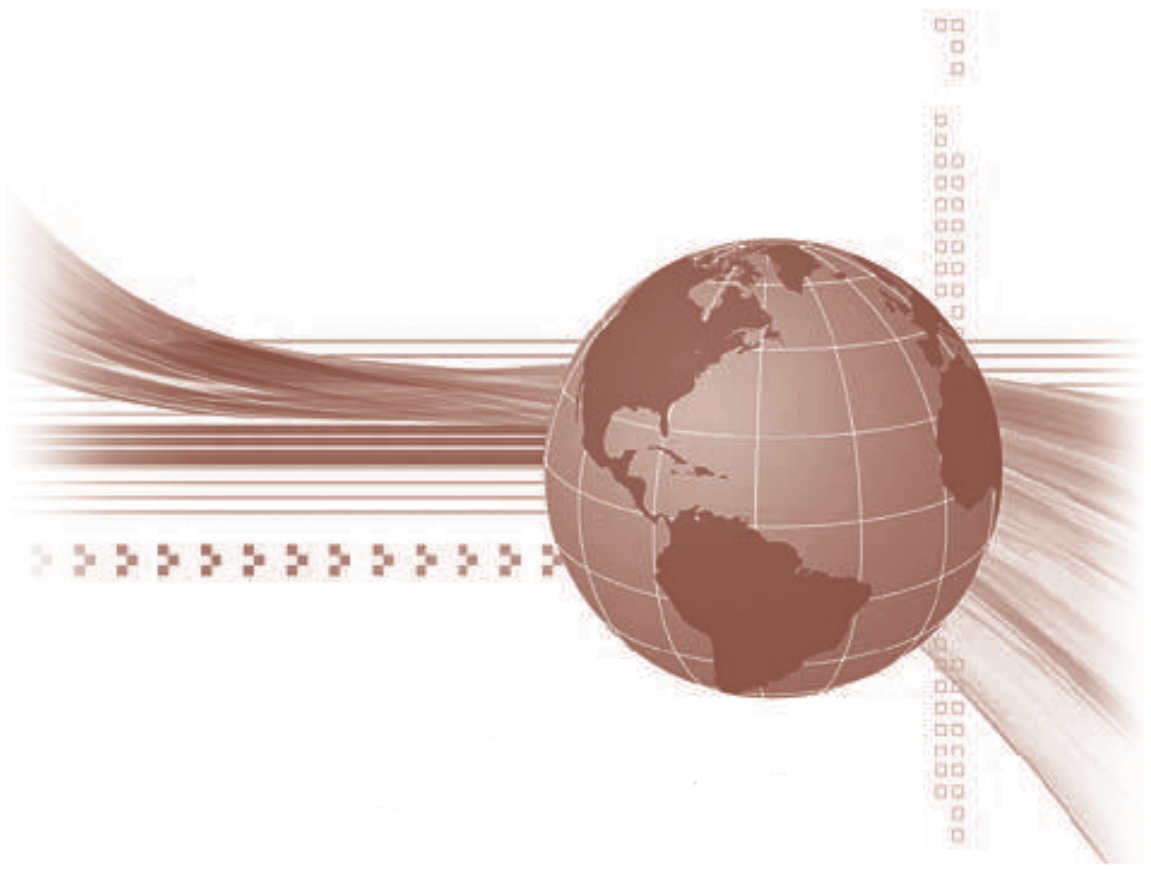
STUDIA UNIVERSITATIS
BABEŞ-BOLYAI

UNIVERSITATEA BABEŞ-BOLYAI
BABEŞ-BOLYAI TUDOMÁNYEGYETEM
BABEŞ-BOLYAI UNIVERSITÄT
BABEŞ-BOLYAI UNIVERSITY
TRADITIO ET EXCELLENTIA



INFORMATICA

1-2/2025

STUDIA

UNIVERSITATIS BABEŞ-BOLYAI
INFORMATICA

No. 1-2/2025

January - December

ISSN (online): 2065-9601; ISSN-L: 1224-869X

Published by Babeș-Bolyai University

EDITORIAL BOARD

YEAR **Volume 70 (LXX) 2025**
MONTH **DECEMBER**
ISSUE **1-2**

S T U D I A
UNIVERSITATIS BABEŞ-BOLYAI
INFORMATICA

1-2

EDITORIAL OFFICE: M. Kogălniceanu 1 • 400084 Cluj-Napoca • Tel: 0264.405300

SUMAR – CONTENTS – SOMMAIRE

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LXX**, Number 1-2, 2025
DOI: 10.24193/subbi.2025.01

NETWORK OPTIMIZATION PROBLEM APPLICABLE FOR BREAST CANCER SCREENING COST MINIMIZATION

ATTILA MESTER AND ANCA ANDREICA

Abstract. We investigate the problem of breast cancer screening optimization, using various techniques applicable in domains where the data format is not defined in advance. The aim is to minimize the cost related to the screening of patients while maximizing the beneficial effect of the process regarding some key breast cancer indices. Our model can be easily adjusted to other similar network optimization tasks where a goal function has to be minimized across a geographical surface. We present the problem's key similarities to the Travelling Salesman Problem and underline the fact why we choose a deterministic algorithm compared to a Simulated Annealing-based solution. Furthermore, we present the usefulness of the Elastic Stack regarding this application and offer a concrete solution to the problem defined by our generated dataset, respecting the European data distributions in this domain.

Introduction

To optimize the breast cancer screening process, it is necessary to analyze the trends of various domain indicators. These can be simple indices like incidence or mortality rate, or other ones such as participation rate at screening process / prevention campaigns, detection rate, malign/benign percentage, period between detection and first medical treatment, etc.

There are several works in the literature related to optimizing the breast cancer screening [7, 6, 1]. The advantage of the proposed method is that we offer a deterministic algorithm for finding the optimal number of mammography machines needed for a nationwide screening, we determine the exact location of the medical units based on the density of the population and the

Received by the editors: 1 March 2023.

2010 *Mathematics Subject Classification.* 91-04, 91-08.

1998 *CR Categories and Descriptors.* G.1.6 [**Optimization**]: Subtopic – *Simulated annealing* G.1.6 [**Optimization**]: Subtopic – *Constrained optimization*.

Key words and phrases. optimization, network, graph traversal, DBSCAN.

distribution of medical personnel, and offer a detailed technical description of the process, including where and when each patient needs to be examined, and how many days the national screening will last according to the algorithm.

In the first part of this work, we present existing literature for the analysis of indicator trends. The second part of the paper presents the proposed framework used to simulate a nationwide screening process and offers a deterministic solution regarding the optimal cost–screening time ratio.

1. Joinpoint regression analysis

Joinpoint regression analysis is in fact a linear regression, built up of multiple segments – the number of which is undefined and has to be determined by the algorithm. This regression approximates a set of points, in our case, on the format $(year, indexvalue)$. In the literature, this problem is known as joinpoint regression with k points. Formally, one can state the problem as follows. Let us note the points $(x_1, y_1), (x_2, y_2), .., (x_n, y_n)$, find the regression model function M, so that the least square error (LSSQ) of the model is minimized – as stated in Equation 1 [5].

$$M(x) = ax + b + s_1(x - t_1)^+ + .. + s_k(x - t_k)^+ \qquad (1)$$
$$LSSQ = \sum (y_i - M(x_i))^2,$$

where t represents the positions of the joining points, and $e^+ = e$ if $e > 0$ else 0.

The linear regression problem is a classic one, having an analytical but also an estimative solution, obtained by optimization methods. In our case, the problem has two components: finding the optimal number of joining points k, and determining the positions (x, y) of these points. These two issues define the directions and approaches in the literature. We will present in turn these two problems. Fig. 1 shows a data set (index), on which we will build the regression in several sequences. This dataset was used in [5], and contains incidence rates for prostate cancer in the US. between 1975 and 1995.

1.1. Dataset.

In order for the experiments to be as relevant as possible, our goal was to use real data from the studied field. The European Cancer Information System[1] offers a REST API, which exposes a huge amount of historical statistics. By sending 60 million requests to this server, we were able to populate a local Elasticsearch database. These requests are answered by a JSON

[1] `https://ecis.jrc.ec.europa.eu/`

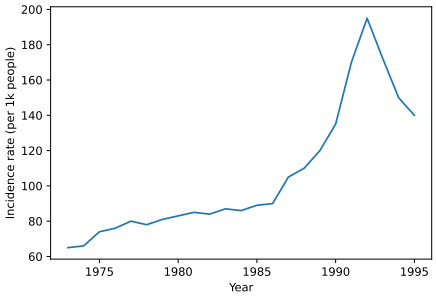


FIGURE 1. Incidence rate of US prostate cancer, 1975 – 1995.

dictionary in the following format: which region reported the indicator (city/-country), age range and sex of the people for whom the index belongs, year, and type of cancer.

