Erzincan Üniversitesi
Fen Bilimleri Enstitüsü Dergisi
2025, 18(2),476-494
ISSN: 1307-9085, e-ISSN: 2149-4584
Araştırma Makalesi

Erzincan University
Journal of Science and Technology
2025, 18(2),476-494
DOI: 10.18185/erzifbed.1590024
Research Article

Leveraging Machine Learning and Transformers to Identify Domain-Specific Services Decomposition in Legacy Systems

Isil KARABEY AKSAKALLI[1*]

[1]Erzurum Technical University, Engineering and Architecture Faculty, Computer Engineering Department

Received: 22/11/2024, **Revised:** 21/02/2025, **Accepted:** 11/03/2025, **Published:** 31/08/2025

Abstract
Service-oriented architecture, one of the popular software architectures that have become very popular in recent years, has scalability, isolation and flexibility as it consists of smaller and independent domain-specific services compared to monolithic systems. For this reason, the transition from monolithic monolithic systems to service-oriented architectures is becoming widespread for large-scale applications with millions of users to have an easily manageable, scalable and flexible structure. In this study, the effectiveness of various machine learning models and different types of tokenization methods were evaluated by analyzing static source code to decompose monolithic legacy systems into domain-specific services. Lightweight machine learning algorithms and transformer-based tokenizers were applied to the FXML-POS legacy system and model performance were evaluated using precision, recall, accuracy, and F1 score metrics. Experimental results indicate that all transformer-based feature extraction models achieve strong performance with an F1 score of 91.9% using Random Forest and Logistic Regression classifiers. Furthermore, it has been observed in the experimental results that the Word2Vec vectorization method outperforms TF-IDF in most scenarios and a maximum F1 score of 97.2% is achieved using the Random Forest Classifier. These results underscore the utility of advanced embedding techniques and classifiers in the accurate identification of domain-specific service components.

Keywords: Decomposition using source code, Static analysis, Transformer-based tokenizers, Word embeddings, Machine learning

Eski Sistemlerde Alana Özgü Hizmet Ayrıştırmasını Belirlemek İçin Makine Öğreniminden ve Transformatörlerden Yararlanma

Öz

Son yıllarda oldukça popüler hale gelen yazılım mimarilerden biri olan servis odaklı mimari monolit sistemlere göre daha küçük ve bağımsız alana özgü hizmetlerden oluştuğundan ölçeklenebilirlik, izolasyon ve esnek yapıya sahiptir. Bu nedenle milyonlarca kullanıcıya sahip büyük ölçekli uygulamaların kolay yönetilebilir, ölçeklenbilir ve esnek bir yapıya sahip olması için monolitk tek parçalı sistemlerden servis odaklı mimarilere geçiş yaygınlaşmaktadır. Bu çalışmada, tek parçalı eski sistemleri alana-özgü hizmetlere ayrıştırmak için statik kaynak kod analizi yapılarak çeşitli makine öğrenimi modelleri ve farklı tokenleştirme yöntemlerinin etkinliği değerlendirilmektedir. Hafif ağırlıklı makine öğrenimi algoritmaları ve dönüştürücü tabanlı tokenleştiriciler FXML-POS eski sistemine uygulanmıştır ve model performansı hassasiyet, geri çağırma, doğruluk ve F1 skor metrikleri kullanarak değerlendirilmiştir. Deneysel sonuçlar, tüm transformatör tabanlı özellik çıkarım modellerinin Rastgele Orman ve Lojistik Regresyon sınıflandırıcılarını kullanarak %91,9'luk bir F1 puanı ile güçlü bir performans elde ettiğini göstermektedir. Ayrıca, Word2Vec vektörleştirme yönteminin çoğu senaryoda TF-IDF'den daha iyi performans gösterdiği ve Rastgele Orman Sınıflandırıcısı kullanılarak %97,2'lik maksimum bir F1 puanı elde edildiği deneysel sonuçlarda görülmüştür. Bu sonuçlar, alan-özgü hizmet bileşenlerinin doğru bir şekilde tanımlanmasında gelişmiş yerleştirme tekniklerinin ve sınıflandırıcıların yararlılığını vurgulamaktadır.
Anahtar Kelimeler: Kaynak kodu kullanarak ayrıştırma, Statik analiz, Transformatör tabanlı belirteçleyiciler, Kelime yerleştirmeleri, Makine öğrenimi

*Corresponding Author: isil.karabey@erzurum.edu.tr
Isil KARABEY AKSAKALLI, https://orcid.org/0000-0002-4156-9098

1. Introduction

With the rapid development of technology, the number of users using web infrastructures is increasing significantly. This necessitates software architectures to be more comprehensible, scalable and flexible [1]. Therefore, companies and organizations with software infrastructures with large user bases are decomposing their monolithic software architectures into smaller, service-based architectures to meet the demands of the market [2-4]. The current market requires flexible architectures that can provide easily scalable solutions to meet frequently changing user demands and in the case of a rapidly growing user base [5-7]. For these reasons, companies are moving towards service-oriented architectures. While monolithic architectures, also known as legacy systems, are being modernized, in service-oriented architectures it is necessary to identify service types and decompose the source code according to these types. Identifying reusable functionalities and evaluating these functionalities as candidate services is a critical step in the transition from legacy systems to service-oriented systems [8]. Therefore, it is important that the Service Identification (SI) process, which is the first stage of the transition process, is carried out completely and effectively [9].

There are many SI approaches in the literature [10-15]; however, the most of these approaches have some limitations due to their service identification accuracy and the need for various types of data such as business process model, Unified Modeling Language (UML), use cases, activity diagrams. Methods that use these diagrams are called domain analysis. On the other hand, static analysis, another legacy code parsing method, provides a faster and more accurate identification process by using the relationships between functions and classes in the source code and the classes and interfaces imported in the source files. This process is based on the accurate identification of service types through associated code patterns. In this study, based on the importance of identifying service types in the decomposition of monolithic legacy systems into service-based architectures, the source code of a monolithic project is decomposed into three main service types named application, utility and entity using static analysis, transformer models and machine learning (ML) methods. On the JavaFX-Point of Sales (FXML-POS) project [16], an open source legacy system, twelve different ML methods and five different embedding methods were applied and their performances were compared based on accuracy, precision, recall and F1 score metrics. Experimental results show that ML and transformer approaches can automatically recognize domain-specific service types in promising performance. The main contributions of this study are listed below:

- A machine learning-based service decomposition approach is proposed as an alternative to time-consuming methods that rely on expert intervention during the transition from legacy systems to service-oriented architectures, as well as to manual or heuristic-based approaches.
- Instead of traditional semantic or rule-based methods that require expert knowledge and documentation such as UML diagrams, business process models, or architectural specifications a static source code analysis approach is adopted to eliminate document dependency.

