

Article

A Meta-Contrastive Optimization Framework for Multilabel Bug Dependency Classification

Jantima Polpinij ¹, Manasawee Kaenampornpan ^{2,*} and Bancha Luaphol ³

¹ Department of Computer Science, Faculty of Informatics, Mahasarakham University, Mahasarakham 44150, Thailand; jantima.p@msu.ac.th

² Department of Computer Engineering, Faculty of Engineering, Khon Kaen University, Khon Kaen 40002, Thailand

³ Department of Business Computer, Faculty of Administrative Science, Kalasin University, Kalasin 46000, Thailand; bancha.lu@ksu.ac.th

* Correspondence: manaska@kku.ac.th; Tel.: +66-842045999

Abstract

Software maintenance and release management demand proper identification of bug dependencies since priority violations or unresolved dependent issues can often lead to a chain of failures. However, dependency annotations in bug reports are extremely sparse and imbalanced. These dependencies are often expressed implicitly through natural language descriptions rather than explicit metadata. This creates challenges for automated multilabel dependency classification systems. To tackle these drawbacks, we introduce a meta-contrastive optimization framework (MCOF). This framework integrates established learning paradigms to enhance transformer-based classification through two key mechanisms: (1) a meta-contrastive objective adapted for enhancing discriminative representation learning under few-shot supervision, particularly for rare dependency types, and (2) dependency-aware Laplacian regularization that captures relational structures among different dependency types, reducing confusion between semantically related labels. Experimental evaluation on a real-world dataset demonstrates that MCOF achieves significant improvement over strong baselines, including BM25-based clustering and standard BERT fine-tuning. The framework attains a micro-F1 score of 0.76 and macro-F1 score of 0.58, while reducing hamming loss to 0.14. Label-wise analysis shows significant performance gain on low-frequency dependency types, with improvements of up to 16% in F1 score. Runtime analysis exhibits only modest inference overhead at 15%, confirming that MCOF remains practical for deployment in CI/AT pipelines. These results demonstrate that integrating meta-contrastive learning and structural regularization is an effective approach for robust bug dependency discovery. The framework provides both practical and accurate solutions for supporting real-world software engineering workflows.



Academic Editor: Florin Leon

Received: 1 December 2025

Revised: 15 January 2026

Accepted: 16 January 2026

Published: 19 January 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

Keywords: multilabel classification; bug dependency analysis; meta-contrastive learning; transformer models; imbalanced data; software maintenance

MSC: 68T07; 62H30; 68T50; 68T05

1. Introduction

Dependent bug analysis plays an important role in issue management and release planning on software projects [1,2]. Unresolved dependency chains can cause cascading failures that delay projects indefinitely because fixing one bug often requires resolving

another dependent bug first [3]. Bugzilla and Jira include issue tracker features such as ‘blocks’ or ‘depends on’ to help developers visualize and track the structured relationships between issues [1,4]. However, these dependencies are mostly assigned manually, leading to incomplete annotations, inconsistencies and significant delay [5,6]. As a result, automating dependency identification from bug reports has become essential for state-of-the-art software maintenance practices [7–9].

Early approaches usually relied on unsupervised learning and information retrieval techniques. For instance, Luaphol et al. [8] utilized BM25 similarity to cluster dependent bug reports around meta-bugs based on a fixed similarity threshold, illustrating that it was feasible to assemble dependencies automatically without labeled data [8,9]. While this approach is cost-effective, it is limited to coarse-grained lexical analysis. It often leads to errors by linking semantically unrelated words during symptom identification in textual bug descriptions [10–12]. Unsupervised clustering also struggles to handle multilabel dependence types, making it ineffective at distinguishing subtle relation subtypes (e.g., blocks versus related) [13,14]. Overall, the lack of semantic modeling and dependency structure understanding shows poor performance, especially for rare dependency labels that are important in practice [13–15].

Recently, advances in natural language processing (NLP) and deep learning have strengthened the effectiveness of supervised text classification-based techniques for exploring semantic information from bug reports [16–18]. Transformer-based models such as BERT and RoBERTa learn contextualized representations of text and have delivered promising results in tasks such as bug triaging and severity prediction. However, existing supervised approaches have not fully examined multilabel classification for dependency, where a single bug can correspond to multiple types of dependency at once [16,19]. Two main challenges occur: (1) imbalanced label distribution, where certain dependency types are severely underrepresented and (2) weak structure modeling between dependency labels, such that their co-occurrence and relational semantics are neglected [20–22].

These restrictions point to a clear research gap. Current methods for dependency identification rely either on unsupervised lexical-similarity clustering, which lacks semantic awareness, or on supervised text classification, which does not explicitly consider the multilabel structure of dependencies nor maximize performance for rare relational types [20,23,24]. These limitations make it difficult to automatically discover complex dependency relationships between software bugs, especially when subtle semantic clues or multiple co-existing dependencies are involved [7,9,24,25].

We address this research gap through a novel integration of meta-learning and contrastive optimization specifically designed for multilabel bug dependency classification. Our framework systematically integrates established learning paradigms while providing empirical validation of performance robustness under conditions of severe label imbalance. While the individual components have been investigated independently in prior studies, their systematic integration within a unified meta-contrastive optimization framework and comprehensive evaluation for fine-grained bug dependency classification remain underexplored. Instead of relying on a single learning objective, the proposed method combines transformer-based embeddings of bug titles and descriptions to capture the rich context semantics [9]. A meta-contrastive loss is used to attract semantically related bug reports and repel unrelated samples in the latent space [26]. In particular, meta-learning strategies are utilized to address the extreme scarcity of certain dependency types. By treating different label subgroups as distinct tasks, the framework promotes robust optimization even for “tail” classes with very few training examples. To further cope with the highly imbalanced label distribution, especially for low-frequency types that are operationally important dependency types, an imbalance-aware regularization term is incorporated into the

optimization criterion [9]. Moreover, dependence-aware structural regularization, based on optional graph constraints from label co-occurrence pattern, helps the model capture label correlations, leading to more robust multilabel prediction. Therefore, dependency classification is formulated as a fully supervised, semantics-driven multilabel learning problem, supported by a theoretically sound optimization framework.

The objective of this study is to demonstrate that supervised semantic learning offers statistically significant improvements over unsupervised dependency kit-assembly for recognizing text-based dependencies, ultimately providing fine-grained support for real-world development cycles. More importantly, the project aims to develop a powerful multilabel dependency classifier that can significantly enhance prediction results for rare relational types and provide an integrated solution that effectively combines the semantic representation learning and the dependency structure modeling in bug repositories. Anticipated results include a significant improvement in macro-F1 and label-wise predictions performance as compared to unsupervised BM25-based baseline. Thus, it demonstrates the importance of semantic modeling for better and more intelligent dependency history learning in practice.

To summarize, this work contributes to automated dependency type classification, including the following: (i) the first direct comparison between unsupervised syntactic assembly of dependencies and supervised semantic multilabel learning; (ii) a unified integration of Reptile-based meta-learning and supervised contrastive learning, empirically validated for improving rare dependency prediction under severe label imbalance; and (iii) substantial performance gains on low-frequency dependency labels achieved through imbalance-aware techniques and dependency-sensitive regularization. Together, these contributions advance the field of bug dependency management in evolving software applications.

The remainder of this paper is organized as follows. Section 2 describes the dataset and its characteristics. Section 3 details the proposed methodology. Section 4 presents the experimental results and analysis. Finally, Section 5 concludes the paper and discusses potential future work.

2. Bug Report Dataset and Dependency Annotation

2.1. Data Source and Annotation

The dataset used in our study was compiled from real-world bug reports scraped from the Bugzilla platform, primarily focusing on Mozilla Firefox project. All reports were obtained from publicly accessible development repositories, ensuring data reproducibility and transparency. To ensure the reliability of our dependency data, only bugs marked as “verified” or “closed” were considered. These bugs were further inspected and validated by software developers or triagers. Therefore, the dependency annotations reflect expert-driven knowledge rather than purely heuristic assumptions.

Every bug report in our dataset comprises three main components: (1) a brief description of the main symptoms, (2) a complete problem statement, and (3) comments from the ongoing discussion during diagnosis and repair.

Based on previous works [8,9,27,28], the summary and description fields are the most useful sources of semantic information for identifying dependencies, as they tend to minimize the noise introduced by comments. Consequently, these two textual fields are selected as the main input features for our models.

Importantly, bugs participate in multiple dependency relationships simultaneously. A single bug report might block several other issues while also depending on additional root issues. This characteristic transforms the couple of class-labels for dependence classification into a multilabeling problem that better reflects the complex interdependencies found in real

software development and dependency management. Given the annotation schema used in BM25 clustering, each bug report is associated with one to four dependency relations, forming a directed acyclic dependency structure where meta-bugs serve as parent nodes imposing constraints on related issues.

The dataset contains a total of 4781 annotated bug reports covering 200 dependency centers. A summary of the dataset is provided in Table 1. All annotations were manually verified by expert triagers, establishing these labels as ground-truth expert annotations; therefore, this dataset is very reliable for the evaluation of contemporary dependency mining methods. Figures 1 and 2 are examples of a bug report.