The advantage of this system is that having the data locally, we can analyze the data set by applying custom visualizations, not only through some plots provided by the official site. Thus, we created a Kibana dashboard containing several views that provide us with valuable information on the trend for several types of cancer, age categories, year of onset, etc., shown on a timeline in which the index was recorded. This timeline gives us a kind of histogram of documents, showing how many records were reported in the selected period in Kibana.

1.2. **Scientific approach - Literature review.** It is easy to prove that the higher the k, the smaller the LSSQ error, so the more accurate the regression. However, taking into account the fact that in the field of medicine, we are looking for a more general trend, extended over a longer period of time, we are looking for k as small as possible - so that the regression error is still as small as possible. To determine the optimal number k, there are several variants in the literature. In a recent paper [3], Gkioulekas and Papageorgiou applied two aspects from information theory to penalize a more complex model, that is, to keep k as small as possible. These are the Akaike Information Criterion (AIC) and the Bayesian Information Criterion (BIC). Both assume that there is no perfect model, and therefore the optimal model must be found by measuring the relative distance between the model and the ground truth. By penalizing a high k value, we can find the model with as few joining points as possible. Another approach found in the literature is described in [5], which favors a small k by applying the hypothesis testing method. In the initial hypothesis *H0*, they assume that the optimal model uses k_0 points, and the alternative

hypothesis *H1* says k_1 points ($k_0 < k_1$). Thus, a higher number of joining points k_1 is accepted only if it results in a statistically relevant improvement to the model, applying permutation testing, as follows. To decide whether the initial hypothesis (using k_0) points can be rejected, we compare fitting models several times using k_0 and k_1 points. But, each time, on a dataset slightly altered from the initial one, introducing noise, by permuting the error vector from the initial model. There are cases in which the relative quality between the two models is at least as good as at the initial fitting, on the initial dataset. If this percentage number (p value) is high enough (above a 5% threshold), then we do not reject the initial hypothesis, we accept it because the changes assumed by H1 do not bring the relevant improvements. For a defined k, we must determine the position of the joining points. In their paper [5], Kim et al. mention an exhaustive search method called the grid-search method, meaning that any combination of positions for points in the range of x_{min} and x_{max} is considered. In practice, this method is too expensive regarding execution time and allows only positions with integer value for abscissa. A demonstration of this method is shown in Fig. 2. Here we can see how the grid-search method works in finding the optimal position for the $k = 2$ joining points. Each subplot represents a unique position for these two points, shown as their title. The blue lines represent the input data – US prostate cancer incidence rate, as shown in Fig. 1. The X and Y axis represent the year and the incidence rate, respectively. The red lines represent the output of the joinpoint regression algorithm using the respective two joining points.

The approach described in [3] is to test of each k, until the AIC or BIC indices improve (i.e. do not decrease). This method is built on the idea described in [8], called Optimal Piecewise Linear Regression Analysis (OPLRA), which can determine the positions of the joining points - the disadvantage is that k must be fixed by the user, and in [3] it is determined by the algorithm. To summarize the necessary steps, we will need the following functions.

i. A function that has as input the points over which we apply the regression (X, Y), and k. Having k, we can estimate the initial positions of the joining points by dividing the sequence $[x_{xmin}, x_{max}]$ into k equal parts. Then we use the simplex method and the LSSQ error function to determine each of the k points.

ii. A function that has as input k_0 and k_1, and determines the optimal model, by direct comparison, either using indices for model complexity as AIC, or hypothesis testing.

After applying the regression for each of the grid-search cases, we obtain the optimal model, having $k = 2$ joining points at the abscissas 1988 and 1992, demonstrated in Fig.3.

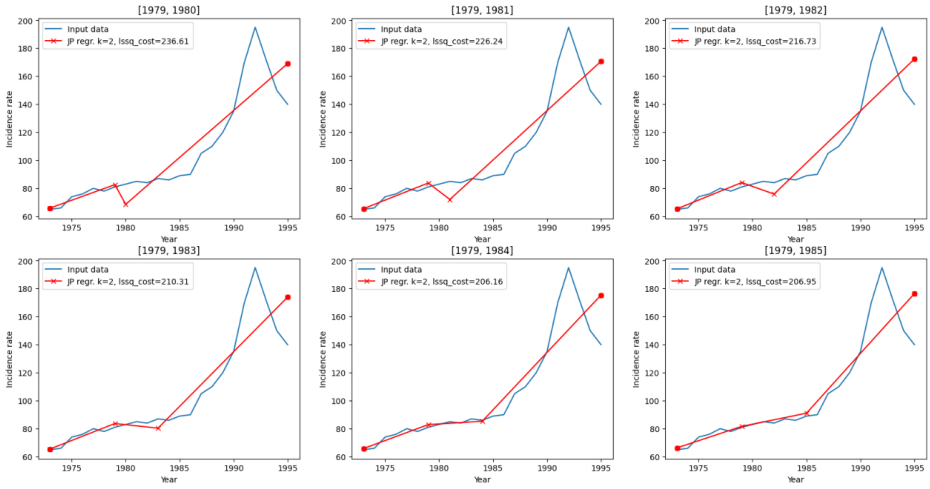


FIGURE 2. Extract from the grid-search method to determine the optimal position for the $k = 2$ joining points.

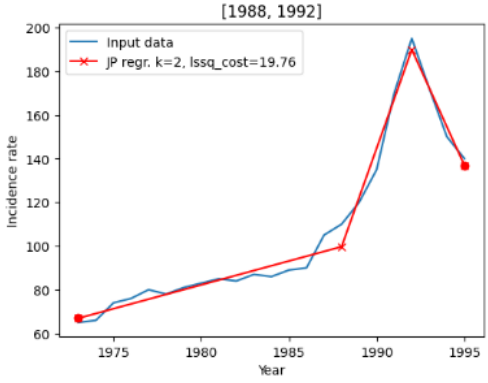


FIGURE 3. The optimal regression model with $k = 2$ joining points in the abscissas 1988 and 1992.

2. Optimizing a national screening process

In this section, we describe our methodology used for optimizing the process of breast cancer screening. Since the task lacks essential specifications, our focus was to determine the dataset, and the goal functions that we will optimize (e.g. minimize the logistical cost – the transport of equipment, maximize the amount of help the application offers to the population).

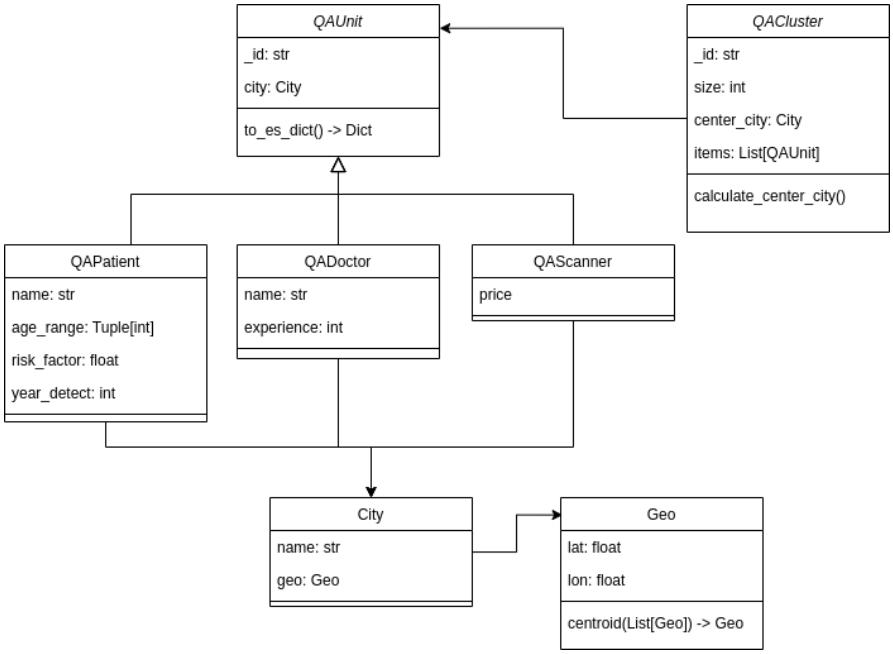


FIGURE 4. Entities of the generated dataset.

2.1. **Defining the data structure.** We need to generate a population with a given age, risk factor, and geographical distribution, generate an appropriate number of healthcare specialists, and also some mammography machines. This project aims the study of Romania's population, so the number of the generated healthcare people in the current experiment is influenced by national statistics. Similarly, the geographical position of doctors and patients is determined by national data. European statistics also take a crucial part in determining the age and risk factor distribution of the generated patients[2]. The generated data is stored in a local Elasticsearch[3] database in order to be able to query the data for subsequent analysis, and a live Kibana dashboard is also created, in order to visualize the nature of the generated data – and possible, to track the runtime of the future algorithm. The dashboard reflecting the current dataset is shown in Fig. 5. The data generated by us follows OOP principles, as shown in Fig. 4.

