Title	**Deep Learning with Multimodal Inputs for Enhanced Issue Resolution**
Team	Marina Reda 221101235 Omar Adly 221101398 Hazem Ahmed 221100343 Ali Sherif 221101562 Loay Gamal 221100419
Field	Computer science and Engineering
Program	Artificial intelligence science
Supervisor	**Dr. Mohamed Ghetas**
Date of Submission	1/1/2025

Table of Contents

Abstract

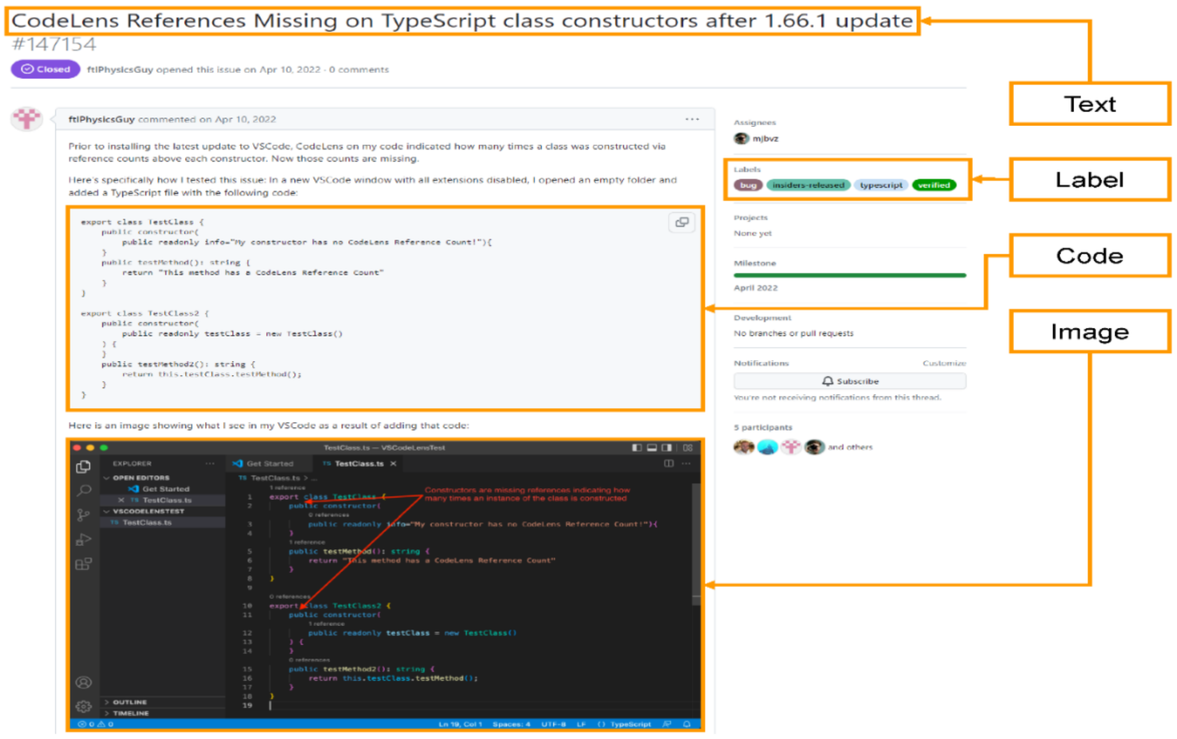
Issue reports serve as essential resources for maintaining and enhancing software systems. However, the management and classification of these reports demand considerable effort from developers. While automated techniques have been proposed to address this challenge, many rely solely on text-based unimodal models, resulting in suboptimal classification accuracy. This project introduces a novel classification approach leveraging a multimodal deep learning model that integrates heterogeneous information from text, images, and code found in issue reports. To assess the efficacy of the proposed approach, we conduct experiments across four distinct projects, comparing its performance against traditional text-based models. The results reveal that our multimodal model outperforms unimodal counterparts, achieving a 5.07% to 14.12% improvement in F1-score. These findings underscore the advantage of incorporating diverse data modalities to enhance issue classification performance.

1. Introduction

To streamline software maintenance, developers rely on issue-tracking systems to integrate user requirements into their software products efficiently. These systems enable users to report bugs, suggest features, and provide feedback through issue reports. These reports serve as a critical reference for developers when making software improvements. However, the sheer volume of issue reports—especially in active open-source projects—poses a significant challenge, with hundreds of reports generated daily. Managing these reports manually demands substantial time and effort.

To address this, researchers have explored automated classification techniques to better organize and manage issue reports [1-10]. Some studies [1,2,3,6,7,9] focused on categorizing reports into broad categories, such as bugs or non-bugs. For instance, Kallis et al. [7] employed FastText to classify issue reports into three categories: Bug, Enhancement, and Question. Others aimed to classify reports into more specific categories [5,8,10]. However, these approaches primarily relied on unimodal data, typically textual information, which limited their ability to fully exploit the diverse content found in issue reports. This reliance on a single modality constrained the models' understanding and reduced their classification performance.

Issue reports, in reality, often include various types of information such as textual descriptions, images, videos, and code snippets. **Figure 1** illustrates an example from VS Code (issue #147154) containing text, code, and images. This example underscores the potential value of leveraging multimodal data. We hypothesize that incorporating image and code data alongside text can enhance the representation and understanding of issue reports, thereby improving classification performance.


CodeLens References Missing on TypeScript class constructors after 1.66.1 update #147154

Closed ftIPhysicsGuy opened this issue on Apr 10, 2022 · 0 comments

ftIPhysicsGuy commented on Apr 10, 2022

Prior to installing the latest update to VSCode, CodeLens on my code indicated how many times a class was constructed via reference counts above each constructor. Now those counts are missing.