Table 1. Summary of dataset used.

Property	Description
Source	Bugzilla, Mozilla Firefox
Total reports	4781
Meta-bug reports	200
Dependency labels	1–4 multilabel per report
Text fields used	Summary + Description
Annotation origin	Verified by bug triagers
Validation	10-fold cross-validation
Data issue	Imbalanced label distribution

The screenshot shows a Bugzilla interface for a bug report. At the top, there's a navigation bar with the Bugzilla logo, a search bar, and links for 'Browse', 'Advanced Search', 'New Bug', 'My Dashboard', and user profile. Below this, the bug list shows 'Bug 1833512' with status 'Open', opened and updated 2 years ago. The title is '[meta] Firefox Add-on Suggestions in the Address Bar'. The 'Categories' section shows 'Product: Firefox' and 'Component: Address Bar'. The 'Tracking' section shows 'Status: NEW'. The 'People' section shows 'Assignee: Unassigned', 'Reporter: daisuke', and 'Triage Owner: daisuke'. The 'References' section lists dependencies on other bugs (e.g., 1835041, 1838877, 1838882, 1834986, 1834995, 1835008, 1835034, 1835224, 1835598, 1837933, 1828610, 1834048, 1832474, 1832515, 1832553, 1833750, 1833760, 1833966, 1834646, 1835979, 1836005, 1836582, 1836585, 1839558) and a URL 'mozilla-hub.atlassian.net/browse/SNT-614'. A 'Dependency tree / graph' link is also present.

Figure 1. An example of bug report.

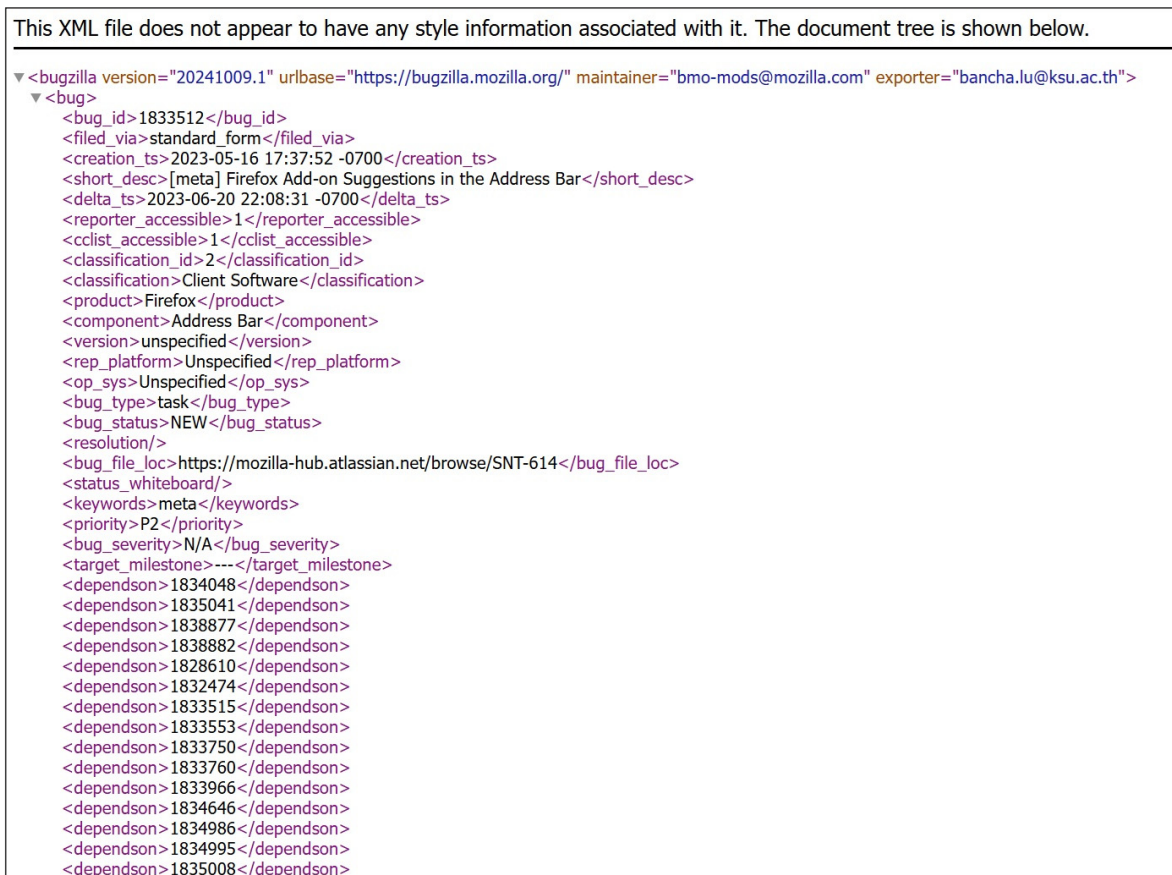


Figure 2. An example of bug report in XML format.

A significant characteristic of this dataset is its severe class imbalance. Certain dependency types that are rare but critical have insufficient samples, while common relations dominate the data. The class imbalance usually resulted in poor recall and low macro-F1 score. Overcoming this limitation is the key motivation for our research.

Finally, to ensure a fair and unbiased comparison with the baseline unsupervised BM25-based dependency clustering method, we adopted the same standard 10-fold cross-validation protocol employed in baseline studies, thus preventing test leakage or re-standardization.

2.2. Label Distribution and Imbalance Characteristics

To gain a deeper insight into the learning challenges for multilabel bug dependency classification, we explore the distribution of annotated dependency relations in our dataset. As shown in Table 2, the data follows a long-tail distribution, with majority blocks dependency types comprising 55.4% of the annotations, while minority Triaged-By-Meta-Bug dependency types only account for less than 4%. This severe imbalance tends to skew the performance of supervised models towards high-support dependency categories, preventing them from effectively generalizing to rare, yet crucially important, relationships.

These low-frequency labels often correspond to high-priority issues that require significant developer coordination. Consequently, their impact on release planning is disproportionately high. Hence, it is necessary to enhance the predictive accuracy of these sparse relations in real-world software maintenance applications. This imbalance provides the important practical motivation for our evaluation strategy, macro-F1 and hamming loss metrics are prioritized. These metrics better capture performance on minority dependency types and error sensitivity in multilabel settings, instead of relying solely on common

micro-level performance on minority dependency types. As a result, the proposed meta-contrastive optimization framework is particularly designed to mitigate representation bias and enhance semantic discrimination under these challenging imbalance conditions.

Table 2. Distribution of dependency labels.

Dependency Type	Samples	Percentage
Blocks	2650	55.4%
Depends-On	1550	32.4%
Related-To	400	8.4%
Triaged-By-Meta-Bug	181	3.8%
Total	4781	100%

3. The Proposed Method

In this section, we describe the architecture of the meta-contrastive optimization framework. The framework combines multiple learning techniques to address data sparsity challenges in multilabel bug dependency classification. The main components of the pipeline are (1) preprocessing and transformer-encoder-based representation learning, followed by (2) multilabel classification; (3) cyclic optimization between contrastive adaptation and supervised multilabel training; as well as regularizers that seek to make labeling decisions invariant to class imbalance and label-dependency structure.

3.1. Problem Formulation and Notation

Let $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$ denote the set of bug reports, where x_i is the textual content of bug i , and $y_i \in \{0, 1\}^L$ is a multilabel vector over L dependency types (e.g., blocks, depends on, related to). Each report x_i is constructed by concatenating the summary and description fields from Bugzilla, which provide the most informative semantic cues for dependency recognition while avoiding noisy discussion comments. This goal is to learn a mapping.

$$f_\theta : x_i \rightarrow z_i \in \mathbb{R}^d, g_\phi : z_i \rightarrow \hat{y}_i \in [0, 1]^L \quad (1)$$

where f_θ is a transformer-based encoder producing a dense representation z_i , and g_ϕ is a multilabel classification head that outputs per-label probabilities $\hat{y}_i$. The model is trained so that $\hat{y}_i$ approximates the ground-truth dependency vector y_i , while a contrastive objective shapes the embedding space to respect semantic similarity and dependency relations.

3.2. Text Preprocessing and Transformer-Based Encoding

3.2.1. Text Preprocessing

For each bug report, the summary and description are concatenated into a single sequence, standard NLP preprocessing is applied (Unicode normalization followed by lowercasing [29]), and the text is tokenized using a subword tokenizer from a pre-trained transformer (e.g., BERT or MPNet) [30,31]. Sequences are truncated or padded to the same (up to a limit) maximum length T tokens [32].