2.2. **Data storage.** We use the Elastic Stack in order to offer three separate containers. *Logstash* corresponds to an ETL process, used in the real-time

[2]`https://ecis.jrc.ec.europa.eu/`

[3]`https://www.elastic.co/elastic-stack/`

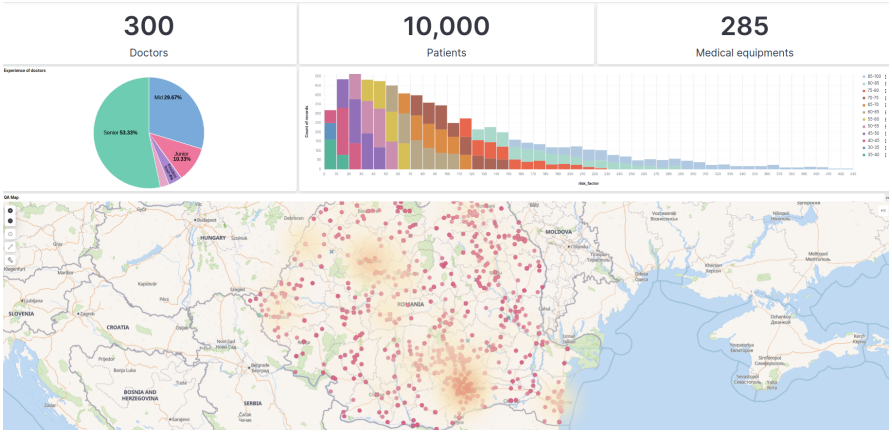


FIGURE 5. Kibana dashboard of the generated healthcare people, medical equipment and patients.

insertion of data. Our generated data will be stored in *Elasticsearch*, a NoSQL database, which will be visualized using *Kibana*. In our Logstash pipeline configuration, we use the *mutate:convert* filter plugin to extract the latitude and longitude values from the input file.

2.3. **Generating the data.** In order to develop a realistic algorithm, we needed to generate a population of entities representing patients with a certain age, risk factor, and geographic distribution, furthermore, also health professionals and mammography machines (scanners). The project is specific to the population of Romania, the data generation module respects the national statistics regarding the number of medical professionals but also their geographical position, as well as of the patients. European statistics determine the age and risk factor distribution of the generated patients. The dashboard created to visualize the generated data is shown in Fig. 5. The first row depicts the number of different entities: 300 doctors with different experiences in the field, 285 scanners, and 10,000 patients with age distribution and breast cancer risk factors according to European data. The second row shows a pie chart regarding the seniority level of the medical professionals: more than half of the personnel are generated with senior experience level, while there are 10% juniors i.e. fresh graduates – these percentages can also be modified within the data generation module. We can also see a distribution plot of the risk factor values separated by age groups, and a heatmap over the generated entities.

2.4. **Simulated annealing (SA).** The optimization of a screening process refers to minimizing the expenses related to the overall operation – possibly both governmental and private costs, and also the time required for the process, measured in days. This optimization resembles the classic Traveling Salesman Problem (TSP). In TSP, we have an agent that traverses a list of cities, each one exactly once, starting from a city and returning to the same location. The goal is to minimize the length of the road – the distance travelled by the agent. TSP is a classic NP-hard problem, which means that there is no polynomial algorithm that finds the optimal solution, and we have to apply different heuristics and optimization methods to find an optimal solution – a classic approach being evolutionary algorithms (GA), or SA. A comprehensive study on the TSP problem is offered in [2]. In this section we will present the similarities of our screening problem with that of TSP, we will present our technical approach, but also the reasons why this SA method could not be used in this project.

In the first step, we assume that we have a single doctor and a modular scanner that can be carried by this doctor. Then, the national screening process consists of the path travelled by this doctor, who has to reach every city and arrive back home, from where they started the traversal. The function to be minimized is the path travelled by the doctor — i.e. an optimal order of cities must be found. We now add the first element of complexity.
* We have multiple agents (doctors), and the union of their route must contain every city (not necessarily just once). The goal is to reduce the length of routes travelled by doctors. In order to simulate the screening as described in this project, we add one more element of complexity.
** The doctor does not necessarily have to visit the city, it is possible that the "the city visits the doctor", in the sense that the patients choose the nearest center of screening, and doctors have a fixed location – a medical center, or healthcare unit.

In classical TSP, a possible solution is to use an SA-based algorithm, which provides a random, initial order of cities, and through iterative changes applied to this list (e.g. mutation), we obtain an optimal order of the cities, according to a certain function, known as the fitness function, e.g. the length of the route. Given the fact that in this project we have the detail *, we cannot use SA with a single list of cities, respectively a single cost function. We need to use one list of cities for each doctor, and simultaneously apply SA on these lists. This approach is possible, but we still need the cost function applied on the combination of all lists, which evaluates if the existing configuration has errors such as cities visited by multiple doctors. This means that those SAs cannot be run independently, and any configuration of different SAs must be

```
1. day:
[ unit1-city1-patient1, unit2-city2-patient2,  u3-c3-p3 ...]
2. day:
[ unit5-city3-patient5, unit3-city1-patient8,  u2-c4-p9 ...]
```

FIGURE 6. Simulated annealing model: the aim is to reach an optimal order of scanning operations.

checked on each route (per doctor), which results in an unacceptable runtime, not applicable in practice (having 300 routes according to 300 doctors, and 537 cities). Besides these clear limitations, the information on required days is completely ignored by this model.

Given that we have one more complexity **, even if we manage to get a result with the method described above, the solution would not reflect a real-life screening process. We need to use another SA model. We can use another list of fixed length, which represents a day in the screening process, where one element represents a scanning operation, i.e. a union of the medical center, doctor, and patient – the goal is to get an optimal order, as shown in Fig. 2.4.

This model solves the problem of the previous model, we will also determine the required days. After SA is completed, we get the list of patients that will need to be scanned on the first day. After which, we repeat the process on the next day, etc. Unfortunately, this method is not practical either, for several reasons:

- runtime – with the final algorithm proposed in this report, the minimum required time is 180 days, which means 180 SA simulations according to this model
- the list mentioned in this model is extremely long, we have a Cartesian product between patients $(10,000)$, cities (537), and centers (see arg. 3)
- the number of centers is unknown. For this model, we need the number of medical centers, which should be determined by the algorithm.

In addition to all of the above, SA is non-deterministic, a property that would not be preferred in an application aimed to optimize a process that impacts the health of thousands of patients. The final algorithm proposed in this paper was formed upon analysis of all of the key points listed above, which we present in the following.

2.5. **DBSCAN clustering.** Optimizing the national screening process requires an algorithm to find the required number of mammography machines (scanners) and medical centers, but also their location. Applying an optimization method to obtain the scanners (their number and location), due to

reasons described in Section 2.4, would result in an unacceptable runtime, furthermore, no guarantee would be offered on the optimality of the solution. Considering that in practice, mammography machines will be placed in already existing medical centers, probably in larger cities, resulted in the basic concept of the proposed solution: fixing the number of scanners. If we assume that we have n scanners, we can determine the cities in which they should be placed – according to the geographical distribution of doctors and patients. For this step, we need to obtain this distribution, i.e. to cluster the doctors, respectively the patients, according to their location - eventually, we obtain a map with *QACluster* documents (Fig. 4) which we can validate as a debugging method by comparing with the heatmap provided by Kibana, depicted in Fig. 5. DBSCAN is an unsupervised learning method, based on the concept of spatial density (according to a distance function), according to which it groups the elements into tight clusters. The advantage over K-means is that the number of clusters does not need to be specified beforehand, which is very useful in this application regarding spatial data [4]. The algorithm works with the following important concepts:

- Eps – parameter that defines the radius of a neighborhood
- minPoints – the minimum number of nodes that must be in the vicinity of the Eps radius of a node n for n to be a base node in the cluster.

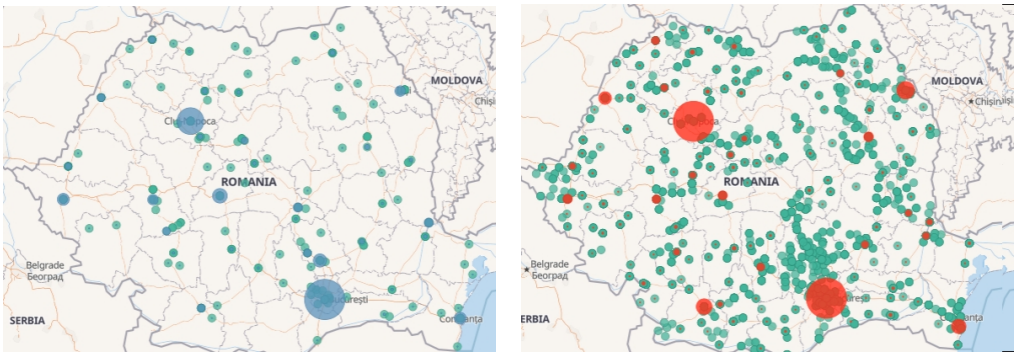
The benefits of using DBSCAN clustering in a TSP problem are detailed in a recent work [1], where the authors argue that this method facilitates a more efficient computation time of the simulation.