Here's specifically how I tested this issue: In a new VSCode window with all extensions disabled, I opened an empty folder and added a TypeScript file with the following code:

```
export class TestClass {
    public constructor(
        public readonly info="My constructor has no CodeLens Reference Count!"){
    }
    public testMethod(): string {
        return "This method has a CodeLens Reference Count"
    }
}

export class TestClass2 {
    public constructor(
        public readonly testClass = new TestClass()
    ) {
    }
    public testMethod2(): string {
        return this.testClass.testMethod();
    }
}
```

Here is an image showing what I see in my VSCode as a result of adding that code:

```
1 reference
export class TestClass {
    public constructor(
        0 references
        public readonly info="My constructor has no CodeLens Reference Count!"){
    }
    1 reference
    public testMethod(): string {
        return "This method has a CodeLens Reference Count"
    }
}

0 references
export class TestClass2 {
    public constructor(
        1 reference
        public readonly testClass = new TestClass()
    ) {
    }
    0 references
    public testMethod2(): string {
        return this.testClass.testMethod();
    }
}
```

Constructors are missing references indicating how many times an instance of the class is constructed

Assignees: mjbvz

Labels: bug, insiders-released, typescript, verified

Projects: None yet

Milestone: April 2022

Development: No branches or pull requests

Notifications: You're not receiving notifications from this thread.

5 participants and others

“Figure 1”

To this end, we propose **FusionNet**, a novel multimodal deep learning model for issue classification that integrates text, image, and code data from issue reports.

To evaluate **FusionNet**, we conducted experiments using four large-scale projects with numerous issue reports: VS Code, Kubernetes, Flutter, and Roslyn. We compared the performance of our model with a state-of-the-art text-based unimodal model proposed by Cho et al. [10]. Our results demonstrate that **FusionNet** consistently outperformed the unimodal approach, achieving an improvement in F1-score ranging from 5.07% to 14.12% across all projects.

Contributions of this project include:

1. Proposing the first multimodal model combining text, image, and code modalities for issue classification tasks.
2. Evaluating the proposed model's effectiveness by comparing it against a text-based unimodal model.
3. Analyzing the impact of image and code modalities through experiments with various modality combinations.

2. Related Works

This section categorizes the related studies into two main groups: (1) classification of issue reports in open-source projects and (2) the application of multimodal deep learning techniques in software engineering and other fields.

2.1. Issue Report Classification

Various approaches, including machine learning and deep learning techniques, have been proposed for classifying issue reports. These methods primarily focus on categorizing issue reports into "bug" or "non-bug" categories. Below is a comparison of key studies in this domain (Look at Table 1):

Study	***Methodology***	***Dataset/Project Examples***	***Key Features***	***Performance Metrics***
Fan et al. [2]	Two-stage approach using SVM, Naive Bayes, Logistic Regression	Not specified	Combines text features with developer information	F1-score: 75.21%
Pandey et al. [3]	Machine learning (Naive Bayes, SVM, Logistic Regression)	Three open-source projects	Uses issue summaries for classification	F1-score: 71.11–72.15%
Zhu et al. [6]	Attention-based BiLSTM with k-NN correction	Apache, JBoss, Spring Framework	Preprocessing with attention mechanisms	Micro F1-score: 85.6%
Kallis et al. [7]	TicketTagger (Automated GitHub label assignment)	GitHub projects	Assigns labels: Bug, Enhancement, Question	F1-score: 82.3–83.1%

Kim et al. [8]	CNN-based classification (Production vs. Test bug reports)	Not specified	Utilizes bug report text and source file features	Macro F1-score: 83.9%
Cho et al. [10]	CNN & RNN for linking issues to relevant user manuals	Notepad++, Komodo, VS Code	Combines title, body text, and user manuals	F1-score: 52.73–67.14%
Zhifang et al. [9]	PIFTNet with BERT and developer traits	GitHub projects	Combines text with submitter's project familiarity traits	F1-score: 85.55%

"Table 1"

Despite these advancements, existing studies primarily utilize a single modality (textual data). To address this limitation, we propose FusionNet, a multimodal approach that integrates text, image, and code data to enhance issue classification.

2.2. Application of Multimodal Deep Learning

Multimodal deep learning techniques have gained traction in various domains, achieving superior performance compared to unimodal approaches.

In Software Engineering

Multimodal techniques in software engineering have been applied to tasks such as bug localization, code editing, and bug report triage. The table below summarizes relevant studies (Look at Table 2):

Study	***Application***	***Modalities Used***	***Methodology/Key Features***	***Performance Highlights***
Hoang et al.	Bug Localization	Text, Graph	NetML with feature extraction, graph construction, integrator	Outperforms 4 baselines on 355 bugs
Chakraborty et al.	Automated Code Editing	Text, Code	MODIT (Multimodal Translator with commit messages)	Outperforms CodeBERT, GraphCodeBERT
Zhang et al.	Bug Report Triage	Text, Metadata	SusTriage with ensemble learning	Improves accuracy and sustainability

"Table 2"

These studies incorporate supplementary data to enhance performance. However, they do not fully utilize diverse modalities, such as images or code, which are critical for comprehensive issue analysis. FusionNet fills this gap by leveraging three modalities: text, code, and images.

