The Journal of Systems and Software 216 (2024) 112131

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

Data preparation for Deep Learning based Code Smell Detection: A systematic literature review[☆]

Fengji Zhang[a], Zexian Zhang[b,c], Jacky Wai Keung[a], Xiangru Tang[d], Zhen Yang[e], Xiao Yu[b,f,*], Wenhua Hu[b]

[a] Department of Computer Science, City University of Hong Kong, Hong Kong, China
[b] School of Computer Science and Artificial Intelligence, Wuhan University of Technology, Wuhan, China
[c] Sanya Science and Education Innovation Park of Wuhan University of Technology, Sanya, China
[d] School of Engineering & Applied Science, Yale University, New Haven, United States
[e] School of Computer Science and Technology, Shandong University, Tsingtao, China
[f] Wuhan University of Technology Chongqing Research Institute, Chongqing, China

ARTICLE INFO

Keywords:
Code smell detection
Deep learning
Data preparation
Systematic literature review

ABSTRACT

Code Smell Detection (CSD) plays a crucial role in improving software quality and maintainability. And Deep Learning (DL) techniques have emerged as a promising approach for CSD due to their superior performance. However, the effectiveness of DL-based CSD methods heavily relies on the quality of the training data. Despite its importance, little attention has been paid to analyzing the data preparation process. This systematic literature review analyzes the data preparation techniques used in DL-based CSD methods. We identify 36 relevant papers published by December 2023 and provide a thorough analysis of the critical considerations in constructing CSD datasets, including data requirements, collection, labeling, and cleaning. We also summarize seven primary challenges and corresponding solutions in the literature. Finally, we offer actionable recommendations for preparing and accessing high-quality CSD data, emphasizing the importance of data diversity, standardization, and accessibility. This survey provides valuable insights for researchers and practitioners to harness the full potential of DL techniques in CSD.

1. Introduction

Code smell refers to certain symptoms or indications in the source code that suggest there may be underlying problems or potential design flaws (Danphitsanuphan and Suwantada, 2012; Santos et al., 2018; Zakeri-Nasrabadi et al., 2023; Li et al., 2023b). It does not necessarily indicate a functional error or bug in the code but rather highlights programming practices that can impair the maintainability, readability, and extensibility of the software (Di Nucci et al., 2018; Alazba et al., 2023; Hu et al., 2023; Liu et al., 2024). Code Smell Detection (CSD) aims to automatically identify code smells in software source code to ensure code quality, improve software maintainability, and promote good programming practices. Recently, Deep Learning (DL) techniques are gaining popularity in the CSD task (Guo et al., 2019; Kim, 2020; Hamdy and Tazy, 2020). The main advantage of DL models is their ability to automatically encode and learn from raw data, eliminating the need for handcrafted rules and feature engineering presented in previous heuristic-based and machine learning-based CSD methods (Sharma and Spinellis, 2018; Alkharabsheh et al., 2019; Jain and Saha, 2021). Despite the outstanding performance of the DL-based CSD methods, they require a substantial amount of training data to model the complexity of code smells (Di Nucci et al., 2018). High-quality training data plays a key role in the validity of model results. Noisy datasets can hinder effective model training and affect result reliability (Fakhoury et al., 2018; Ardimento et al., 2021a). Data quality could also impact the scalability of models (Allal et al., 2023). Early decisions when constructing CSD datasets, such as the choice of language (Virmajoki et al., 2022; Siddiq et al., 2022) and application scenarios (Zhang et al., 2022; Kaur and Singh, 2023), can affect the scaling and generalization ability of models. Consequently, the availability of high-quality code smell datasets is crucial for building effective DL-based CSD models.

Despite the critical importance of data to CSD models, little attention has been paid to systematically analyzing the CSD data preparation process. Recent literature surveys (Alazba et al., 2023; Naik et al., 2023; Malhotra et al., 2023) comprehensively analyzed DL-based

[☆] Editor: Dr. Hongyu Zhang.

[*] Corresponding author at: School of Computer Science and Artificial Intelligence, Wuhan University of Technology, Wuhan, China.

E-mail addresses: fengji.zhang@my.cityu.edu.hk (F. Zhang), zexianzhang@whut.edu.cn (Z. Zhang), jacky.keung@cityu.edu.hk (J.W. Keung), xiangru.tang@yale.edu (X. Tang), zhenyang@sdu.edu.cn (Z. Yang), xiaoyu@whut.edu.cn (X. Yu), whu10@whut.edu.cn (W. Hu).

https://doi.org/10.1016/j.jss.2024.112131
Received 28 December 2023; Received in revised form 27 May 2024; Accepted 8 June 2024
Available online 12 June 2024

CSD methods. They covered many facets, including code smell types, deep learning techniques, model features, evaluation methods, and the datasets used. They indicated insights on programming language preferences, model effectiveness, and dataset characteristics. However, they overlooked crucial aspects of data preparation, such as requirements, collection, cleaning, and labeling techniques. Consequently, they lack a comprehensive view of data preparation challenges and strategies, limiting researchers and practitioners in harnessing the complete potential of DL techniques in CSD.

To address these gaps, this survey systematically analyzes the existing data preparation processes for DL-based CSD. We identify relevant papers through a Systematic Literature Review (SLR) process, collecting 36 papers on DL-based CSD studies published until December 2023. We then carefully analyze the collected papers concerning data preparation considerations, encountered challenges, and proposed solutions. Additionally, we provide recommendations for preparing and accessing high-quality CSD data. This survey is organized around three main Research Questions (RQs):

RQ1 *What are the critical considerations in constructing CSD datasets?* This research question aims to understand the critical factors researchers consider when building DL-based code smell detection datasets. We analyze papers regarding the four main phases of the established machine learning workflow (Amershi et al., 2019): data requirements, collection, labeling, and cleaning. For data requirements, we examine the programming language, code smell types, and detection scenarios addressed. For data collection, we analyze the data sources and types. For data labeling, we summarize the costs and efficiency of automatic, manual, and semi-automatic approaches. Finally, for data cleaning, we identify issues of code noise and redundancy.

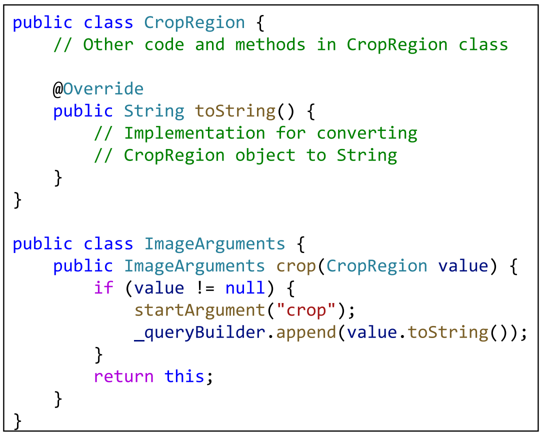
RQ2 *What are the challenges in existing CSD datasets?* This research question identifies challenges that may hinder the performance and reliability of DL-based CSD methods due to data issues. We analyze seven primary challenges: data scarcity, limited generalization, inaccessibility, heavy expert dependency, difficulty in labeling, data imbalance, and redundancy.

RQ3 *What are the solutions presented in the literature?* Given the challenges identified in RQ2, this research question summarizes solutions proposed in the literature. To the challenges addressed, we map five approaches - cross-project datasets, two-phase data utilization, resampling, semi-automatic labeling, and data cleaning methods.

Finally, we provide recommendations based on this survey. Future work should focus on creating more diverse, publicly available datasets that address current limitations. Researchers could leverage multiple programming languages, data sources, and domains to improve generalizability. Semi-automatic labeling and automated real-world data collection may help scale datasets while maintaining quality. Adopting best practices for data governance, including documenting the data collection and pre-processing details, would enhance transparency and reproducibility. Establishing standard criteria to evaluate datasets could help standardize their construction and quality assessment. These efforts aim to generate larger, higher-quality datasets allowing DL-based models to better learn complex code smell patterns across different application scenarios.

The main contributions of this research are:

- Introduce the first systematic review of data preparation processes for DL-based CSD methods (RQ1 in Section 4).
- Provide thorough solutions mapped to identified data challenges to guide future dataset preparation (RQ2 in Section 5 and RQ3 in Section 6).
- Propose recommendations on diversifying and standardizing datasets through multi-language modeling, semi-automatic labeling, and best practices for data governance and accessibility (Section 7).



```
public class CropRegion {
    // Other code and methods in CropRegion class

    @Override
    public String toString() {
        // Implementation for converting
        // CropRegion object to String
    }
}

public class ImageArguments {
    public ImageArguments crop(CropRegion value) {
        if (value != null) {
            startArgument("crop");
            _queryBuilder.append(value.toString());
        }
        return this;
    }
}
```

Fig. 1. An example of *Feature Envy* code smell.

The rest of the paper is organized as follows. Section 2 describes background and related work. Section 3 details our SLR methodology. Sections 4 to 6 presents results addressing each RQ. Section 7 provides recommendations based on findings. Section 8 discusses threats to validity. Section 9 concludes the paper.

2. Related work

This section provides context on code smells, code smell detection techniques, and related CSD SLRs. The background informs the goals and context of our study.

2.1. Code smell detection

Code smells represent undesirable design or implementation constructs that can degrade code maintainability and quality, indicating structural patterns correlated with increased defect risk (Al-Shaaby et al., 2020; Fowler, 2018; Kim, 2017). One example is the *Feature Envy* shown in Fig. 1. In this case, the *crop* method belongs to the *ImageArguments* class but is coupled to the *CropRegion* class. The *CropRegion* class has a *toString()* method that converts the *CropRegion* object to a string. While the *crop* method calls *value.toString()*, which focuses too much on the internal details of the *CropRegion* class about its string conversion logic. Since *crop* is defined in *ImageArguments*, it should not read and manipulate attributes of *CropRegion* rather than its owning class. The *crop* method would be better defined as a method in *CropRegion* since it primarily operates on that class's data rather than *ImageArguments*. This excessive dependency on another class's implementation indicates the presence of *Feature Envy*, which can negatively impact code understandability, modification, and testing.

Early code smell detection approaches are generally heuristic-based or machine learning-based models (Sharma and Spinellis, 2018; Alkharabsheh et al., 2019). However, heuristic-based methods are criticized for their subjectivity due to manually adjusted heuristic rules (Di Nucci et al., 2018). Machine learning-based methods require careful construction and selection of code smell features (Fontana and Zanoni, 2017), which also heavily relies on human expertise.

With advancements in deep learning technology in the field of both artificial intelligence and software engineering (Chen et al., 2023; Yang et al., 2023; Gao et al., 2023; Chen et al., 2020; Ma et al., 2023; Qiao et al., 2023), recent researchers have introduced various deep learning techniques for automatically extracting code smell features from code (Tarwani and Chug, 2022). Model architectures like convolutional neural networks (Fakhoury et al., 2018; Das et al., 2019; Hamdy and Tazy, 2020; Yin et al., 2021), long short-term memory networks (Guo et al., 2019; Wang et al., 2020; Yu et al., 2021; Ardimento et al., 2021a; Li and Zhang, 2022; Siddiq et al., 2022; Ho et al., 2023), and recurrent neural networks (Das et al., 2019; Hamdy and Tazy, 2020; Siddiq et al., 2022) have achieved state-of-the-art CSD accuracy (Zhang et al., 2022).

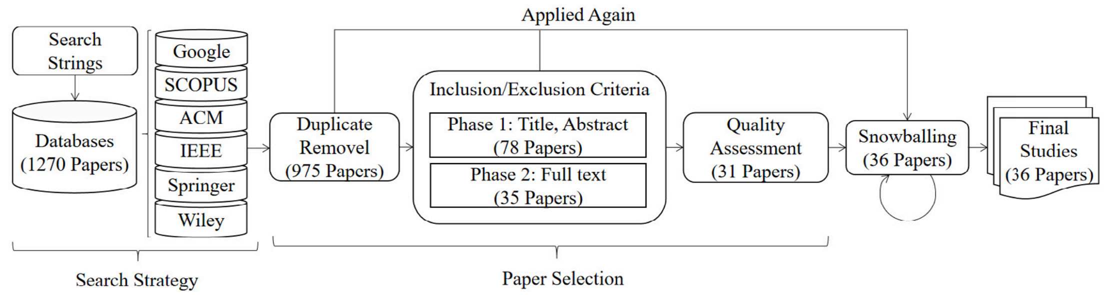

Applied Again
Search Strings
Databases (1270 Papers)
Google
SCOPUS
ACM
IEEE
Springer
Wiley
Duplicate Removel (975 Papers)
Inclusion/Exclusion Criteria
Phase 1: Title, Abstract (78 Papers)
Phase 2: Full text (35 Papers)
Quality Assessment (31 Papers)
Snowballing (36 Papers)
Final Studies (36 Papers)
Search Strategy
Paper Selection

Fig. 2. The overall process of our systematic literature review.