For the application of DBSCAN clustering, we use the *sklearn* library[4].

The result of this clustering applied on $10,000$ patients and 300 doctors is shown in Fig. 5. Having obtained these clusters, we can continue the description of the proposed algorithm by distributing the scanners in the largest clusters.

2.6. **Deterministic grid-search algorithm.** The essence of the proposed algorithm consists of trying each n for the number of scanners, and simulating the screening process having n scanners. First, we distribute the machines in the largest n cities in each possible way, evaluating the cost related to this processes. Then, we select the optimal number n for which the cost functions are optimal, according to certain hybrid criteria. Although the proposed method is an exhaustive search algorithm, the runtime is absolutely acceptable, running several thousand simulations in under 5 minutes, and it has the

[4]`https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html`



Doctors Patients

FIGURE 7. DBSCAN clustering on the generated dataset of 10,000 patients and 300 doctors.

advantage that the obtained result is deterministic – offering a guarantee that for a certain set of generated data (doctors, patients), the optimal simulation chosen by the algorithm is constant. This is a clear advantage over an SA-type method or other metaheuristics.

2.7. **Simulation with n mammographic machines.** The proposed algorithm is based on the assumption that if we determine the examination centers, then the screening process is well defined, we have no other aspects to optimize – the centers determine the patients' decision, and each patient will choose the one that is closest. In other words, all we need to optimize through this proposed algorithm is the choice of the number of centers, their location, and the distribution of n scanner machines in these centers, so that the cost related to this model is minimal.

The algorithm starts with distributing the scanners in examination centers (Section 2.8). For each distribution, the screening process is simulated: until we still have patients to examine in the queue, we start a new day, we select a batch of patients who can be scanned based on the number of machines, we determine the nearest city for each patient, and we send them to be examined, removing that patient from the queue. In this process, we take into account the maximum capacity of a center, but also the number of possible scans with a single machine, per day – detailed in Section 2.9. If a center if filled according to its maximum capacity, the model is penalized due to the associated cost of an overcrowded medical center. If – for some reason – the patient cannot be scanned on the current day (e.g. overcrowded center), then we postpone them to the next day. These are decisive factors to select the optimal model, because if a medical center in a very dense area of patients does not have

enough scanners, then the number of days needed for screening increases, precisely because patients are continuously rescheduled. The algorithm can determine the optimal number of machines to avoid this, also considering the initial investment cost of the machines.

2.8. **Distribution of scanner machines in centers – backtracking.** The simulation function receives as input the number of scanners, the list of doctors and patients, but also the list of clusters of doctors and patients, determined by DBSCAN. The basic concept is that the scanners will be distributed in the largest cities, thus, for n machines, we must distribute them to the largest n clusters of doctors. A hybrid selection might be applied that also takes into account patient clusters (Section 3), for the time being it was assumed that the natural distribution of doctors and patients is not significantly different. Since we want to provide a deterministic solution, we will try each combination of the distribution of the scanner machines in the largest cities, with one constraint: a smaller city cannot have more scanners than a larger city. Algorithmically, the problem translates to finding all non-increasing sequences of fixed length, having the sum equal to a number n. For this, we apply the backtracking method. An example for the distribution of cars in cities, having $n = 4$ cars is depicted in Fig. 8.

2.9. **Fitness functions & Constants.** Fitness functions, in a general metaheuristic context, measure how close a single unit of solution is to the desired achievements according to some fixed criteria. Once we have several simulations, we need to sort them according to certain criteria (i.e. the output of our fitness function) so that we can choose the top k of them. For a certain number of n scanners, we will select the top 3 runs, according to a hybrid criterion, and for all simulations, we select the top 5 – it is important to mention this aspect because although the algorithm selects the top k solutions, also marked on the final plot, the user may choose any other run, regardless of the algorithm's decision. The proposed algorithm optimizes two aspects: the cost related to national screening, and the days needed to examine all patients. These two aspects are not comparable, their importance relies on some

```
{bucuresti: 4, clujnapoca: 0, sibiu: 0, ploiesti: 0}
{bucuresti: 3, clujnapoca: 1, sibiu: 0, ploiesti: 0}
{bucuresti: 2, clujnapoca: 2, sibiu: 0, ploiesti: 0}
{bucuresti: 2, clujnapoca: 1, sibiu: 1, ploiesti: 0}
{bucuresti: 1, clujnapoca: 1, sibiu: 1, ploiesti: 1}
```

FIGURE 8. Distribution of $n = 4$ mammography machines in the largest n clusters of doctors.

outer factors which we add by weight constants to express one single score of weighted sum of these measures. All constants used in these cost functions are shown in Fig. 9. We present the cost functions in the following.

(1) Governmental cost. Naturally, if we have 537 scanners, we can create an examination center in each city, so no patient will have to travel to another city, and the screening will be extremely fast. However, this way the government cost increases a lot because many scanners have to be bought and more centers are needed to be prepared. The algorithm increases the government cost with each scanner purchased, and each test center has a daily maintenance cost (set to a small value, because centers exist in larger cities anyway), so a model with multiple scanners and centers is penalized.
(2) Private cost (Patient cost). That is, the cost paid by patients. Optimally, a patient should not travel a lot for a screening - therefore, the algorithm selects the nearest medical center, and penalizes the model with the cost of travel. If the distance is long enough that the patient will need accommodation, then the model is also penalized with an accommodation cost.
(3) Total cost. The sum of the above two. Ideally, both costs should be minimized.
(4) Days required. A national screening should not take years – as is the result of running with 3 scanners (560 days). Thus, the model is rewarded for offering a solution which finishes the screening of all patients in as few days as possible.
(5) Hybrid criterion. Each of the mentioned aspects must be taken into account, thus, for the selection of the best runs, we take into account the necessary days, and the total cost (with 70% weight on the government cost).

3. Improvement ideas

The distribution of mammography machines is based on the location of doctors, assuming that a dense region of doctors probably means a dense

```
PRICE_PER_KM = 1 # RON
PRICE_PER_OVERNIGHT_TRAVEL = 300
PRICE_PER_SCANNER = 5000
PRICE_PER_CENTER_PER_DAY = 1000
PRICE_PER_CENTER = 1000
PRICE_PER_FULL_CENTER = 2000
```

Figure 9. Constants used in the screening simulation.

region of patients as well. The algorithm is easily extensible with another weighting parameter that would mean taking into account the geographical distribution of patients as well.

Machines are placed into the largest clusters of doctors (i.e. medical centers), based on a backtracking algorithm, presented in Section 2.8. Unfortunately, this solution grows exponentially, for $n = 10$ scanners, we have 42 possibilities of placing them in 10 cities, and for $n = 20$ scanners, we have 627 possibilities, each resulting in a new simulation. An optimization method could be applied that considers only those configurations that have a chance that during the simulation, the model will have a minimum cost.

At the moment, a series of metadata-level information is not used by the proposed algorithm, precisely to simplify the problem and generalize the proposed solution. Information such as the price of a scanner, the age and risk category of the patient, as well as the experience of doctors, are available at the level of data generation and their storage in Elasticsearch (searchable in Kibana), but at the level of the optimization algorithm, they are not used. These constants can be adjusted according to global standards [6]. The scanner has a fixed price and patients are queued in a randomized order, not taking into consideration their risk factor – which is a must in an ideal screening programme [7]. The algorithm is easily extensible to take this information into account as well.

4. Results and discussion – validating the health economics tool

In this section, we present different simulations for the dataset shown in Fig. 5. The algorithm started with $n = 1$ mammography machines and stopped at 20 because the cost function no longer improved. On the generated plots shown in Fig. 10, 11, 12, 13, one dot represents one screening simulation, the x and y axes will show the days needed for screening, respectively the total cost, and the size of the points represents the percentage of the government cost in the total cost – it will be important at higher n values. Furthermore, the header contains all the technical details related to the result of the particular simulation. We can observe that these images also show the three best runs according to our hybrid criteria: taking into account both the cost and the days required to run the screening.

We analyze $n = 2$ scanners in Fig. 10. The selected centers will be in Bucharest and Cluj-Napoca, and the most optimal run regarding the required days is when we have both machines placed in Bucharest - the density of patients is higher there.

We analyze $n = 5$ scanners in Fig. 11. We can see that the algorithm selects the most optimal simulations with 2, 3 and 4 medical centers, marked with

X. The cost associated with them is interesting: the total cost decreases as there are more examination centers due to the patient cost (lower travel costs). From the size of the points we can see that the percentage of government cost in the total cost has increased from 2 centers to 4 centers, which means that in total, the final cost has become lower.