- The feature extraction performance of transformer-based tokenizers and word embedding models is evaluated using ML methods, providing a more lightweight approach compared to traditional techniques.

The remaining sections of the paper are organized as follows: In the second section, the literatüre studies regarding the service-type decomposition are presented as comperatively. In the third section, the process of identifying domain-specific services, the transformation of source code into vectors using transformer-based models and word embedding methods, and the service-based classification process are described in detail. In the fourth section, experimental results are presented by comparing the performance of various embedding methods and ML algorithms. In the fifth and sixth sections, the findings, the limitations of the case study and applied methods, the results obtained and future directions are discussed respectively.

2. Related Work

Due to the costly and complex nature of transitioning from a monolithic architecture to service-oriented and microservices architectures, the software engineering literature proposes various methods for analyzing the codebase and decomposing it into service components. Research findings indicate that static analysis approaches based on source code, dynamic analysis leveraging runtime information, word embedding techniques, and ML models play a crucial role in facilitating the migration of legacy systems to service-oriented and microservices architectures. In the transformation of legacy systems into Service-Oriented Architecture (SOA) or Microservices Architecture (MSA), Abdellatif et al. [8] proposed the ServiceMiner method, which applies a codebase analysis-driven approach. This method aims to extract specific functional clusters from the source code of legacy systems by identifying service types. As a result of the experiments, it was stated that the ServiceMiner method achieved 77.9% precision, 66.4% recall and 71.7% F1 score metrics in decomposing monolith structures into service types. The topic modeling approach proposed by Brito et al. [17] aims to create microservice clusters by analyzing textual information to identify functional components in the system. In terms of cohesion and domain level metrics, the independence value of 200 open source projects on Github was determined as 0.6 and the conceptual modularity value as 0.4. Although this method offers promising results in enhancing service independence and modularity, its exclusive reliance on textual analysis may lead to overlooking dependencies and semantic relationships within the code. In another study by Trabelsi et al. [18], a method called MicroMiner was proposed by using both static code relations analysis and semantic analyses from source codes. The proposed method was tested on four different open source systems. One of the open source systems tested was FXML-POS and an accuracy rate of 92% was obtained with the SVM method using the dataset generated from this system.

In the literature, ML and natural language processing (NLP) methods are frequently used for code decomposition. In the study by Al-Debagy and Martinek [1], a hierarchical clustering algorithm and a Code2Vec based NLP feature extraction method were used to group vectorially similar components by creating meaningful vector representations from the code. In terms of the metrics Cohesion at Message Level (CHM) and Cohesion at Domain Level (CHD), the algorithm shows promising results compared to other decomposition systems. This approach simply consider dependencies within the code and does not directly model contextual service relationships, despite the fact that it is more capable of analyzing semantic similarities within the code. On the other hand, Trabelsi et al. [19] proposes the MAGNET approach, and presents a microservice identification mechanism that uses Graph Neural Networks (GNNs) to integrate the static, semantic, and structural aspects of the code. The experimental findings show that on open-source systems, GNN with semantic analysis have 68% recall and 56% precision. Although this method offers an automated method for identifying service components, the model's training can be time-consuming, and the system requires higher recall, accuracy, and precision values. Another approach used for microservice decomposition is evolutionary optimization methods. The MSExtractor method proposed by Sellami et al. [19] focuses on optimizing metrics such as service granularity, independence, and cohesion using multi-objective optimization techniques. The proposed method has 0.6-0.7 CHD and 0.4-0.5 CHM using the open-source systems. However, the applicability of such methods to large-scale systems is often limited due to their high computational costs.

Unlike existing studies, this research proposes an approach that combines static code analysis with transformer-based feature extraction and employs lightweight ML models to more accurately and less costly identify service components. The feature extraction performance of transformer-based models is compared against Word2Vec and the traditional TF-IDF vectorization method, and the classification models that best capture service types are evaluated using ML techniques. In the proposed model, pre-trained models in transformers are not used as classifiers to minimize computational complexity and provide a scalable approach for large-scale monolithic architectures. In this context, the proposed model not only reduces computational costs but also improves service decomposition accuracy.

3. Material and Methods

To decompose legacy systems into smaller services, service types must first be identified. For this purpose, SI taxonomies that deal with different aspects of service types have been analyzed. The SI taxonomies developed by Abdellatif et al. [10] divided SI approaches into four main categories named Input, Process, Output and Usability. Within the scope of this study, we focused on Domain-specific service types under the Output category. These service types include Application, Utility and Entity services. The goal is to decompose the legacy system into these type-services by performing source code analysis. The steps of the applied methodology are described in the following subsections. Furhermore, a general flowchart of the service-type classifiction process of a monolithic legacy system presented in Figure 1.

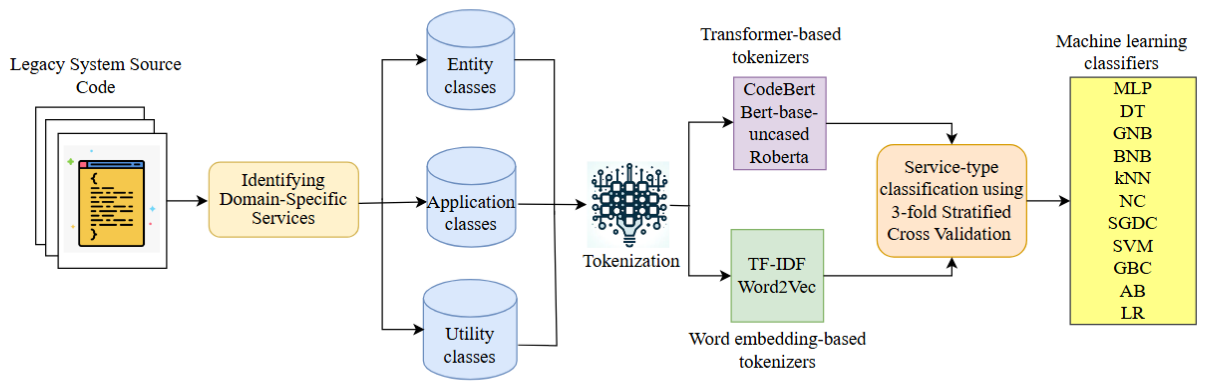
Legacy System Source Code
Identifying Domain-Specific Services
Entity classes
Application classes
Utility classes
Tokenization
Transformer-based tokenizers
CodeBert
Bert-base-uncased
Roberta
TF-IDF
Word2Vec
Word embedding-based tokenizers
Service-type classification using 3-fold Stratified Cross Validation
Machine learning classifiers
MLP
DT
GNB
BNB
kNN
NC
SGDC
SVM
GBC
AB
LR