2.2. Related CSD SLRs

Despite data preparation playing a pivotal role in CSD, comprehensive investigations have been lacking. Several surveys in CSD methods explored machine learning and deep learning techniques. While these surveys discussed CSD datasets to varying extents, most of them lacked detailed examination of the data preparation process and systematic identification of challenges or solutions. Specifically, Gupta et al. (2017) delivered a comprehensive review of CSD studies from 1999 to 2016, stressing the pivotal role of code smells in software maintenance. Azeem et al. (2019) and Al-Shaaby et al. (2020) explored machine learning-based CSD studies until 2018, focusing on the types and performance of machine learning techniques. They both found that the random forest emerged as the most effective technique for detecting various code smells and emphasized the significance of manually validating datasets, noting a scarcity of available datasets. Kaur et al. (2021) reviewed CSD studies up to 2020, concentrating on simple and hybrid machine learning techniques and their evaluation methods. They revealed that support vector machine and decision tree algorithms were frequently used by the researchers, and much of the research focused on open-source software. Additionally, they noted that most of the researchers used small and medium-sized datasets and lacked valid industrial datasets. Lewowski and Madeyski (2022) assessed the reproducibility of CSD research from 1999 to 2020, focusing on machine learning-based studies.

Alazba et al. (2023), Naik et al. (2023), and Malhotra et al. (2023) are all devoted to systematically reviewing the research progress in the field of DL-based CSD. They provided a comprehensive survey and summary of the developments in the field from various perspectives such as code smell types, deep learning techniques, datasets, and model performance evaluation. They highlighted supervised learning as the most commonly used learning method and pointed out the importance of models such as convolutional neural networks, recurrent neural networks, and long and short-term memory networks in CSD. In addition, they generally observed the prevalence of Java datasets and that method-level code smell is most often detected. Although these reviews discussed the datasets, there is a certain lack of detailed exploration and systematic analysis of the data preparation stages. They focused more on what type of code smell dataset was used, the programming language of the dataset, and the size of the dataset, while the impact of the dataset preparation process was not examined in depth.

Compared to general CSD surveys, we specifically target the understudied domain of data preparation to advance understanding and inform future practices. There is one particular survey studying CSD datasets (Zakeri-Nasrabadi et al., 2023). They meticulously compared CSD datasets across properties like size, supported smells, programming languages, and construction methods. Their findings highlighted several limitations within existing datasets, notably imbalances in samples, absence of severity levels for smells, and constraints related to Java-based datasets. However, their analysis predominantly focused on machine learning datasets and did not explore the challenges identified in our survey, i.e., *Data Scarcity*, *Limited Generalization Ability*, *Limited Data Accessibility*, *Heavy Expert Dependency*, *Difficulty of Data Labeling*, and *Redundancy*. In addition, they lacked a detailed exploration of solutions or recommendations to address the challenges found, particularly in the context of deep learning applications within CSD. Instead, we address these challenges by proposing diverse solutions. Furthermore, we provide a set of recommendations to advance the field. These recommendations aim to foster progress by addressing critical issues and promoting standardized practices within the domain of DL-based CSD.

3. Research methodology

Our SLR process strictly adheres to established guidelines (Kitchenham, 2004; Zhang et al., 2011) to ensure an objective review. We also adopt a *snowballing* approach (Wohlin, 2014) to include additional literature and enhance the completeness of our review. Fig. 2 outlines our SLR process. The first two authors conduct the work closely with review from the other authors. This process occurred in 2023 and identified 36 relevant papers, as detailed in Table 1.

3.1. Search strategy

To design the search string for our SLR, we utilize the PICO (Population, Intervention, Comparison, Outcomes) framework (Schardt et al., 2007). This framework is widely adopted in systematic reviews to formulate research questions and develop search strategies. PICO helps in breaking down the research topic into four key components:

- Population (P): The population of interest in our study is represented by “Code Smell”.
- Intervention (I): The intervention refers to the “DL Technique” (Deep Learning Technique).
- Comparison (C): Comparison is not applicable in our study, hence it is omitted.
- Outcomes (O): The expected outcomes are related to “Code Smell Detection”.

Table 2 details the key terms and synonyms associated with each PICO component used in our search strategy. We constructed the query by combining these PICO components using the Boolean operator “AND”. This approach ensures a comprehensive search by encompassing a broad spectrum of research related to our topic. We include variants of key terms facilitated through the use of wildcard matching. For instance, the term “detect*” covers “detect”, “detection”, “detecting”, etc. We combine the key terms using the “OR” operator. The detailed search string in the SCOPUS format is as follows:

TITLE-ABS-KEY((“Code smell” OR “Bad smell” OR “Design smell*” OR “Design flaw*” OR “Antipattern*” OR “Model smell*”) AND (“DL Technique” OR “Deep learning” OR “Transfer learning” OR “CNN” OR “RNN” OR “Auto-encoder*” OR “Deep neural network*”) AND (“Code Smell Detection” OR “Detect*” OR “Predict*” OR “Identif*”)*

We adapt the search string as necessary to match each database. The databases queried include Google Scholar, SCOPUS, ACM Digital Library, IEEE Xplore, Springer, and Wiley. These databases are chosen based on recommendations from previous SLRs (Yang et al., 2022; Martínez-Fernández et al., 2022), which highlighted them as

Table 1
The primary studies analyzed in our systematic literature review.

ID	Reference	Title
S1	Kim (2017)	Finding bad code smells with neural network models
S2	Hadj-Kacem and Bouassida (2018)	A hybrid approach to detect code smells using deep learning
S3	Fakhoury et al. (2018)	Keep it simple: Is deep learning good for linguistic smell detection?
S4	Guo et al. (2019)	Deep semantic-based feature envy identification
S5	Barbez et al. (2019)	Deep learning anti-patterns from code metrics history
S6	Liu et al. (2019)	Deep learning based code smell detection
S7	Hadj-Kacem and Bouassida (2019a)	Deep representation learning for code smells detection using variational auto-encoder
S8	Das et al. (2019)	Detecting code smells using deep learning
S9	Hadj-Kacem and Bouassida (2019b)	Improving the identification of code smells by combining structural and semantic information
S10	Hamdy and Tazy (2020)	Deep hybrid features for code smells detection
S11	Wang et al. (2020)	Feature envy detection based on Bi-LSTM with self-attention mechanism
S12	Yu et al. (2021)	A novel tree-based neural network for android code smells detection
S13	Gupta et al. (2021)	An empirical study on predictability of software code smell using deep learning models
S14	Sharma et al. (2021)	Code smell detection by deep direct-learning and transfer-learning
S15	Xu and Zhang (2021)	Multi-granularity code smell detection using deep learning method based on abstract syntax tree
S16	Ren et al. (2021)	Exploiting multi-aspect interactions for god class detection with dataset fine-tuning
S17	Yin et al. (2021)	Local and global feature based explainable feature envy detection
S18	Ardimento et al. (2021a)	Temporal convolutional networks for just-in-time design
S19	Sidhu et al. (2022)	A machine learning approach to software model refactoring
S20	Tarwani and Chug (2022)	Application of deep learning models for code smell prediction
S21	Khleel and Nehéz (2022)	Deep convolutional neural network model for bad code smells detection based on oversampling method
S22	Zhang et al. (2022)	Code smell detection based on deep learning and latent semantic analysis
S23	Yedida and Menzies (2022)	How to improve deep learning for software analytics
S24	Li and Zhang (2022)	Multi-label code smell detection with hybrid model based on deep learning
S25	Dewangan et al. (2022)	Code smell detection using ensemble machine learning algorithms
S26	Zhang and Jia (2022)	Feature envy detection with deep learning and snapshot ensemble
S27	Bhave and Sinha (2022)	Deep multimodal architecture for detection of long parameter list and switch statements using DistilBERT
S28	Ardimento et al. (2021b)	Transfer learning for just-in-time design smells prediction using temporal convolutional networks
S29	Jeevanantham and Jones (2022)	Extension of deep learning based feature envy detection for misplaced fields and methods
S30	Virmajoki et al. (2022)	Detecting code smells with AI: a prototype study
S31	Imam et al. (2022)	The automation of the detection of large class bad smell by using genetic algorithm and deep learning
S32	Siddiq et al. (2022)	An empirical study of code smells in transformer-based code generation techniques
S33	Afrin et al. (2022)	A hybrid approach to investigate anti-pattern from source code
S34	Ho et al. (2023)	Fusion of deep convolutional and LSTM recurrent neural networks for automated detection of code smells
S35	Kaur and Singh (2023)	Improving the quality of open-source software
S36	Liu et al. (2023)	Deep learning based feature envy detection boosted by real-world examples

Table 2
The key terms and synonyms for paper search.

Category	Subject	Search terms
Population	Code Smell	"Bad smell[a]" OR "Design smell[a]" OR "Design flaw[a]" OR "Antipattern[a]" OR "Model smell[a]"
Intervention	DL Technique	"Deep learning" OR "Transfer learning" OR "CNN" OR "RNN" OR "Auto-encoder[a]" OR "Deep neural network[a]"
Comparison	–	–
Outcomes	Code Smell Detection	"Detect[a]" OR "Predict[a]" OR "Identif[a]"

[a] Denotes the wildcard matching pattern.

sources containing high-quality, peer-reviewed research in software engineering.

Furthermore, we apply additional filters to the retrieved papers, including the language (i.e., English-only) and publication status (i.e., The paper should be a peer-reviewed full research paper published in a conference proceeding or a journal). The initial search identifies 1270 papers. We then remove the duplicate records, resulting in 975 papers for further screening.

3.2. Paper selection

We aim to identify high-quality studies that could provide valuable insights into data preparation for DL-based CSD. Additional criteria and quality assessment are applied to screen eligible papers.

3.2.1. Inclusion/exclusion criteria

We propose three inclusion and four exclusion criteria, as shown in Table 3. A paper is only included if it meets all the inclusion criteria and does not conform to any exclusion criteria. The inclusion criteria require that papers utilize deep learning techniques for CSD and propose novel deep learning-based models or solutions for the task. Papers also need to describe the datasets used clearly. For exclusion, papers that use heuristic or machine learning-based detection techniques, review existing models, or only perform statistical/correlational analyses are removed. Papers with unclear descriptions of the datasets are also excluded. To help validate the consistent application of the criteria, the first two authors also conduct an initial screening of 50 randomly selected papers. They independently assess whether each paper meets or does not meet the inclusion and exclusion criteria. The absence of discrepancies between the authors' assessments lends additional confidence in the reliability and validity of the criteria used in this study.

The criteria are applied in two phases. First, the titles and abstracts of retrieved papers are screened according to the criteria. This process leaves 78 papers for potential inclusion. Then, the full texts of the remaining 78 papers are thoroughly reviewed against the criteria. After a full assessment, we have 35 papers that suit the inclusion and exclusion criteria.

3.2.2. Quality assessment

We conduct a quality assessment on the remaining papers using the checklist in Table 4. Quality criteria are essential for assessing the reliability of extracted information, though there is no standardized approach (Kitchenham et al., 2009). Following the Alazba et al. (2023), Croft et al. (2022), and Zakeri-Nasrabadi et al. (2023), our checklist mainly examines independent/dependent variables, validation methods, datasets, and experimental complexity.

One author initially performs the quality assessment, with two additional authors conducting another round of results validation. This process excludes four papers for failing to meet one or more quality

Table 3
The inclusion and exclusion criteria for screening eligible papers.

Inclusion criteria	Exclusion criteria
IC1: The paper proposes new deep learning techniques.	**EC1:** The paper is not in English.
IC2: The paper reports on empirical results.	**EC2:** The paper is a literature review only.
IC3: The paper has undergone peer review.	**EC3:** The full text of the paper is unavailable.
	EC4: The paper provides no dataset(s) details.