We analyze $n = 14$ scanners in Fig.12. We can see that the total price of the screening process has decreased, although we now have more scanners and centers to maintain, which means an increased percentage of government cost in the total cost -– reflected by the point size, compared to previous simulations. According to the government cost criterion, the best model has only 2 centers, and according to the private cost we have 14 centers. The selected optimal model has 12 medical centers, with 2 machines each in Bucharest and Cluj-Napoca, and 1 in every other center.

We analyze all of the simulations in Fig. 13. We have nearly 2000 simulations, for scanners between $n = 1$ and $n = 20$, the execution time being only a few minutes (3 minutes). We can see that as we add mammography machines, the government cost increases 0000and the private cost decreases. But since more centers and scanners result in fewer rescheduled patients and overcrowded centers (both penalized by the model), the government cost does not increase as much as the private cost decreases. This optimization trend is observed up to $n = 19$, where it is not worth adding more scanners – the selected optimal models have 15, 16, 17, 18, and 19 mammography machines, respectively, in $14 - 15$ centers, and they require ≈ 180 days for a complete screening of the patients.

5. Conclusions and future work

This paper presents a novel framework for optimizing a nationwide breast cancer screening process. The key aspects which are offered by this algorithm are the deterministic nature of the solution it provides, and the precise description of the simulation details, offering guidelines for each patient on which day and which examination center they should visit in order for the screening to be carried out in the shortest timespan while minimizing governmental and private expenses. Future work includes and is not limited to implementation details specified in Section 3.

6. Acknowledgments

This work was supported by a grant of the Romanian Ministry of Education and Research, CCCDI - UEFISCDI, project number PN-III-P2-2.1-PED-2019-2607, within PNCDI III.

```
        == Simulating 2 scanners, in 2 ways across the cities
====== Best configurations: ======
  1. BALANCED #0     :: 2 centers |  756 days | all cost  2611269.1 ;
  gov. cost   163200.0 (6.2%) ; civil cost  2448069.1
  1. BALANCED #1     :: 1 centers |  625 days | all cost  3919550.0 ;
  gov. cost    73500.0 (1.9%) ; civil cost  3846050.0
  2. GOV. COST       :: 1 centers |  625 days | all cost  3919550.0 ;
  gov. cost    73500.0 (1.9%) ; civil cost  3846050.0
  3. CIVIL COST      :: 2 centers |  756 days | all cost  2611269.1 ;
  gov. cost   163200.0 (6.2%) ; civil cost  2448069.1
  4. ALL COST        :: 2 centers |  756 days | all cost  2611269.1 ;
  gov. cost   163200.0 (6.2%) ; civil cost  2448069.1
  5. FEWEST DAYS     :: 1 centers |  625 days | all cost  3919550.0 ;
  gov. cost    73500.0 (1.9%) ; civil cost  3846050.0
1.0 {bucuresti: 1, clujnapoca: 1}
1.1 {bucuresti: 2}
2. {bucuresti: 2}
3. {bucuresti: 1, clujnapoca: 1}
4. {bucuresti: 1, clujnapoca: 1}
5. {bucuresti: 2}
```

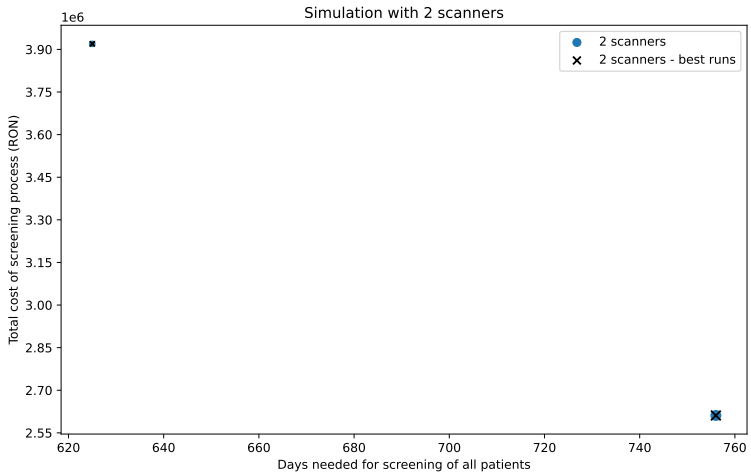


FIGURE 10. Simulation with $n = 2$ scanner machines: 2 centers, 625 days needed.

```
    == Simulating 5 scanners, in 7 ways across the cities
====== Best configurations: ======
  1. BALANCED #0     :: 4 centers |  360 days | all cost  2210048.4 ;
  gov. cost   173000.0 (7.8%) ; civil cost  2037048.4
  1. BALANCED #1     :: 3 centers |  355 days | all cost  2457990.6 ;
  gov. cost   134500.0 (5.5%) ; civil cost  2323490.6
  1. BALANCED #2     :: 2 centers |  314 days | all cost  2887869.1 ;
  gov. cost   439800.0 (15.2%) ; civil cost  2448069.1
  2. GOV. COST       :: 3 centers |  355 days | all cost  2457990.6 ;
  gov. cost   134500.0 (5.5%) ; civil cost  2323490.6
  3. CIVIL COST      :: 5 centers |  581 days | all cost  2124036.2 ;
  gov. cost   320500.0 (15.1%) ; civil cost  1803536.2
  4. ALL COST        :: 5 centers |  581 days | all cost  2124036.2 ;
  gov. cost   320500.0 (15.1%) ; civil cost  1803536.2
  5. FEWEST DAYS     :: 2 centers |  314 days | all cost  2887869.1 ;
  gov. cost   439800.0 (15.2%) ; civil cost  2448069.1
1.0 {bucuresti: 2, clujnapoca: 1, oradea: 1, sibiu: 1}
1.1 {bucuresti: 2, clujnapoca: 2, oradea: 1}
1.2 {bucuresti: 3, clujnapoca: 2}
2. {bucuresti: 2, clujnapoca: 2, oradea: 1}
3. {bucuresti: 1, clujnapoca: 1, oradea: 1, sibiu: 1, craiova: 1}
4. {bucuresti: 1, clujnapoca: 1, oradea: 1, sibiu: 1, craiova: 1}
5. {bucuresti: 3, clujnapoca: 2}
```

