VFAST Transactions on Software Engineering
https://vfast.org/journals/index.php/VTSE@ 2025, ISSN(e): 2309-3978, ISSN(p): 2411-6246
Volume 13, Number 1, January-March 2025 *pp: 88-98*

Analyzing the Impact of Machine Learning Algorithms on Software Requirements Classification

Muhammad Bin Saleem [1], **Nosheen Qamer** [1], **Rehana Danial** [1], **Kinza Sardar** [1], **Maham Noor** [2], **Uzma Omer** [3]

[1]Department of Software Engineering, University of Management and Technology, Lahore, Pakistan ; [2]Department of Computer Science, University of Education, Lahore, Pakistan ; [3]Department of Software Engineering, University of Central Punjab, Lahore, Pakistan

Keywords: *Machine Learning, Prototypical Networks, Matching Networks, MAML, Software Requirements, Software Engineering.*

Journal Info:
Submitted:
February 22, 2025
Accepted:
March 3, 2025
Published:
March 18, 2025

Abstract

Along with the rapid growth of the world, the demand for efficient and successful software has increased swiftly. Any software has many steps for developing software and the most important step is software requirements engineering. Requirements classification can be applied manually, which requires great effort, time, and cost and the accuracy may vary. Many previous studies utilized machine learning algorithms to automate the classification process but traditional classification algorithms often require a large amount of labeled data, which can be expensive and time-consuming to collect. Few-Shot Learning (FSL) excels in situations with limited data, making it a promising alternative. This paper investigates the potential of applying Few-Shot Learning (FSL) algorithms for classifying software requirements. This study explores three prominent FSL algorithms: Prototypical Networks, Matching Networks, and Model-Agnostic Meta-Learning (MAML). These algorithms are evaluated on their ability to classify software requirements using a publicly available dataset. The results demonstrate that Prototypical Networks outperforms Matching Networks and MAML in this specific application. Matching Networks, designed for visual similarity tasks, struggle with textual data. Prototypical Networks achieve a remarkable accuracy of 82 percent, suggesting their effectiveness in learning class representations from a small number of samples. MAML also shows promising results with an accuracy of 76.9 percent. While acknowledging limitations in data pre-processing, the study concludes that FSL holds significant potential for efficient and cost-effective software requirement classification, particularly when dealing with limited labeled data.

***Correspondence author email address:** nosheen.qamar@umt.edu.pk
DOI: 10.21015/vtse.v13i1.2051

Introduction

The software, has become ubiquitous and is playing an important role in the advancement of various fields like education, business, entertainment, transportation and medicine. Requirements engineering is one of the critical phases of the software development life cycle (SDLC). The requirements engineering (RE) process is the first and crucially important step of the SDLC. RE deals with discovering, documenting, and maintaining multiple requirements and the RE process has four core activities, i.e., elicitation, analysis and negotiation, documentation, and validation [1].

Requirement engineering techniques, both traditional and advanced, play a vital role in organizing software requirements and aligning essential project needs with innovative strategies to address the complexities of modern software development. The software requirements classification deals with the segregation of the client requirements and demands found in the Software Requirements Specification (SRS) document into different categories. It is a key step in the software development pipeline, and it can be automated using machine learning techniques. This allows the industry to save on labor expenses, as a domain expert is often required, while also optimizing the process and saving crucial time [2].

A key problem that needs to be addressed during requirements classification is the inconsistency and ambiguity in the terminology used by the clients and the software engineers. This may lead to misclassification of the software requirements [3]. There are a lot of obstacles to classifying software requirements effectively.

Figuring out the real demands of the consumer is one of the biggest challenges. It is common for requirements to be vague, prone to ambiguity, and subject to change as the project progresses. Because of this, engineers have a difficult time understanding the motivation behind these requirements [4]. Software requirement classification has historically mainly relied on manual analysis, which is a laborious and error-prone process.

An encouraging approach to automate this activity was provided by the development of Deep Learning (DL) algorithms [5]. Large datasets are no problem for these algorithms, which enable them to recognize intricate patterns within specifications and classify them appropriately. However, the availability of massive volumes of labeled training data is critical to the efficacy of Deep Learning (DL) techniques. This creates a major bottleneck in the field of software requirements, where datasets may be scarce and domain-specific. Here's where Few-Shot Learning (FSL) shows itself as a potent substitute [6]. Unlike classic deep learning, FSL performs well in situations where there is a shortage of training data. Even when given a small number of labeled instances, FSL models may quickly adapt to new classification tasks by utilizing pre-existing information and effective learning strategies [7].