Table 4
The quality criteria checklist for screening eligible papers.

Quality criteria
QC1: Are the code smells being detected clearly defined?
QC2: Are the deep learning models sufficiently described?
QC3: Are the performance metrics specified?
QC4: Are the independent and dependent variables clearly defined?
QC5: Are the data sources and statistics fully described?
QC6: Is the data labeling method clearly explained?
QC7: Is the validation methodology specified?
QC8: Are potential threats to validity clearly outlined?

Table 5
The data extraction form for collecting information from reviewed papers.

	Item	Description
Metadata	Study ID	Unique identifier for the paper
	Title	Title of the paper
	Author	Author(s) of the paper
	Year	Year of publication
	References	Number of references in the paper
	Publication	Journal or conference of publication
Datasets	Data source	Real-world or synthetic data
	Dataset name	Name of dataset(s) used
	Multiple datasets	Number of datasets for experiments
	Data integration	How multiple datasets were used
	Availability	Reproducibility of datasets
	Source type	Open-source or exclusive license
	Code smell types	Types of code smells considered
	Programming language	Language of code in datasets
	Size	Number of samples in datasets
	Ratio	Ratio of smelly to non-smelly samples
	Labeling method	Approach to labeling smelly samples
	Required expertise	Expertise needed for labeling
Methods	Data cleaning	Pre-processing approaches
	Transformation	Data representation techniques
	Partitioning	How data was split for training/evaluation
	Resampling	Methods to handle class imbalance
	DL techniques	Proposed deep learning approaches
Others		Additional relevant findings

criteria. Specifically, Lin et al. (2021), Virmajoki (2020), Malathi and Jabez (2023), and Grodniyomchai et al. (2019) lack sufficient descriptions of independent/dependent variables, validation approaches, or datasets used. The details of these papers that do not meet our quality standards are omitted for brevity. This assessment aims to screen papers with incomplete reporting that could limit the extraction of meaningful insights.

3.3. Snowballing

To ensure comprehensive coverage of all relevant literature, we perform manual *snowballing* as per the guidelines by Wohlin (2014). This involves both forward and backward snowballing techniques. Forward snowballing involves examining the citation lists of all papers that meet our inclusion criteria to locate additional relevant papers. Backward snowballing reviews the reference lists to uncover any pertinent studies not previously identified. During the initial round of snowballing, we successfully identify five new relevant papers. Subsequent rounds of snowballing are conducted following the same rigorous screening process, adhering to our predefined inclusion and exclusion criteria and maintaining our quality assessment standards. Despite the additional rounds, no new papers are found that meet our criteria, leading us to conclude that we have reached a saturation point. This is due to the limited scope of current research on DL-based code smell detection, which is a relatively nascent field.

In total, our search and snowballing processes yield 36 papers. Of these, 31 papers are initially retrieved through keyword searches across various databases. The remaining five papers are found through manual snowballing of references and citations. This dual-phase approach helps provide a more comprehensive examination of the literature.

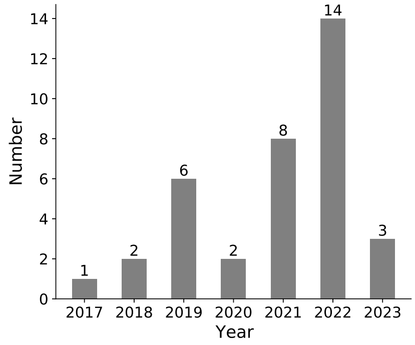


Fig. 3. The number of primary studies by year.

3.4. Data extraction

We have designed a data extraction form to systematically analyze the identified papers, shown in Table 5. The form design is adapted from prior SLR guidelines (Garousi and Felderer, 2017; Kitchenham, 2004) and pilot-tested before finalizing.

The form captures qualitative and quantitative attributes across four aspects: metadata, datasets, methods, and other information. One author performs the initial data extraction to organize information collected from each paper. Then, two additional authors verify the extracted data through independent examination. Any disagreements are resolved through group discussion to reach a consensus. Most data extracted is qualitative, such as deep learning techniques applied, data sources, pre-processing approaches, and code smells addressed. Some quantitative data is also collected, like imbalance ratios within datasets. This formal extraction process aims to investigate the research questions proposed in our study comprehensively. The publication details of the 36 primary studies are listed in Table 6. The number of these 36 papers over the years is shown in Fig. 3.

4. RQ1 - Critical considerations in CSD data preparation

Through a comprehensive literature review, we extract and analyze the various considerations taken to construct datasets for code smell detection. Following the machine learning workflow introduced by Amershi et al. (2019), our data preparation analysis centers around four main phases in Fig. 4, including data requirements, collection, labeling, and cleaning. By thoroughly reviewing these preparation aspects, we clarify current practices and guide practitioners and researchers on effectively addressing critical factors when building datasets. This will help standardize the construction of high-quality datasets for code smell detection.

Table 6
The publication statistics of the primary studies in our SLR. *J* and *C* denote journal and conference publication, respectively. *N.* denotes the number of publications.

Sources	N.	Study
J: IEEE transactions on software engineering	1	S[6]
J: Journal of systems and software	1	S[14]
J: Neurocomputing	1	S[18]
J: Knowledge-based systems	1	S[22]
J: International journal of electrical and computer engineering	1	S[1]
J: Journal of theoretical and applied information technology	1	S[10]
J: International journal of computers and applications	1	S[19]
J: Indonesian journal of electrical engineering and computer science	1	S[21]
J: Applied sciences	1	S[25]
J: International journal of intelligent engineering and systems	1	S[29]
J: Journal of King Saud University-computer and information sciences	1	S[31]
J: Agile software development: Trends, challenges and applications	1	S[35]
C: International conference on software engineering and knowledge engineering	2	S[15, 24]
C: International computer software and applications conference	2	S[16, 17]
C: IEEE international working conference on source code analysis and manipulation	2	S[27, 32]
C: ACM joint european software engineering conference and symposium on the foundations of software engineering	1	S[36]
C: International conference on software maintenance and evolution	1	S[5]
C: International conference on software quality, reliability and security	1	S[12]
C: International conference on neural information processing	1	S[9]
C: International conference on evaluation and assessment in software engineering	1	S[34]
C: IEEE international symposium on parallel and distributed processing with applications	1	S[11]
C: Asia-Pacific symposium on internetware	1	S[4]
C: International conference on evaluation of novel approaches to software engineering	1	S[2]
C: International conference on software analysis, evolution and reengineering	1	S[3]
C: International joint conference on neural networks	1	S[7]
C: IEEE region 10 conference	1	S[8]
C: International conference on advanced information networking and applications	1	S[13]
C: International conference on reliability, Infocom technologies and optimization (trends and future directions)	1	S[20]
C: International conference on mining software repositories	1	S[23]
C: International conference on dependable systems and their applications	1	S[26]
C: Jubilee international convention on information, communication and electronic technology	1	S[30]
C: International conference on computer and information technology	1	S[33]
C: International conference on software technologies	1	S[28]

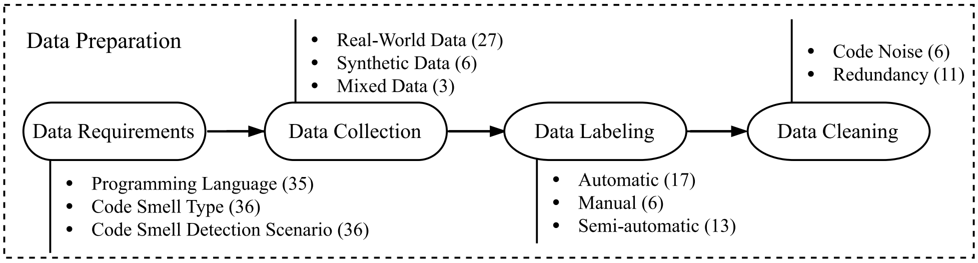


Fig. 4. The critical considerations in CSD data preparation (RQ1). The number of papers for each category is indicated.

4.1. Data requirements

When constructing high-quality datasets to train and evaluate DL-based code smell detection models, it is crucial to determine which programming language code will undergo smell detection and the specific types of code smells to be detected. Moreover, we should also consider the code smell detection scenario, i.e., whether to use within-project or cross-project data to build the datasets. Therefore, three key factors should be considered when preparing datasets for code smell detection research: programming language, code smell type, and code smell detection scenario.

Programming language: The choice of programming language is an essential early decision in dataset construction. Several aspects influence this choice, including the availability of openly accessible code samples and the types of code smells to be studied for that particular language. As depicted in Fig. 5, our analysis shows that the vast majority of papers [S1–12, S14–18, S20–31, S33–36] utilize Java datasets due to the widespread use of the Qualitas Corpus — an open-source collection of Java projects. The higher availability of Java datasets helps accelerate research in this area. Besides, two papers [S14, S34] focus on C# to investigate the feasibility of transfer learning across languages. There is one paper [S32] studying Python datasets and one paper [S19] studying UML datasets, where [S32] specifically examines Python security smells, and [S19] studies the presence of functional decomposition in UML models of object-oriented software. S13 does not provide details on the programming language studied or the dataset used, which is not categorized within this section.

Code smell type: Another critical consideration is the types of code smells. In our study, we categorize the code smells into the *Class* level, the *Method* level, and the code smells that are relevant to *Both* levels. The *Class* level code smells typically pertain to the design and structure of entire classes, which concerns class-level refactoring or redesign. The *Method* level code smells are usually related to the internal implementation and behavior of the methods or functions, which concerns refactoring or decomposition of the methods. The *Both* level represents code smells that may result in both class- and function-level refactoring. We list the details of the *Method* level code smells, the *Class* level, and the *Both* level code smells in Table 7.

As can be seen, certain code smell types have garnered considerable attention from researchers, with four prevalent types covered by ten or

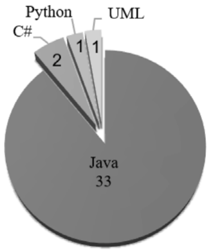


Fig. 5. The frequency of programming languages addressed in analyzed primary studies.

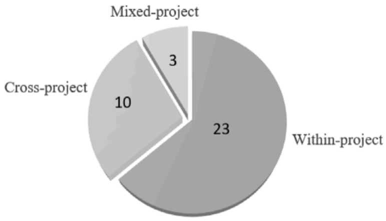


Fig. 6. The distribution of code smell detection scenarios in analyzed primary studies.

more papers. Among them, *Feature Envy* is the most investigated code smell, which denotes methods that access the data of another object rather than its data. Another prevalent code smell *God Class*[1] means classes that have many members and implement different behaviors. The *Long Method* code smell refers to the methods that are too long and contain too much code logic. And *Data Class* means a class containing only data fields and methods. The prevalence of these code smell types can be attributed to their distinctiveness, ease of identification, and relatively large number of samples in real-world codebases.

However, some code smell types have not received enough attention. One of them is *Type checking*, which means frequently using type checking to determine the type of an object. Another example is *Dummy handler*, an exception handler that only logs an error message without taking any meaningful corrective actions. Notably, there has been a recent effort (Tarwani and Chug, 2022) studying these code smells to broaden the scope of experimental research and enhance empirical investigations in these domains.

Code smell detection scenario: There are three main CSD scenarios. First is within-project detection, splitting a project into the training and testing data with no intersection. The second is cross-project detection. Contrary to the previous, the training and testing data come from different projects. This approach solves the problem of lacking enough training data from a single project. The third is mixed-project detection, which utilizes mixed data from multiple projects to get the training and testing data partitions. In this way, it can create enough data for training and evaluation. We categorize the reviewed papers based on their CSD scenario and draw a pie chart in Fig. 6. We can find that 23 papers [S1–4, S7, S9, S12–13, S15, S17–23, S25–27, S29–30, S32–33] belong to the within-project scenario; Ten papers [S5–6, S8, S11, S14, S16, S28, S34–36] belong to the cross-project scenario; And three papers [S10, S24, S31] belong to the mixed-project scenario. Within-project detection is the most popular practice for training and testing CSD models. There is a scarcity of papers using mixed-project datasets because unifying the feature extraction from different projects is still an open challenge.

4.2. Data collection

The primary considerations during data collection vary based on the data source, which we categorize as real-world, synthetic, or mixed.