In Other Fields

Multimodal deep learning has been extensively explored in domains such as emotion recognition, visual question answering (VQA), and fake news detection. Below is a summary(Look at Table 3):

Study	***Domain/Application***	***Modalities Used***	***Methodology/Key Features***	***Performance Highlights***
Antol et al.	Visual Question Answering	Image, Text	VGGNet for images, LSTM for text	Improved VQA accuracy
Lopez-Fuentes et al.	Disaster Analysis	Image, Text	CNN and BiLSTM for social media post retrieval	Better retrieval of flood-related posts
Palani et al.	Fake News Detection	Image, Text	CB-Fake with CapsNet and BERT	Enhanced early-stage fake news detection
Huang et al.	Emotion Recognition	Facial Expressions, EEG	Fusion of classifiers for emotions and intensity levels	Improved emotion recognition accuracy

“Table 3”

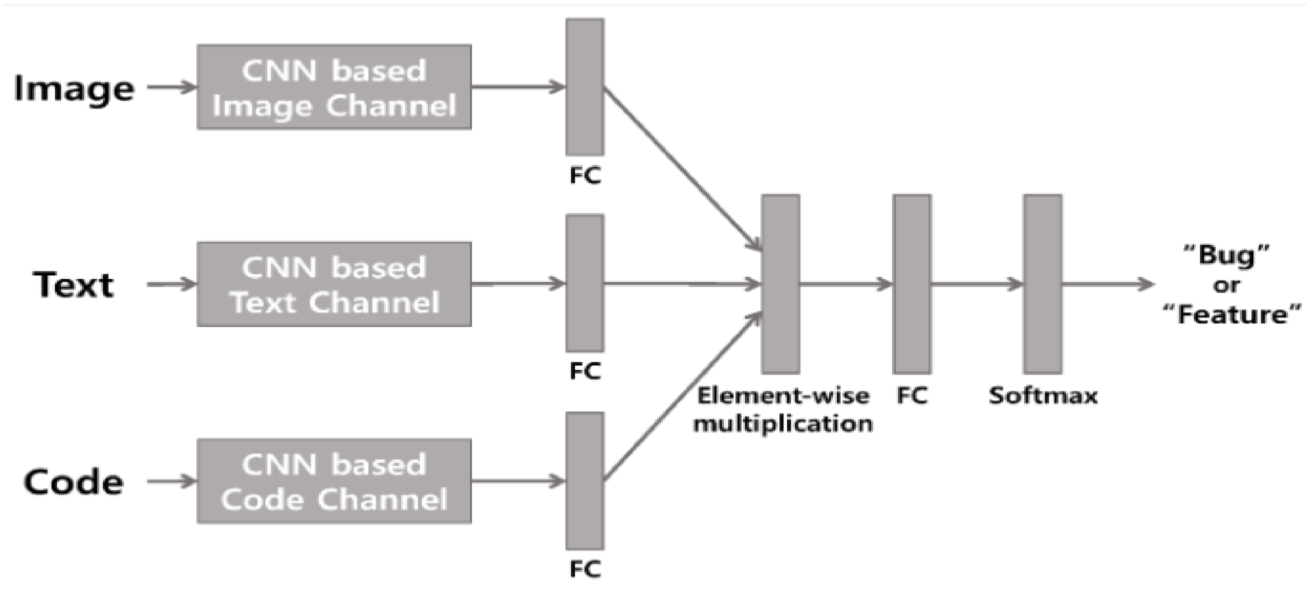
The above studies demonstrate the benefits of combining different modalities for diverse tasks. However, as far as we know, FusionNet is the first multimodal approach tailored for issue classification in software engineering, integrating text, code, and image data for comprehensive analysis.

3. The Proposed Multimodal Approach

This section elaborates on our proposed multimodal methodology for issue report classification.

3.1 Overview

We introduce **FusionNet**, a multimodal model designed for classifying issue reports into two categories: "bug" and "feature." The architecture of FusionNet is depicted in **Figure 2**, where "FC" signifies "Fully Connected." The model processes data from three input modalities: **text**, **image**, and **code**. By employing CNN-based channels, FusionNet preserves the unique features of each modality.



"Figure 2"

Key Stages of FusionNet:

1. **Data Pre-Processing**: Text, image, and code inputs are pre-processed individually.
2. **Feature Extraction**: Pre-processed data are passed through CNN-based channels to generate feature vectors for each modality.
3. **Fusion**: Feature vectors from all modalities are integrated through element-wise multiplication to create a unified multimodal representation.
4. **Classification**: The fused representation is classified into "bug" or "feature" using a Softmax operation.

The choice to limit the classification to two categories allows us to test the potential of a multimodal model in improving issue classification accuracy.