Figure 1. A general flowchart diagram of the legacy source code analysis into domain specific service-type classification (MLP: Multilayer Perceptron, DT: Decision Tree, GBN: Gaussian Naive Bayes, BNB: Bernoulli Naive Bayes, kNN: k Nearest Neighbor, NC: Nearest Centroid, SGDC: Stochastic Gradient Descent, SVM: Support Vector Machine, GBC: Gradient Boosting, RF: Random Forest, AB: AdaBoost; LR: Logistic Regression)

Figure 1 illustrates the process of decomposing legacy source code into domain-specific services within the scope of this study. Initially, the legacy system source code is processed by categorizing all Java files from a legacy e-commerce application named FXML-POS [16] into application, entity, and utility classes. Within the application layer, classes containing "inventory.controller" are included. The entity layer comprises classes under "inventory.dao", "inventory.entity", and "inventory.model". Finally, the utility layer consists of classes such as "HibernateUtil", "PrintInvoice", and "login.LoginController". Then the tokenization procedures are applied to the extracted code components, followed by feature extraction using transformer-based architectures and vector space-based models. After the feature extraction process, each tokenized dataset is subjected to training and testing using the 3-fold Stratified Cross-Validation method. Subsequently, service-type classification is performed utilizing various ML techniques.

3.1. Identifying Domain-Specific Services

Layered architectures such as Model-View-Controller [20] do not sufficiently decouple function-based systems compared to the SOA and microservices [21] leading to a decrease in the performance of web applications in terms of scalability, flexibility and deployability performed against the number of users or faults. In monolithic applications, it is almost impossible to scale a particular functionality independently or to update it without affecting other functionalities. Each update to a monolithic functionality may require changes to other parts of the application [22]. In this study, we focus on Application, Utility and Entity from four main domain-specific service categories identified by Abdellatif et al. [10]. In order to separate the Enterprise services from the Application services, dynamic analysis must be used together with static analysis. The description of these services separated by functionality and domain is given below [18].

- *Enterprise services:* This service, positioned in the Presentation Layer, typically refers to highly critical services that provide and support core business functions [23]. These services are reusable, scalable and business critical. ERP services and Customer Relationship Management (CRM) services are examples of such services.

- *Application services:* The services in the Business layer are domain-specific and only provide functionality for a specific application. In addition, these services can also use or create functionality provided by Entity services [8].
- Utility services: They have common discrete functionalities such as logging, mail, print function and authentication that are necessary for domain specific services [18, 24].
- Entity services: Entity Services are designed to handle persistent data and manage all the functions related to that data (such as storage, retrieval, locking and transaction management), so they are known as CRUD (Create, Read, Update, Delete) services. This type of service manages a single data entity by encapsulating CRUD operations and related business logic [24].

3.2. Generating Embeddings from Source Codes

To generate embedding vectors from the source code, the Java extension source files of the MVC project named FXML-POS is first assigned to the “application”, “entity” and “utility” classes using the ground truth files provided by Trabelsi et al. [18]. The dictionary created as a result of this assignment maps all file names to the appropriate class labels. Thus, it is ensured that the file names are used as classified for further processing. After this process, the content of each Java file is split into tokens using transformer-based tokenizers such as CodeBERT [25], BERT-based uncased [26] and RoBERTa-base [27], as well as word embedding-based TF-IDF [28] and Word2Vec [29] vectorization models. The content is divided into chunks according to the maximum token length and the embedding vectors of each chunk are calculated. The calculated vectors are added to a list, which is transformed into a numpy matrix of all embedding vectors and the average embedding vector is calculated. This average value is added to the relevant dataset of the file. Then, the embedding vectors are saved in a dataset in csv format. This dataset contains the name of each Java file, its domain class and the embedding vectors. Since there are 55 Java files in the FXML-POS project, the dataset has 55 rows and 770 columns depending on the line of code of the files. Among these columns, all data except the class name and the domain where the class is located are embedding vectors.

3.3. Service Type Classification

The dataset generated through transformer-based tokenizer and word embedding vectorizers were classified using Multilayer Perceptron (MLP), Decision Tree (DT), Gaussian Naive Bayes (GBN), Bernoulli Naive Bayes (BNB), k-Nearest Neighbor (kNN), Nearest Centroid (NC), Stochastic Gradient Descent (SGDC), Support Vector Machine (SVM), Gradient Boosting (GBC), Random Forest (RF), AdaBoost (AB) and Logistic Regression (LR) ML methods and 3-fold Stratified Cross Validation data separation method. To analyze the classification results, accuracy, precision, recall and F1-Score metrics were evaluated to compare the performance of the machine learning methods. The algorithms used to classify service types are briefly explained in the subsections.

3.3.1. Multilayer Perceptron (MLP)

MLP is one of the most widely used types of artificial neural networks and has a feed-forward network structure consisting of an input layer, one or more hidden layers and an output layer [30]. The main feature of MLP is that it is effective in solving problems that cannot be linearly decomposed. This is possible thanks to the activation functions used in the hidden layers. The working principle of MLP is based on producing an output by processing input data through weighted connections. Each neuron applies weights to the incoming inputs, combines them and produces its output by passing it through the activation function. Mathematically, this process is expressed as equation (1):

$$h_j = f(\sum(w_{ij} * x_i) + b_j) \qquad (1)$$

In this equation, x_i represents the input variables, w_{ij} represents the weight between the input and hidden layer, b_j is the bias term and f is the activation function. A similar calculation is made for the output layer shown in the equation (2):

$$y_k = g(\sum(v_{jk} * h_i) + c_k) \qquad (2)$$

In this equation, h_i represents the output of hidden layer neurons, v_{jk} represents the weight between hidden and output layer, c_k is the bias term of output layer and g is the activation function of output layer (softmax or sigmoid).