This study investigates the possibilities of FSL for classifying software requirements, concentrating on sophisticated models like Prototypical Networks [20], Matching Networks [21], Model-Agnostic Meta-Learning(MAML) [22]. At the vanguard of FSL research, these algorithms show great promise in terms of adapting to novel classification tasks with little to no labeled data. The objective of this research is to examine the effectiveness of several FSL algorithms in this field and assess their superiority over conventional Deep Learning (DL) 2 methods. We will also look into the opportunities and difficulties of incorporating FSL into the software development lifecycle [5].

The remainder of the paper is structured as the next section gives an overview of the selected algorithms, section 3 elaborates and provides an overview of the related work, whereas section 4 describes the methodology of the paper. Section 5 discusses the results and lastly, section 6 concludes the paper along with some future directions.

literature review

In the initial study by Jonas Wikler and Andreas Vogelsang [8] the authors utilized conventional Neural Networks (CNNs) to automate the categorization of software requirements. However, they did not distinguish between functional and non-functional requirements. While the use of CNNs for text classification if innovative, the study's limitation lies in its failure to differen-

tiate between these two important types of requirements.

The second paper by Gaith Y.Quba et.al [9] focused on automating the classification of software requirements into functional and non-functional categories using machine learning techniques such as Support vector Machines (SVM) and K-Nearest Neighbors (KNN). Using the PROMISE exp dataset, they evaluated their approach based on precision, recall, and F-measure. Although their technique provides an efficient way to categorize software requirements, it may be constrained by the representativeness of the dataset and lacks a comprehensive comparison of the selected machine learning algorithms. In summary, both studies aim to automate the requirement classification. One uses CNNs and the other uses machine learning- the first failed to distinguish between the functional and non-functional requirements, while the second's reliance on the PROMISE exp dataset my limit generalizability and could benefit from a deeper algorithmic analysis. Our study expands upon these works by reviewing additional research that uses various methods and algorithms for requirement classification, summarizing the approaches, algorithms, dataset and results from each study.

In another paper [10] the authors highlight the importance of security in software development, nothing that security requirements are often inadequately addressed in Software Requirements Specifications (SRS). To tackle this issue, they propose a method to extract and classify security-related requirements from SRS using text mining and the J48 decision tree algorithm. This approach seeks to improve software reliability by identifying security requirements early in the development process. The classification methodology consists of three steps: gathering security requirements, applying text mining techniques for processing, feature selection, and TF-IDF weighting, and developing four binary predication models using J48 to identify specific security requirement types. Model performance is evaluated through Receiver Operating Characteristic (ROC) analysis.

A different study [11] addressed the automated extraction and classification of the functional and non-functional requirements (FRs and NFRs) in software development, using supervised machine learning. The authors developed a supervised learning approach based on metadata, lexical and syntactic features, and applied the Support Vector Machine (SVM) classifier. To address class imbalances, they employed under and over-sampling strategies and enriched their dataset with user comments from software reviews. They achieve impressive precision and recall rates of approximately 92% for both FRs and NFRs and about 92% precision and 90% recall for specific types of NFRs, such as security and performance.

In [12] Verma, S discusses how software engineering applies methodical and quantitative techniques to the creation, operation, and maintenance of software. Over the years, various approaches, procedures, and actions have been developed to improve software systems with varying degrees of success.

Sabale et. al [13] emphasized the importance of lifecycle models in the framework and highlighted the need to follow a coordinated methodology to improve system development. Models such as Waterfall, Prototype, and V-shaped methods were suggested. As organizations grew, the need to automate the tasks increased. The transition from manual to automated processes required the development of structured procedures in the industry.

In [14] Wang et al focused on Support Vector Machine (SVM) techniques. The process involves analyzing the characteristics of SVMs followed by a learning method that introduces SVM classification to expand the learning portion. Academic research and experimental results show that this algorithm can maintain organizational accuracy while speeding up the training process and reducing storage costs.

Almanza et. al [15] explored the use of Artificial intelligence (AI) and Deep Learning (DL) to classify software requirements. They used Convolutional Neural Networks (CNNs) which are the forefront of natural language processing tasks. The PROMISE corpus dataset was used to evaluate the model, which contained labeled data for both functional and non-functional requirements. Encouraging results were achieved without any manually crafted features, showcasing the po-

tential of CNNs for Software Requirements Classification (SRC).

