized sigmoid activation function: Competitive supervised learning, Information Sciences 99 (1) (1997) 69–82. doi:https://doi.org/10.1016/S0020-0255(96)00200-9.

[18] Y. Alparslan, E. J. Moyer, I. M. Isozaki, et al., Towards searching efficient and accurate neural network architectures in binary classification problems (2021). arXiv:2101.06511.

[19] R. M. Schmidt, Recurrent neural networks (rnns): A gentle introduction and overview (2019). arXiv:1912.05911.

[20] J. A. Hartigan, M. A. Wong, A k-means clustering algorithm, JSTOR: Applied Statistics 28 (1) (1979) 100–108.

[21] J. Wang, Y. Dong, Measurement of text similarity: A survey, Information 11 (9) (2020). doi:10.3390/info11090421.

[22] P. Zhang, Evaluating accuracy of community detection using the relative normalized mutual information, Journal of Statistical Mechanics: Theory and Experiment 2015 (11) (2015) P11006. doi:10.1088/1742-5468/2015/11/p11006.

[23] A. Banerjee, I. S. Dhillon, J. Ghosh, S. Sra, Clustering on the unit hypersphere using von mises-fisher distributions, Journal of Machine Learning Research 6 (46) (2005) 1345–1382.
URL http://jmlr.org/papers/v6/banerjee05a.html

[24] M. O. F. Rokon, P. Yan, R. Islam, M. Faloutsos, Repo2vec: A comprehensive embedding approach for determining repository similarity (2021). doi:10.48550/ARXIV.2107.05112.

[25] C. Williams, R. Filgueira, Repograph: A novel semantic code exploration tool for python repositories based on knowledge graphs and deep learning, in: 19th IEEE International Conference on e-Science, e-Science 2023, Limassol, Cyprus, October 9-13, 2023, IEEE, 2023, pp. 1–10. doi:10.1109/E-SCIENCE58273.2023.10254843.

[26] G. Gousios, D. Spinellis, Ghtorrent: Github's data from a firehose, in: 2012 9th IEEE Working Conference on Mining Software Repositories (MSR), 2012, pp. 12–21. doi:10.1109/MSR.2012.6224294.

[27] B. Paaßen, J. McBroom, B. Jeffries, I. Koprinska, K. Yacef, ast2vec: Utilizing recursive neural encodings of python programs, CoRR abs/2103.11614 (2021). arXiv:2103.11614.

[28] H. Sajnani, V. Saini, J. Svajlenko, C. K. .Roy, C. V. Lopes, Sourcerercc: Scaling code clone detection to big code, in: 38th International Conference on Software Engineering (ICSE 2016), ACM, ACM, Austin, TX, 2016, pp. 1157–1168. doi:10.1145/2884781.2884877.