3.3.2. Decision Tree (DT)

DT is a popular ML algorithm that classifies or regresses data by branching it according to certain rules. Each internal node represents a decision rule, branches represent possible decision paths, and leaf nodes represent the final result [31]. Decision trees can be applied in different types, especially to handle discrete or continuous variables on data. For example, Classification Trees used in classification problems help to determine the possible classes of a given variable, while Regression Trees focus on predicting for a specific target variable. In addition, more reliable results can be obtained by combining multiple decision trees with methods such as Decision Tree Forests. Among the reasons why this algorithm is preferred in this study are that it requires fewer parameters compared to other ML algorithms and that pre- and post-pruning techniques can be applied to prevent the risk of overfitting on the training data. Furthermore, decision trees can work especially effectively in situations with data incompleteness and noise, thus having a wide range of applications in various fields such as health, finance, education and text mining.

3.3.3. Gaussian Naive Bayes (GBN)

GNB is a probabilistic classifier based on Bayes theorem and assumes strong independence between features. It is particularly effective when the data are normally distributed and offers high speed and low computational cost in classification processes. The GNB algorithm is widely favoured for its scalability, especially on large datasets. Its ability to cope with missing data and its robustness against noisy data are among the most important advantages of the model. However, the independence assumption of the model may not always hold in real-world data and this may limit the accuracy of the model. However, it has been observed that it performs particularly well in areas such as text classification, biomedical data analysis and financial forecasting [32].

3.3.4. Bernoulli Naive Bayes (BNB)

BNB is a probabilistic model that belongs to the Naïve Bayes classifier family and calculates the presence or absence of features. This model, which is widely used especially in text classification problems, treats the presence or absence of each word as a binary variable. This approach gives effective results in applications such as spam filtering, document classification and sentiment analysis [33]. In this study, BNB is preferred due to its low computational cost, good performance on small datasets and its ability to deal with noisy data. However, the fact that it does not take into account word frequency may lead to information loss in some cases. Therefore, when compared to alternative methods such as Multinomial Naïve Bayes (MNB), it seems to be more suitable especially for short texts.

3.3.5. k-Nearest Neighbor (kNN)

kNN algorithm is an unsupervised ML algorithm that classifies data according to its nearest neighbours. The basic principle is that in order to determine the class of a data point, the closest is to look at the class of its k neighbours. The algorithm calculates the distance between data points, usually using metrics such as Euclidean distance, and performs classification by majority voting [34]. It is preferred in this study for service type separation because it uses a memory-based method by working in the prediction phase instead of the learning phase, allows direct use of data and does not require parameter optimization, and offers a flexible classification method by capturing non-linear relationships.

3.3.6. Nearest Centroid (NC)

NC classifier is a ML method that calculates the centroid of each class and classifies new instances according to the nearest centroid. This method is effective when the class distributions can be clearly separated from each other and offers a fast and computationally efficient alternative, especially in high-dimensional datasets. The NC classifier calculates the mean vector of each class in the training data and assigns the test data to the nearest centre [35]. This method is particularly favoured for its low computational cost and interpretability. However, its accuracy may decrease when there are significant overlapping classes or non-linear distinctions in the dataset. However, Nearest Centroid, which is seen as one of the bridges between statistical methods and machine learning, is preferred due to its simplicity and efficiency, especially in large datasets.

3.3.7. Stochastic Gradient Descent Classifier (SGDC)

SGDC is a fundamental ML algorithm for solving optimisation problems on large datasets. Unlike the classical Gradient Descent method, SGDC optimises the weights by updating only a randomly selected subset (mini-batch) instead of the entire dataset at each iteration. This approach provides a significant performance improvement, especially in training deep learning models, and greatly reduces computational costs [36]. SGDC is preferred in this study, because it provides faster model training by reducing the computational cost on such large-sized source code files. In addition, the fact that it requires less memory usage compared to traditional gradient descent methods and offers a dynamic learning process with mini-batch updates increases its effectiveness in microservice decomposition processes.

3.3.8. Support Vector Machine (SVM)

SVM is a powerful supervised learning algorithm used in classification and regression problems. It is based on finding the optimal hyperplane that best discriminates between classes and thus works effectively, especially in high-dimensional and complex data sets [37]. SVM facilitates classification by transforming linearly separable data into a higher dimensional space using kernel methods. It can be customised for different data structures thanks to different kernel functions such as linear, polynomial, radial basis functions (RBF) and sigmoid kernels. This flexibility helps to achieve the best classification performance in accordance with the structure of the dataset. It is preferred in this study because of its potential to produce more accurate results by filtering out noisy or irrelevant components in the code base during the service type separation process, to separate code components into the correct services by determining the dividing line between classes in the best way, and to separate service components more effectively by generalising even in large and complex code bases.

3.3.9. Gradient Boosting Classifier (GBC)

GBC is an ensemble learning method that builds a stronger model by iteratively training weak learners. This algorithm improves the overall accuracy of the model by focusing on reducing the error rate at each iteration and is often preferred because it achieves high accuracy rates, especially in large datasets [38]. Gradient Boosting improves prediction accuracy by creating new trees that focus on the errors of the previous model at each step. It reduces model complexity by filtering out unnecessary features with the Weighted Feature Importance (WFI) method to prevent overlearning (overfitting). Thus, faster and more efficient models can be created by using fewer but more meaningful features. In this study, GBC is preferred since it eliminates redundant data by feature selection and highlights more meaningful code components in service decomposition, provides reliable and consistent results in the service decomposition process by reducing the risk of overfitting, and optimises the processing time by using fewer computational resources.