Real-world data: Most studies [S1–5, S7–9, S12–15, S18–22, S24–25, S27–28, S30, S32–36] utilize real-world data by collecting open-source projects/repositories or using existing datasets. Real-world data is the best testbed for validating CSD techniques in practical applications. The most commonly used dataset is Qualitas (Tempero et al., 2010), which contains many Java open-source projects. It is used by five papers [S2, S4, S21, S25, S27]. Other well-processed corpora are constructed using multi-lingual source code from Github, Bitbucket, Apache, etc. Examples include LandFill (Palomba et al., 2015) [S7, S9], MUSE (Yu et al., 2021) [S12], MLCQ (Madeyski and Lewowski, 2020) [S30], CodeXGlue (Lu et al., 2021) [S32], and the Benchmark (Sharma et al., 2021) [S14, S34]. Furthermore, there is also a study [S19] investigating alternative data type, using the Img2UML (Karasneh and Chaudron, 2013) corpus, which consists of XMI file of the UML class models parsed from images. Utilizing these real-world datasets is crucial in CSD research as they provide valuable insights into the complexities and challenges faced in practical software development.

Synthetic data: In the context of CSD, researchers need to generate synthetic data to tackle challenges like insufficient real-world code smell samples or severe data imbalance. We identify six papers [S6, S11, S17, S23, S26, S29] using synthetic methods to overcome these challenges. They usually follow a unified process for synthesizing the needed data. The first is to collect usable code snippets. The second is to assess whether each code snippet can be transformed into a code smell. The final step is to generate positive and negative samples using the identified code snippets in the second step. Negative samples are the unchanged code snippets. Positive samples are artificially altered fragments of the original code to make it smelly. For example, to create feature envy smells, they can perform unnecessary move refactoring, moving methods from one class to another (Liu et al., 2019).

Mixed data: Another way to create datasets is to mix real-world and synthetic data. This is typically employed to address the challenge of having limited samples while preserving real-world data distribution (Di Nucci et al., 2018). Our survey identifies three papers [S10, S16, S31] that use mixed data. Specifically, [S10] mixes the real-world data from the Qualitas and synthetic data. [S16] mixes synthetic data created by Liu et al. (2019) with real-world data from the LandFill (Ren et al., 2021). [S31] mixes real-world (Arcelli Fontana et al., 2016; Sousa et al., 2017) and synthetic datasets (Liu et al., 2019) from previous references used. These mixed datasets provide researchers with a valuable resource for conducting experiments that balance real-world complexity's benefits with synthetic data's controlled environment.

4.3. Data labeling

The scale and quality of data labeling significantly influence the results and reliability of empirical studies. We summarize three labeling methods for constructing CSD datasets: automatic, manual, and semi-automatic ways.

Automatic: A common practice of labeling CSD datasets is using automatic tools. Such practice can bring several benefits, including convenience and time efficiency (Liu et al., 2019). 17 papers [S4–6, S8, S10–13, S15, S21–24, S26, S29–30] use automatic tools to label the datasets. One of the frequently used tools is JDeodorant (Tsantalis et al., 2008). It is an Eclipse plug-in that detects code smells in Java software and recommends appropriate refactorings to resolve them. For the moment, the tool supports five code smells, namely *Feature Envy*, *Type/State Checking*, *Long Method*, *God Class*, and *Duplicated Code*. Another example is Checkstyle (Checkstyle, 2013), which is a development tool for Java, which checks many aspects of the source code. It

[1] Also known as the *Blob Class*, which is studied in [S7, S9, S13, S33].

Table 7
The code smell types addressed in analyzed studies.

Code smell type	Description	Study
Method level		
Feature Envy	Methods accessing another object's data.	S[1–2, 4, 6–7, 9, 11, 14–15, 17, 20–21, 23, 25–26, 29–30, 34–36]
Long Method	Methods with excessive code logic.	S[2, 6–7, 9, 13, 20–21, 23–25, 30]
Complex Method	Methods with high cyclomatic complexity.	S[14, 24, 34]
Empty Catch Block	Empty exception catch blocks.	S[15, 20, 24]
Brain Method	Methods concentrating class intelligence excessively.	S[8, 22, 35]
Complex Conditional	Long or complex conditional expressions.	S[14, 24, 34]
Member Ignoring Method	Ordinary methods not accessing member attributes.	S[12–13]
Internal Getter and Setter	Methods accessing properties via get/setters.	S[12–13]
Long Parameter Lists	Methods with excessively long parameter lists.	S[24, 27]
Magic Number	Unexplained numeric literals in expressions.	S[24]
Type checking	Frequent object type checking.	S[20]
Shotgun Surgery	Similar changes in multiple places for requirements.	S[22]
Over logging	Excessive logging causing large log files.	S[20]
No Low Memory Resolver	Lack of proper low memory handling.	S[13]
Nested try statement	Multiple layers of nested try-catch blocks.	S[20]
Linguistic Antipatterns	Poor language in code, comments, or documentation.	S[3]
Exception in finally block	Throwing exceptions within a finally block.	S[20]
Dummy handler	Handling exceptions by just printing error messages.	S[20]
Careles Cleanup	Mishandling exceptions or resource leaks in cleanup.	S[20]
SpaghettiCode	Confusing and intricate code structure.	S[33]
Intensive Coupling	Methods calling too many other member methods.	S[35]
Extensive Coupling	Methods calling scattered member methods.	S[35]
Switch Statements	Heavy use of switch statements.	S[27]
Long Identifier	Excessively long identifiers.	S[24]
Long Statement	Individually lengthy statements.	S[24]
Missing default	Lack of a default case branch in switch statements.	S[24]
Class level		
God Class (Blob Class)	Classes with numerous behaviors.	S[1–2, 5, 7, 9, 10, 13, 16, 20–21, 25, 33, 35]
Data Class	Classes containing only fields and access methods.	S[1–2, 21–22, 25, 35]
Large Class	Classes with many methods and data members.	S[1, 6, 23, 31]
Multifaceted Abstraction	Classes having multiple responsibilities.	S[14, 34]
Misplaced Class	Classes improperly distributed.	S[6, 23]
Leaking Inner Class	Inner classes referencing outer classes.	S[12–13]
Brain Class	Overly complex classes.	S[8, 22]
Swiss Army Knife	Classes using multiple interfaces for functionalities.	S[13, 33]
Functional Decomposition	Class functionality spread across multiple classes.	S[19, 33]
Unprotected main	Core logic in an unprotected main function.	S[20]
Parallel Inheritance Hierarchies	Inheritance tree dependencies.	S[1]
Lazy Class	Classes not performing enough.	S[1]
Insufficient Modularization	Incomplete class decomposition.	S[15]
Deficient Encapsulation	Over-permissive member accessibility.	S[15]
Complex Class	Classes with intricate logic.	S[13]
Schizophrenic Class	Classes with unrelated functions.	S[35]
Refused Parent Bequest	Subclasses resisting parent class methods.	S[35]
Both level		
Design Smell	Poor design choices in software systems.	S[18, 28]
Security Smells	Potential security holes or vulnerabilities in the code.	S[32]

Table 8
The summary of the automatic tool for code smell detection.

Tool	Code smell	Study
JDeodorant (Tsantalis et al., 2008)	Feature Envy, Type/State Checking, Long Method, God Class, Duplicated Code	S[5–6, 11–12, 16, 18, 22, 26, 28, 29, 36]
iPlasma (Marinescu et al., 2005)	Duplicated Code, God Class, Feature Envy, Refused Bequest	S[2, 4, 8, 10, 16, 21, 22, 25]
PMD (PMD, 2017)	Large Class, Long Method, Long Parameter List, Duplicated Code	S[2, 10, 12, 21, 25, 28]
AntiPattern (Wieman, 2011)	Data Class, Feature Envy, Long Method	S[2, 10, 21, 25]
Checkstyle (Checkstyle, 2013)	Large Class, Long Method, Long Parameter List, Duplicated Code	S[12]
UCDetector (Ucdetector, 2008)	Data Class, Large Class, Long Method, Long Parameter List, Message Chains, Refused Bequest, Speculative Generality, Tradition Breaker	S[12]

can find class and method design problems. It also has the ability to check code layout and formatting issues. We provide a summary of all identified automatic tools in Table 8.

Manual: Manual labeling is time-consuming and labor-intensive, demanding substantial human resources and specialized knowledge. However, manual effort is sometimes necessary because humans can easily generalize to different domains and achieve higher reliability. We identify six papers [S1, S7, S9, S17, S19–20] that manually label the datasets. The manual labeling process first requires experts to manually analyze the source code based on various code smells. Secondly, additional experts are required to validate the accuracy of the previously identified smelly samples. Any disputed samples should be reviewed again with respect to code smell definitions, source code, and change

Table 9
The summary of data challenge in CSD.

Challenge	Description	Study
Data Scarcity	· Lack of large, real-world datasets to train DL models.	S[3, 5–6, 14, 16, 22, 36]
	· Synthetic data cannot represent real code smells.	S[6, 11, 16–17]
Limited Generalization Ability	· A single programming language during training hinders model generalization.	S[1, 3, 7, 12, 15, 22, 24–25, 28, 30, 32, 36]
	· Including only a few code smell types in datasets restricts models from detecting other smell types.	S[1, 5–6, 9, 12, 15-18, 22, 25, 30, 32, 35]
	· Using datasets solely from open-source projects limits generalizing to proprietary codebases.	S[2, 7, 15, 18, 28, 35]
Limited Data Accessibility	· Lack of clarity about dataset construction makes it difficult to reproduce.	S[3, 8, 13–15, 22, 24, 28–30]
	· Using private datasets limits independent validation and extension of approaches.	S[6]
Heavy Expert Dependency	· Manual labeling by domain experts is crucial but demanding given the scale of needed datasets.	S[3, 6, 14, 17–18, 26, 34, 36]
Difficulty of Data Labeling	· Labeling is time-consuming and error-prone, with challenges in accuracy for automated labeling and potential noise in human labels.	S[5–6, 18, 24, 36]
Data Imbalance	· Uneven distribution of code smell samples hinders model training.	S[4, 12–16, 21–23, 25, 27–28, 31, 33]
Redundancy	· Duplicate samples inflate dataset sizes without adding value for learning.	S[2, 14–15]
	· Redundant and uninformative features makes model training more difficult.	S[2, 13, 21, 27]

history information to reach a conclusion. Such protocol enhances the reliability and reduces the subjectivity of the labeling process (Palomba et al., 2015).

Semi-automatic: There are 13 papers [S2–3, S14, S16, S18, S25, S27, S31–36] that explore a hybrid approach that combines both methods to address the reliability issue associated with automatic labeling and the labor-intensive nature of manual labeling. Generally, they use automatic tools to label all samples and verify the labeled results by experts. Specifically, a subset of samples regarding code smells is chosen for individual analysis by multiple experts. The experts perform individual analyses without discussing them with others. Then, Cohen's Kappa is calculated to measure inter-expert agreement. Disagreements are discussed to reach a consensus on a final labeled set. Finally, the manually labeled results are compared to the automatic tools' labeled results. If the differences are negligible, the dataset labeled by the automatic tools for all samples is ultimately adopted.

4.4. Data cleaning

The final stage of data preparation is data cleaning. Though not all studies comprehensively address this stage, we identify two prevalent cleaning steps involving code noise and data redundancy.

Code noise: Six papers [S3, S18, S21, S27–28, S33] indicate that code noise may introduce irrelevant or erroneous information that can mislead models. [S21, S27–28] identifies several noise types, including outliers, missing data, and mismatching feature types. Textual noise like blank lines and non-ASCII characters are also found in [S3]. [S18, S27] find that incomplete or erroneous data sessions and non-normalized features can introduce additional noise. Object data types instead of expected numerical types are also identified in one dataset [S33]. Such noise can be removed through pre-processing to improve dataset usability for CSD models, which will be detailed in the subsequent sections of this survey.