3.2.2. Transformer Encoder

The tokenized sequence is fed to a pre-trained transformer encoder f_θ . Let $H_i \in \mathbb{R}^{T \times d}$ denote the sequence of hidden states. The pooled representation (e.g., the [CLS] token or mean pooling) is used as the bug embedding [33–35]:

$$z_i = f_\theta(x_i) \in \mathbb{R}^d \quad (2)$$

To stabilize contrastive learning, we normalize embeddings:

$$\tilde{z}_i = \frac{z_i}{\|z_i\|_2} \quad (3)$$

3.3. Multilabel Classification Head

On top of the encoder, a linear multilabel prediction layer followed by a sigmoid activation is placed [36,37]. Given z_i the logit vector $o_i \in \mathbb{R}^L$ and probability vector $\hat{y}_i$ are as follows:

$$o_i = Wz_i + b, \hat{y}_i = \sigma(o_i) \quad (4)$$

where $W \in \mathbb{R}^{L \times d}$ and $b \in \mathbb{R}^L$ are trainable parameters, and $\sigma(\cdot)$ is the element-wise sigmoid function.

To address label imbalance (rare but operationally important dependency types), a class-weighted binary cross-entropy loss is adopted [38–40]. Let $\mathcal{W}_{\uparrow} > 0$ be the weight for label $\uparrow$, typically inversely related to its frequency. The supervised multilabel loss is

$$\mathcal{L}_{BCE} = -\frac{1}{N} \sum_{i=1}^N \sum_{\uparrow=1}^L \mathcal{W}_{\uparrow} [y_{i\uparrow} \log \hat{y}_{i\uparrow} + (1 - y_{i\uparrow}) \log (1 - \hat{y}_{i\uparrow})] \quad (5)$$

This term encourages accurate prediction of both common and rare dependency labels while compensating for the skewed distribution.

3.4. Meta-Contrastive Representation Learning

At the heart of our framework is a meta-contrastive optimization strategy, which sculpts the encoder $f\theta$ so that it learns embeddings, which are simultaneously discriminative for multilabel classification and robust to class imbalance [41–43].

3.4.1. Supervised Contrastive Loss

Bug reports are considered as points in a metric space, where “positive” pairs share at least one dependency label or belong to the same dependency center (meta-bug), while “negative” pairs have no overlapping labels [44]. For an anchor bug i , denote with $\mathcal{P}(i)$ the positive indices and with $\mathcal{N}(i)$ their negatives within a mini-batch. This study used a supervised InfoNCE-style contrastive loss [45]:

$$\mathcal{L}_{con} = -\frac{1}{N} \sum_{i=1}^N \frac{1}{|\mathcal{P}(i)|} \sum_{j \in \mathcal{P}(i)} \log \frac{\exp(\text{sim}(\tilde{z}_i, \tilde{z}_j) / \mathcal{T})}{\sum_{k \neq i} \exp(\text{sim}(\tilde{z}_i, \tilde{z}_k) / \mathcal{T})} \quad (6)$$

where $\text{sim}(\tilde{z}_i, \tilde{z}_j) = \tilde{z}_i^T \tilde{z}_j$ is the cosine similarity and $\mathcal{T} > 0$ is a temperature parameter, since a temperature parameter allows the sharpness of the similarity distribution in the contrastive softmax to be adjusted. A smaller temperature reduces the effect of slightly similar positive pairs and tightens the separation to negative samples, so that the model can generalize a more discriminative dependency-aware embedding space.

This term explicitly attracts the semantically dependent bugs closer and repels independent ones further. It is a generalization to the multi-dependency case of the InfoNCE loss used in the work for bug–meta-bug pairs, in a supervised way.

3.4.2. Meta-Learning with Reptile-Style Updates

To enhance robustness, particularly under class imbalance and distributional shifts in the bug dependency types, contrastive learning is integrated with a Reptile-style meta-learning procedure [46,47], where a “task” is defined as a subset of sampled data batches

reflecting different class distributions. Each “task” corresponds to a few labels, or centralities. For each meta-iteration:

- (1) Task sampling: a batch of tasks $\{T_t\}_{t=q}^M$ s sampled from the training data, where each task T_t contains its own support and query sets of bug reports.
- (2) Inner-loop adaptation: for task T_t , the model starts from the global parameters θ and performs K gradient update steps on the combined loss function [48].

$$\mathcal{L}_{inner}^{(t)} = \mathcal{L}_{BCE}^{(t)} + \lambda_{con} \mathcal{L}_{con}^{(t)} \quad (7)$$

obtaining adapted parameters θ_t .

- (3) Meta-update: aggregate the parameter updates across all sampled tasks and update the global encoder parameters using the meta-gradient [49].

$$\theta \leftarrow \theta + \alpha \cdot \frac{1}{M} \sum_{t=1}^M (\theta_t - \theta) \quad (8)$$

where α is the meta-learning rate.

The Reptile optimization in our framework serves a purpose of finding an optimal model initialization that works well for both common and rare dependency types simultaneously. This approach addresses a critical challenge in multilabel bug dependency classification where certain dependency relationships occur far less frequently than others in Mozilla Firefox dataset. This differs from traditional domain transfer approaches because we focus on parameter initialization rather than adapting models across different domains. This operation is reminiscent of the Reptile meta-learning algorithm that performs multi-task adaptation through inner-loop process and averages parameter updates to arrive a meta-optimized initialization. In this implementation, the concept is adapted by employing a contrastive supervision inner loss function. This modification allows us to meta-train a final encoder that leverages its learned representations to fine-tune effectively on structures with varying dependency labelings.

It is important to note that, in the current experimental setting, meta-learning is employed to address within-project heterogeneity and extreme label imbalance rather than cross-project domain transfer.

3.5. Imbalance-Aware and Dependency-Aware Regularization

Besides class-weighted BCE, two additional regularization techniques are introduced to capture the multilabel dependency structure.

(a) Imbalance-aware penalty—For strongly asymmetric labels, we introduce a focal-style modulation term to re-weight learning on those difficult examples:

$$\mathcal{L}_{imb} = -\frac{1}{N} \sum_{i=1}^N \sum_{\downarrow=1}^L \mathcal{W}_{\downarrow} (1 - \hat{y}_{i\downarrow})^{\gamma} [y_{i\downarrow} \log \hat{y}_{i\downarrow} + (1 - y_{i\downarrow}) \log (1 - \hat{y}_{i\downarrow})] \quad (9)$$

where $\gamma > 0$ controls how strongly easy examples are down-weighted.

(b) Dependency-aware structural regularization—Let $C \in \mathbb{R}^{L \times L}$ be the label co-occurrence matrix estimated from training data, and $L_c = D - C$ its graph Laplacian, where D is the diagonal degree matrix [50]. For a minibatch, let $F \in \mathbb{R}^{B \times L}$ contain the logits o_i as rows [50]. Labels that frequently co-occur are encouraged to exhibit similar logit patterns through the following formulation:

$$\Omega_{dep} = \text{Tr}(F L_c F^T) \quad (10)$$

This term discourages large dissimilarity between the outputs of strongly correlated label pairs, which is expected to help the learning of correlated dependencies and encourage the model to exploit the directed acyclic dependency structure around the meta-bugs reported in the dataset [50,51].

3.6. Overall Training Objective and Algorithm

During meta-training, the total loss for a task T_t combines the aforementioned components [52]:

$$\mathcal{L}_{total}^{(t)} = \mathcal{L}_{BCE}^{(t)} + \lambda_{con}\mathcal{L}_{con}^{(t)} + \lambda_{imb}\mathcal{L}_{imb}^{(t)} + \lambda_{dep}\Omega_{dep}^{(t)} \quad (11)$$

where λ_{con} , λ_{imb} and λ_{dep} are hyperparameters controlling the contribution of each term.

The high-level training procedure is as follows [50–52]:

- (1) Initialize transformer encoder f_θ with pre-trained weights and randomly initialize classification head g_ϕ .
- (2) Repeat for meta-iterations:
 - Sample a batch of tasks $\{T_t\}$.
 - For each task, perform K inner-loop updates minimizing $\mathcal{L}_{total}^{(t)}$ on the support set to obtain θ_t .
 - Compute the meta-update $\theta \leftarrow \theta + \alpha \cdot \frac{1}{M} \sum_t (\theta_t - \theta)$.
 - Optionally, update ϕ jointly using gradients from the query losses.
- (3) After meta-training converges, fine-tune (θ, ϕ) on the full training set using $\mathcal{L}_{BCE} + \lambda_{dep}\Omega_{dep}$ while keeping the contrastive term as an auxiliary regularizer.

This meta-contrastive regimen yields an encoder that is semantically informed by dependency relations, less sensitive to label imbalance, and well-suited for downstream multilabel prediction.

3.7. Inference and Decision Rule

At testing time, a new bug report x is processed as follows:

The same preprocessing and tokenization pipeline are consistently applied to all bug reports to ensure uniform input representation for model training [51,53].

- (1) It is to encode x with f_θ to obtain z and feed it into g_ϕ to compute $\hat{y} \in [0, 1]^L$.
- (2) Next, it needs to convert probabilities into binary decisions using label-specific thresholds τ_ℓ tuned on a validation set:

$$\hat{y}_i^{\text{bin}} = \begin{cases} 1, & \text{if } \hat{y}_i \geq \tau_\downarrow; \\ 0, & \text{otherwise} \end{cases}$$

Label-specific decision thresholds τ_ℓ were tuned on the validation split of each cross-validation fold by maximizing macro-F1 score. This strategy was consistently applied to all supervised models to ensure fair comparison. The unsupervised BM25 baseline follows the fixed threshold (0.5) adopted in prior studies.