3.3.10. Random Forest (RF)

RF is an ensemble learning algorithm consisting of many decision trees. Each tree is trained on different subsets of the training dataset and the final prediction is made by majority voting in the classification stage [39]. This approach reduces the overfitting tendency of the individual decision trees and enables a more generalised model to be constructed. Random Forest is widely used due to its advantages such as high classification accuracy, robustness to noise and resistance to overfitting. The voting process of independently generated trees increases the reliability of the model and provides advantages in terms of variable selection and error tolerance. It is preferred in this study since it minimises the negative impact of irrelevant or noisy data in code analysis on the performance of the model and the combination of multiple decision trees increases the generalisation ability of the model and minimises erroneous decompositions.

3.3.11. AdaBoost (AB)

AdaBoost (Adaptive Boosting) is an ensemble learning method that can make more accurate predictions by iteratively strengthening weak learners. This algorithm improves the accuracy of the model by focusing on the errors of the previous model in each iteration, increasing the importance of misclassified instances by weighting. AdaBoost is a preferred method due to its high success especially in small and unbalanced datasets [40]. The reasons why it is preferred in this study are that it offers high performance with low computational cost in small and medium-sized datasets, and more accurately identifies service components by combining multiple weak learners.

3.3.12. Logistic Regression (LR)

LR is a probability-based classification method, a statistical model that assigns data points to specific categories using linear discriminators. It is widely used especially in binary classification problems and allows the probability estimated by the sigmoid function to be classified according to a certain threshold value [41]. The advantages of logistic regression are that it is easy to interpret, requires low computational cost and performs strongly on small or medium-sized datasets. Its reduced susceptibility to overfitting and ability to eliminate irrelevant variables through feature selection offer a significant advantage in the decomposition of monolithic architectures into microservices.

3.4. Hyperparameter Selection

The proposed method incorporates several hyperparameters that influence the performance of classifiers. The Table 1 summarizes the key parameters used, their tuning approach, and the rationale for selection.

Table 1. Hyperparameters of ML Methods

Parameter	Value(s)	Tuning Method	Impact on Performance
Random Seed	1	Fixed for reproducibility	Ensures stability across runs

Cross-Validation	StratifiedKFold(n_splits=3, shuffle=True, random_state=1)	Stratified to maintain class distribution	Prevents data imbalance bias
Embedding Size	Not explicitly set (depends on POS embeddings)	Fixed per dataset configuration	Affects representation quality
MLP Hidden Layers	(5,2)	Manually set	Controls network complexity
MLP Optimizer	adam	Default (Adam optimizer)	Ensures stable gradient updates
MLP Regularization	alpha=1e-5	Default	Reduces overfitting
SGD Loss Function	Hinge (default for SGDClassifier)	Default	Optimized for linear separation
Random Forest Trees	100	Default	Balances accuracy and efficiency
Gradient Boosting Trees	100	Default	Reduces variance
AdaBoost Estimators	50	Default	Controls model complexity
Logistic Regression Solver	liblinear	Default (suitable for small datasets)	Ensures convergence

In this study, a fixed seed (random_state=1) was used across experiments to ensure consistent and reproducible results. Stratified k-fold validation (n_splits=3) was employed to prevent class imbalance from affecting model performance. For MLP configuration, a small network structure (hidden_layer_sizes=(5,2)) was used to avoid overfitting on a relatively small dataset. Adam optimizer and regularization (alpha=1e-5) were applied to stabilize training. As the SVM and SGD parameters, the Radial Basis Function (RBF) kernel was chosen for SVM due to its ability to handle non-linearly separable data, while SGDC used the default hinge loss for linear separability. For the tree-based classifiers such as RF and GBC, the number of trees was selected as 100 to balance computational cost and classification accuracy while AB used 50 weak learners to avoid overfitting. Besides, The liblinear solver was chosen as it is optimized for small-to-medium datasets, ensuring convergence and numerical stability fort he LR classifier.

4. Experimental Results

After generating embeddings from source codes using different types of tokenizers, various ML classification models were applied using 3-fold cross validation. Table 2 presents the performance metrics for source-code classification using different transformer-based tokenizers across these models. Among the models evaluated, LR consistently achieves the highest performance, with an F1 Score of 91.9% when using CodeBERT and BERT-based uncased tokenizers. RF and AB also demonstrate strong performance, especially with CodeBERT and RoBERTa-base tokenizers, showing high Precision, Recall, and F1 Scores. In contrast, MLP performs the weakest overall, with particularly low F1 Scores of 36.4% across all tokenizers. SGDC shows very poor performance with RoBERTa-base, with notably low Precision and F1 Scores. Among the tokenizers, CodeBERT generally provides better results compared to others, while RoBERTa-base shows slightly superior F1 Scores in some cases. Thus, the choice of tokenizer and model significantly influences classification accuracy, with CodeBERT and logistic regression emerging as the most effective combinations.

Table 2. Performance metrics of FXML-POS legacy system's source-code classification results with various transformer-based tokenizers according to different ML classifiers

Algorithm	**Transformer-Based Tokenizers/Performance Metrics (%-Average weights)**								
	CodeBERT			**Bert-based uncased**			**Roberta-base**		
	Precision	**Recall**	**F1 Score**	**Precision**	**Recall**	**F1 Score**	**Precision**	**Recall**	**F1 Score**
MLP	47.3	65.6	53.8	27.8	52.7	36.4	27.8	52.7	36.4
DT	82.7	85.2	83	77.5	79.8	78.2	82.2	81.7	81.1
GNB	84	87.5	84.8	83.8	87.2	84.8	85.6	89	86.7
BNB	80.6	81.9	79.2	75.1	76.5	73.7	78.3	80.2	77.6
kNN	87.1	90.8	88.3	78.4	76.4	72.7	88.2	92.6	90.1
NC	56.4	59.7	56.1	73.3	72.8	72.1	81.5	81.9	80.2
SGDC	81.7	81.9	79.7	81.2	83.7	81	26.1	50.8	34.4
SVM	27.8	52.7	36.4	75.1	69.1	64.3	27.8	52.7	36.4
GBC	87	90.8	88.4	81.3	85.3	83	82.4	83.5	82.1
RF	88.3	92.6	90.2	**89.6**	**94.5**	**91.9**	**89.6**	**94.5**	**91.9**
AB	88.3	92.6	90.2	80.1	79.9	79.1	81	83.5	81.3
LR	**89.6**	**94.5**	**91.9**	**89.6**	**94.5**	**91.9**	83.7	85.4	82.9