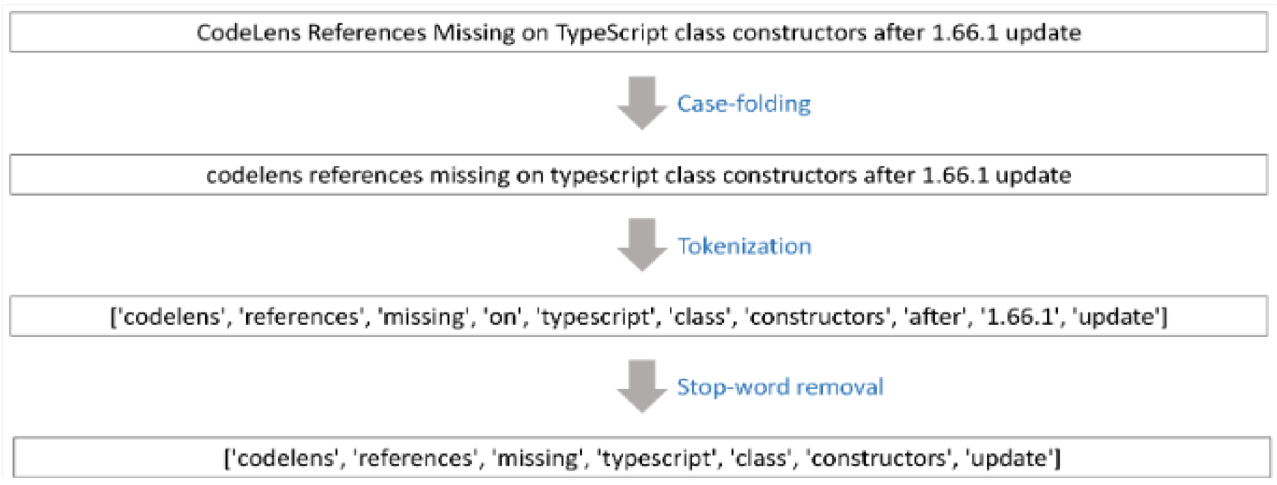
3.2 Data Pre-Processing

Text Pre-Processing

Text data is pre-processed to enhance its relevance and usability. The steps include:

1. **Case-Folding**: Converts text to lowercase for consistency.
2. **Tokenization**: Splits text into tokens using NLTK's word_tokenize.
3. **Stop-Word Removal**: Eliminates common, non-informative words using predefined NLTK stop-word lists. Non-alphabetic tokens such as numbers are also removed.

Refer to **Figure 3** for an example of this process.


CodeLens References Missing on TypeScript class constructors after 1.66.1 update
Case-folding
codelens references missing on typescript class constructors after 1.66.1 update
Tokenization
['codelens', 'references', 'missing', 'on', 'typescript', 'class', 'constructors', 'after', '1.66.1', 'update']
Stop-word removal
['codelens', 'references', 'missing', 'typescript', 'class', 'constructors', 'update']

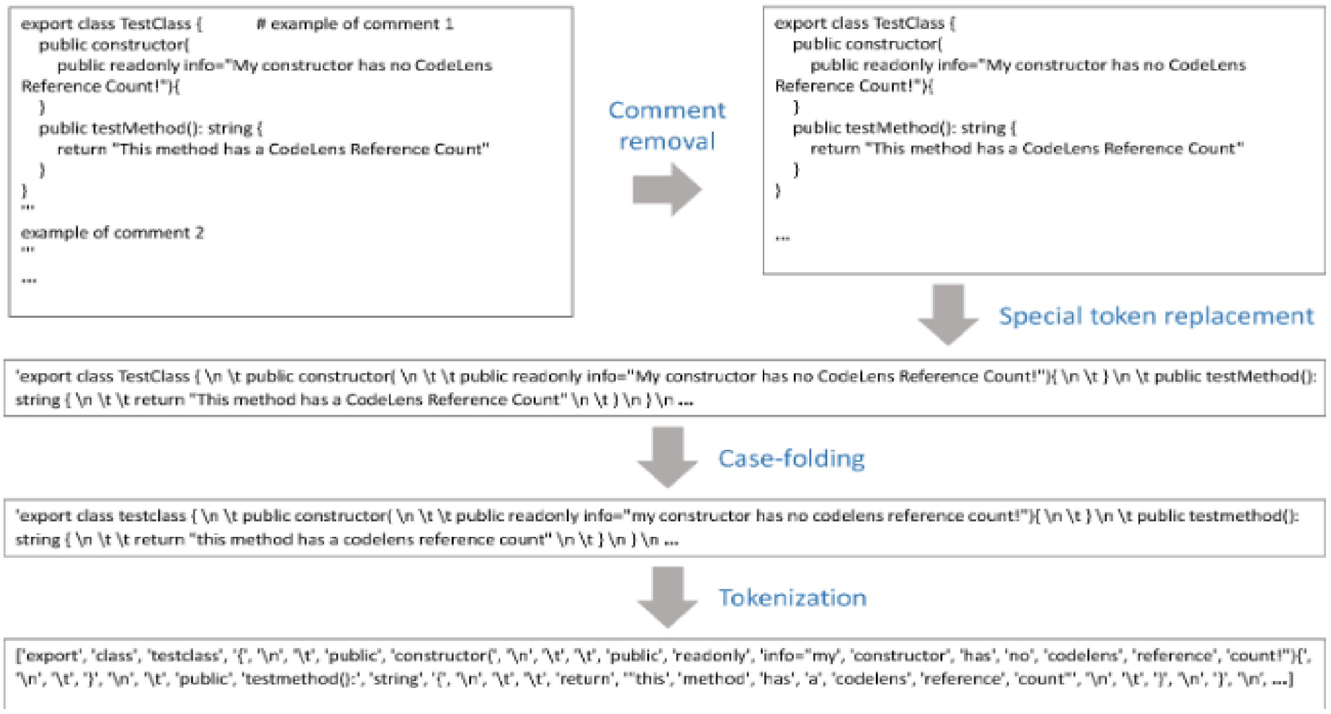
“Figure 3”

Code Pre-Processing

Code data, with its unique syntax, undergoes specialized pre-processing steps*:*

1. **Comment Removal**: Strips natural language comments from code.
2. **Special Token Replacement**: Uses placeholders like \n and \t to preserve structural elements.
3. **Case-Folding**: Converts code to lowercase.
4. **Tokenization**: Breaks code into tokens using NLTK’s SpaceTokenizer.