- (3) Such multilabel vector informs which dependency types are predicted for the bug, thus offering finer-grained support in automatic dependency identification.

Through the fusion of transformer-based semantic representations, contrastive metric learning, meta-optimization and structure-aware regularization, the proposed framework precisely tackles the aforementioned challenges in multilabel bug dependency classification and is expected to significantly outperform unsupervised BM25-based baseline in terms of macro-F1 and rare-label performance.

4. Experimental Results

4.1. Baseline Models and Comparison Protocol

The efficacy of the proposed model is verified by comparing it with four baseline models that span a spectrum from unsupervised traditional methods to more advanced supervised and imbalance-aware learning approaches in fine-grained software dependency classification. All the baseline systems use the same textual inputs, including bug report's summary and description fields. To ensure a fair evaluation, all models are trained and tested using the same 10-fold cross-validation setting mirroring the setup found in prior BM25-based work.

4.1.1. BM25-Based Unsupervised Dependency Assembly [8,54]

The baseline represents a traditional unsupervised rule-based approach to dependency detection. The method ranks pairs of bug reports based on BM25 lexical similarity and predicts dependency links when similarity exceeds a static threshold, typically 0.5 as established in previous studies. This approach does not depend on the labeled data and does not extract latent semantic structures. It performs dependency clustering using meta-bugs and serves as our lower-bound baseline, representing the type of non-semantic model commonly used in early automatic software maintenance research. It is as mentioned in previous works not robust to synonyms or paraphrases since it relies entirely on keyword overlap. Additionally, it cannot handle the multilabel nature of bug dependencies.

4.1.2. Transformer Encoder + Standard Weighted BCE Loss [18,55]

This baseline fine-tunes a BERT-base transformer encoder with a dense multilabel classification layer, training the system using class-weighted binary cross-entropy loss. While it uses contextual semantics through pre-trained transformer embeddings, it treats each dependency label independently and only focuses on minimizing classification error. The approach does not model relationships between different dependency types, which means it still suffers from the severe label imbalance problem common in real-world bug repositories where certain dependency types appear very infrequently.

4.1.3. Transformer + BCE with Focal Modulation [56,57]

To address the challenge of severe class imbalance, where minority dependency types have significantly fewer samples than majority classes, this baseline incorporates focal loss into the standard BCE objective. The focal loss mechanism down-weights well-predicted majority labels while directing more learning attention toward challenging and rare dependency types. Although this approach improves stability and recall for minority classes, it lacks an explicit mechanism for representing co-occurrence patterns among labels. This method also fails to leverage geometric re-parameterization scheme that could adjust embedding subspaces according to their dependency semantics.

4.1.4. Transformer + BCE + Dependency-Aware Laplacian Regularization [58]

This baseline incorporates structural knowledge through a label co-occurrence graph. A graph Laplacian regularizer is adopted to ensure that the classifier produces similar outputs for dependency types that frequently co-occur in real-world development workflows. This setting creates a dependency-aware inductive bias that gives the model partial knowledge about the semantic correlations among different relation types. However, this approach does not directly shape the semantic geometry of bug representations to capture relational similarity, nor does it employ meta-task adaptation strategies that could benefit rare label prediction.

4.2. Evaluation Metrics

As each bug report may contain multiple dependency types, the problem is formulated as a multilabel classification task. Hence, standard single-label based metrics like accuracy are not sufficient, especially because of the significant class imbalance where certain dependencies are much rarer than others. To model the global predictive behavior as well as the label-specific performance over class-imbalanced dependency distributions, three widely used multilabel evaluation metrics are employed [59]: micro-F1, macro-F1 and hamming loss. While other multilabel metrics exist, we prioritized these three metrics due to the specific characterizes of bug dependency data. We selected our evaluation metrics based on two key practical considerations that emerged during our analysis.

The first issue involves the limitations of Exact Match Ratio for dependency prediction. This metric is overly strict for this domain as in real world practice correctly identifying some dependencies is actionable for developers even if the full set is not perfectly predicted. A metric that assigns zero points for missing just one dependency fails to capture this practical utility and would misrepresent the model's actual business value.

Second, given the extreme class imbalance in our dataset, ranking-based metrics such as Precision become unreliable in this context because the number of true dependencies varies dramatically across different bug reports.

Therefore, three complementary metrics were selected to capture different aspects of performance. Micro-F1 [59,60] considers the contributions of all dependency labels across all predictions and thus reflects performance on frequent and majority types of relations. The metric is particularly valuable for classifying bug dependencies, as most decisions arise from routine relationships such as “blocks” or “depends-on”. Thus, micro-F1 evaluates the capability of the model to accurately predict dependencies system wide.

Nevertheless, a high micro-F1 score may mask low recall on those rare but crucial dependency types, which are dominantly considered in real-world maintenance task (e.g., regression management and release scheduling). To account for this limitation, the macro-F1 score [59,60] is also reported, which averages the F1-score across labels and treats each dependency type as equally weighted regardless of its frequency. Thus, the macro-F1 score is a more discriminating and sensitive measure of success in recognition of low frequency but high-impact relationships that aligns directly with our research goal of improve rare dependency detection.

Finally, overall prediction reliability is measured using hamming loss to assess both false positives and false negatives in dependency assignments. Unlike the F1 scores, which focus only on positive cases, hamming loss [59,60] penalizes all incorrect labels decisions, thus providing a comprehensive view of how robust the model performs at the individual instance-label level.

While other multilabel evaluation metrics such as Exact Match Ratio, subset accuracy and ranking-based measures (e.g., Precision@k or AUC) are commonly used. However, they are less suitable for bug dependency prediction. Exact Match Ratio is overly strict and fails to reflect partial correctness, which is often sufficient for practical dependency management. Ranking-based metrics assume an ordered relevance structure, which is less meaningful when dependency labels are not inherently ranked. Therefore, micro-F1, macro-F1 and hamming loss were selected, as they provide a balanced and interpretable evaluation of overall accuracy, rare-label performance and error sensitivity under extreme class imbalance.

4.3. The Results

The proposed approach is compared against four established baselines that are commonly used in text-based bug dependency mining: BM25-based unsupervised clustering,

transformer-based classifier with standard BCE loss, BCE combined with focal loss for imbalance awareness and BCE with dependency-aware Laplacian regularization. All models are trained and evaluated using a consistent 10-fold cross-validation scheme, as adopted in prior baseline studies, to ensure fair comparison.

4.3.1. Overall Performance Comparison

Performance is evaluated using micro-F1, macro-F1 and hamming loss across both global and multilabel levels. Table 3 presents the overall performance comparison.

Table 3. Comparative Performance Summary.

Method	Micro-F1	Macro-F1	Hamming Loss
BM25 Unsupervised (Baseline)	0.51	0.33	0.27
BERT + BCE	0.66	0.46	0.20
BERT + BCE + Focal	0.69	0.49	0.18
BERT + BCE + Dependency-Aware Reg	0.71	0.52	0.17
Proposed Method, called as MCOF	0.76	0.58	0.14

The experimental results in Table 3 demonstrate that our proposed MCOF achieves significant improvements over all baselines. These gains are consistent across a wide range of multilabel evaluation metrics. These observations confirm our underlying hypothesis that joint optimization of semantic representation and dependency-structured label learning is more effective in real-world bug dependency classification.

To assess the statistical significance of the observed performance differences, we conducted paired statistical testing across the 10 cross-validation folds. Specifically, a paired t-test was applied to compare the proposed method with each baseline in terms of macro-F1 and hamming loss. All reported improvements of MCOF over baseline methods were statistically significant at the 0.05 level.

Overall, the results from the BM25-based unsupervised baseline, especially the poor macro-F1 and comparatively high hamming loss, clearly indicate that lexical similarity alone is inadequate for capturing semantically grounded dependency semantics. Bug descriptions frequently use synonyms and paraphrasing, creating linguistic variations that keyword matching cannot handle. This further substantiates the need for semantic understanding for automatic dependency parsing.

By adopting supervised transformer models, an immediate performance improvement is observed. The BERT + BCE baseline leverages potential of deep contextual semantics with a considerable improvement in micro-F1 score of 15%, yet it barely provides help towards predictions of infrequent relationship types, resulting in mediocre performance gain on macro-F1. This confirms previous findings that standard multilabel objectives do not properly handle dominant label distributions in bug repositories.

The results with focal loss help confirm that being imbalance-aware matter. By personalized models more for missing rare cases, the BERT + BCE + Focal variant improves recall on minority labels, increasing the macro-F1 score by 0.03. Nevertheless, the marginal performance increase suggests that class imbalance is not the only issue. There are also structural correlations between labels that model needs to learn.