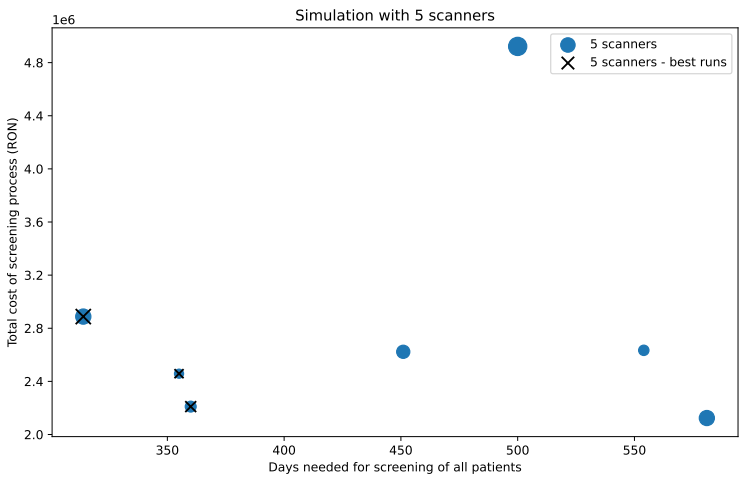


FIGURE 11. Simulation with $n = 5$ scanner machines: $2 - 4$ centers, $314 - 360$ days needed.

```
    == Simulating 14 scanners, in 135 ways across the cities
====== Best configurations: ======
  1. BALANCED #0      :: 12 centers |  207 days | all cost   935075.8 ;
  gov. cost   330400.0 (35.3%) ; civil cost   604675.8
  1. BALANCED #1      :: 12 centers |  211 days | all cost   985875.8 ;
  gov. cost   381200.0 (38.7%) ; civil cost   604675.8
  1. BALANCED #2      :: 10 centers |  206 days | all cost  1012609.7 ;
  gov. cost   286000.0 (28.2%) ; civil cost   726609.7
  2. GOV. COST        :: 2 centers |  554 days | all cost  2678869.1 ;
  gov. cost   230800.0 (8.6%) ; civil cost  2448069.1
  3. CIVIL COST       :: 14 centers |  332 days | all cost  1087803.5 ;
  gov. cost   548800.0 (50.5%) ; civil cost   539003.5
  4. ALL COST         :: 12 centers |  207 days | all cost   935075.8 ;
  gov. cost   330400.0 (35.3%) ; civil cost   604675.8
  5. FEWEST DAYS      :: 6 centers |  192 days | all cost  2095934.3 ;
  gov. cost   567200.0 (27.1%) ; civil cost  1528734.3
1.0 {bucuresti: 2, clujnapoca: 2, oradea: 1, sibiu: 1, craiova: 1, constanta: 1,
albaiulia: 1, iasi: 1, ploiesti: 1, arad: 1, calan: 1, baiamare: 1}
1.1 {bucuresti: 3, clujnapoca: 1, oradea: 1, sibiu: 1, craiova: 1, constanta: 1,
albaiulia: 1, iasi: 1, ploiesti: 1, arad: 1, calan: 1, baiamare: 1}
1.2 {bucuresti: 2, clujnapoca: 2, oradea: 2, sibiu: 2, craiova: 1, constanta: 1,
albaiulia: 1, iasi: 1, ploiesti: 1, arad: 1}
2. {bucuresti: 13, clujnapoca: 1}
3. {bucuresti: 1, clujnapoca: 1, oradea: 1, sibiu: 1, craiova: 1, constanta: 1,
albaiulia: 1, iasi: 1, ploiesti: 1, arad: 1, calan: 1, baiamare: 1, timisoara: 1,
brasov: 1}
4. {bucuresti: 2, clujnapoca: 2, oradea: 1, sibiu: 1, craiova: 1, constanta: 1,
albaiulia: 1, iasi: 1, ploiesti: 1, arad: 1, calan: 1, baiamare: 1}
5. {bucuresti: 3, clujnapoca: 3, oradea: 2, sibiu: 2, craiova: 2, constanta: 2}
```

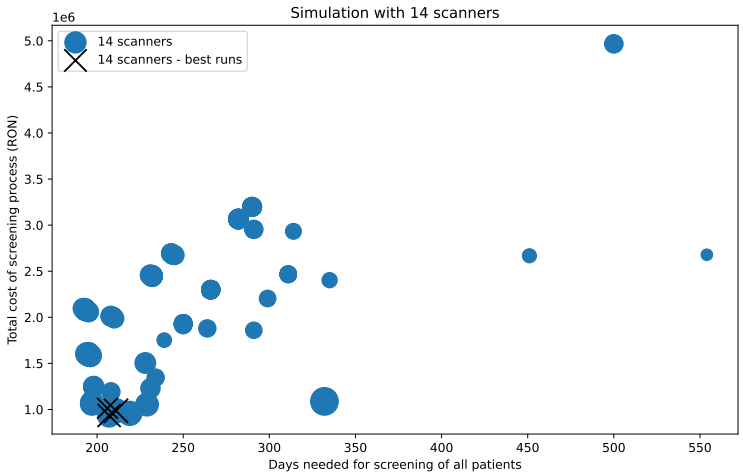


FIGURE 12. Simulation with $n = 14$ scanner machines: 12 centers, 207 days needed.

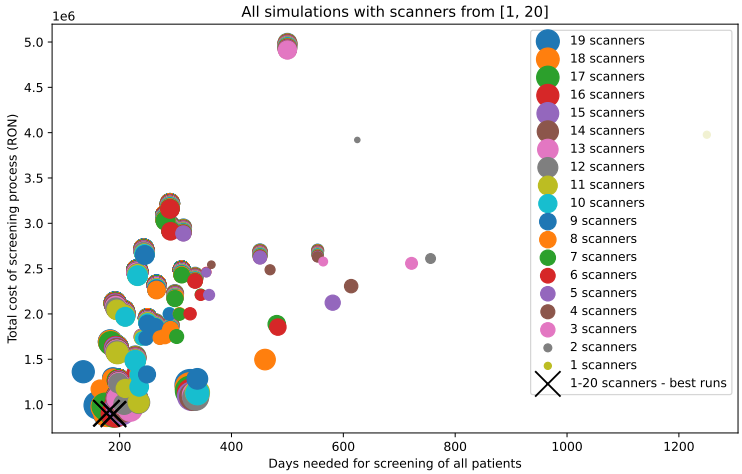


FIGURE 13. Simulation with $n = 1-20$ scanner machines; best result yield by $16 - 18$ machines, ≈ 180 days needed.

References

[1] Matthew Agostinelli. Density-based clustering heuristics for the traveling salesman problem. 2017.

[2] Omar Cheikhrouhou and Ines Khoufi. A comprehensive survey on the multiple traveling salesman problem: Applications, approaches and taxonomy. *Computer Science Review*, 40:100369, 2021.

[3] Ioannis Gkioulekas and Lazaros G Papageorgiou. Piecewise regression analysis through information criteria using mathematical programming. *Expert Systems with Applications*, 121:362–372, 2019.

[4] Kamran Khan, Saif Ur Rehman, Kamran Aziz, Simon Fong, and Sababady Sarasvady. Dbscan: Past, present and future. In *The fifth international conference on the applications of digital information and web technologies (ICADIWT 2014)*, pages 232–238. IEEE, 2014.

[5] Hyune-Ju Kim, Michael P Fay, Eric J Feuer, and Douglas N Midthune. Permutation tests for joinpoint regression with applications to cancer rates. *Statistics in medicine*, 19(3):335–351, 2000.

[6] Wenhui Ren, Mingyang Chen, Youlin Qiao, and Fanghui Zhao. Global guidelines for breast cancer screening: a systematic review. *The Breast*, 64:85–99, 2022.

[7] Adam Yala, Peter G Mikhael, Constance Lehman, Gigin Lin, Fredrik Strand, Yung-Liang Wan, Kevin Hughes, Siddharth Satuluru, Thomas Kim, Imon Banerjee, et al. Optimizing risk-based breast cancer screening policies with reinforcement learning. *Nature medicine*, 28(1):136–143, 2022.

[8] Lingjian Yang, Songsong Liu, Sophia Tsoka, and Lazaros G Papageorgiou. Mathematical programming for piecewise linear regression analysis. *Expert systems with applications*, 44:156–167, 2016.

Faculty of Mathematics and Computer Science, Babeş-Bolyai University, Cluj-Napoca

Email address: {attila.mester, anca.andreica}@ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LXX**, Number 1-2, 2025
DOI: 10.24193/subbi.2025.02

OLDIES BUT GOLDIES: THE POTENTIAL OF CHARACTER N-GRAMS FOR ROMANIAN TEXTS

DANA LUPŞA, SANDA-MARIA AVRAM, AND RADU LUPŞA

Abstract. This study addresses the problem of authorship attribution for Romanian texts using the ROST corpus, a standard benchmark in the field. We systematically evaluate six machine learning techniques — Support Vector Machine (SVM), Logistic Regression (LR), k-Nearest Neighbors (k-NN), Decision Trees (DT), Random Forests (RF), and Artificial Neural Networks (ANN), employing character n-gram features for classification. Among these, the ANN model achieved the highest performance, including perfect classification in four out of fifteen runs when using 5-gram features. These results demonstrate that lightweight, interpretable character n-gram approaches can deliver state-of-the-art accuracy for Romanian authorship attribution, rivaling more complex methods. Our findings highlight the potential of simple stylometric features in resource-constrained or under-studied language settings.

1. Introduction

The authorship determination of a text requires distinguishing the author particularities that are reflected in the written text. Instruments used to do automated authorship attribution (AA) can be characterized along several dimensions, as detailed in the following. *Dataset characteristics* include language, dataset size, class imbalance, and text type (e.g., emails, novels, social media posts, or cross-domain/genre scenarios) [22]. *Preprocessing steps* include text normalization and punctuation handling [17]. *Feature extraction* encompasses character-based features (e.g., character types - letters, digits; character n-grams), lexical features (e.g., word frequencies, word n-grams, stop words, function words), syntactic features (e.g., parts of speech, sentence and phrase

Received by the editors: 25 June 2025.
2010 *Mathematics Subject Classification.* 68T50.
1998 *CR Categories and Descriptors.* I.2.7 [**Artificial Intelligence**]: Natural Language Processing – *Text Analysis*; I.2.6 [**Artificial Intelligence**]: Learning – *Induction*.
Key words and phrases. Authorship Attribution, Machine Learning, Character n-gram.

structure), and semantic features (e.g., topic modeling, word embeddings, contextual representations from large language models) [10, 14, 22, 24]. *Computational methods* rang from traditional statistical techniques (e.g., stylometry, PCA) to machine learning algorithms (e.g., Support Vector Machines, Random Forests), deep learning models (e.g., Recurrent Neural Networks, transformers), and modern large language model (LLM) approaches [12]. *Evaluation metrics* include accuracy, macro-averaged accuracy (especially for imbalanced datasets), precision, recall, F1-score, area under the curve, and confusion matrices [22].

English remains the most studied language in AA research due to the availability of data and the focus on computational linguistics [22, 10]. However, research on Romanian texts has seen growing interest in recent years, both by efforts to support under-resourced languages and by practical applications such as historical and literary analysis, plagiarism detection, and cybercrime investigations [5, 3, 2, 15].