The Table 3 compares the performance of different ML models for source-code classification using TF-IDF and Word2Vec vectorizers. Word2Vec consistently delivers superior results compared to TF-IDF across most models. For instance, Logistic Regression (LR) and Support Vector Machine (SVM) show notably higher Precision, Recall, and F1 Scores with Word2Vec, achieving an F1 Score of 91% and 36.4%, respectively, with TF-IDF. GBC and RFboth achieve the highest F1 Scores of 97.2% with Word2Vec, demonstrating exceptional performance in Precision and Recall. NC and kNN also perform well with Word2Vec, showing improved metrics over TF-IDF. Conversely, models like SGDC and SVM perform poorly with Word2Vec, particularly in F1 Score, highlighting that Word2Vec may not be universally effective for all models. Overall, Word2Vec outperforms TF-IDF in most cases, particularly for models like GBC and RF, while TF-IDF yields variable results depending on the model**.**

Table 3. Performance metrics of FXML-POS legacy system's source-code classification results with TF-IDF and Word2Vec vectorizers according to different ML models

Algorithm	**Word Embedding Vectorizers/ Performance Metrics (%-Average Weights)**					
	TF-IDF			**Word2Vec**		
	Precision	**Recall**	**F1 Score**	**Precision**	**Recall**	**F1 Score**
MLP	83.5	83.6	82	89.5	94.5	91
DT	87.5	90	89.1	91.4	92.5	91.8
GNB	83	85.3	83.7	96.4	98.1	97.2
BNB	80	78.2	76	80	80	77.2
kNN	83.3	87.2	84.6	87.8	92.6	90.1
NC	88.4	92.7	90.3	89.1	83.5	84.4
SGDC	90.3	87.2	87.5	73.5	72.4	71.4
SVM	89.6	94.5	91.9	27.8	52.7	36.4
GBC	87.7	92.6	90.1	96.4	98.1	97.2
RF	**89.6**	**94.5**	**91.9**	**96.4**	**98.1**	**97.2**
AB	87.7	92.6	90.1	94.6	96.2	95.4
LR	89.6	94.5	91.9	27.8	52.7	36.4

Comparing the two tables reveals that Word2Vec consistently delivers higher performance metrics than both TF-IDF and transformer-based tokenizers. In terms of F1 Scores, GBC and RF achieve exceptional results of 97.2% with Word2Vec, significantly surpassing the highest F1 Scores of 91.9% seen in Logistic Regression (LR) and Random Forest (RF) using Bert-based uncased in the first table. Similarly, Precision and Recall values are notably higher with Word2Vec, with models like GBC and RF reaching 96.4% Precision and 98.1% Recall. In contrast, the best results with transformer-based tokenizers show lower metrics, highlighting that Word2Vec provides superior overall performance for classification tasks.

The accuracy metrics shown in Table 4 for the FXML-POS legacy system's source-code classification across various tokenizers and ML models highlight significant differences in performance. LR and RF achieve the highest accuracy with 94.5% and 92.6% respectively, using both CodeBERT and TF-IDF tokenizers. Word2Vec vectorizers generally deliver superior accuracy metrics across all models, with the highest accuracy of 98.1% observed for GBC and RF. In contrast, models such as SVM and SGDC show varied performance, with SVM reaching a maximum accuracy of 94.5% using TF-IDF, while SGDC performs best with TF-IDF but remains lower overall compared to Word2Vec results. Overall, Word2Vec outperforms other tokenizers, demonstrating its efficacy in achieving higher accuracy in source-code classification.

Table 4. Accuracy metrics of FXML-POS legacy system's source-code classification results of all tokenizers according to different ML models

Algorithm	**Accuracy Metrics**				
	Transformer-based Tokenizers			**Word Embedding Vectorizers**	
	CodeBERT	**Bert-based uncased**	**Roberta-base**	**TF-IDF**	**Word2Vec**
MLP	65.6	52.7	52.7	83.6	94.5
DT	85.2	79.8	81.7	90	92.5
GNB	87.5	87.2	89	85.3	98.1
BNB	81.9	76.5	80.2	78.2	80
kNN	90.8	76.4	92.6	87.2	92.6
NC	59.7	72.8	81.9	92.7	83.5
SGDC	81.9	83.7	50.8	87.2	72.4
SVM	52.7	69.1	52.7	94.5	52.7
GBC	90.8	85.3	83.5	92.6	98.1
RF	**92.6**	**94.5**	**94.5**	**94.5**	**98.1**
AB	92.6	79.9	83.5	92.6	96.2
LR	94.5	94.5	85.4	94.5	52.7

5. Discussion

This study presents a ML-based approach for domain-specific service decomposition in legacy systems, leveraging transformer-based tokenizers and word embedding techniques. The proposed methodology effectively classifies source code into application, entity, and utility services, demonstrating competitive performance across multiple ML classifiers. Numerous studies have proposed methodologies for service identification in legacy systems, including domain analysis-based, dynamic analysis-based, and static analysis-based approaches. For example, Abdellatif et al. [10] introduced a taxonomy for service identification, highlighting that most existing methods require supplementary artifacts such as business process models, UML diagrams, or activity diagrams. Similarly, Trabelsi et al. [18] proposed a semantic analysis-based approach for microservice identification, integrating function similarity and ML techniques. In contrast, the proposed approach in this study does not rely on additional external artifacts such as UML models or business process diagrams. Instead, it utilizes static analysis techniques combined with transformer-based and word embedding models, making it faster and more scalable than domain analysis-based methods. Furthermore, our findings indicate that Word2Vec embeddings outperform transformer-based models, particularly when combined with Random Forest and Gradient Boosting classifiers, providing a more accurate and computationally efficient solution for source code classification.

Compared to studies that employ rule-based or heuristic-based methods for service decomposition, the proposed approach provides greater generalization capability, as ML models automatically learn patterns from the source code. Moreover, this method achieves higher classification accuracy than static analysis-based studies that rely solely on traditional NLP methods such as TF-IDF, as demonstrated by the significant performance improvements observed in our experimental results. The effectiveness of fine-tuned transformer models in text classification tasks has been widely discussed in recent literature, particularly in domain adaptation scenarios [42] [43]. While our study evaluates transformer-based tokenizers with traditional NLP methods, they were not specifically fine-tuned for service decomposition. Future research could focus on fine-tuning transformers on larger software engineering datasets, similar to how domain-specific BERT models have been optimized for text categorization tasks.