Figure 4 illustrates this process.


```
export class TestClass {          # example of comment 1
   public constructor(
      public readonly info="My constructor has no CodeLens
Reference Count!"){
   }
   public testMethod(): string {
      return "This method has a CodeLens Reference Count"
   }
}
"""
example of comment 2
"""
...
```

```
export class TestClass {
   public constructor(
      public readonly info="My constructor has no CodeLens
Reference Count!"){
   }
   public testMethod(): string {
      return "This method has a CodeLens Reference Count"
   }
}

...
```

```
'export class TestClass { \n \t public constructor( \n \t \t public readonly info="My constructor has no CodeLens Reference Count!"){ \n \t } \n \t public testMethod():
string { \n \t \t return "This method has a CodeLens Reference Count" \n \t } \n } \n ...
```

```
'export class testclass { \n \t public constructor( \n \t \t public readonly info="my constructor has no codelens reference count!"){ \n \t } \n \t public testmethod():
string { \n \t \t return "this method has a codelens reference count" \n \t } \n } \n ...
```

```
['export', 'class', 'testclass', '{', '\n', '\t', 'public', 'constructor(', '\n', '\t', '\t', 'public', 'readonly', 'info="my', 'constructor', 'has', 'no', 'codelens', 'reference', 'count!"){',
'\n', '\t', '}', '\n', '\t', 'public', 'testmethod():', 'string', '{', '\n', '\t', '\t', 'return', '"this', 'method', 'has', 'a', 'codelens', 'reference', 'count"', '\n', '\t', '}', '\n', '}', '\n', ...]
```

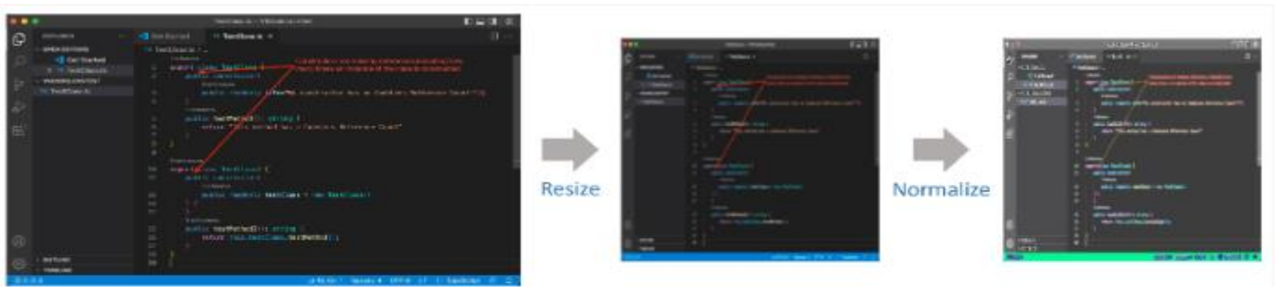
"Figure 4"

Image Pre-Processing

To standardize image data, the following steps are performed:

1. **Resizing**: Images are resized to 258 × 258 pixels using Torchvision's transforms.
2. **Normalization**: Pixel values are scaled to fall within the range [-1, 1] for consistency.

An example is shown in **Figure 5**.

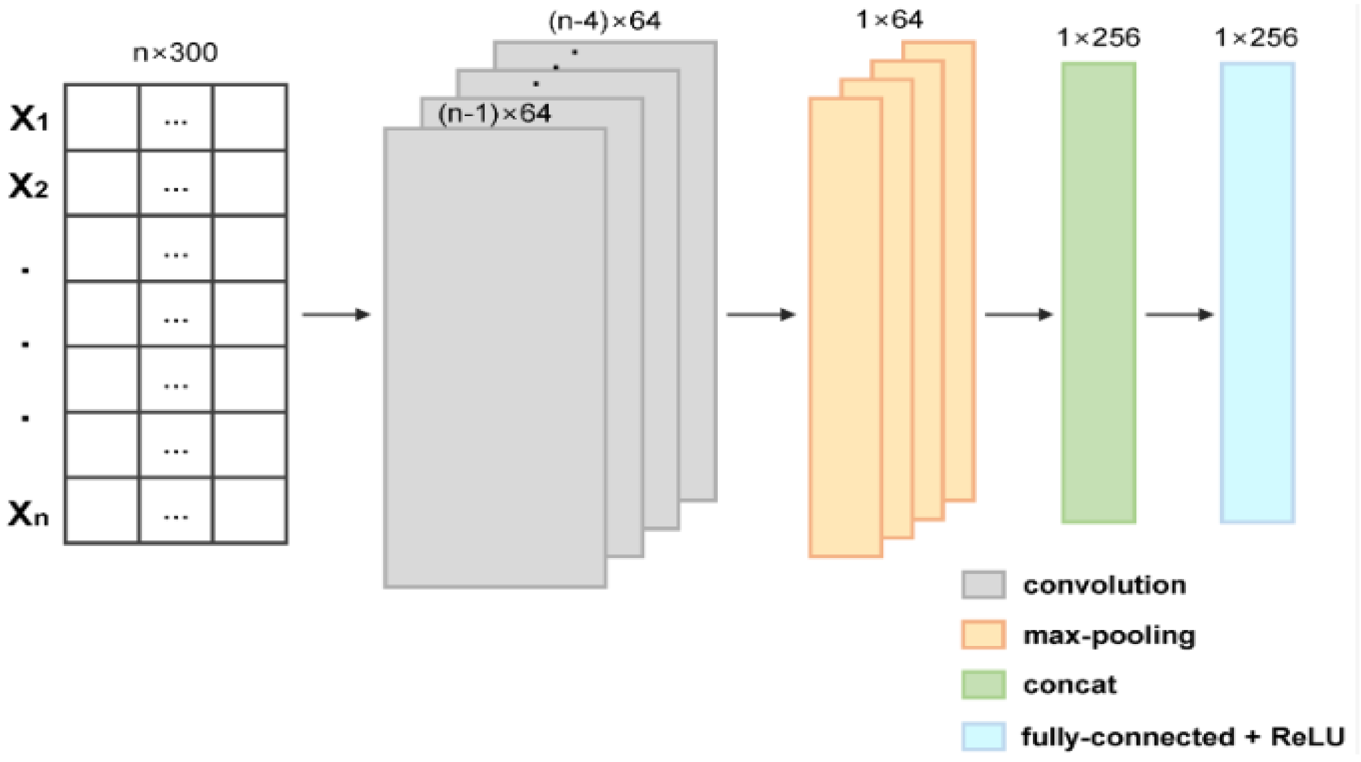


"Figure 5"