Redundancy: Data redundancy refers to identical or highly similar code samples and redundancy features. This adversely impacts analysis and model performance. To mitigate these effects, two papers [S14–15] remove duplicate code samples from identical code files or fragments. Nine papers [S6, S9–10, S13, S18–21, S25] use various feature selection methods to remove redundant features, including:

- Convolutional Neural Networks (CNN): Applied in [S6, S10, S18, S21], CNNs are leveraged for their ability to automatically identify and discard redundant features through generalizing from relevant data.
- Gain Ratio: Utilized in [S9, S21], the gain ratio is an extension of the information gain criterion, which normalizes the information gain by the intrinsic information of a split, making it effective in choosing features that provide the most significant discrimination.
- Cross-Correlation Analysis: As described in Podobnik and Stanley (2007) and used in [S13], this method involves analyzing the cross-correlation function to identify and eliminate features that exhibit high redundancy with other features.
- Chi-Square Test: Employed in [S19, S25], the chi-square test evaluates the independence of features with respect to the target variable, enabling the selection of features that have a statistically significant association with the outcome, thus removing irrelevant or redundant features.
- Information Gain: Applied in [S20], information gain measures the reduction in entropy or uncertainty by partitioning the data according to different features, helping to identify and retain the most informative features.

5. RQ2 - Challenges in CSD data preparation

This section presents seven prominent challenges encountered by researchers during the creation of CSD datasets, including *Data Scarcity*, *Limited Generalization Ability*, *Limited Data Accessibility*, *Heavy Expert Dependency*, *Difficulty of Data Labeling*, *Data Imbalance*, and *Data Redundancy*. Each challenge presents a unique hurdle in the quest for effective DL-based CSD. We briefly summarize these challenges in Table 9.

5.1. Data scarcity

Data scarcity in DL-based CSD refers to the inadequacy of available real-world data for training and testing DL models. Seven papers [S3, S5–6, S14, S16, S22, S36] point out this issue. In cases where the datasets lack sufficient samples, the model may struggle to acquire essential features and patterns, ultimately leading to a decline in performance (Fakhoury et al., 2018; Sharma et al., 2021). For example, [S5] states that their sample size may limit the generalizability of the results and hope further to evaluate the approach on a larger set of systems. In addition, several researchers have opted to use synthetic datasets due to the scarcity of real-world data. These synthetically generated datasets are large in size and low in labor effort. However, four papers [S6, S11, S16–17] argue that these synthesized datasets can threaten the validity of the proposed methods. It is underscored that the generated data could be significantly different from real-world code smells (Yin et al., 2021). The generated smelly samples are essentially different from real-world ones that are often more challenging to identify (Liu et al., 2019).

5.2. Limited generalization ability

The limited generalization ability of CSD models is related to the single programming language of the dataset, the limited number of code smell types, and the choice of data source. Several papers [S1, S3, S7, S12, S15, S22, S24–25, S28, S30, S32, S36] find that the programming language of training data affects model effectiveness. Ramos et al. (2022) also find that Python models have lower performance in transfer learning. This performance degradation becomes even more pronounced when evaluated on datasets containing *switch* statements. In addition, several papers [S1, S5–6, S9, S12, S15–18, S22, S25, S30, S32, S35] have focused on the different code smell types in the datasets. For example, three papers [S12, S22, S35] recognize performance degradation when applying their approach to other code smells. Six papers [S2, S7, S15, S18, S28, S35] indicate that the choice of dataset sources can also limit the generalization ability of models. [S2] points out that using only datasets collected from open-source projects cannot generalize to close-source industrial projects. The narrowed scope of datasets and classification scenarios may lead to model performance degradation in unseen contexts.

5.3. Limited data accessibility

The limited data accessibility refers to the unreproducible or unavailable datasets used in CSD. Many papers [S3, S8, S13–15, S22, S24, S29–30] do not provide access to their source code or the constructed datasets. Some even do not reveal the dataset construction details. For example, [S6] uses private libraries to build datasets. While other researchers cannot access the same datasets to validate or extend the study. Lack of reproducibility in scientific research means reduced impact of the results (Lewowski and Madeyski, 2022). For example, it is difficult for the industry to trust, invest in, or apply ideas or findings that cannot be replicated in practice. We suggest that future studies should select publicly available and representative data sources when constructing the datasets to ensure that the study is replicable, scalable, and widely applicable.

5.4. Heavy expert dependency

Heavy Expert Dependency refers to the manual labeling of the code smell datasets described in the previous subsection, which imposes a substantial demand for expertise on the experts (Ho et al., 2023). Many papers [S3, S6, S14, S17–18, S26, S34, S36] have mentioned that data experts should deeply understand the distinct characteristics and intricate concepts underpinning various code smell types. For example, [S17] proposes to train inspectors and enhance their conceptual and cognitive grasp of the code smell domain, evaluating their aptitude to select excellent graduate students for the manual evaluation. Moreover, it is worth noting that the identification of certain complex code smells can pose formidable challenges to researchers. These intricacies can make the task exceedingly difficult. Even experts can find it hard to agree on the presence of a smell sample (Palomba et al., 2015). For example, Bavota et al. (2013) invite 105 experts to evaluate all refactoring suggestions generated by their models and identify whether they agree with the smelly samples. Results show that 44% of the experts disagree with each other.

5.5. Difficulty of data labeling

Data labeling poses unique challenges for code smell detection, including time costs for manual labeling, accuracy issues with automatic approaches, and potential label noise. Five papers [S5–6, S18, S24, S36] emphasize difficulties with manual and automatic labeling. Manually building labeled datasets is time-consuming due to the extensive effort required [S5]. Though automatic labeling could help with scale, tools have limitations related to predefined heuristics and lower accuracy than human judgments [S6]. Label noise originating during dataset construction also impacts quality. Manual labeling relies on subjective developer perspectives and inconsistent interpretations of smell definitions, which can lead to ambiguous or conflicting labels for the same code [S7, S33, S35]. To experiment on manually validated datasets, [S5] observes significant performance decreases compared to generated data. This highlights the risk of models overfitting to potentially noisy human-generated labels. In summary, both manual and automatic approaches present difficulties that hinder effective labeling at scale. The subjective nature of code smells also makes datasets susceptible to label noise. These challenges point to the need for labeling methods that balance accuracy, efficiency, and consistency in dataset construction.

5.6. Data imbalance

Data imbalance refers to an uneven distribution of code smell samples within datasets, where smelly samples are generally outnumbered by non-smelly code. Such imbalance poses challenges for model training and performance (Gong et al., 2019; Yu et al., 2017; Feng et al., 2021). For example, the widely-used dataset Qualitas [S2, S10, S21, S25, S27] exhibits a significant imbalance, containing only 33% smelly samples. Some datasets are even more skewed, with four studies [S5, S11, S16–17] utilizing data comprising just 2% smelly samples. [S28] indicates severe performance degradation caused by data imbalance. And [S6] highlights imbalance slows down model convergence during training due to the overabundance of non-smelly samples, prolonging the training time. Moreover, models trained on such imbalanced data tend to over-predict samples as non-smelly, which hinders the detection of the underrepresented yet important smelly samples.

5.7. Data redundancy

Data redundancy refers to duplicate or overlapping samples and features within datasets used for model training (Li et al., 2023a; Jian et al., 2019). Sample redundancy means multiple identical code samples exist in the dataset. Several papers [S2, S14–15] observe this occurrence. Redundant samples unnecessarily consume storage and computational resources during training, as they provide no additional value (Xu and Zhang, 2021; Sharma et al., 2021). Redundant features refer to excessive or overlapping code smell features within the dataset. Overlapping features can complicate models and hurt performance, as highlighted by [S2]. Such redundancy can arise when feature extraction tools derive similar code smell features from source code. We identify four papers [S2, S13, S21, S27] that initially use such tools to characterize code smells and construct deep learning models based on these extracted features.

5.8. The interplay of challenges

In previous discussion, we have addressed each identified challenge in isolation to explain their specific impacts and solutions. However, it is crucial to recognize that these challenges often do not occur independently but interplay in ways that can compound their effects. For example, data scarcity often leads to a disproportionately smaller number of smelly instances than non-smelly instances within datasets, thereby exacerbating data imbalance (referenced in [S14, S16]). Similarly, heavy reliance on expert knowledge not only hinders the efficiency of data labeling but also contributes to overall data scarcity, further complicating the challenges (as noted in [S6, S18, S36]). Furthermore, issues such as limited data accessibility combined with high data redundancy can severely limit the generalization ability of the solutions (discussed in [S3, S6, S15, S22, S24, S28, S30]).

To effectively tackle these multifaceted problems, it is apparent that solutions need to be designed with an understanding of these dynamics. Therefore, we propose that a more systematic approach, possibly incorporating integrated solutions, is essential to address the interrelated challenges comprehensively. This holistic perspective is crucial for developing robust and effective strategies to address the complexities.

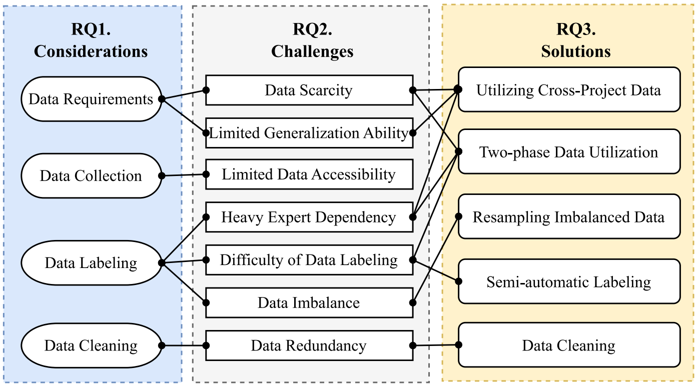


Fig. 7. The mappings between three research questions.

6. RQ3 - Solutions presented in the literature

The previous sections have outlined several key challenges in constructing datasets for DL-based CSD. This section summarizes solutions presented in the literature for mitigating these issues. Fig. 7 summarizes the solutions we have identified and marks the challenges in RQ2 that they address. The following analysis examines these proposed approaches, assessing their potential, as well as their limitations.

6.1. Utilizing cross-project data

(Target Challenges in RQ2: Data Scarcity, Limited Generalization Ability, and Heavy Expert Dependency)

Leveraging data from multiple software projects is an approach for constructing more high-quality datasets (Yu et al., 2018; Xu et al., 2019; Gong et al., 2019). We identify 13 papers [S5–6, S8, S10–11, S14, S16, S24, S28, S31, S34–36] that apply this approach to address challenges related to manual dataset creation efforts. Key benefits of cross-project datasets include improved efficiency, diversity, and generalizability.

Specifically, drawing from various codebases significantly reduces time and resources spent on manual labeling compared to single-project datasets (Barbez et al., 2019). This approach also seamlessly accommodates differing languages and smell types to address dataset homogeneity issues (Sharma et al., 2021). Moreover, validating models across projects further enhances their assessed generalizability. Rather than overfitting to narrow contexts, cross-project datasets support identifying smells in new codebases. This confirms a model's versatility versus the limitations of project-specific evaluations.

In summary, utilizing multi-project sources presents an effective strategy for constructing more high-quality, diverse, and representative datasets. The enhanced scale, heterogeneity, and generalizability provided by cross-project data help advance research by enabling the development of detection models applicable to broad codebase populations. This solution directly addresses key challenges around dataset construction efforts.

6.2. Two-phase data utilization

(Target Challenges in RQ2: Data Scarcity, Heavy Expert Dependency, and Difficulty of Data Labeling)

Aiming to enhance model performance, there are ten papers [S2–3, S14, S18, S25, S27, S32, S33–35] employing a two-phase pre-training and fine-tuning approach.

During pre-training, models learn patterns from synthetically generated data. This provides a foundation of code smell characteristics despite potential differences from real data distributions. The primary goal is exposure rather than perfect replication. Subsequently, fine-tuning involves further refining the model using real-world datasets. Adjusting the learnable parameters of deep learning models helps specialize them to real-world code smells.

Research has shown deep learning can cope with noise in training data (Liu et al., 2019). Thus, synthetic data facilitates dataset expansion while fine-tuning addresses challenges of low reliability and scarce real-world examples. Pre-training establishes a general understanding before fine-tuning customizes performance for real-world accuracy (Ren et al., 2021). Overall, this staged process maximizes the value of generated data. The two-phase approach cultivates models capable of detecting smells across diverse codebases addressing practical scenarios. By integrating synthetic and real data synergistically, this strategy helps advance the field.

6.3. Resampling imbalanced data

(Target Challenges in RQ2: Data Imbalance)

Data resampling aims to address the imbalance by rebalancing class distributions. The appropriate technique depends on the imbalance severity and requirements.