In [16] M. Lu et al. presented a method for classifying users reviews into categories of non-functional (e.g., usability, portability, performance, reliability) and functional requirements. The study combined multiple organizational strategies, such as Term Frequency-inverse Documents Frequency (TF-IDF), Chi-squared, and Bag of Words, with various machine learning algorithms, including Bagging, Naïve Bayes, and J48. The comparison of F-measures for different methodology-algorithm combinations showed that the Bagging machine learning method was most effective for classifying non-functional requirements from user reviews.

In [17] the paper by Gaith Y.Quba et al. again focused on automating the classification of software requirements into functional and non-functional categories using machine learning methods such as SVM and KNN. They evaluated their models using precision, recall, and F-measure, relying on the PROMISE exp dataset. Although the approach is effective, it is limited by the representativeness of the dataset and lacks an in-depth comparison of the chosen algorithms.

Another study [18] introduced a novel approach for classifying non-functional requirements (NFRs) using a CNN-based multi-label classifier. The goal was to automate the classification of stakeholder requirements into five distinct NFR categories: reliability, usability, portability, maintainability, and efficiency.

Lastly in [19] the authors proposed a method for detecting requirements smells by combining deep learning with traditional natural language processing (NLP) techniques. The approach focused on five types of smells defined by the ISO/IEC/IEEE 29148 standards. However the model suffered from overfitting due to an imbalance dataset, insufficient instances, incorrect labeling, and noise.

Selected Models

In modern machine learning research, there is a growing interest in finding efficient approaches to deal with the problems that arise when there are few labeled examples available for each class. In the area of few-shot learning, which is a subfield that deals with classification tasks, two landmark articles provide an overview of the state of the art and future directions in this field, providing in-depth knowledge of the approaches and uses intended to alleviate the limitations brought about by data scarcity.

few-shot learning models' resilience and ability to generalize. Overfitting is reduced and model performance is improved by artificially growing pre-existing datasets using augmentation techniques. Furthermore, feature augmentation techniques are essential for improving the quality of features that are extracted, which strengthens the model's discriminative power. By utilizing strategies like feature engineering, dimensionality reduction, and feature selection, these approaches enable few-shot learning models to identify subtle patterns in the few labeled data. Few-shot learning is a viable approach for developing predictive models with confined labeled cases in the healthcare domain, where rare diseases and data privacy concerns are prevalent [23].

Furthermore, even in situations where huge datasets are available, the computational cost of training on large datasets and the labor-intensive nature of data annotation call for the investigation of more effective learning approaches. In light of this, zero-, one-, and few-shot learning techniques provide a workable answer by allowing models to extract useful information from sparse data while reducing computing demands and annotation loads. By utilizing methods like generative modeling, episodic training,and meta-learning, these strategies enable machine learning models to efficiently generalize from small numbers of labeled samples, leading to more precise medical diagnoses and prognoses. To summarize, the combination of cutting-edge techniques and practical applications highlight the growing importance of few shot learning strategies in tackling the problems caused by sparse labeled data [24].

A new era in machine learning research is being ushered in by the development of transfer learning, data augmentation, and feature enhancement techniques, which include anything from picture classification to medical prediction systems. Models of this age are skilled at producing exact predictions in

contexts with limited resources and at deriving useful insights from sparse datasets. Fewshot learning has the potential to revolutionize a wide range of fields, from computer vision to healthcare, and this promise is evident as research in this area advances. This will open the door for future machine learning systems that are more resilient and flexible.

Prototypical Networks:

Jake Snell et al. presented Prototypical Networks (Prototypical Networks in 2017 [20] with help of learning a metric space where classification may be carried out by computing distances between prototype representations of each class, it is intended to overcome the problem of learning from sparsely labeled data. By calculating a centroid (prototype) for every class in the embedding space, Prototypical Networks gains the ability to generalize to new classes. It allocates a query sample to the class with the closest prototype during inference [20].

Prototypical networks designed for few-shot classification, tackle the challenge of classifying new classes with limited data by learning a metric space where class prototypes represent the mean embeddings of class-specific examples. This approach involves a neural network that maps inputs into an embedding space, where distances between query points and prototypes are calculated to assign classifications. The model employs episodic training, mimicking the test scenario by sampling small datasets, and uses squared Euclidean distance to optimize its performance. Unlike complex meta-learning methods, prototypical networks adopt a simple inductive bias, yielding superior results on few-shot tasks like Omniglot and mini ImageNet datasets. Furthermore, the model extends to zero-shot learning by embedding class metadata as prototypes, achieving state-of-the-art accuracy on datasets such as CU-Birds. This demonstrates its versatility and efficiency in limited-data settings [20].