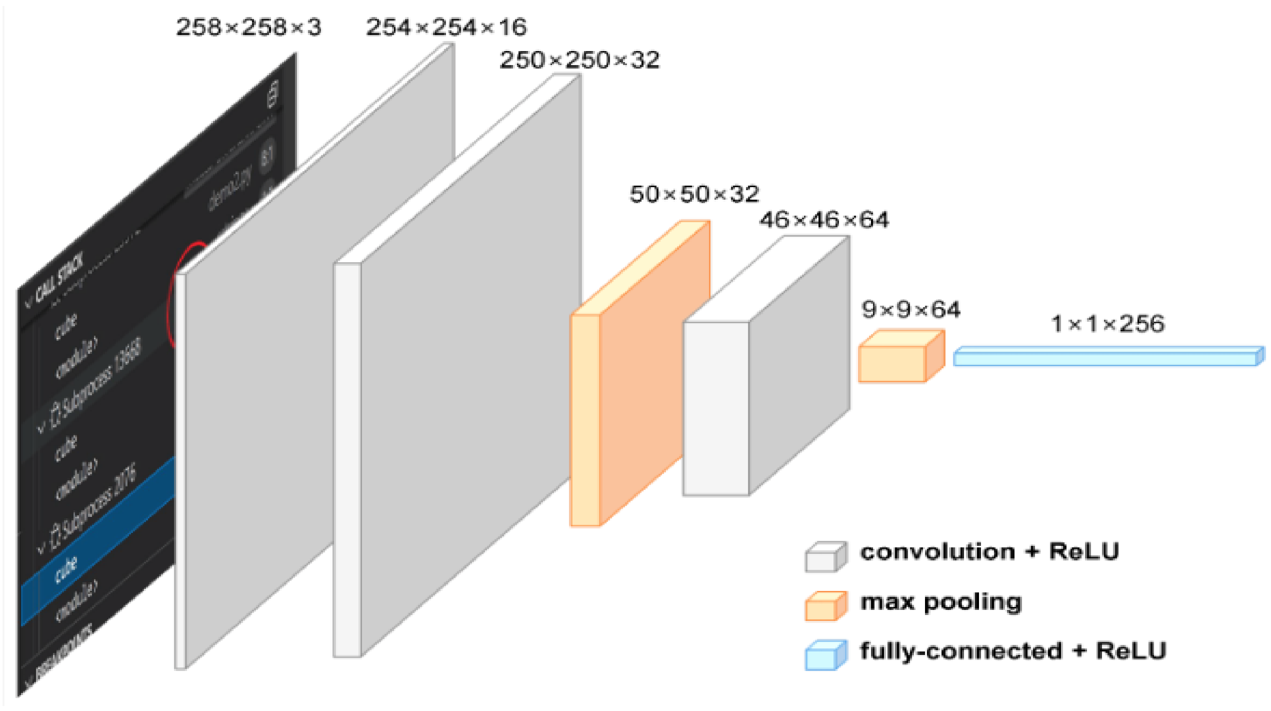
3.3 Feature Extraction

Feature vectors are extracted from each modality as follows:

1. **Embedding**: Text and code data are vectorized using an embedding layer. Images are already vectorized during pre-processing.
2. **CNN Processing**:
    a. Text and code share a CNN-based structure with kernels of varying sizes, performing convolution, max-pooling, and concatenation. The output is passed through a fully connected layer with ReLU activation (see **Figure 6**).
    b. Images are processed using a CNN with three convolution layers, max-pooling, and a fully connected layer to extract feature vectors (see **Figure 7**).



"Figure 6"



“Figure 7”

3.4 Fusion of Feature Vectors

Fusion combines modality-specific features into a unified representation using **element-wise multiplication**. This method, chosen for its simplicity and effectiveness, integrates the features into a common space, enabling seamless multimodal learning.

3.5 Classification into "Bug" or "Feature"

In the final stage, the fused representation is processed through a fully connected layer and Softmax operation, yielding probabilities for "bug" or "feature." While the current model performs binary classification, its structure can be extended for multiclass tasks. The model is optimized using CrossEntropyLoss.

This step concludes the multimodal pipeline, providing a robust framework for issue report classification.

4. Experimental Setup

4.1 Datasets

Project Selection

We selected open-source projects from GitHub for our experiment, as GitHub provides an integrated issue management system, with users frequently submitting issue reports. We considered projects with 'bug' and 'feature' labels for binary classification. After reviewing the top 100 active projects based on issue count, we chose four projects: VS Code, Kubernetes, Flutter, and Roslyn. These projects had sufficient issue reports containing text, images, and code. Table 4 summarizes the number of issue reports per label for each project.

Project	Total Number of Issues	Label	Num
VS Code	160,218	Bug	28,353
		Feature	20,074
		Total	48,427
Kubernetes	115,035	Bug	13,059
		Feature	5184
		Total	18,243
Flutter	118,576	Bug	13,037
		Feature	9967
		Total	23,004
Roslyn	66,464	Bug	12,882
		Feature	3824
		Total	16,706

“Table 4”

Data Sampling

Due to class imbalance in the collected data, we applied downsampling to balance the number of samples in each class. The 'feature' class data were retained, while we selected a subset of the most recent 'bug' class data to match the 'feature' class size. We used an 80:20 training/test split for each dataset.