The size of the dataset significantly influences the choice of methodology. Deep learning and LLM-based approaches typically require large datasets for effective training, validation, and testing. In contrast, classical machine learning methods are often more suitable for small to mid-scale datasets (e.g., 10,000–20,000 texts, tens of authors) [11].

Key challenges associated with AA datasets include [23, 14]:

- limited text availability (e.g., few samples per author),
- class imbalance (e.g, in representation of authors or genres),
- generalizability issues (e.g., on cross-domain or cross-genre texts).

The aforementioned challenges are especially pronounced for under-resourced languages like Romanian [3, 2, 15].

Dataset partitioning into training, testing, and validation subsets also affects performance outcomes, as different splits can yield varying results due to random sampling effects. Thus, researchers commonly employ multiple splits or cross-validation techniques to obtain more reliable performance estimates.

While preprocessing transforms raw text into a cleaner format, the feature representation stage, where texts are converted to numerical vectors, is critical to classification success. The choice of features can have a profound impact on the results, often independent of the classification algorithm used, making feature engineering a central focus of AA research.

Different classification algorithms may yield varying results on the same data and features. Consequently, studies often compare multiple algorithms or maintain algorithm consistency when evaluating features. Furthermore, optimizing algorithm hyperparameters can significantly enhance performance.

In this paper, we investigate the effectiveness of character n-gram features to improve authorship attribution performance on the Romanian ROST dataset.

The remainder of this paper is organized as follows. Section 2 reviews related work on authorship attribution, with a focus on approaches relevant to Romanian texts and the ROST dataset. Section 3 presents the experimental setup and results, including dataset description, model configurations, and detailed analyses of classification performance across different feature and parameter settings. Finally, Section 4 concludes the paper with a summary of key findings and outlines directions for future research.

2. Related Work

2.1. **Challenges in Romanian Authorship Attribution.** Authorship attribution (AA) in under-resourced languages, such as Romanian, presents a distinct set of challenges that set it apart from work in high-resource languages. One of the primary obstacles is the limited availability of annotated corpora, which restricts both the scale and diversity of training data available for model development and evaluation. This scarcity is compounded by a significant class imbalance, as datasets such as the ROST [3] corpus exhibit uneven representation of authors, ranging from as few as 27 to as many as 60 texts per author, and substantial variation in text length, from short stories of 90 words to extensive works exceeding 39,000 words. The ROST dataset also encompasses a wide range of genres, including stories, fairy tales, novels, articles, and sketches, and spans a broad historical period from 1850 to 2023, further increasing heterogeneity and complicating model generalization.

These factors introduce biases and variability that are difficult to control, making it challenging to develop robust and generalizable AA models for Romanian [3, 2]. Addressing such issues is essential for advancing the field, as models trained on unbalanced or limited data may fail to perform reliably across different authors, genres, or time periods.

Recent research has begun to address these challenges by developing hybrid models that combine handcrafted linguistic features with contextualized embeddings, tailored specifically for Romanian and other under-resourced languages. However, the linguistic complexity of Romanian, characterized by rich morphology and flexible word order, means that effective solutions for English or other major languages often do not transfer directly [15]. Similar difficulties have been reported in other under-resourced languages, such as Albanian, where the lack of large annotated corpora continues to impede progress in authorship attribution research [13].

Moreover, the scarcity of large, publicly available Romanian corpora limits the applicability of data-intensive deep learning methods, making careful

feature engineering and rigorous evaluation protocols especially important for achieving reliable results in this context.

2.2. **N-gram Features in Authorship Attribution.** N-grams are contiguous sequences of N items, such as characters, words, or other tokens, extracted from text to capture patterns indicating an author's unique style. Typically, n-grams are constructed at the character, word, or syntactic level [24].

Character n-grams have been extensively used in authorship attribution research due to their ability to encode stylistic nuances, lexical patterns, word order tendencies, and punctuation or capitalization habits [22]. Despite their widespread use in languages such as English, Romanian texts remain relatively understudied using this approach [22, 3].

2.2.1. *Advantages of Character N-grams.*

- **Robustness:** Character n-grams are resilient to infrequent errors such as grammatical mistakes or punctuation slips because their discriminative power derives from frequent, recurring patterns [23].
- **Preservation of Stylistic Traits:** Subtle author-specific variations, such as repeated punctuation marks or distinctive capitalization, are naturally encoded within character n-grams [12]. For instance, some authors rarely use exclamation marks, while others employ them frequently. Similarly, preferences for sentence length can be reflected through punctuation usage patterns, such as the relative frequency of periods versus commas, which correspond to shorter or longer sentence structures, respectively [9].
- **Language Independence:** Character n-grams do not require deep linguistic knowledge of grammar or semantics, making them applicable across diverse languages without modification.
- **Computational Efficiency:** Extracting character n-grams is both straightforward and computationally inexpensive, as it involves direct processing of raw text without the need for complex preprocessing.

2.2.2. *Limitations of Character N-grams.*

- **Redundancy:** Due to the overlapping nature of n-grams, each n-gram shares $N-1$ characters with adjacent ones. Therefore, many n-grams represent slight variations of the same lexical unit (e.g., `"in_"`, `"in."`, `"in!"`). While this redundancy can reinforce stylistic signals such as affixation (prefix/suffix) or punctuation preferences, it may also lead to overfitting if not properly managed.
- **Sparsity and High Dimensionality:** Compared to word-level n-grams, character n-grams, especially for larger N, tend to generate

very high-dimensional and sparse feature spaces, which can pose challenges for model training and generalization [23, 20].

2.3. **Computational Methods.** A wide range of classification algorithms has been applied to authorship attribution (AA) leveraging character n-gram features [4, 6, 7, 1, 18, 26]. Next, we provide a concise overview of the principal methods utilized in this study:

Support Vector Machine (SVM) is a supervised learning model that identifies an optimal hyperplane to separate author classes by maximizing the margin between data points of different classes. It excels in handling high-dimensional feature spaces such as those generated by n-grams and is robust against overfitting and noise, making it a popular choice in stylometric analysis.

Logistic Regression (LR) is a probabilistic linear model used for binary and multiclass classification. It maps input features to class probabilities via the logistic (sigmoid) function and applies regularization (L1 or L2 penalties) to prevent overfitting. LR provides interpretable coefficients, which can be valuable for understanding the contribution of specific n-grams, particularly in smaller datasets.

k-Nearest Neighbors (k-NN) is a non-parametric, instance-based classifier that assigns a class label based on the majority vote of the k nearest neighbors in the feature space. It relies on distance metrics such as Euclidean, cosine, or Minkowski distance. While simple to implement and intuitive, k-NN can be computationally expensive for large datasets and high-dimensional n-gram features.

Decision Trees (DT) recursively partition the feature space by selecting feature thresholds that maximize class purity, typically using criteria like entropy or Gini impurity. The resulting tree structure yields interpretable decision rules that can highlight stylometric patterns, such as frequent character sequences. However, DTs are prone to overfitting, especially with high-dimensional n-gram data.

Random Forest (RF) is an ensemble method that constructs multiple decision trees using random subsets of the data and features, aggregating their predictions via majority voting. This approach reduces overfitting through bagging and feature randomness. RF handles noisy, high-dimensional n-gram data effectively and provides measures of feature importance, enabling identification of discriminative n-grams.

Artificial Neural Networks (ANN) consist of interconnected layers of neurons that learn hierarchical feature representations through backpropagation. Activation functions such as ReLU enable the modeling of complex, non-linear relationships among features. While deep ANNs typically require

large datasets, shallow architectures (1–2 hidden layers) are well-suited for moderate-sized corpora like ROST [3].

Among the aforementioned methods, SVM and RF have traditionally dominated AA research [23] due to their strong performance with stylometric features such as n-grams. ANNs, particularly deep learning models, are gaining traction but generally demand larger datasets. Logistic Regression and k-NN often serve as baseline benchmarks, while standalone Decision Trees are less commonly employed due to their susceptibility to overfitting.

2.4. **Evaluation Metrics.** Authorship attribution studies commonly employ the following evaluation metrics to assess model performance:

2.4.1. *Accuracy.* Accuracy measures the proportion of correctly classified texts across all classes. While widely used, accuracy can be misleading for imbalanced datasets, as it may overemphasize performance on majority classes.

2.4.2. *Macro-Accuracy (Balanced Accuracy).* Macro-accuracy addresses class imbalance by calculating accuracy for each class independently and then averaging the results:

$$\text{Macro-Accuracy} = \frac{1}{C}\sum_{i=1}^{C}\text{Accuracy}_{Class_i}$$

where C is the number of classes (authors). This metric ensures all authors contribute equally to the final score, making it more representative for datasets with uneven class distributions.