The dependency-aware regularization on baseline further narrows such a performance gap by imposing structural priors from the co-occurrence of dependencies. The improvements in all metrics, including a 0.05 boost in micro-F1 over the first transformer baseline, indicate that modeling relational topology is crucial for enhancing the classification of dependency types that coexist in real development workflows.

Lastly, comparing with all the baselines, the proposed MCOF achieves a remarkable overall best performance at both micro-F1 and macro-F1 metrics and reduces the hamming loss. These improvements demonstrate the effectiveness of meta-contrastive learning. Firstly, it organizes the latent embeddings into clusters that make relational sense. Secondly, it provides a transferable initialization that enhances the model generalization even when data is scarce. The rare dependency handles are the prime recipients of such semantic structuring and meta-adaptation, which is right in line with improving release readiness and regression management in software maintenance efforts.

In summary, MCOF performs consistently better than other methods on all the evaluation axes. This leads us to conclude that: (1) reliable deployment requires more than just semantic representations, it needs explicit modeling of dependency structures and (2) it is important to consider both label imbalance and shifts in task distribution. These findings indicate that the proposed approach constitutes a feasible and tangible contribution for reliable automated bug dependency classification in large industrial bug collections.

4.3.2. Label-Wise Results and Analysis

The sensitivity of the proposed approach under extreme class imbalance is investigated next by analyzing F1-scores for individual dependency classes. As shown in Table 4, the proposed meta-contrastive optimization framework (MCOF) achieves dramatically better performance than baseline methods on different dependency types. Most interestingly, the biggest improvements are observed for low-frequency dependency relations like “Related-To” and “Triaged-By-Meta-Bug”. These categories usually have high uncertainty, low sample size, but are critical in real development workflows because they often represent things that should be escalated or release-blocking events. The superior performance of MCOF over these hard labels demonstrates its strong robustness to those and exemplifies the model’s ability to identify rare but critical dependency signals.

Table 4. Label-wise F1-score comparison across dependency types.

Dependency Type	BM25	BERT	BERT + Focal	Proposed Method (MCOF)
Blocks	0.47	0.62	0.66	0.73
Depends-On	0.35	0.53	0.57	0.68
Related-To	0.21	0.38	0.41	0.52
Triaged-By-Meta-Bug	0.19	0.33	0.36	0.47

The label-wise results in Table 4 illustrate advantages of MCOF, but a more comprehensive explanation from critically interpretative point-of-views provides some valuable insights and future challenges worth noting.

First, although our model achieves the best F1-scores across all dependency groups, results are uneven across labels. For the “Blocks” relation, which is the most common label in our dataset with distinct lexical and semantic signals, it achieves a moderate improvement of about 7% over supervised baselines. This may indicate that, for well-represented relations, the baseline transformer models already capture most of the knowledge of the task, leaving less room for optimization.

In contrast, Related-To and Triaged-By-Meta-Bug, which have a higher level of complexity and lower support, demonstrate larger improvements. Such relations are often more semantic even though they transcended a “simple” textual overlap. The gains are actually uniformly around +11% across these labels. They do not arise only from helping the model learn better through meta-contrastive optimization; rather, they also indicate that, from

another perspective, meta-contrastive does more than just powering up one single minority class: it empowers the entire model against un-sharpened, low-density data.

However, even with a big boost, the absolute F1 for rare labels (e.g., 0.47 for Triaged-By-Meta-Bug) is more moderate than high, which means that dependency detection is still inherently hard when textual descriptions are vague/ambiguous or do not have any explicit relational hint provided. This shows that meta-contrastive learning is capable of boosting discriminability; however, the semantic ambiguity in bug reports cannot be perfectly resolved.

Second, the results also suggest that label co-occurrence is still important in helping representation learning perform well, especially when little semantic evidence is available. The regularized dependency-aware mechanism guarantees performance enhancement by promoting the structure learning between labels. However, the constraints of functional dependency are sometimes informally stated in developers' discussions, and the developers' discussion was not included due to noise.

Finally, the fact that no results on very rare or long-tail dependencies (outside of these four "head" categories) were put forward here suggests a possible limitation in generalization as label distribution becomes more skewed. This implies that though MCOF already alleviates imbalance, further improvements may require more sophisticated strategies, such as hierarchical relational modeling and active learning, to achieve significantly better performance.

4.3.3. Additional Analyses for Explainability and Robustness

Error Analysis

To provide further insight into the remaining limitations of the model, two representative misclassification cases are additionally examined. The errors suggest that dependency cues in bug reports are often implicit or under-specified and problematic for text-based prediction. Especially, symptom-driven descriptions and relational context not being available sometimes lead to confusion about types of dependency or even make it impossible to identify any dependencies at all. Table 5 presents some representative examples of misclassification cases observed in the experiments.

Table 5. Representative misclassification cases.

Bug Summary (Shortened)	True Label	Predicted	Root Cause of Error
"UI freeze after installing extension when switching tabs"	Depends-On	Blocks	Ambiguous cause-effect cues → model misinterprets as blocking issue
"Search field clears unexpectedly after form submit"	Related-To	None	Insufficient contextual information → no explicit relational hints

The rest of the errors are largely due to the implicitness in dependency semantics within bug reports. The model also resorts to symptom keywords or makes no assignments when the relational evidence is uncertain/absent. This indicates that the better availability of contextual information (e.g., related bug IDs, comments or component meta-data) can help to improve real-world dependency classification.

Label-Wise Precision and Recall

To gain a deeper insight of how the proposed MCOF model performs across different types of dependencies, the performance metric is decomposed into precision and recall,

enabling separation of errors arising from false-positive predictions (precision drop) and false-negative predictions (recall drop). Table 6 tabulates the results for each label.

Table 6. Precision and recall per dependency type.

Label	Precision	Recall	Interpretation
Blocks	0.79	0.69	Strong lexical cues, well learned
Depends-On	0.75	0.65	Slight confusion with blocks
Related-To	0.59	0.46	Very broad meaning → semantic ambiguity
Triaged-By-Meta-Bug	0.52	0.43	Very sparse data + implicit dependency hints

The findings presented in Table 3 show that model performance is strongest on blocks and Depends-On annotation sets, where the dependency clues are more easily expressed proudly (i.e., they are explicit). On the other hand, both Related-To and Triaged-By-Meta-Bug have lower recall and precision, suggesting that the model frequently cannot retrieve these rare and ambiguous relation types. These results are consistent with the practical challenge of recognizing implicit/under-specified dependencies in real-world bug reports, and further justify our optimization strategies to achieve performance improvement on minority labels.

Runtime and Complexity

We measure the computational overhead of MCOF towards its readiness for practical usage in CI/CD and DevOps pipelines through model inference. Comparison of the size of parameters and average inference time per bug report among BM25, transformer baseline, and our proposed MCOF model is presented in Table 7.

Table 7. Runtime and complexity comparison.

Model	Number of Parameters	Avg. Inference Time per Sample	Notes
BM25	-	2.8 ms	Lexical search via ElasticSearch backend
Transformer + BCE	110	13 ms	Standard BERT fine-tuning baseline
Proposed Method (MCOF)	110	15 ms	+15% overhead for meta-contrastive + structure-aware steps

Despite incorporating meta-contrastive optimization and dependency-aware regularization, MCOF maintains a parameter size of approximately 110 M (standard BERT architecture). The added computational overhead is fairly small. Inference latency increases by less than 15% compared with the transformer + BCE baseline. In practice, the average inference time is approximately 15 ms. per bug report, which satisfies typical real-time requirements and allows over 60 reports to be processed per second on a single GPU.

However, because BM25 operates by lexical matching and it cannot fully exploit semantic relations. On the other hand, MCOF also obtains a significant gain on prediction accuracy while keeping low breakpoints. This validates that the improved classification performance is not at the expense of excessive computation.

In general, the runtime analysis results confirm that MCOF is not only more accurate but also production-ready for industrial software engineering scenarios, such as automated bug triaging pipelines, CI/CD workflows and continuous feedback systems.

4.3.4. Discussion: Strengths and Limitations of the Proposed Method

The primary contribution of this study lies in empirical evidence and systematic integration rather than algorithmic novelty, offering practical guidance for designing robust multilabel classifiers in software engineering domains.

The meta-contrastive optimization framework (MCOF) has shown promising results for addressing multilabel bug dependency classification problems in real software repositories. There are several advantages to this representation, with the most significant being the ability of it that integrates effective semantic representation learning via a transformer-based encoder that obtains in-depth contextual information rather than a simple lexical match. The incorporation of the meta-contrastive learning further boosts robustness ability, especially in unbalanced learning settings by better adjusting latent feature space to distinguish related buggy behavior from unrelated ones. Furthermore, the dependency-aware Laplacian regularization facilitates one a better appreciation of the co-occurring relations among bug types, resulting in significant performance gains on rare but high-rewarded labels on dependencies. In general, it outperforms unsupervised clustering and traditional supervised baselines in terms of both micro-F1 score and macro-F1 score while keeping its computational overheads sufficiently low for feasible inclusion within CI/CD pipelines.