Matching Networks:

Another few-shot learning technique is Matching Networks, presented by Oriol Vinyalsin 2016 figuring out a similarity function between a query sample and a support set of labeled samples [21], it solves the issue of one-shot learning. In order to determine the classification result, Matching Networks computes the weighted sum of the support set embeddings using an attention technique. This result is then compared to the query sample embedding. It works especially well in situations where the size of the support set is minimal [21].

Matching Network [21] tackles the challenge of one-shot learning by combining metric learning with external memory augmentation. It uses a deep neural network to map a small labeled support set alongside an unlabeled query to a feature space, allowing it to pre-dict the query's label without requiring fine-tuning. Metric learning enables the network to compare examples within an embedding space, while the external memory helps store and recall information from previous tasks.

Model-Agnostic Meta-Learning(MAML):

A meta-learning technique called MAML was presented by Chelsea Finn et al. in 2017 [2022]. Its goal is to learn a model's initialization parameters so that it may quickly adapt to new tasks with a small amount of training data. Through gradient descent updates, MAML acquires a strong initialization that enables quick adaptation. It has achieved state-of-the-art performance when applied to a variety of few-shot learning applications, such as image classification and regression [22].

The model-agnostic meta-learning approach aimed at enabling models to quickly adapt to new tasks with minimal data. The core idea is to optimize the model's parameters during training so that it can generalize few-shot after just a few gradient updates on a new task, even with limited examples. This technique involves two levels of optimization: an outer loop that adjusts the model's initial parameters across multiple tasks, and an inner loop where the model is fine-tuned with a small number of gradient steps for each new task. The method has demonstrated state-of-the-art performance on few-shot image classification benchmarks, and outperformed previous models in few-shot regression, and improved the efficiency of reinforcement learning by accelerating policy fine-tuning. Overall, the approach offers a versatile solution for a range of tasks by preparing models to adapt quickly and ef-

fectively, even with sparse data.

These algorithms represent the state-of-the-art in few-shot learning research and have shown promise in a number of fields, such as natural language processing, computer vision, and now the classification of software needs, which you are considering. [23].

Research Methodology

This research employed a four-phased methodology to classify software requirements using few-shot learning algorithms. Figure 1 illustrates the methodology of the research which is further explained below.

Data Aqquisition

We used dataset on Kaggle called "Software Requirements Dataset," by Iam Vaibhav [99]. The dataset contains 977 rows (Excluding headers) and two columns, Type and Requirement. Type refers to the type of software requirement and the Requirement column describes the requirements. Labeling in the data set refers to the following: Functional (F), Functional Requirement (FR), Availability (A), Fault Tolerance (FT), Legal (L), Look and Feel (LF), Maintainability (MN), Operational (O), Performance (PE), Portability (PO), Scalability (SC), Security (SE), Usability (US), and Non-Functional Requirement (NFR). We removed duplicate requirements and come up with 976 requirements. The data set was kept in two column labels that featured single words (such as 'FR' or 'US') that indicated the type of demand. The goal variables for the classification job were these labels.

Table 1. Attributes Details

Platform	Precision
Platform	60 Google Colab
Libraries	Numpy, Pandas, Scikit learn, Matplotlib
Models	Prototypical Networks Matching Network Model-Agnostic Meta-Learning
Iterations	20
Total records	978
Training dataset	782 instances
Testing dataset	195 instances