We identify 14 papers [S4, S12–16, S21–23, S25, S27, S31, S33, S36] exploring resampling. The approach used by most studies [S4, S13, S21, S25, S27, S33] is Synthetic Minority Over-sampling TechniquE (SMOTE) (Chawla et al., 2002). This oversampling technique generates synthetic smelly samples to help balance the datasets. Apart from SMOTE, one study [S23] uses fuzzy sampling, and [S22] develops an automatic refactoring tool to transform non-smelly samples into smelly ones. Five papers [S12, S14–16, S36] apply undersampling to reduce the non-smelly samples.

6.4. Semi-automatic labeling

(Target Challenges in RQ2: Difficulty of Data Labeling)

To address labeling challenges, 14 papers [S2–3, S14, S16, S18, S25, S27–28, S31–36] apply a semi-automatic methodology combining automatic tools with manual validation for high-quality labeled data.

Generally, this approach uses automatic tools to label all samples and verifies the results that are labeled by experts. Specifically, multiple experts chose a subset of samples for individual analysis regarding code smells. The experts perform individual analyses without discussing them with others. Then, Cohen's Kappa is calculated to measure inter-expert agreement. Disagreements are discussed to reach a consensus on a final labeled set. Finally, the manually labeled results are compared to the automatic tools' labeled results. If the differences are negligible, the dataset labeled by the automatic tools for all samples is ultimately adopted.

This approach not only facilitates large-scale labeling but also ensures quality through expert verification. For instance, study S[2] achieves balanced datasets by integrating tool-based advice with expert validation, demonstrating improved robustness in model performance.

Similarly, S[16] and S[36] underscore the method's efficacy in refining datasets for enhanced model learning outcomes. Such empirical evidence highlights the semi-automatic approach as both effective and efficient.

6.5. Data cleaning

(Target Challenges in RQ2: Data Redundancy)

The goal of cleaning is to derive datasets optimized for accurate detection. Data cleaning involves removing redundancy, inconsistencies, and irrelevant content. Our survey identifies several common cleaning methods, including:

- Comment/blank line removal to filter non-code contextual data [S3, S21, S27].
- Missing value imputation or sample removal to handle gaps [S21, S27–28].
- Outlier detection/replacement to manage abnormal distributions [S21, S27].
- Data type conversion to fix incorrect format [S33].
- Feature scaling/normalization to standardize attribute ranges [S18, S27–28].
- Feature selection techniques to remove redundancy features [S6, S9–10, S13, S18–21, S25].
- De-duplication to remove replicate samples [S27].

These techniques facilitate downstream smell feature capture by pruning problematic samples and preparing clean, consistent data. Models can then better learn from focused, high-quality inputs. Notably, S[9] and S[13] have shown that careful feature selection is crucial for model accuracy. Further, S[20] and S[25] demonstrate that reducing feature redundancy not only improves model performance but also reduces computational demands. Additionally, S[33] highlights the significant performance enhancements achievable through data standardization, affirming the critical role of these techniques in preparing high-quality datasets for deep learning. Further refinement of these cleaning practices remains an active area of research to develop high-quality CSD datasets.

7. Recommendation

Fig. 7 summarizes our findings on key data preparation considerations, challenges, and potential solutions based on the literature review. This section aims to provide recommendations for researchers and practitioners guided by these results.

Develop datasets across languages and sources. Most papers [S1–12, S15–33, S35] focus on a single programming language, limiting external generalizability and cross-language applicability. Only two papers [S14, S34] examined transfer learning across languages. Manual dataset creation also poses expertise and labor challenges. Automated generation relies on subjective tools restricting new smell detection. To address these, we recommend utilizing cross-project datasets leveraging multiple codebases. This reduces manual effort while enhancing the diversity of languages and smells represented. Researchers should also focus on expanding language support beyond the dominant Java studies to enable transfer learning assessments. We further suggest applying a two-phase pre-training and fine-tuning strategy. This marries the benefits of plentiful synthetic data for pre-training with refinement on real-world examples, improving generalizability.

Standardize cross-study dataset. Lack of consistency hinders reproducibility and progress. Researchers should consider standardizing aspects like identifier naming, feature representations, and metadata tracking across publically available datasets. This will facilitate continued research efforts on cross-dataset challenges. Integrating dataset construction into end-to-end pipelines also helps. Many studies optimized individual preparation phases in isolation. Developing consolidated pipelines covering data sourcing, labeling, cleaning, and modeling could promote co-optimization of these interrelated tasks. Automating pipelines would further decrease manual overhead. In addition, we also recommend setting up centralized data repositories. Finding, preparing, and reusing existing datasets requires significant effort. Researchers should consider developing open centralized repositories and standardized metadata to overcome these barriers.

Adopt semi-automated labeling approaches. Dataset labeling plays a critical role in the preparation process. While manual labeling guarantees high accuracy, it requires significant time and expertise that risks scalability issues (i.e., expertise requirements, labor intensity). Meanwhile, fully automated labeling raises accuracy concerns, especially for complex smells. To address these challenges, we recommend increased utilization of semi-automated labeling approaches. Combining automated prior generation with expert validation, these hybrid methods capture the benefits of both worlds. They considerably reduce manual effort compared to pure manual labeling while maintaining relatively high-quality labels superior to fully automated techniques alone. This makes semi-automated labeling particularly suitable for training deep-learning models targeting complicated smell types. Additionally, datasets constructed from publicly available sources may not precisely represent industrial contexts due to project-specific differences (i.e., external generalizability challenge). We thus suggest practitioners leverage semi-automated strategies to build customized, organization-focused datasets, improving application scenario reflection. Standardizing such hybrid workflows could advance research reproducibility and industrial adoption of detection solutions.

Enhance data transparency and sharing. Current studies face data privacy and replicability challenges. Some datasets solely rely on open-source repositories, introducing subjectivity issues. The inability to fully access or reproduce original private databases also limits validation. We recommend researchers clearly document their full data preparation process. This transparency allows others to comprehensively understand and replicate methodologies. Researchers should also utilize open data platforms to publicly share datasets while preserving privacy. Key metadata around sources and quality assurances enhances usability and trust for the research community. Establishing centralized repositories incentivizing data contributions could help amass more comprehensive benchmark resources. Engaging the industry collaboratively in curating realistic problem snapshots likewise benefits the field. Standardizing metadata schemas and licensing models promotes long-term data maintenance. Proper governance balances privacy, reproducibility, and continued community-driven progress in solving critical problems. Overall, data availability remains paramount for advancing this impactful research domain.

Establish data quality evaluation standards. Dataset quality impacts the credibility and reproducibility of findings. As our review shows, some papers exhibit class imbalance, limited generalizability, noise, and redundancy. We recommend the research community develop a standardized set of data quality criteria. Metrics should assess the key attributes mentioned above and more. For example, balance criteria could specify acceptable sample size ratios between smells/non-smells. Diversity standards may require code drawn from different projects/domains to ensure representativeness. Noise and redundancy checks aim to flag and remove problematic samples. Researchers should systematically apply the proposed criteria when curating and publishing datasets. Studies could also quantitatively benchmark resources pre/post quality improvements to validate enhancement approaches. Establishing clear quality baselines empowers comparative assessment and continued refinement. It promotes replication by formalizing important methodological aspects. Overall, endorsed evaluation practices can help judiciously manage preparation trade-offs and advance the field through high-confidence shared resources.

Potentials of large language models. Large Language Models (LLMs) exhibit excellent zero-shot and in-context learning capabilities, can automatically leverage large-scale data, and are applicable across multiple domains (Gutierrez et al., 2022). In software engineering, LLMs are increasingly recognized for their utility in various applications. For instance, in vulnerability detection, LLMs assist in identifying potential security risks (Akuthota et al., 2023), while in code completion, they facilitate the automatic completion of code fragments based on contextual cues (Roziere et al., 2023).

Specifically, in the realm of code smell detection, LLMs have the potential to identify common code smell patterns by learning from extensive code samples. This capability can aid programmers in detecting and rectifying issues more efficiently. LLMs offer adaptability across different programming languages and project scales compared to existing techniques. Despite the current absence of research directly combining LLMs with CSD, the potential for LLMs to reduce data labeling workloads and tackle data scarcity challenges significantly is compelling. This makes them a promising technology for addressing the needs of high-quality CSD datasets.

Advancing CSD with proven data techniques. Various studies within the software engineering field highlight the importance of robust data preparation. Croft et al. (2022) discuss the challenges of data imbalance and labeling noise, common to our own findings in Code Smell Detection (CSD). They advocate for using class rebalancing and specific data cleaning techniques, such as removing blank lines, non-ASCII characters, comments from code, and duplicate code instances. A unique approach they propose, which differs from our current methodology, is the replacement of user-defined variables and function names with generic tags to reduce code noise further. This strategy could potentially enhance the generalizability of our CSD models by minimizing overfitting to specific code styles or developer idiosyncrasies. According to Yang et al. (2022), a classification of the dataset based on data types — such as code data and metric data — is essential for maintaining relevance and accuracy in data analysis. For code data, it is necessary to filter out irrelevant elements while preserving valuable source code and to remove duplicated instances that can skew the analysis. For metric data, normalization is crucial when values span different orders of magnitude, to ensure that no single metric disproportionately influences the model. As explored by Shi et al. (2022), the development of automatic data cleaning tools represents a significant advancement in handling noisy data within software engineering projects. These tools utilize heuristic rules to automatically identify and remove common issues such as empty functions and duplicated code. Incorporating such tools into our data preparation process could streamline our workflows and improve the quality of our datasets, ultimately leading to more reliable CSD detection models. We believe that integrating cross-disciplinary techniques could improve the overall effectiveness and robustness of CSD models.

8. Threats to validity

As with any systematic literature review, this study faces potential threats to validity that could influence results and conclusions. We classify threats based on construct, internal, external, and conclusion validity (Zhou et al., 2016). While not exhaustive, reporting these threats promotes transparency. The study's context is DL-based code smell detection techniques to establish a foundation for methodological discussions over the specified period — general conclusions require corroborating evidence. We aim to contextualize results by openly reporting on these threats and mitigation efforts.

Construct validity. This is about the connection between the research hypothesis and the findings associated with the RQs. Threats about this category are usually related to the RQs, search strategy, and paper selection process. To mitigate this threat, we create a comprehensive paper selection strategy. Specifically, first, we use the search strings and their alternative spellings and synonyms to ensure that most of the key papers are retrieved. For example, some papers do not necessarily include the term deep learning in the title, abstract, or keywords. We may choose to use the name of the techniques (e.g., RNN or CNN) to ensure that the search string is comprehensive. Second, we create a paper quality assessment strategy to ensure retrieved papers satisfy the RQs. Finally, we employ the *snowballing* (Wohlin, 2014) approach to obtain further any relevant research that may have been missed.

Internal validity. This is related to the consistency of research findings. In this survey, it mainly affects the results of the paper quality assessment and data extraction. To mitigate this threat, we collaborate with multiple authors to reduce subjectivity in the quality assessment and data extraction processes. Specifically, during the quality assessment phase, one author assesses the quality of all papers, and two other authors validate the results. We will talk about any disagreements and resolve them. Also, one author extracts the data during data extraction, and the other authors validate the extracted data for all the papers. The data are compared, and any conflicts are discussed and resolved.

External validity. This is related to the generalizability of the reported results. This survey focuses on only one area of software engineering — the data preparation process for DL-based CSD. Therefore, the results cannot be generalized to other areas. Furthermore, deep learning is a rapidly evolving field with new techniques being introduced every day (Alazba et al., 2023). Our survey is limited to December 2023. Results may not apply to ranges outside the timeline.

Conclusion validity. This is related to the likelihood of reproducing the research and obtaining the same results. To mitigate this threat, we describe the entire research process in detail, including the RQs, the search string, the inclusion/exclusion criteria, the quality assessment form, and the data extraction form. In addition, our findings, i.e., considerations, challenges, and solutions, are based on data extracted from the original papers. We ensure the integrity of our survey and the reality of our findings through rigorous paper selection.

9. Conclusion

Through a systematic analysis of 36 relevant studies on data preparation approaches for deep learning-based code smell detection until December 2023, our survey illuminates key aspects of the data preparation process. We explore considerations across the data requirements, collection, labeling, and cleaning phases, as well as prevalent challenges and proposed solutions from the literature. Key challenges identified include data scarcity, limited generalizability, difficulties in labeling, data imbalance, and redundancy issues. The solutions proposed focused on leveraging cross-project data, two-phase data utilization, semi-automated labeling, resampling, and data cleaning techniques. Based on these results, our primary recommendations are for researchers to establish standardized practices around dataset quality assessment, transparency, and centralized resources. We also recommend techniques for practitioners to construct industrial-strength datasets representative of real-world codebases. This systematic review provides a foundation for rigorously evaluating CSD data preparation efforts. Adopting its recommendations aims to foster continued optimizations towards better detection capabilities with real-world application potential.