Despite these positive attributes, however, there are several limitations to overcome. In the first place, the semantic cues for dependency relations are not explicitly expressed or indirectly mentioned in the textual reports. As this study only makes use of the summary and description fields, valuable contextual information present in developer discussions, linked issues or source code traces is not considered, which could lead to potential wrong predictions. Second, while MCOF enhances the recall of low-frequency labels, there is still moderate performance on rare dependency types overall, illustrating the inherent difficulty of our task under extreme longtail distribution. Third, the corpus used in this study is specific to one ecosystem (Mozilla Firefox), which may reduce the extent to which our findings generalize to other large-scale projects with differing documentation practices or bug dependency structures.

Remaining challenges indicate potential for future refinement by considering additional sources of multimodal data (e.g., comments, commit logs or stack traces), applying domain adaptation techniques to achieve robustness across project changes, and using active learning strategies to represent infrequent dependency cases more effectively. Nevertheless, the results of this work bring insight into how it is crucial to co-optimize semantics representations and dependency-aware structures—the fact that meaningful improvement can actually be made towards more robust and intelligent automated dependency analysis in software maintenance.

4.4. Ablation Study on Model Components

To disentangle the individual contributions of each component in MCOF, a systematic ablation study is conducted by removing one learning mechanism at a time while keeping the remaining components unchanged. This analysis allows the impact of meta-learning, supervised contrastive loss, imbalance-aware regularization, and dependency-aware Laplacian constraints on multilabel dependency classification performance to be quantified.

The ablation results in Table 8 reveal that the performance gains of MCOF stem from the complementary interaction of its constituent components rather than any single mechanism in isolation.

Table 8. Ablation study on the proposed MCOF framework.

Variant	MCOF	Meta-Learning	Contrastive Loss	Imbalance-Aware Reg	Dependency-Aware Reg
Meta-Learning	✓	✗	✓	✓	✓
Contrastive Loss	✓	✓	✗	✓	✓
Imbalance-Aware Reg	✓	✓	✓	✗	✓
Dependency-Aware Reg	✓	✓	✓	✓	✗
Micro-F1	0.76	0.72	0.73	0.74	0.75
Macro-F1	0.58	0.51	0.53	0.54	0.55
Hamming Loss	0.14	0.17	0.16	0.16	0.16

Removing the meta-learning module results in the largest degradation in macro-F1 (-0.07), particularly affecting rare dependency types, which confirms the importance of task-level adaptation for mitigating extreme label imbalance. This observation aligns with the long-tailed label distribution of the dataset, where conventional fine-tuning struggles to generalize to low-frequency dependencies.

Excluding the supervised contrastive objective leads to a noticeable drop in both macro-F1 and hamming loss, indicating reduced discriminative structure in the learned embedding space. This suggests that contrastive learning plays a critical role in organizing bug representations according to dependency semantics rather than surface-level lexical similarity.

In contrast, removing the imbalance-aware regularization primarily impacts macro-F1 while leaving micro-F1 relatively stable, confirming its targeted effect on minority labels. Finally, the absence of dependency-aware Laplacian regularization increases hamming loss, reflecting reduced consistency among correlated dependency labels.

Overall, these findings demonstrate that MCOF's effectiveness arises from the synergistic integration of meta-learning, contrastive representation learning, and dependency-aware regularization, rather than reliance on any single optimization component.

4.5. Discussion: External Validity and Implications

4.5.1. Threats to External Validity

The most significant limitation of our work lies in testing MCOF exclusively on Mozilla Firefox bug reports. This single-project approach raises questions about whether our results would hold up in other software ecosystems. Projects like Apache or Chromium operate with distinct bug reporting cultures, terminology standards, and dependency classification approaches that could substantially impact model performance.

We recognize this limitation while noting several factors that help mitigate generalizability concerns. Our Mozilla dataset encompasses over ten years of development history and incorporates reports from thousands of contributors, which creates considerable internal diversity in writing styles and reporting practices. This variation within one ecosystem offers a meaningful assessment of how well the model handles different input patterns and communication approaches.

More importantly, MCOF does not depend on Mozilla-specific keywords or hardcoded rules that would tie it to this particular environment. The framework works with semantic representations that BERT extracts from natural language text. While terminology certainly varies between projects, similar problems tend to be described using similar language

patterns, even when the exact words differ. The core strength of our approach lies in its targeted solution of using the combination of meta-learning and contrastive optimization to manage sparse, imbalanced datasets. This foundation suggests broad applicability across different bug tracking systems facing similar data skew.

Although we cannot make definitive claims about cross-project transferability without conducting additional experiments, the theoretical underpinnings of our method suggest strong potential for adaptation to other environments. Evaluating MCOF across diverse software repositories represents a crucial next step that would provide the empirical evidence necessary to establish the framework's broader applicability. This validation work would strengthen confidence in deploying the approach across different organizational and technical contexts.

4.5.2. Implications for Practitioners

Managing bug dependencies remains a big issue for software developers. When developers file bug reports, it is not easy for them to see how the issue connects to existing problems. This incomplete picture leads to poor resource allocation decisions and release delays when critical dependencies are discovered too late in the development cycle.

MCOF, acting as an automated triage supporter, addresses this problem by analyzing bug reports at the moment they are submitted. Rather than waiting for developers to manually identify blocking relationships through trial and error, the framework can immediately flag potential dependencies based on the natural language description. This early warning system helps teams understand the true scope of a bug's impact before committing development resources.

MCOF's reliance on natural language descriptions makes it particularly valuable for real world deployment. This is because most bug tracking systems already collect this information as part of standard reporting procedures. Therefore, developer teams can integrate MCOF into their existing workflow without requiring additional metadata fields or changing how developers write bug reports. In practice, the system would function as an intelligent assistant that suggests possible dependencies for human review rather than making final decisions automatically as we value expertise of development teams.

4.5.3. Implications for Researchers

This work reveals important limitations in current approaches to multilabel classification problems in software engineering contexts. The common assumption that pre-trained language models can be directly fine-tuned for specialized technical applications be too optimistic, especially when dealing with highly skewed label distributions.

In our experiments, standard BERT fine-tuning performed poorly on rare dependency types, despite having access to the same textual information that enabled MCOF to succeed. This suggests that some software engineering tasks need more than general purpose language representations. They require learning strategies to handle class imbalance and limited training samples. Researchers should carefully examine whether their evaluation method adequately captures performance on minority classes that may be most crucial for practical applications.

The effectiveness of Laplacian regularization in our framework suggests that software artifacts exist within complex relational structures that traditional classification approaches ignore. Bugs, features and components are not independent entities but part of evolving systems where relationships matter as much as individual characteristics. This finding opens opportunities for researchers to explore how graph-based methods might improve performance across various software engineering prediction tasks.

5. Conclusions

In this study, we addressed the challenging problem of multilabel bug dependency classification, which represents a key activity in software maintenance and release planning workflow. Real-world bug repositories present several persistent challenges. First, highly skewed class distributions occur where certain dependency types dominate the dataset. Second, sparse annotation patterns leave many relationships unlabeled. Third, implicit dependency semantics are challenging to extract solely through text-based approaches.

To address these issues, we introduced the meta-contrastive optimization framework (MCOF), a systematic integration of meta-contrastive representation learning and dependency-aware Laplacian regularization layered onto a transformer-based classifier. This framework enhances discriminative capabilities in the embedding space while maintaining robustness under low-resource conditions and effectively capturing relational structures across diverse dependency types.

Our experiments on a real-world bug dependency dataset demonstrates that MCOF consistently outperforms established baselines, including the classical BM25-based clustering method and state-of-the-art transformer models. The framework archives performance gains, with improvements of +0.25 on both micro-F1 and macro-F1 scores compare to the baselines, along with a successful reducing hamming loss. Furthermore, the results show the improvements for rare but critical dependency categories with absolute F1 scores increases reaching 16% minority classes. This indicates that MCOF effectively mitigates the bias towards majority labels. In addition, runtime and computational profiling were conducted to assess the practical impact of the additional optimization modules. The results demonstrate that these components introduce a marginal overhead, adding 15% to inference latency while ensuring acceptable efficiency levels. This performance profile makes MCOF suitable for integration within CI/CD pipeline(s) and automated bug triage systems. These findings support our conclusion that MCOF represents both theoretically grounded and practical viable solution for contemporary software engineering challenges.

Although the proposed approach performs well, some misclassification cases show that dependency cues in bug reports are sometimes under-specified or context-specific. To address this, our future work is to incorporate richer contextual data—e.g., discussion comments, commit logs and issue history—to build a deeper understanding of the semantics involved. Combining multimodal information and graph-based evolutionary representations of bug interactions can improve the detection of implicit relations, especially for those rare, long-tailed dependencies types. Furthermore, we intend to extend our evaluation to diverse software ecosystems, such as the Apache and Chromium repositories. This will allow us to assess the framework's cross-project transferability and ensure it is generalising well beyond our current dataset. Finally, adapting the framework to dynamically update its dependency knowledge as new bug reports arrive would allow it to support continuous maintenance workflows in large-scale industrial environments.