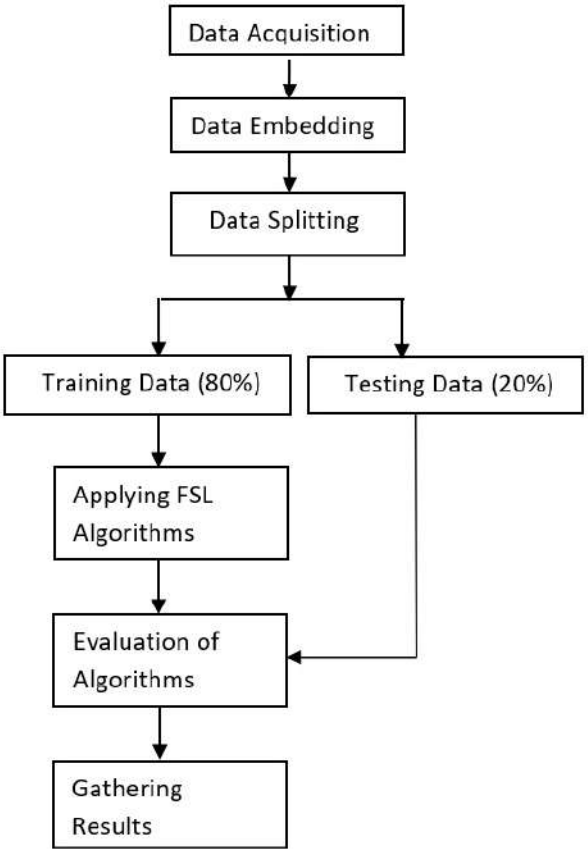

Figure 1. Research Methodology

Data Embedding

For supervised learning models to function at their best, a substantial amount of labeled data is usually needed. Nevertheless, learning new concepts from sparse data presents a problem for few-shot learning systems. In order to overcome this obstacle and improve model performance, the study used data embedding methods. Word embedding captures a word's semantic links with other words in a vocabulary by condensing the meaning of a word into a dense vector. Since the compressed representation already contains contextual information within the vector, it enables the model to learn from a smaller dataset [4]. Furthermore, word embeddings help with similarity matching, which is an essential component of few-shot learning. Comparing new data points to a small number of labeled samples is a common step in the classification process. The model can calculate the

dis-tance between the labeled instances' embeddings and the new data point's embedding thanks to word em-beddings. The closest match can then be found using methods like cosine similarity, which will help with the classification choice [2]. Sent2Vec, a well-known NLP method for creating sentence embeddings, was used in the study because of these benefits. This method captured the semantic content of each demand by converting the natural language descriptions of the requirements (Feature column) into dense numerical vectors.

Data Splitting

Following data collection, the dataset was divided into training and testing sets using a standard data-splitting technique. It is probable that stratified random sampling was employed to preserve the distribution of classes in both sets. An eighty percent training set and a twenty percent testing set of data were used in this study. This division guarantees that the model is trained on a representative sample of the data, with the remaining portion set aside for an objective assessment of the model's performance on untested data.

Data Training

The preprocessed and embedded data was subsequently employed to train three distinct few-shot learning algorithms: Prototypical Networks (Prototypical Networks): Prototypes are basically averaged embeddings that represent each class in the few-shot context, and this approach makes use of them. The degree to which a fresh data point resembles these prototypes determines its classification [9].

Application of FSL models

In this stage, the method learns a distance metric between embedded data points using a Siamese neural network architecture. The network is trained to recognize dissimilar pairings from distinct classes and comparable pairs from the same class of data points. A new data point is classified into the class with the most similar embedding by the model after it has been compared to the support set, which consists of a small number of labeled examples. A relation module discovers a relationship between a query data point (new classification requirement) and a support set (few labeled instances). The model is trained to discriminate between relationships between data points from distinct classes and those from the same class. Based on the learned relations in the support set, the relation network uses classification to predict the class label for the query data point [6].

Data Testing

Following the dataset's training on the aforementioned algorithms, 20% of the data was tested to ensure each algorithm's accuracy. The results section goes into further detail about the obtained results.

Results and Discussion

Table 2-5 shows our results for the three (i.e. Prototypical network, matching networks, and MAML) few shot learning models' results. This study evaluated the capacity to categorize unseen text input and learn from a limited number of labeled instances.

Table 2 and Figure 2 shows the results summary of the prototypical model based on three selected metrics i.e. accuracy, precision, and recall. Results reveal that during the training period, the values of metrics are a bit lower than validation and testing.

Table 3 demonstrates the results summary of the Matching networks. The results show that this model does not perform good as the values are very low in all the stages like training, validation and testing as depicted in Figure 3.

Table 4 points out the results of the Model Agnostic Metra-Learning model. The results show that this model outperformed in all the stages like training validation and testing as shown in Figure 4.

It is clear from the data available in Table 5 that Prototypical Network lies in the top position and the Model-agnostic meta-learning (MAML) performed well with second whereas the Matching network model perform least in this comparative analysis. The reason behind this low performance could be that it was mainly intended for visual similarity tests. This is to be expected as Matching Networks' fundamental workings are designed to capture associations between visual elements. Regarding text classification, Matching Networks results were as follows: a respectable 33.3% accuracy, a worrisome 20% recall, and a meager 10% precision. These findings emphasize how crucial it is to use algorithms designed especially for textual data, since generic methods may not be able to adequately capture the subtleties of language.