4.2 Models

We compared our multimodal deep learning model, **FusionNet**, against a baseline model that only uses text data. The multimodal models tested were: **FusionNet**TI (text and image), **FusionNet**TC (text and code), and **FusionNet**TIC (text, image, and code). These were evaluated against the baseline **Text Only** model, which uses only text data.

Unimodal Model: Text Only

The **Text Only** model utilizes only text data for issue classification, serving as the baseline for comparison.

Multimodal Models

- **FusionNet**TI: This model combines text and image data for issue classification.
- **FusionNet**TC: This model combines text and code data for issue classification.
- **FusionNet**TIC: This model combines text, image, and code data for issue classification. It was compared to both **FusionNet**TI and **FusionNet**TC to evaluate the combined impact of all three modalities.

4.3 Experimental Design

We comparing the three multimodal models with the baseline **Text Only** model. The models were trained using modality-specific pre-processing on the sampled data. Features were extracted from each modality, and in the case of multimodal models, features from each modality were combined. The resulting feature vectors were classified, and performance was compared to the ground truth.

4.4 Evaluation Metrics

We measured classification performance using **Precision**, **Recall**, and **F1-score**. These metrics were calculated for each class and averaged using the weighted mean based on class distribution:

- **Precision**: The proportion of correctly predicted instances for each class.
- **Recall**: The proportion of actual instances correctly predicted for each class.
- **F1-score**: The harmonic mean of Precision and Recall.

Equations for the metrics are as follows:

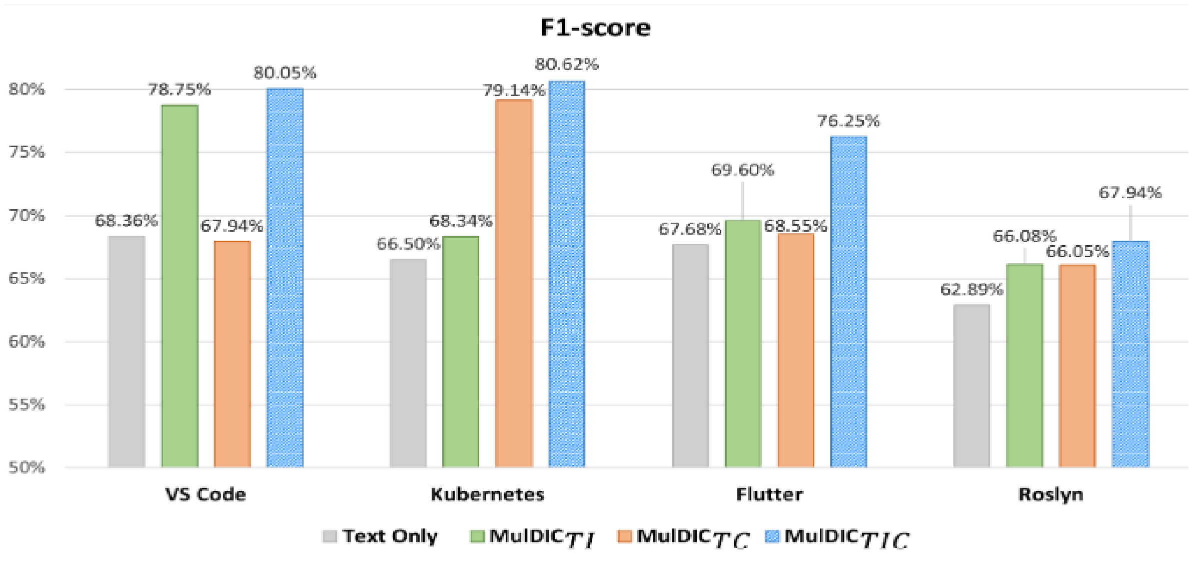
$$\text{Precision} = \frac{\sum_{i=0}^{n} \text{precision}_i \times \text{number of issues in class}_i}{\sum_{i=0}^{n} \text{number of issues in class}_i}$$

$$\text{Recall} = \frac{\sum_{i=0}^{n} \text{recall}_i \times \text{number of issues in class}_i}{\sum_{i=0}^{n} \text{number of issues in class}_i}$$

$$\text{F1-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

5. Results

This section summarizes the key findings from the experiments conducted in this project, as illustrated in Figure 8. The multimodal model **FusionNet**TIC, which integrated text, image, and code data, outperformed other models in terms of F1-score across all projects. Detailed performance metrics are discussed below.



"Figure 8"

5.1. Results of the Text-Image Experiment

The experiment compared the **Text Only** model with the multimodal **FusionNet**TI model, which combined text and image data. The results indicated that the **FusionNet*I*** model consistently outperformed the **Text Only** model across all projects. The **FusionNet*I*** model showed significant improvements in **Precision** (up to 11.15%), **Recall** (up to 9.65%), and **F1-score** (up to 10.39%) compared to the **Text Only** model, especially for **VS Code** and **Roslyn**. These findings suggest that images provide valuable information for improving issue classification, as the multimodal model leverages both text and images to better understand the issue reports.

5.2. Results of the Text-Code Experiment

In the second experiment, we compared the **Text Only** model with the multimodal **FusionNetTC** model, which combined text and code data. The results showed that **FusionNet$C$** outperformed the **Text Only** model in most projects in terms of **Precision** (up to 13.06%) and **F1-score** (up to 12.64%), though the **Text Only** model had a slight edge in **VS Code**. The inclusion of code data improved the model's ability to classify issues, reinforcing the importance of incorporating diverse modalities such as code along with text.