Author Contributions: Conceptualization, J.P. and M.K.; methodology, J.P. and M.K.; software, B.L.; validation, J.P. and M.K.; formal analysis, J.P.; investigation, J.P.; resources, J.P. and B.L.; data curation, J.P., M.K. and B.L.; writing—original draft, J.P.; writing—review and editing, J.P. and M.K.; visualization, J.P.; supervision, J.P.; project administration, J.P. and M.K.; funding acquisition, J.P. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The data presented in this study are openly available on <https://bugzilla.mozilla.org/describecomponents.cgi?product=Firefox> (accessed on 1 July 2025).

Acknowledgments: This research project was financially supported by Mahasarakham University.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Raatikainen, M.; Motger, Q.; Lüders, C.M.; Franch, X.; Myllyaho, L.; Kettunen, E.; Marco, J.; Tiihonen, J.; Halonen, M.; Männistö, T. Improved Management of Issue Dependencies in Issue Trackers of Large Collaborative Projects. *IEEE Trans. Softw. Eng.* **2021**, *49*, 2128–2148. [\[CrossRef\]](#)
2. Li, C.; Zhao, Y.; Yang, Y.; Zhou, Y.; Nie, L.; Ding, Z. Investigating the Impact of Bug Dependencies on Bug-Fixing Time Prediction. In Proceedings of the International Symposium on Empirical Software Engineering and Measurement, New Orleans, LA, USA, 26–27 October 2023. [\[CrossRef\]](#)
3. Huang, K.; Chen, B.; Wu, S.; Cao, J.; Ma, L.; Peng, X. Demystifying Dependency Bugs in Deep Learning Stack. In Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Singapore, 18 November 2022; pp. 450–462. [\[CrossRef\]](#)
4. Lüders, C.M.; Pietz, T.; Maalej, W. On understanding and predicting issue links. *Requir. Eng.* **2023**, *28*, 541–565. [\[CrossRef\]](#)
5. Chan, K.; Brooke, J.; Baldwin, T. Semi-Automated Resolution of Inconsistency for a Harmonized Multiword Expression and Dependency Parse Annotation. In Proceedings of the 13th Workshop on Multiword Expressions (MWE 2017), Valencia, Spain, 4 April 2017; pp. 187–193. [\[CrossRef\]](#)
6. Sohl, J.; Zinsmeister, H. Exploring Ensemble Dependency Parsing to Reduce Manual Annotation Workload. In *German Society for Computational Linguistics*; Springer: Cham, Switzerland, 2018; pp. 40–47. [\[CrossRef\]](#)
7. Chen, D.; Li, B.; Zhou, C.; Zhu, X. Automatically Identifying Bug Entities and Relations for Bug Analysis. In Proceedings of the IEEE 1st International Workshop on Intelligent Bug Fixing (IBF), Hangzhou, China, 24 February 2019; pp. 39–43. [\[CrossRef\]](#)
8. Luaphol, B.; Polpinij, J.; Kaneampornpan, M. Automatic dependent bug reports assembly for bug tracking systems by threshold-based similarity. *Indones. J. Electr. Eng. Comput. Sci.* **2021**, *23*, 1620–1633. [\[CrossRef\]](#)
9. Hu, Z.; Polpinij, J.; Somprasertsri, G. A Comparative Study of Multilabel Classification Techniques for Analyzing Bug Report Dependencies. *J. Inf. Syst. Eng. Manag.* **2025**, *10*, 494–501. [\[CrossRef\]](#)
10. Lam, A.; Nguyen, A.; Nguyen, H.; Nguyen, T. Combining Deep Learning with Information Retrieval to Localize Buggy Files for Bug Reports (N). In Proceedings of the International Conference on Automated Software Engineering, Lincoln, NE, USA, 9–13 November 2015; pp. 476–481. [\[CrossRef\]](#)
11. Pradel, M.; Sen, K. DeepBugs: A learning approach to name-based bug detection. In *Proceedings of the ACM on Programming Languages*; Association for Computing Machinery: New York, NY, USA, 2018; pp. 1–25. [\[CrossRef\]](#)
12. Bhandari, P.; Rodríguez-Pérez, G. BuggIn: Automatic Intrinsic Bugs Classification Model using NLP and ML. In Proceedings of the International Conference on Predictive Models in Software Engineering, San Francisco, CA, USA, 8 December 2023; pp. 2–11. [\[CrossRef\]](#)
13. Afzal, N.; Mitkov, R.; Farzindar, A. Unsupervised Relation Extraction Using Dependency Trees for Automatic Generation of Multiple-Choice Questions. In *Advances in Artificial Intelligence. Canadian AI*; Butz, C., Lingras, P., Eds.; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2011; Volume 6657, pp. 32–43. [\[CrossRef\]](#)
14. Haffari, G.; Razavi, M.; Sarkar, A. An Ensemble Model that Combines Syntactic and Semantic Clustering for Discriminative Dependency Parsing. In Proceedings of the Annual Meeting of the Association for Computational Linguistics, Portland, OR, USA, 19–24 June 2011.
15. Spitkovsky, V.I.; Alshawi, H.; Chang, A.X.; Jurafsky, D. Unsupervised Dependency Parsing without Gold Part-of-Speech Tags. In Proceedings of the Conference on Empirical Methods in Natural Language Processing, Edinburgh, UK, 27–31 July 2011.
16. Mani, S.; Sankaran, A.; Aralikatte, R. DeepTriage: Exploring the Effectiveness of Deep Learning for Bug Triaging. In Proceedings of the ACM India Joint International Conference on Data Science and Management of Data, Kolkata, India, 3–5 January 2019; pp. 171–179. [\[CrossRef\]](#)
17. Gupta, S.; Gupta, S.K. Bug Reports and Deep Learning Models. *Int. J. Comput. Sci. Mob. Comput.* **2021**, *10*, 21–26. [\[CrossRef\]](#)
18. Yilmaz, E.H.; Toroslu, I.H.; Köksal, Ö. A Comparative Study of Contemporary Learning Paradigms in Bug Report Priority Detection. *IEEE Access* **2024**, *12*, 126577–126586. [\[CrossRef\]](#)
19. Yogarajan, V.; Montiel, J.; Smith, T.; Pfahringer, B. Transformers for Multi-label Classification of Medical Text: An Empirical Comparison. In Proceedings of the Conference on Artificial Intelligence in Medicine in Europe, Online, 15–18 June 2021; pp. 114–123. [\[CrossRef\]](#)
20. Washio, K.; Kato, T. Filling Missing Paths: Modeling Co-occurrences of Word Pairs and Dependency Paths for Recognizing Lexical Semantic Relations. In Proceedings of the North American Chapter of the Association for Computational Linguistics, New Orleans, LA, USA, 1–6 June 2018; pp. 1123–1133. [\[CrossRef\]](#)
21. Deng, K.; Wu, S. Improving Relation Classification by Incorporating Dependency and Semantic Information. In Proceedings of the IEEE International Joint Conference on Neural Network, Glasgow, UK, 19–24 July 2020. [\[CrossRef\]](#)