Unlike manual domain analysis-based methods, which require substantial human intervention and domain knowledge, this approach offers an automated and scalable alternative for large-scale monolithic systems without the need for additional metadata such as business process models or UML diagrams. The results confirm that ML models, particularly Random Forest and Logistic Regression with Word2Vec, significantly enhance the accuracy and efficiency of service decomposition. Recent research has shown that the choice of embedding method significantly influences classification performance in various NLP tasks, including sentiment analysis and text categorization [44]. Similarly, in this study, Word2Vec demonstrated superior performance over TF-IDF in distinguishing domain-specific services from monolithic systems, supporting previous findings on the importance of embedding quality.

Despite the promising results, this study has certain limitations that should be addressed in future work. One notable limitation is the dataset size and generalizability, as the study was conducted using the FXML-POS legacy system, which consists of a relatively small number of Java classes. While the results are promising, further validation is necessary on larger and more complex legacy systems to assess the robustness and scalability of the approach. Additionally, the decomposition was performed into three service types (application, entity, and utility), whereas future research could explore more granular service categorization, including business logic and presentation layer services. Another limitation of this study is its reliance solely on static source code analysis, which may not fully capture dynamic interactions between software components. Incorporating dynamic analysis techniques, such as runtime monitoring and execution traces, could improve the accuracy of service identification by considering real-time dependencies and function calls. While transformer-based tokenizers were evaluated, they were not fine-tuned specifically for service decomposition. Future studies could focus on fine-tuning transformer models on larger software engineering datasets, potentially improving their classification performance. Moreover, while this study demonstrates strong performance in an offline experimental setup, its applicability in real-world software migration projects remains an open challenge. Future research should explore the integration of this approach into software reengineering pipelines and evaluate its effectiveness in industrial case studies to further assess its practical applicability.

6. Conclusion

In this study, we focused on domain-specific service decomposition to facilitate the transformation of monolithic architectures into service-oriented architectures. The motivation behind this research stems from the increasing need for modular, scalable, and maintainable software structures, particularly as legacy systems struggle with adaptability and performance constraints. By leveraging ML models and transformer-based tokenization techniques, we demonstrated an automated and effective methodology for identifying and classifying service types within a legacy system. The FXML-POS project, used as a case study, presented an inherent dataset imbalance due to the relatively small number of utility classes compared to application and entity classes. Despite this challenge, the proposed classification models successfully identified service types with high accuracy, showcasing the potential of data-driven approaches in software modernization efforts.

The findings indicate that transformer-based tokenizers, such as CodeBERT, RoBERTa, and BERT-based uncased, performed well, achieving precision, recall, and F1 scores of 89.6%, 94.5%, and 91.9%, respectively. However, the results also highlight that Word2Vec embeddings, a classical word embedding method, exhibited superior performance in source code classification, surpassing transformer-based models in various evaluation metrics. This suggests that traditional word embeddings still hold significant value in software engineering applications, particularly when combined with ensemble-based classifiers. Among the ML models tested, the Random Forest classifier demonstrated the most robust results, achieving an accuracy of 98.1%, precision of 96.4%, recall of 98.1%, and an F1 score of 97.2%. These findings underscore the importance of selecting appropriate vectorization strategies and classification models in the context of static source code analysis.

One of the major contributions of this study is the demonstration that ML-driven service decomposition can effectively replace manual or heuristic-based methods, which are often time-consuming and reliant on expert intervention. Unlike traditional semantic or rule-based techniques, which require extensive external documentation such as UML diagrams, business process models, or architectural specifications, the proposed approach operates purely on static source code analysis. This independence from external artifacts enhances the applicability of the method across various domains and software ecosystems. Furthermore, the comparative analysis of transformer-based tokenizers and word embedding models provides valuable insights into their strengths and limitations in the context of software decomposition. While transformer-based models offer contextual richness, their computational overhead remains a concern, making Word2Vec a more lightweight and effective alternative in many scenarios.

Overall, this study demonstrates that ML techniques, particularly Word2Vec combined with ensemble classifiers, provide an efficient and accurate solution for automated domain-specific service decomposition in legacy systems. By eliminating the need for manual intervention and external documentation, this approach offers a scalable and high-performance alternative to traditional static and semantic analysis techniques. Future research will focus on decomposing larger monolithic projects into both service-oriented and microservice architectures, enhancing model generalizability, incorporating dynamic analysis, and evaluating real-world deployment feasibility to further improve service decomposition in large-scale legacy software systems.

References

1. Al-Debagy, O., P.J.S.C.P. Martinek, and Experience, *A microservice decomposition method through using distributed representation of source code.* 2021. **22**(1): p. 39-52.
2. Akl, M., *Exploring Software Architectural Transitions: From Monolithic Applications to Microfrontends enhanced by Webpack library and Cypress Testing*. 2024, Politecnico di Torino.
3. Balalaie, A., A. Heydarnoori, and P.J.I.S. Jamshidi, *Microservices architecture enables devops: Migration to a cloud-native architecture.* 2016. **33**(3): p. 42-52.
4. Razzaq, A. and S.A.J.C.A.i.E.E. Ghayyur, *A systematic mapping study: The new age of software architecture from monolithic to microservice architecture—awareness and challenges.* 2023. **31**(2): p. 421-451.
5. Cerny, T., M.J. Donahoo, and M.J.A.S.A.C.R. Trnka, *Contextual understanding of microservice architecture: current and future directions.* 2018. **17**(4): p. 29-45.
6. Stojanov, Z., et al., *A Tertiary Study on Microservices: Research Trends and Recommendations.* 2023. **49**(8): p. 796-821.
7. Verma, R., D.J.S.C.P. Rane, and I.f.R.-T.S.-O. Computing, *Service-Oriented Computing: Challenges, Benefits, and Emerging Trends.* 2024: p. 65-82.
8. Abdellatif, M., et al. *A type-sensitive service identification approach for legacy-to-SOA migration*. in *Service-Oriented Computing: 18th International Conference, ICSOC 2020, Dubai, United Arab Emirates, December 14–17, 2020, Proceedings 18*. 2020. Springer.
9. Khadka, R., et al. *A structured legacy to SOA migration process and its evaluation in practice*. in *2013 IEEE 7th International Symposium on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems*. 2013. IEEE.
10. Abdellatif, M., et al., *A taxonomy of service identification approaches for legacy software systems modernization.* 2021. **173**: p. 110868.
11. Abebe, S., H.J.I.J.o.P.M. Twinomurinzi, and Benchmarking, *Identifying business services from small and micro enterprises' collaboration: the activity-based service identification framework.* 2023. **15**(3): p. 373-399.
12. Ayalew, S.A.J.A.P., *Identifying reusable services from collaborative activities using activity theory (AT): The Activity-Based Service Identification Framework (ASIF).* 2023.
13. Si, H., et al. *A service-oriented analysis and modeling using use case approach*. in *2009 International Conference on Computational Intelligence and Software Engineering*. 2009. IEEE.
14. Yousef, R., et al., *Extracting SOA Candidate Software Services from an Organization's Object Oriented Models.* 2014. **2014**.
15. Zhao, Y., et al. *A service-oriented analysis and design approach based on data flow diagram*. in *2009 International Conference on Computational Intelligence and Software Engineering*. 2009. IEEE.
16. Rafsanjani, S. *JavaFX Point of Sales*. 2024; Available from: https://github.com/sadatrafsanjani/JavaFX-Point-of-Sales.