A bar graph comparing the recall, accuracy, and precision of three machine learning models—Matching Networks, Prototypical Networks, and MAML—is displayed in Figure 2. The three models are displayed along with the percentage of the three models. Figure two bars show that Matching Networks had the lowest accuracy, precision, and recall values. MAML came out on stage as the leader, in sharp contrast. This algorithm makes use of a technique that involves learning class prototypes from the few few-shot samples that are given.

Figure 2 depicts the Prototypical Network's remarkable 82% accuracy, 60% recall, and 72% precision, it looks like this method is surprisingly well-suited for text classification tasks. These results imply that Prototypical Networks is an effective tool for handling scenarios with minimal labeled data because of its capacity to extract the essence of a class from a small number of samples. Further results showed encouraging outcomes, obtaining 76.9% accuracy, 68% recall, and 65% precision. The fundamental strength of MAML is its capacity to learn an appropriate initialization for the parameters of the model. This enables quick adaptability, even with sparse data, to novel categorization tasks. Even though MAML didn't quite match Prototypical Networks' performance in this particular study, it nevertheless offers a strong substitute for few-shot text classification, especially where quick adjustment to new categories is crucial.

Table 2. Summary of Prototypical Networks

Model	Training (%)	Validation (%)	Testing (%)
Accuracy	60	64	89
Precision	72	77	79
Recall	82	85	83

Table 3. Summary of Matching Networks

Model	Training (%)	Validation (%)	Testing (%)
Accuracy	20	34	49
Precision	10	27	29
Recall	33	45	39

Table 4. Summary of Model-Agnostic Meta-Learning (MAML)

Model	Training (%)	Validation (%)	Testing (%)
Accuracy	68	67	77
Precision	65	67	69
Recall	77	75	73

Conclusion and Future Work

While the study's data pre-processing was limited due to time restrictions, the outcomes show that Prototypical Networks have a distinct advantage in the few-shot text categorization space. It is necessary to conduct more research to determine how significant text pre-processing might affect Prototypical Networks' functionality and possibly increase its performance.

Overall, our work demonstrates the enormous potential of few-shot learning algorithms for text classification that is both economical and time-efficient, particularly in situations when traditional machine learning techniques are severely hampered by a lack of data. These results are highly valuable for many situations where efficient text categorization is hampered by a lack of labeled data.