5.3. Results of the Text-Image-Code Experiment

The final experiment evaluated the performance of the **FusionNetTIC** model, which integrates text, image, and code data, in comparison to the other models. the **FusionNetTC** model outperformed both the **Text Only** and **FusionNet$T$** models in all three evaluation metrics (Precision, Recall, and F1-score). The **FusionNetTC** model demonstrated improvements of up to 14.34% in **Precision**, 13.90% in **Recall**, and 14.12% in **F1-score** for **Kubernetes**, and showed consistent improvements across other projects. This highlights the synergistic effect of combining multiple data types, enhancing the model's ability to understand and classify issue reports.

Discussion

6.1 Effects of Using Multiple Modalities

The integration of text, image, and code data in the FusionNetTIC model significantly outperformed other models. It achieved an F1-score improvement of 5.07–14.12% over the Text Only model, demonstrating the synergistic benefits of combining modalities.

This performance highlights that leveraging heterogeneous information enriches issue classification. For example, textual details provide reproduction steps, code data offers specific lines, and images illustrate buggy results. Together, these inputs enable the FusionNetTIC model to classify reports like issue #147154 (**Figure 1**)accurately.

6.2 Text-Code Experiment Challenges in VS Code

The FusionNetTC model underperformed for the VS Code project, with a slightly lower F1-score (67.94%) than the Text Only model (68.36%). Factors investigated include:

1. **Token Length**: Analysis showed no significant data loss in VS Code, eliminating this as a factor.
2. **Weight Assignment**: Equal weights assigned to modalities might have weakened text-code synergy, as text data appeared more informative.
3. **Data Quality**: Variability in contributor-written reports across projects likely impacted performance, with VS Code data being of comparatively lower quality.

6.3 Implications and Future Work

Implications:

- **Researchers**: Extend multimodal classification techniques to multi-class and multi-label tasks.
- **Developers**: Save time and avoid misclassification with automated tools.
- **Companies**: Employ models to track systematic issue trends more accurately.

Future Directions:

1. **State-of-the-Art Feature Extraction**: Employ models like BERT for text, CodeBERT for code, and Transformer-based vision models for images to improve accuracy.
2. **Advanced Fusion Methods**: Explore bilinear pooling techniques like MCB, MLB, MUTAN, and MFB to enhance feature interaction while maintaining computational efficiency.

7. Conclusions

This project focused on issue reports that encompass multiple data types, such as text, images, and code, and introduced **FusionNet**, a multimodal deep learning framework tailored to classify these reports effectively. Through extensive experimentation, we evaluated the performance of the proposed model and analyzed the contribution of each modality. The results highlighted the exceptional performance of the **FusionNet**TIC variant, which integrates text, image, and code data, delivering an F1-score improvement ranging from 5.07% to 14.12% over baseline models.

Our findings revealed that:

1. Incorporating images alongside text consistently enhanced classification performance.
2. While the code modality generally improved results, there were instances where it did not.
3. Combining all three modalities outperformed any two-modality combinations, demonstrating the advantages of leveraging diverse data sources for issue classification.

These results underscore the importance of adopting multimodal approaches for handling heterogeneous issue report content, emphasizing the synergistic benefits of combining text, images, and code.

References

1. Pandey, N.; Sanyal, D.K.; Hudait, A.; Sen, A. Automated classification of software issue reports using machine learning techniques: An empirical study. *Innov. Syst. Softw. Eng.* 2017, *13*, 279–297.
2. Fan, Q.; Yu, Y.; Yin, G.; Wang, T.; Wang, H. Where is the road for issue reports classification based on text mining? In Proceedings of the *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, Toronto, ON, Canada, 9–10 November 2017; pp. 121–130.
3. Pandey, N.; Hudait, A.; Sanyal, D.K.; Sen, A. Automated classification of issue reports from a software issue tracker. In *Progress in Intelligent Computing Techniques: Theory, Practice, and Applications: Proceedings of ICACNI 2016*; Springer: Berlin/Heidelberg, Germany, 2018; Volume 1, pp. 423–430.
4. Panichella, A. A systematic comparison of search algorithms for topic modelling—A study on duplicate bug report identification. In *Search-Based Software Engineering: 11th International Symposium, SSBSE 2019*, Tallinn, Estonia, 31 August–1 September 2019; Springer: Berlin/Heidelberg, Germany, 2019; pp. 11–26.
5. Lu, M.; Liang, P. Automatic classification of non-functional requirements from augmented app user reviews. In Proceedings of the *21st International Conference on Evaluation and Assessment in Software Engineering*, Karlskrona, Sweden, 15–16 June 2017; pp. 344–353.

6. Zhu, Y.; Pan, M.; Pei, Y.; Zhang, T. A bug or a suggestion? An automatic way to label issues. *arXiv* 2019, arXiv:1909.00934.
7. Kallis, R.; Di Sorbo, A.; Canfora, G.; Panichella, S. Ticket tagger: Machine learning-driven issue classification. In Proceedings of the *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, Cleveland, OH, USA, 29 September–4 October 2019; pp. 406–409.
8. Kim, M.; Kim, Y.; Lee, E. Deep learning-based production and test bug report classification using source files. In Proceedings of the *ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings*, Pittsburgh, PA, USA, 22–24 May 2022; pp. 343–344.
9. Zhifang, L.; Kun, W.; Qi, Z.; Shengzong, L.; Yan, Z.; Jianbiao, H. Classification of open source software bug reports based on transfer learning. *Expert Syst.* 2022, *e13184*.
10. Cho, H.; Lee, S.; Kang, S. Classifying issue reports according to feature descriptions in a user manual based on a deep learning model. *Inf. Softw. Technol.* 2022, *142*, 106743.