Limitations and future work. To address the limitations inherent in a systematic literature review, we acknowledge the absence of empirical validation of the recommendations made in this paper. While our study provides a comprehensive synthesis of available literature on data preparation processes for deep learning-based code smell detection, the conclusions drawn remain hypothetical. Recognizing this, we commit to staying updated on the latest developments in DL-based CSD. We plan to conduct practical experiments and detailed case studies for empirical validation in future studies. Such empirical evidence will be crucial to substantiate the effectiveness of the proposed solutions and further advance the field.

CRediT authorship contribution statement

Fengji Zhang: Writing – review & editing, Writing – original draft, Validation, Project administration, Methodology, Investigation, Data curation, Conceptualization. **Zexian Zhang:** Writing – original draft, Validation, Investigation, Data curation. **Jacky Wai Keung:** Writing – review & editing, Supervision, Project administration. **Xiangru Tang:** Writing – review & editing, Validation. **Zhen Yang:** Writing – review & editing, Validation, Investigation. **Xiao Yu:** Writing – review & editing, Writing – original draft, Validation, Supervision, Investigation. **Wenhua Hu:** Writing – review & editing, Validation, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

This work is partially supported by the National Natural Science Foundation of China (62202350), the Natural Science Foundation of Chongqing, China (cstc2021jcyj-msxmX1115), and the General Research Fund of the Research Grants Council of Hong Kong and the research funds of the City University of Hong Kong (6000796, 9229109, 9229098, 9220103, 9229029).

References

Afrin, M., Asma, S.A., Akhter, N., Ridoy, J.H., Sauda, S.S., Taher, K.A., 2022. A hybrid approach to investigate anti-pattern from source code. In: 2022 25th International Conference on Computer and Information Technology. ICCIT, IEEE, pp. 888–892.

Akuthota, V., Kasula, R., Sumona, S.T., Mohiuddin, M., Reza, M.T., Rahman, M.M., 2023. Vulnerability detection and monitoring using LLM. In: 2023 IEEE 9th International Women in Engineering (WIE) Conference on Electrical and Computer Engineering. WIECON-ECE, IEEE, pp. 309–314.

Al-Shaaby, A., Aljamaan, H., Alshayeb, M., 2020. Bad smell detection using machine learning techniques: a systematic literature review. Arab. J. Sci. Eng. 45, 2341–2369.

Alazba, A., Aljamaan, H., Alshayeb, M., 2023. Deep learning approaches for bad smell detection: a systematic literature review. Empir. Softw. Eng. 28 (3), 77.

Alkharabsheh, K., Crespo, Y., Manso, E., Taboada, J.A., 2019. Software design smell detection: a systematic mapping study. Softw. Qual. J. 27, 1069–1148.

Allal, L.B., Li, R., Kocetkov, D., Mou, C., Akiki, C., Ferrandis, C.M., Muennighoff, N., Mishra, M., Gu, A., Dey, M., et al., 2023. SantaCoder: don't reach for the stars!. arXiv preprint arXiv:2301.03988.

Amershi, S., Begel, A., Bird, C., DeLine, R., Gall, H., Kamar, E., Nagappan, N., Nushi, B., Zimmermann, T., 2019. Software engineering for machine learning: A case study. In: 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice. ICSE-SEIP, IEEE, pp. 291–300.

Arcelli Fontana, F., Mäntylä, M.V., Zanoni, M., Marino, A., 2016. Comparing and experimenting machine learning techniques for code smell detection. Empir. Softw. Eng. 21, 1143–1191.

Ardimento, P., Aversano, L., Bernardi, M.L., Cimitile, M., Iammarino, M., 2021a. Temporal convolutional networks for just-in-time design smells prediction using fine-grained software metrics. Neurocomputing 463, 454–471.

Ardimento, P., Aversano, L., Bernardi, M.L., Cimitile, M., Iammarino, M., et al., 2021b. Transfer learning for just-in-time design smells prediction using temporal convolutional networks. In: ICSOFT. pp. 310–317.

Azeem, M.I., Palomba, F., Shi, L., Wang, Q., 2019. Machine learning techniques for code smell detection: A systematic literature review and meta-analysis. Inf. Softw. Technol. 108, 115–138.

Barbez, A., Khomh, F., Guéhéneuc, Y.-G., 2019. Deep learning anti-patterns from code metrics history. In: 2019 IEEE International Conference on Software Maintenance and Evolution. ICSME, IEEE, pp. 114–124.

Bavota, G., Oliveto, R., Gethers, M., Poshyvanyk, D., De Lucia, A., 2013. Methodbook: Recommending move method refactorings via relational topic models. IEEE Trans. Softw. Eng. 40 (7), 671–694.

Bhave, A., Sinha, R., 2022. Deep multimodal architecture for detection of long parameter list and switch statements using distilbert. In: 2022 IEEE 22nd International Working Conference on Source Code Analysis and Manipulation. SCAM, IEEE, pp. 116–120.

Chawla, N.V., Bowyer, K.W., Hall, L.O., Kegelmeyer, W.P., 2002. SMOTE: synthetic minority over-sampling technique. J. Artif. Intell. Res. 16, 321–357.

Checkstyle, http://checkstyle.sourceforge.net.

Chen, Y., Huang, J., Mou, L., Jin, P., Xiong, S., Zhu, X.X., 2023. Deep saliency smoothing hashing for drone image retrieval. IEEE Trans. Geosci. Remote Sens. 61, 1–13.

Chen, Y., Lu, X., Wang, S., 2020. Deep cross-modal image–voice retrieval in remote sensing. IEEE Trans. Geosci. Remote Sens. 58 (10), 7049–7061.

Croft, R., Xie, Y., Babar, M.A., 2022. Data preparation for software vulnerability prediction: A systematic literature review. IEEE Trans. Softw. Eng. 49 (3), 1044–1063.

Danphitsanuphan, P., Suwantada, T., 2012. Code smell detecting tool and code smell-structure bug relationship. In: 2012 Spring Congress on Engineering and Technology. IEEE, pp. 1–5.

Das, A.K., Yadav, S., Dhal, S., 2019. Detecting code smells using deep learning. In: TENCON 2019-2019 IEEE Region 10 Conference. TENCON, IEEE, pp. 2081–2086.

Dewangan, S., Rao, R.S., Mishra, A., Gupta, M., 2022. Code smell detection using ensemble machine learning algorithms. Appl. Sci. 12 (20), 10321.

Di Nucci, D., Palomba, F., Tamburri, D.A., Serebrenik, A., De Lucia, A., 2018. Detecting code smells using machine learning techniques: are we there yet? In: 2018 Ieee 25th International Conference on Software Analysis, Evolution and Reengineering. Saner, IEEE, pp. 612–621.

Fakhoury, S., Arnaoudova, V., Noiseux, C., Khomh, F., Antoniol, G., 2018. Keep it simple: Is deep learning good for linguistic smell detection? In: 2018 IEEE 25Th International Conference on Software Analysis, Evolution and Reengineering. SANER, IEEE, pp. 602–611.

Feng, S., Keung, J., Yu, X., Xiao, Y., Bennin, K.E., Kabir, M.A., Zhang, M., 2021. COSTE: Complexity-based OverSampling TEchnique to alleviate the class imbalance problem in software defect prediction. Inf. Softw. Technol. 129, 106432.

Fontana, F.A., Zanoni, M., 2017. Code smell severity classification using machine learning techniques. Knowl.-Based Syst. 128, 43–58.

Fowler, M., 2018. Refactoring. Addison-Wesley Professional.

Gao, Y., Wang, X., He, X., Feng, H., Zhang, Y., 2023. Rumor detection with self-supervised learning on texts and social graph. Front. Comput. Sci. 17 (4), 174611.

Garousi, V., Felderer, M., 2017. Experience-based guidelines for effective and efficient data extraction in systematic reviews in software engineering. In: Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering. pp. 170–179.

Gong, L., Jiang, S., Bo, L., Jiang, L., Qian, J., 2019. A novel class-imbalance learning approach for both within-project and cross-project defect prediction. IEEE Trans. Reliab. 69 (1), 40–54.

Grodniyomchai, B., Chalapat, K., Jitkajornwanich, K., Jaiyen, S., 2019. A deep learning model for odor classification using deep neural network. In: 2019 5th International Conference on Engineering, Applied Sciences and Technology. ICEAST, IEEE, pp. 1–4.

Guo, X., Shi, C., Jiang, H., 2019. Deep semantic-based feature envy identification. In: Proceedings of the 11th Asia-Pacific Symposium on Internetware. pp. 1–6.

Gupta, H., Kulkarni, T.G., Kumar, L., Neti, L.B.M., Krishna, A., 2021. An empirical study on predictability of software code smell using deep learning models. In: International Conference on Advanced Information Networking and Applications. Springer, pp. 120–132.

Gupta, A., Suri, B., Misra, S., 2017. A systematic literature review: code bad smells in java source code. In: Computational Science and Its Applications–ICCSA 2017: 17th International Conference, Trieste, Italy, July 3-6, 2017, Proceedings, Part V 17. Springer, pp. 665–682.

Gutierrez, B.J., McNeal, N., Washington, C., Chen, Y., Li, L., Sun, H., Su, Y., 2022. Thinking about gpt-3 in-context learning for biomedical ie? think again. arXiv preprint arXiv:2203.08410.

Hadj-Kacem, M., Bouassida, N., 2018. A hybrid approach to detect code smells using deep learning. In: ENASE. pp. 137–146.

Hadj-Kacem, M., Bouassida, N., 2019a. Deep representation learning for code smells detection using variational auto-encoder. In: 2019 International Joint Conference on Neural Networks. IJCNN, IEEE, pp. 1–8.

Hadj-Kacem, M., Bouassida, N., 2019b. Improving the identification of code smells by combining structural and semantic information. In: Neural Information Processing: 26th International Conference, ICONIP 2019, Sydney, NSW, Australia, December 12–15, 2019, Proceedings, Part IV 26. Springer, pp. 296–304.

Hamdy, A., Tazy, M., 2020. Deep hybrid features for code smells detection. J. Theor. Appl. Inf. Technol. 98 (14), 2684–2696.

Ho, A., Bui, A.M., Nguyen, P.T., Di Salle, A., 2023. Fusion of deep convolutional and LSTM recurrent neural networks for automated detection of code smells. In: Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering. pp. 229–234.

Hu, W., Liu, L., Yang, P., Zou, K., Li, J., Lin, G., Xiang, J., 2023. Revisiting“ code smell severity classification using machine learning techniques”. In: 2023 IEEE 47th Annual Computers, Software, and Applications Conference. COMPSAC, IEEE, pp. 840–849.

Imam, A.T., Al-Srour, B.R., Alhroob, A., 2022. The automation of the detection of large class bad smell by using genetic algorithm and deep learning. J. King Saud Univ.-Comput. Inf. Sci. 34 (6), 2621–2636.

Jain, S., Saha, A., 2021. Improving performance with hybrid feature selection and ensemble machine learning techniques for code smell detection. Sci. Comput. Program. 212, 102713.

Jeevanantham, M., Jones, J., 2022. Extension of deep learning based feature envy detection for misplaced fields and methods. Int. J. Intell. Eng. Syst. 15 (1), 563–574.

Jian, Y., Yu, X., Xu, Z., Ma, Z., 2019. A hybrid feature selection method for software fault prediction. IEICE Trans. Inf. Syst. 102 (10), 1966–1975.

Karasneh, B., Chaudron, M.R., 2013. Img2uml: A system for extracting uml models from images. In: 2013 39th Euromicro Conference on Software Engineering and Advanced Applications. IEEE, pp. 134–137.

Kaur, A., Jain, S., Goel, S., Dhiman, G., 2021. A review on machine-learning based code smell detection techniques in object-oriented software system (s). Recent Adv. Electr. Electron. Eng. (Formerly Recent Pat. Electr. Electron. Eng.) 14 (3), 290–303.