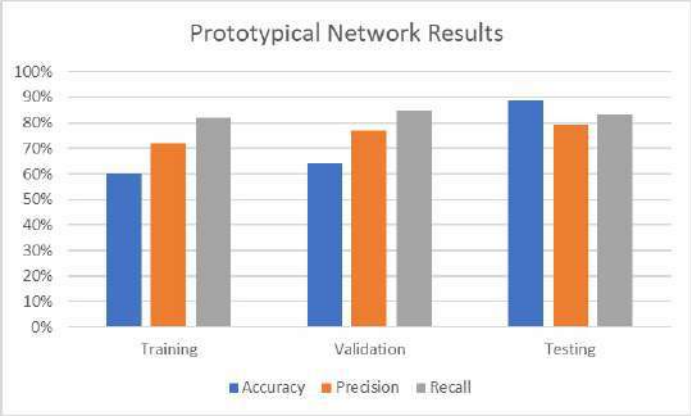

Figure 2. Results Summary of Prototypical Networks

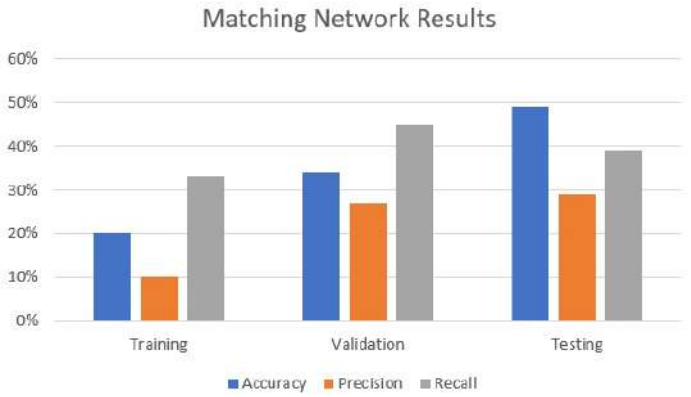

Figure 3. Results Summary of Matching Networks

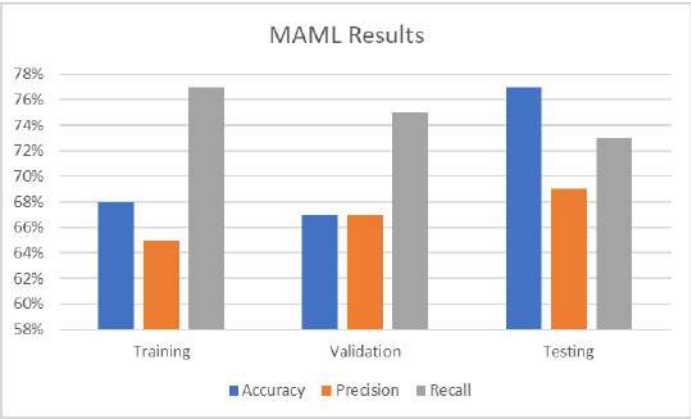

Figure 4. Results Summary MAML

Table 5. Detailed comparative Analysis of the Models

Model	Recall (%)	Precision (%)	Accuracy (%)
Prototypical	60	72	82
Matching	20	10	33.3
MAML	68	65	77

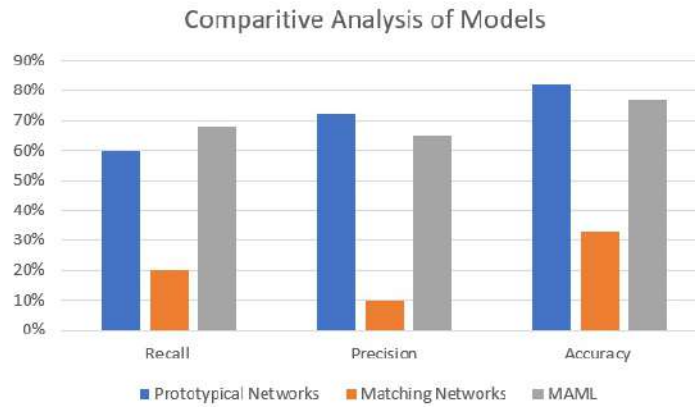

Figure 5. Comparative Analysis of Prototypical Networks, Matching Networks, and MAML Models

Future research in few-shot learning for software requirements classification can focus on improving model generalization across diverse domains by leveraging advanced meta-learning techniques and domain adaptation strategies. It would also be worthwhile to incorporate contrastive learning and prompt-based methods to enhance classification performance with minimal labeled data. Additionally, it would be interesting to explore hybrid approaches that combine rule-based heuristics with neural models. The use of synthetic data generation and augmentation techniques, along with reinforcement learning may further optimize model adaptability.

Author Contributions

Muhammad Bin Saleem: Conceptualization, Methodology, Software **Nosheen Qame**: Supervision. **Rehana Dania**: Visualization, Investigation. **Kinza**

Sarda:Data curation, Writing- Original draft preparation.**Maham Noor**: Software, Validation. **Uzma Omer**: Writing- Reviewing and Editing

Compliance with Ethical Standards

It is declare that all authors don't have any conflict of interest. It is also declare that this article does not contain any studies with human participants or animals performed by any of the authors. Furthermore, informed consent was obtained from all individual participants included in the study.

Funding

No external funding was received for this study.

References

[1] K. Pohl, *Requirements Engineering: Fundamentals, Principles, and Techniques*. New York: Springer, 2011, pp. 53–55.

[2] M. V. Mäntylä, F. Calefato, and M. Claes, "Natural language or not (NLON): A package for software engineering text analysis pipeline," in *Proc. 15th Int. Conf. Mining Softw. Repositories (MSR '18)*, New York, NY, USA: ACM, 2018, pp. 387–391.

[3] N. Qamar, N. Sabahat, A. Mashmool, and A. Mosavi, "Evaluating the impact of pair documentation on requirements quality and team productivity," *arXiv preprint arXiv:2304.14255*, 2023.

[4] D. Canedo and B. C. Mendes, "Software requirements classification using machine learning algorithms," presented at the *2020 IEEE International Conference on Machine Learning*, Sept. 2020.

[5] H. Xu and G. Xu, "Software requirements classification with deep learning: A bibliometric review," presented at the *2024 IEEE International Conference on Software Engineering*, Mar. 2024.