22. Shi, W.; Li, F.; Li, J.; Fei, H.; Ji, D. Effective Token Graph Modeling using a Novel Labeling Strategy for Structured Sentiment Analysis. In Proceedings of the Annual Meeting of the Association for Computational Linguistics, Dublin, Ireland, 22–27 May 2022. [\[CrossRef\]](#)
23. Yao, L.; Haghighi, A.; Riedel, S.; McCallum, A. Structured Relation Discovery using Generative Models. In Proceedings of the Conference on Empirical Methods in Natural Language Processing, Scotland, UK, 27–31 July 2011.
24. Liu, Y.; Li, S.; Wei, F.; Ji, H. Relation Classification Via Modeling Augmented Dependency Paths. *IEEE/ACM Trans. Audio Speech Lang. Process.* **2016**, *24*, 1589–1598. [\[CrossRef\]](#)
25. Chen, C.; Shoga, M.; Boehm, B. Exploring the Dependency Relationships between Software Qualities. In Proceedings of the IEEE International Conference on Software Quality, Reliability and Security Companion, Sofia, Bulgaria, 22–26 July 2019; pp. 105–108. [\[CrossRef\]](#)
26. Zhang, P.; Wu, M. Multi-Label Supervised Contrastive Learning. In Proceedings of the AAAI Conference on Artificial Intelligence, Vancouver, BC, Canada, 26–27 February 2024; Volume 38. [\[CrossRef\]](#)
27. Zhou, Y.; Tong, Y.; Gu, R.; Gall, H. Combining Text Mining and Data Mining for Bug Report Classification. In Proceedings of the IEEE International Conference on Software Maintenance and Evolution, Raleigh, NC, USA, 2–7 October 2016. [\[CrossRef\]](#)
28. Carneiro, G.; Ferreira, J.; Ramalho, F.; Massoni, T. Similar Bug Reports Recommendation System using BERT. In Proceedings of the XXXVII Brazilian Symposium on Software Engineering, Campo Grande, Brazil, 25–29 September 2023; pp. 378–387. [\[CrossRef\]](#)
29. Torunoğlu, D.; Eryiğit, G. A Cascaded Approach for Social Media Text Normalization of Turkish. In Proceedings of the 5th Workshop on Language Analysis for Social Media (LASM), Gothenburg, Sweden, 26–27 April 2014; pp. 62–70. [\[CrossRef\]](#)
30. Song, K.; Tan, X.; Qin, T.; Lu, J.; Liu, T.-Y. MPNet: Masked and Permuted Pre-training for Language Understanding. In Proceedings of the Neural Information Processing Systems, Vancouver, BC, Canada, 6–12 December 2020.
31. Aguilar, G.; McCann, B.; Niu, T.; Rajani, N.; Keskar, N.; Solorio, T. Char2Subword: Extending the Subword Embedding Space Using Robust Character Compositionality. In Proceedings of the Conference on Empirical Methods in Natural Language Processing, Online, 16–20 November 2020; pp. 1640–1651. [\[CrossRef\]](#)
32. Erkan, A.; Güngör, T. Analysis of Deep Learning Model Combinations and Tokenization Approaches in Sentiment Classification. *IEEE Access* **2023**, *11*, 134951–134968. [\[CrossRef\]](#)
33. Reddy, D.M.; Reddy, S. Effects of padding on LSTMs and CNNs. *arXiv* **2019**, arXiv:1903.07288. Available online: <https://api.semanticscholar.org/CorpusID:81983221> (accessed on 16 August 2025).
34. Kosec, M.; Fu, S.; Krell, M.M. Packing: Towards 2x NLP BERT Acceleration. *arXiv* **2021**, arXiv:2107.02027. Available online: <https://api.semanticscholar.org/CorpusID:235732160> (accessed on 16 August 2025).
35. Kumar, A.; Ware, N.; Gupta, S. Leveraging Transfer Learning: Fine-Tuning methodology for Enhanced Text Classification using BERT. In Proceedings of the IEEE Pune Section International Conference (PuneCon), Pune, India, 13–15 December 2024; pp. 1–5. [\[CrossRef\]](#)
36. Pan, D.; Lu, Y.; Kang, P. A Deep Learning Model for Multi-label Classification Using Capsule Networks. In *International Conference on Intelligent Computing*; Springer: Cham, Switzerland, 2019; pp. 144–155. [\[CrossRef\]](#)
37. Lanchantin, J.; Wang, T.; Ordonez, V.; Qi, Y. General Multi-label Image Classification with Transformers. In Proceedings of the Computer Vision and Pattern Recognition, Nashville, TN, USA, 20–25 June 2020; pp. 16473–16483. [\[CrossRef\]](#)
38. Dembczynski, K.; Waegeman, W.; Cheng, W.; Hüllermeier, E. On label dependence and loss minimization in multi-label classification. *Mach.-Mediat. Learn.* **2012**, *88*, 4–45. [\[CrossRef\]](#)
39. Rezaei-Dastjerdehei, M.; Mijani, A.; Fatemizadeh, E. Addressing Imbalance in Multi-Label Classification Using Weighted Cross Entropy Loss Function. In Proceedings of the Iranian Conference on Biomedical Engineering, Tehran, Iran, 26–27 November 2020; pp. 333–338. [\[CrossRef\]](#)
40. Huang, Y.; Giledereli, B.; Köksal, A.; Özgür, A.; Ozkirimli, E. Balancing Methods for Multi-label Text Classification with Long-Tailed Class Distribution. In Proceedings of the Conference on Empirical Methods in Natural Language Processing, Punta Cana, Dominican Republic, 7–11 November 2021; pp. 8153–8161. [\[CrossRef\]](#)
41. Bai, J.; Kong, S.; Gomes, C.P. Gaussian Mixture Variational Autoencoder with Contrastive Learning for Multi-Label Classification. In Proceedings of the International Conference on Machine Learning, Baltimore, MD, USA, 17–23 July 2021.
42. Khvatskii, G.; Moniz, N.; Doan, K.D.; Chawla, N.V. Class-Aware Contrastive Optimization for Imbalanced Text Classification. *arXiv* **2024**. [\[CrossRef\]](#)
43. Labd, Z.; Bahassine, S.; Housni, K. Constractive Multilabel Adaptation with Graph-Aware Transformers (CMAGT). In Proceedings of the IEEE International Conference on Circuits and Systems for Communications, Fez, Morocco, 19–20 June 2025; pp. 1–5. [\[CrossRef\]](#)
44. Cui, D.; Liu, T.; Cai, Y.; Zheng, Q.; Feng, Q.; Jin, W.; Guo, J.; Qu, W. Investigating the Impact of Multiple Dependency Structures on Software Defects. In Proceedings of the International Conference on Software Engineering, Montreal, QC, Canada, 25–31 May 2019; pp. 584–595. [\[CrossRef\]](#)

45. Barbano, C.A.; Dufumier, B.; Tartaglione, E.; Grangetto, M.; Gori, P. Unbiased Supervised Contrastive Learning. *arXiv* **2022**. [CrossRef]
46. Yuan, H.; Lu, Z. Robust Task Representations for Offline Meta-Reinforcement Learning via Contrastive Learning. *arXiv* **2022**. [CrossRef]
47. Nichol, A.; Achiam, J.; John Schulman, J. On First-Order Meta-Learning Algorithms. *arXiv* **2018**. [CrossRef]
48. Ji, K.; Lee, J.; Liang, Y.; Poor, H. Convergence of Meta-Learning with Task-Specific Adaptation over Partial Parameters. *Adv. Neural Inform. Process. Syst.* **2020**. Available online: https://proceedings.neurips.cc/paper_files/paper/2020/hash/84c578f202616448a2f80e6f56d5f16d-Abstract.html (accessed on 16 August 2025).
49. Rajasegaran, J.; Khan, S.M.; Hayat, M.; Khan, F.; Shah, M. Meta-learning the Learning Trends Shared Across Tasks. In Proceedings of the British Machine Vision Conference, Manchester, UK, 7–11 September 2020. [CrossRef]
50. Mojoo, J.; Kurosawa, K.; Kurita, T. Deep CNN with Graph Laplacian Regularization for Multi-label Image Annotation. In Proceedings of the International Conference on Image Analysis and Recognition, Montreal, QC, Canada, 5–7 July 2017; pp. 19–26. [CrossRef]
51. Zhang, M.-L.; Zhang, K. Multi-label learning by exploiting label dependency. In Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, 25–28 July 2010; pp. 999–1008. [CrossRef]
52. Shu, Y.; Cao, Z.; Gao, J.; Wang, J.; Long, M. Omni-Training for Data-Efficient Deep Learning. *arXiv* **2021**, arXiv:2110.07510. Available online: <https://api.semanticscholar.org/CorpusID:267816781> (accessed on 16 August 2025).
53. Zhang, M.-L.; Zhou, Z.-H. A Review on Multi-Label Learning Algorithms. *IEEE Trans. Knowl. Data Eng.* **2014**, *26*, 1819–1837. [CrossRef]
54. Luaphol, B.; Polpinij, J.; Kaenampornpan, M. Text mining approaches for dependent bug report assembly and severity prediction. *Int. Arab J. Inf. Technol.* **2022**, *19*, 915–924. [CrossRef]
55. Cui, X. Addressing Data Imbalance in Transformer-Based Multi-Label Emotion Detection with Weighted Loss. *arXiv* **2025**. Available online: <https://arxiv.org/abs/2507.11384> (accessed on 16 August 2025). [CrossRef]
56. Liu, J.; Duan, X.; Zhang, R.; Sun, Y.; Guan, L.; Lin, B. Relation classification via BERT with piecewise convolution and focal loss. *PLoS ONE* **2021**, *16*, e0257092. [CrossRef] [PubMed]
57. Yuan, Y.; Luo, X.; Peng, Q.; Chen, M.; Shen, H.; Huang, Z. Enhancing Relation Classification Using Focal Loss. In Proceedings of the ACM Cloud and Autonomic Computing Conference, Seattle, WA, USA, 1–4 November 2021; pp. 2680–2685. [CrossRef]
58. Fallah, H.; Bruno, E.; Bellot, P.; Murisasco, E. Exploiting Label Dependencies for Multi-Label Document Classification Using Transformers. In Proceedings of the ACM Symposium on Document Engineering, Limerick, Ireland, 22–25 August 2023; pp. 1–4. [CrossRef]
59. Koyejo, O.; Natarajan, N.; Ravikumar, P.; Dhillon, L. Consistent Multilabel Classification. *Adv. Neural Inf. Process. Syst.* **2015**. Available online: <https://api.semanticscholar.org/CorpusID:6348016> (accessed on 16 August 2025).
60. Pereira, R.B.; Plastino, A.; Zadrozny, B.; Merschmann, L. Correlation analysis of performance measures for multi-label classification. *Inf. Process. Manag.* **2018**, *54*, 359–369. [CrossRef]

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.