Kaur, S., Singh, S., 2023. Improving the quality of open source software. In: Agile Software Development: Trends, Challenges and Applications. Wiley Online Library, pp. 309–323.

Khleel, N.A.A., Nehéz, K., 2022. Deep convolutional neural network model for bad code smells detection based on oversampling method. Indones. J. Electr. Eng. Comput. Sci. 26 (3), 1725–1735.

Kim, D.K., 2017. Finding bad code smells with neural network models. Int. J. Electr. Comput. Eng. 7 (6), 3613.

Kim, D.K., 2020. A deep neural network-based approach to finding similar code segments. IEICE Trans. Inf. Syst. 103 (4), 874–878.

Kitchenham, B., 2004. Procedures for Performing Systematic Reviews, Vol. 33, Keele, UK, Keele University, pp. 1–26.

Kitchenham, B., Brereton, O.P., Budgen, D., Turner, M., Bailey, J., Linkman, S., 2009. Systematic literature reviews in software engineering–a systematic literature review. Inf. Softw. Technol. 51 (1), 7–15.

Lewowski, T., Madeyski, L., 2022. How far are we from reproducible research on code smell detection? A systematic literature review. Inf. Softw. Technol. 144, 106783.

Li, F., Lu, W., Keung, J.W., Yu, X., Gong, L., Li, J., 2023a. The impact of feature selection techniques on effort-aware defect prediction: An empirical study. IET Softw. 17 (2), 168–193.

Li, Y., Zhang, X., 2022. Multi-label code smell detection with hybrid model based on deep learning. In: SEKE. pp. 42–47.

Li, F., Zou, K., Keung, J.W., Yu, X., Feng, S., Xiao, Y., 2023b. On the relative value of imbalanced learning for code smell detection. Softw. - Pract. Exp. 53 (10), 1902–1927.

Lin, T., Fu, X., Chen, F., Li, L., 2021. A novel approach for code smells detection based on deep leaning. In: Applied Cryptography in Computer and Communications: First EAI International Conference, AC3 2021, Virtual Event, May 15-16, 2021, Proceedings 1. Springer, pp. 171–174.

Liu, H., Jin, J., Xu, Z., Zou, Y., Bu, Y., Zhang, L., 2019. Deep learning based code smell detection. IEEE Trans. Softw. Eng. 47 (9), 1811–1837.

Liu, L., Lin, G., Zhu, L., Yang, Z., Song, P., Wang, X., Hu, W., 2024. Revisiting code smell severity prioritization using learning to rank techniques. Expert Syst. Appl. 123483.

Liu, B., Liu, H., Li, G., Niu, N., Xu, Z., Wang, Y., Xia, Y., Zhang, Y., Jiang, Y., 2023. Deep learning based feature envy detection boosted by real-world examples. In: Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. pp. 908–920.

Lu, S., Guo, D., Ren, S., Huang, J., Svyatkovskiy, A., Blanco, A., Clement, C., Drain, D., Jiang, D., Tang, D., et al., 2021. Codexglue: A machine learning benchmark dataset for code understanding and generation. arXiv preprint arXiv:2102.04664.

Ma, X., Keung, J.W., Yu, X., Zou, H., Zhang, J., Li, Y., 2023. AttSum: A deep attention-based summarization model for bug report title generation. IEEE Trans. Reliab..

Madeyski, L., Lewowski, T., 2020. MLCQ: Industry-relevant code smell data set. In: Proceedings of the 24th International Conference on Evaluation and Assessment in Software Engineering. pp. 342–347.

Malathi, J., Jabez, J., 2023. Class code smells detection using deep learning approach. In: AIP Conference Proceedings. AIP Publishing.

Malhotra, R., Jain, B., Kessentini, M., 2023. Examining deep learning’s capability to spot code smells: a systematic literature review. Cluster Comput. 1–29.

Marinescu, C., Marinescu, R., Mihancea, P., Ratiu, D., Wettel, R., 2005. Iplasma: An integrated platform for quality assessment of object-oriented design. In: IEEE International Conference on Software Maintenance-Industrial & Tool Volume. DBLP.

Martínez-Fernández, S., Bogner, J., Franch, X., Oriol, M., Siebert, J., Trendowicz, A., Vollmer, A.M., Wagner, S., 2022. Software engineering for AI-based systems: a survey. ACM Trans. Softw. Eng. Methodol. (TOSEM) 31 (2), 1–59.

Naik, P., Nelaballi, S., Pusuluri, V.S., Kim, D.-K., 2023. Deep learning-based code refactoring: A review of current knowledge. J. Comput. Inf. Syst. 1–15.

Palomba, F., Di Nucci, D., Tufano, M., Bavota, G., Oliveto, R., Poshyvanyk, D., De Lucia, A., 2015. Landfill: An open dataset of code smells with public evaluation. In: 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories. IEEE, pp. 482–485.

PMD, https://pmd.github.io/.

Podobnik, B., Stanley, H.E., 2007. Detrended cross-correlation analysis: A new method for analyzing two non-stationary time series. arXiv preprint arXiv:0709.0281.

Qiao, B., Wu, Z., Ma, L., Zhou, Y., Sun, Y., 2023. Effective ensemble learning approach for SST field prediction using attention-based PredRNN. Front. Comput. Sci. 17 (1), 171601.

Ramos, M., de Mello, R., Fonseca, B., 2022. On Transfer Learning in Code Smells Detection. Technical Report, EasyChair.

Ren, S., Shi, C., Zhao, S., 2021. Exploiting multi-aspect interactions for god class detection with dataset fine-tuning. In: 2021 IEEE 45th Annual Computers, Software, and Applications Conference. COMPSAC, IEEE, pp. 864–873.

Roziere, B., Gehring, J., Gloeckle, F., Sootla, S., Gat, I., Tan, X.E., Adi, Y., Liu, J., Remez, T., Rapin, J., et al., 2023. Code llama: Open foundation models for code. arXiv preprint arXiv:2308.12950.

Santos, J.A.M., Rocha-Junior, J.B., Prates, L.C.L., Do Nascimento, R.S., Freitas, M.F., De Mendonça, M.G., 2018. A systematic review on the code smell effect. J. Syst. Softw. 144, 450–477.

Schardt, C., Adams, M.B., Owens, T., Keitz, S., Fontelo, P., 2007. Utilization of the PICO framework to improve searching PubMed for clinical questions. BMC Med. Inform. Decis. Mak. 7, 1–6.

Sharma, T., Efstathiou, V., Louridas, P., Spinellis, D., 2021. Code smell detection by deep direct-learning and transfer-learning. J. Syst. Softw. 176, 110936.

Sharma, T., Spinellis, D., 2018. A survey on software smells. J. Syst. Softw. 138, 158–173.

Shi, L., Mu, F., Chen, X., Wang, S., Wang, J., Yang, Y., Li, G., Xia, X., Wang, Q., 2022. Are we building on the rock? on the importance of data preprocessing for code summarization. In: Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. pp. 107–119.

Siddiq, M.L., Majumder, S.H., Mim, M.R., Jajodia, S., Santos, J.C., 2022. An empirical study of code smells in transformer-based code generation techniques. In: 2022 IEEE 22nd International Working Conference on Source Code Analysis and Manipulation. SCAM, IEEE, pp. 71–82.

Sidhu, B.K., Singh, K., Sharma, N., 2022. A machine learning approach to software model refactoring. Int. J. Comput. Appl. 44 (2), 166–177.

Sousa, B.L., Souza, P.P., Fernandes, E.M., Ferreira, K.A., Bigonha, M.A., 2017. FindSmells: flexible composition of bad smell detection strategies. In: 2017 IEEE/ACM 25th International Conference on Program Comprehension. ICPC, IEEE, pp. 360–363.

Tarwani, S., Chug, A., 2022. Application of deep learning models for code smell prediction. In: 2022 10th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions). ICRITO, IEEE, pp. 1–5.

Tempero, E., Anslow, C., Dietrich, J., Han, T., Li, J., Lumpe, M., Melton, H., Noble, J., 2010. The qualitas corpus: A curated collection of java code for empirical studies. In: 2010 Asia Pacific Software Engineering Conference. IEEE, pp. 336–345.

Tsantalis, N., Chaikalis, T., Chatzigeorgiou, A., 2008. JDeodorant: Identification and removal of type-checking bad smells. In: 2008 12th European Conference on Software Maintenance and Reengineering. IEEE, pp. 329–331.

Ucdetector, http://ucdetector.sourceforge.net/update.

Virmajoki, J., 2020. Detecting code smells using artificial intelligence: a prototype.

Virmajoki, J., Knutas, A., Kasurinen, J., 2022. Detecting code smells with AI: a prototype study. In: 2022 45th Jubilee International Convention on Information, Communication and Electronic Technology. MIPRO, IEEE, pp. 1393–1398.

Wang, H., Liu, J., Kang, J., Yin, W., Sun, H., Wang, H., 2020. Feature envy detection based on bi-lstm with self-attention mechanism. In: 2020 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom). IEEE, pp. 448–457.

Wieman, R., 2011. Anti-Pattern Scanner: an Approach to Detect Anti-Patterns and Design Violations. LAP Lambert Academic Publishing.

Wohlin, C., 2014. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In: Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering. pp. 1–10.

Xu, Z., Pang, S., Zhang, T., Luo, X.-P., Liu, J., Tang, Y.-T., Yu, X., Xue, L., 2019. Cross project defect prediction via balanced distribution adaptation based transfer learning. J. Comput. Sci. Tech. 34, 1039–1062.

Xu, W., Zhang, X., 2021. Multi-granularity code smell detection using deep learning method based on abstract syntax tree. In: Proc. 33rd Int. Conf. Software Engineering and Knowledge Engineering. pp. 503–509.

Yang, Z., Keung, J.W., Yu, X., Xiao, Y., Jin, Z., Zhang, J., 2023. On the significance of category prediction for code-comment synchronization. ACM Trans. Softw. Eng. Methodol. 32 (2), 1–41.

Yang, Y., Xia, X., Lo, D., Grundy, J., 2022. A survey on deep learning for software engineering. ACM Comput. Surv. 54 (10s), 1–73.

Yedida, R., Menzies, T., 2022. How to improve deep learning for software analytics: (a case study with code smell detection). In: Proceedings of the 19th International Conference on Mining Software Repositories. pp. 156–166.

Yin, X., Shi, C., Zhao, S., 2021. Local and global feature based explainable feature envy detection. In: 2021 IEEE 45th Annual Computers, Software, and Applications Conference. COMPSAC, IEEE, pp. 942–951.

Yu, X., Liu, J., Yang, Z., Jia, X., Ling, Q., Ye, S., 2017. Learning from imbalanced data for predicting the number of software defects. In: 2017 IEEE 28th International Symposium on Software Reliability Engineering. ISSRE, IEEE, pp. 78–89.

Yu, J., Mao, C., Ye, X., 2021. A novel tree-based neural network for android code smells detection. In: 2021 IEEE 21st International Conference on Software Quality, Reliability and Security. QRS, IEEE, pp. 738–748.

Yu, X., Wu, M., Jian, Y., Bennin, K.E., Fu, M., Ma, C., 2018. Cross-company defect prediction via semi-supervised clustering-based data filtering and MSTrA-based transfer learning. Soft Comput. 22, 3461–3472.

Zakeri-Nasrabadi, M., Parsa, S., Esmaili, E., Palomba, F., 2023. A systematic literature review on the code smells datasets and validation mechanisms. ACM J. Comput. Cult. Herit..

Zhang, H., Babar, M.A., Tell, P., 2011. Identifying relevant studies in software engineering. Inf. Softw. Technol. 53 (6), 625–637.

Zhang, Y., Ge, C., Hong, S., Tian, R., Dong, C., Liu, J., 2022. DeleSmell: code smell detection based on deep learning and latent semantic analysis. Knowl.-Based Syst. 255, 109737.

Zhang, M., Jia, J., 2022. Feature envy detection with deep learning and snapshot ensemble. In: 2022 9th International Conference on Dependable Systems and their Applications. DSA, IEEE, pp. 215–223.

Zhou, X., Jin, Y., Zhang, H., Li, S., Huang, X., 2016. A map of threats to validity of systematic literature reviews in software engineering. In: 2016 23rd Asia-Pacific Software Engineering Conference. APSEC, IEEE, pp. 153–160.