17. Brito, M., J. Cunha, and J. Saraiva. *Identification of microservices from monolithic applications through topic modelling*. in *Proceedings of the 36th annual ACM symposium on applied computing*. 2021.
18. Trabelsi, I., et al., *From legacy to microservices: A type-based approach for microservices identification using machine learning and semantic analysis.* 2023. **35**(10): p. e2503.
19. Sellami, K., et al., *Improving microservices extraction using evolutionary search.* Information and Software Technology, 2022. **151**: p. 106996.
20. Bastidas Fuertes, A., M. Pérez, and J.J.A.S. Meza, *Transpiler-based architecture design model for back-end layers in software development.* 2023. **13**(20): p. 11371.
21. Hu, M., et al., *Collaborative Deployment and Routing of Industrial Microservices in Smart Factories.* 2024.
22. Karabey Aksakallı, I., et al., *Micro-IDE: A tool platform for generating efficient deployment alternatives based on microservices.* 2022. **52**(7): p. 1756-1782.
23. Woods, D., *Enterprise services architecture*. 2003: " O'Reilly Media, Inc.".
24. Xiao, Z., I. Wijegunaratne, and X. Qiang. *Reflections on SOA and Microservices*. in *2016 4th International Conference on Enterprise Systems (ES)*. 2016. IEEE.
25. Feng, Z., et al., *Codebert: A pre-trained model for programming and natural languages.* 2020.
26. Behera, S.K. and R. Dash, *Fine-Tuning of a BERT-Based Uncased Model for Unbalanced Text Classification*, in *Advances in Intelligent Computing and Communication: Proceedings of ICAC 2021*. 2022, Springer. p. 377-384.
27. Liu, Y., et al., *RoBERTa: A robustly optimized BERT pretraining approach. arXiv [Preprint](2019).* 1907.
28. Abubakar, H.D., et al., *Sentiment classification: Review of text vectorization methods: Bag of words, Tf-Idf, Word2vec and Doc2vec.* 2022. **4**(1): p. 27-33.
29. Rong, X., *word2vec Parameter Learning Explained.* 2014.
30. Popescu, M.-C., et al., *Multilayer perceptron and neural networks.* WSEAS Transactions on Circuits and Systems, 2009. **8**(7): p. 579-588.
31. Navada, A., et al. *Overview of use of decision tree algorithms in machine learning*. in *2011 IEEE control and system graduate research colloquium*. 2011. IEEE.
32. Ampomah, E.K., et al., *Stock market prediction with gaussian naïve bayes machine learning algorithm.* Informatica, 2021. **45**(2).
33. Abbas, M., et al., *Multinomial Naive Bayes classification model for sentiment analysis.* IJCSNS Int. J. Comput. Sci. Netw. Secur, 2019. **19**(3): p. 62.
34. Zhang, Z., *Introduction to machine learning: k-nearest neighbors.* Annals of translational medicine, 2016. **4**(11).
35. Thulasidas, M. *Nearest centroid: A bridge between statistics and machine learning*. in *2020 IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE)*. 2020. IEEE.
36. Tian, Y., Y. Zhang, and H. Zhang, *Recent advances in stochastic gradient descent in deep learning.* Mathematics, 2023. **11**(3): p. 682.
37. Hearst, M.A., et al., *Support vector machines.* IEEE Intelligent Systems and their applications, 1998. **13**(4): p. 18-28.
38. Upadhyay, D., et al., *Gradient boosting feature selection with machine learning classifiers for intrusion detection on power grids.* IEEE Transactions on Network and Service Management, 2020. **18**(1): p. 1104-1116.
39. Liu, Y., Y. Wang, and J. Zhang. *New machine learning algorithm: Random forest*. in *Information Computing and Applications: Third International Conference, ICICA 2012, Chengde, China, September 14-16, 2012. Proceedings 3*. 2012. Springer.

40. Weidong, L., et al., *Implementation of AdaBoost and genetic algorithm machine learning models in prediction of adsorption capacity of nanocomposite materials.* Journal of Molecular Liquids, 2022. **350**: p. 118527.
41. Nusinovici, S., et al., *Logistic regression was as good as machine learning for predicting major chronic diseases.* Journal of clinical epidemiology, 2020. **122**: p. 56-69.
42. Coban, O., M. Yağanoğlu, and F. Bozkurt, *Domain Effect Investigation for Bert Models Fine-Tuned on Different Text Categorization Tasks.* Arabian Journal for Science and Engineering, 2024. **49**(3): p. 3685-3702.
43. Alawi, A.E.B., F. Bozkurt, and M. Yağanoğlu. *BERT-AraPeotry: BERT-based Arabic Poems Classification Model*. in *2024 4th International Conference on Emerging Smart Technologies and Applications (eSmarTA)*. 2024. IEEE.
44. Ba Alawi, A. and F. Bozkurt, *Performance Analysis of embedding methods for deep learning-based Turkish sentiment analysis models.* Arabian Journal for Science and Engineering, 2024: p. 1-23.