[6] Y. Wang, Q. Yao, J. T. Kwok, and L. M. Ni, "Generalizing from a few examples: A survey on few-shot learning," unpublished, June 12, 2020.

[7] Y. Liu, H. Zhang, W. Zhang, G. Lu, Q. Tian, and N. Ling, "Few-shot image classification: Current status and research trends," *Electronics*, vol. 11, no. 11, p. 1752, 2022.

[8] J. Winkler and A. Vogelsang, "Automatic classification of requirements based on convolutional neural networks," in *Proc. 2016 IEEE 24th Int. Requirements Eng. Conf. Workshops (REW)*, 2016, pp. 39–45.

[9] G. Y. Quba, H. Al Qaisi, A. Althunibat, and S. AlZu'bi, "Software requirements classification using machine learning algorithms," in *Proc. 2021 Int. Conf. Inf. Technol. (ICIT)*, 2021, pp. 685–690.

[10] R. Jindal, R. Malhotra, and A. Jain, "Automated classification of security requirements," in *Proc. 2016 Int. Conf. Adv. Comput., Commun. Informatics (ICACCI)*, 2016, pp. 2027–2033.

[11] Z. Kurtanovic and W. Maalej, "Automatically classifying functional and non-functional requirements using supervised machine learning," in *Proc. 2017 IEEE 25th Int. Requirements Eng. Conf. (RE)*, 2017, pp. 490–495.

[12] S. Verma, "Analysis of strengths and weaknesses of SDLC models," *Int. J. Adv. Res. Comput. Sci. Manage. Stud.*, vol. 2, no. 3, 2014.

[13] R. G. Sabale and A. R. Dani, "Comparative study of prototype model for software engineering with system development life cycle," *IOSR J. Eng.*, vol. 2, no. 7, pp. 21–24, 2012.

[14] R. Xiao, J. Wang, and F. Zhang, "An approach to incremental SVM learning algorithm," in *Proc. ISECS Int. Colloq. Comput., Commun., Control, Manage.*, 2008, vol. 1, pp. 352–354.

[15] R. Navarro-Almanza, R. Juarez-Ramirez, and G. Licea, "Towards supporting software engineering using deep learning: A case of software requirements classification," in *Proc. 2017 5th Int. Conf. Softw. Eng. Res. Innov. (CONISOFT)*, 2017, pp. 116–120.

[16] M. Lu and P. Liang, "Automatic classification of non-functional requirements from augmented app user reviews," in *Proc. 21st Int. Conf. Eval. Assess. Softw. Eng.*, Karlskrona, Sweden, 2017, pp. 344–353. doi:10.1145/3084226.3084241.

[17] G. Y. Quba, H. Al Qaisi, A. Althunibat, and S. AlZu'bi, "Software requirements classification using machine learning algorithms," in *Proc. 2021 Int. Conf. Inf. Technol. (ICIT)*, 2021, pp. 685–690.

[18] M. Sabir, C. Chrysoulas, and E. Banissi, “Multi-label classifier to deal with misclassification in non-functional requirements,” in *Trends Innov. Inf. Syst. Technol.*, vol. 1, Springer, 2020, pp. 486–493.

[19] M. K. Habib, S. Wagner, and D. Graziotin, “Detecting requirements smells with deep learning: Experiences, challenges and future work,” in *Proc. 2021 IEEE 29th Int. Requirements Eng. Conf. Workshops (REW)*, 2021, pp. 153–156.

[20] J. Snell, K. Swersky, and R. Zemel, “Prototypical networks for few-shot learning,” in *Adv. Neural Inf. Process. Syst.*, vol. 30, 2017.

[21] O. Vinyals, C. Blundell, T. Lillicrap, and D. Wierstra, “Matching networks for one-shot learning,” in *Adv. Neural Inf. Process. Syst.*, vol. 29, 2016.

[22] C. Finn, P. Abbeel, and S. Levine, “Model-agnostic meta-learning for fast adaptation of deep networks,” in *Proc. Int. Conf. Mach. Learn.*, PMLR, 2017, pp. 1126–1135.

[23] S. Kadam and V. Vaidya, “Review and analysis of zero, one and few-shot learning approaches,” in *Proc. Int. Conf. Intell. Syst. Design Appl.*, 2018.

[24] A. Melnikov *et al.*, “A guide to few-shot learning with embeddings,” Medium, Aquarium Learning, Mar. 27, 2020.