2.4.3. *Precision, Recall, and F1-Score.* These metrics provide complementary insights into model performance, particularly for imbalanced data:

- **Precision** The proportion of correctly attributed texts for an author out of all texts predicted as that author. High precision minimizes false attributions.
- **Recall** The proportion of correctly attributed texts for an author out of all texts actually written by that author. High recall indicates strong retrieval of true positives.
- **F1-Score** The harmonic mean of precision and recall. It reflects how well a system is at finding relevant items (precision) and all of them (recall), without being skewed by abundant irrelevant items. Particularly important for imbalanced datasets, where, if one class significantly outnumbers the others, a model can achieve high accuracy by simply predicting the majority class.

2.4.4. *Confusion Matrix.* A table showing the number of correct and incorrect predictions for each class. Reveals which authors are frequently confused. It summarizes the true positives, true negatives, false positives, and false negatives, from which accuracy is derived.

3. Experiments and results

3.1. **The dataset.** Motivated by the relative scarcity of research on Romanian authorship attribution (AA) and the challenges posed by the ROST corpus, our work aims to advance the field by exploring lightweight yet effective feature representations. The ROST dataset, comprising approximately 400 texts authored by 10 writers, is characterized by significant class imbalance and diverse text lengths and genres, making it a challenging benchmark for AA [3, 2].

Previous studies have established important baselines on ROST: Avram et al. (2022) introduced the dataset and evaluated five classification methods, including Artificial Neural Networks (ANN), Multi Expression Programming (MEP), k-Nearest Neighbors (k-NN), Support Vector Machines (SVM), and C5.0 Decision Trees, using inflexible part-of-speech tags as features. Their best macro-accuracy was 80.94%, with an overall error rate of 20.40% [3]. Avram (2023) leveraged a Romanian-specific BERT model, achieving approximately 87% macro-accuracy [2]. Nitu et al. (2024) proposed a hybrid Transformer architecture combining handcrafted linguistic features (lexical, syntactic, semantic, discourse markers) with BERT embeddings, reaching a state-of-the-art F1-score of 0.95 and reducing the error rate to 4% [15].

While transformer-based approaches demonstrate impressive performance, their reliance on large pretrained models and substantial computational resources limits their accessibility, especially in resource-constrained environments. In contrast, our study investigates character n-grams, a lightweight, interpretable feature set, to assess whether they can achieve comparable performance on the ROST dataset. By systematically evaluating n-gram sizes ranging from 2 to 5, we aim to demonstrate that simpler, computationally efficient methods can provide competitive accuracy while offering greater interpretability and ease of deployment.

3.2. **Data Preprocessing.** In the raw text preprocessing phase, we apply *normalization* to standardize the textual data. Therefore, texts were initially preprocessed to address specific character encoding variations like:

- **Diacritic standardization:** converting Romanian characters such as Ş/ş and Ţ/ţ to their canonical forms Ș/ș and Ț/ț;
- **Punctuation unification:**
  - convert smart quotes („ “ ” ’) to straight quotes (")
  - convert en/em dashes (——) to hyphens (-)

  - convert ellipses (. . .) to triple periods (. . .)
- **Whitespace regularization:** collapse multiple contiguous whitespace into a single space

These steps ensure consistency and reduce noise in the dataset, facilitating more reliable downstream analysis.

In addition to normalization, we performed the following preprocessing steps:

- **Case Handling:** We conducted experiments using both lowercase text and the original case, to assess the impact of letter casing on authorship attribution.
- **Digit Replacement:** Since the specific values of the numbers were not relevant to our analysis, all digits were replaced with a special character (@), which does not occur in the original texts.
- **Punctuation Preservation:** All other special characters (punctuation marks) were left unchanged, as patterns in punctuation usage may reflect individual authorial style.
- **Whitespace Encoding:** Recognizing the potential importance of whitespace as a stylistic marker, we replaced spaces and tabs with the underscore character (_), and newlines with the dollar sign ($). This allows us to explicitly encode paragraph boundaries and whitespace usage into the N-grams as distinct features.

3.3. **Feature Extraction Process.** We employ character N-grams to construct stylometric representations of texts, following these steps:

3.3.1. *N-gram Definition and Range.* Drawing on prior stylometric studies (Houvardas et al., 2006; Ramezani et al., 2013; Ntoulas et al., 2006; Smith & Jones, 2022) [8, 19, 21, 24], we analyze character sequences of length $N = 2$ to $N = 5$. This range captures both local orthographic patterns (e.g., bigrams) and longer morphological features (e.g., pentagrams).

3.3.2. *Vectorization via TF-IDF.* Each document is transformed into a numerical vector using Term Frequency-Inverse Document Frequency (TF-IDF) weighting. This approach emphasizes N-grams that are statistically distinctive to individual documents while down-weighting common sequences.

3.3.3. *Comprehensive Feature Inclusion.* To avoid potential information loss from premature feature selection, we retain all extracted character N-grams during initial modeling. This ensures maximal preservation of potential stylistic markers.

3.3.4. *Feature Matrix Construction.* The final representation is a $D \times F$ matrix, where:

- D = Number of documents
- F = Total unique N-grams across all N values
- Cell values = TF-IDF weights for N-gram/document pairs

3.4. **Experimental Setup and Classification Methods.**

3.4.1. *Implementation Details.* All experiments were conducted in Python, utilizing the following libraries and tools:

- **Data Processing and Manipulation:** `numpy` and `pandas` for numerical operations and data handling.
- **Feature Extraction:** using `TfidfVectorizer` from `scikit-learn` to generate character N-gram feature representations.
- **Model Training and Evaluation:** `scikit-learn` for implementing and evaluating machine learning classifiers.

3.4.2. *Classification Algorithms.* We explored several supervised learning algorithms: Support Vector Machine (SVM), Logistic Regression (LR), k-Nearest Neighbor (k-NN), Decision Trees (DT), Random Forest (RF), and Artificial Neural Networks (ANN).

3.4.3. *Model Evaluation.* We use standard metrics including *accuracy*, *macro-accuracy* (aka. balanced accuracy or macro-averaged accuracy), and the *classification report* (precision, recall, F1-score), as provided by `scikit-learn`.

3.5. **Experimental Procedure.** For each of the classification methods described above, we conducted a set of experiments to identify the configuration yielding the best performance. Our investigation included the following key aspects:

- **Parameter Exploration:** For k-NN, we evaluated five different values of k to determine the optimal neighborhood size.
- **Robustness to Randomness:** To mitigate the effects of stochastic variability inherent in certain models, we repeated the training and evaluation of DT, RF, and ANN five times, each with a different random seed. This procedure allowed us to assess the stability and reliability of the results.
- **Evaluation Metrics:** Performance was assessed using multiple metrics, including accuracy and balanced accuracy, to provide a comprehensive understanding of classifier effectiveness across potentially imbalanced classes.

- **Data Splitting:** We employed randomly selected train-test splits to preserve class distribution during evaluation, ensuring fair and representative performance estimates.

This rigorous experimental protocol ensures that observed performance differences reflect genuine model capabilities rather than artifacts of data sampling or initialization.

ML Model	Hyperparameters	Variation Parameters
SVM	kernel: linear	—
LR	solver: lbfgs, penalty: l2	—
k-NN	metric: minkowski	n_neighbors = $\{3, 5, 7, 9, 11\}$
DT	criterion: gini	randomstate = $\{7, 17, 42, 67, 101\}$
RF	n_estimators: 100 criterion: gini	randomstate = $\{7, 17, 42, 67, 101\}$
ANN	activation: relu hidden layer sizes: (100,50)	randomstate = $\{7, 17, 42, 67, 101\}$

TABLE 1. Summary of machine learning models and their hyperparameters. To ensure that performance differences reflect model characteristics rather than random variation, DT, RF, ANN were each trained and evaluated five times using different random seeds. For k-Nearest Neighbors (k-NN), five different values of the number of neighbors were tested.

3.6. **Parameter Tuning and Model Selection.** The dataset was partitioned into training (80%) and testing (20%) subsets. This splitting procedure was repeated five times to generate distinct train-test partitions, ensuring robustness and reliability of the evaluation. For each partition, all six machine learning models (SVM, LR, k-NN, DT, RF, ANN) were trained and evaluated using the hyperparameter configurations detailed in Table 1.

Furthermore, to assess the impact of text casing on model performance, experiments were conducted on both the original case and fully lowercased versions of the texts.

3.7. **Results Overview and Key Findings.**

3.7.1. *Impact of Letter Casing on Classification Performance.* We conducted experiments comparing classification performance on original-case versus fully lowercased texts. The observed differences in Macro-Accuracy (MAcc) were generally minor. The largest difference, 0.031, occurred with the k-Nearest Neighbors (k-NN) classifier at n-gram size $N = 3$, where the lowercase representation slightly outperformed the original case.

Figure 1 depicts the variation in MAcc differences across all classifiers and n-gram sizes. Overall, lowercase texts tend to yield marginally better results,

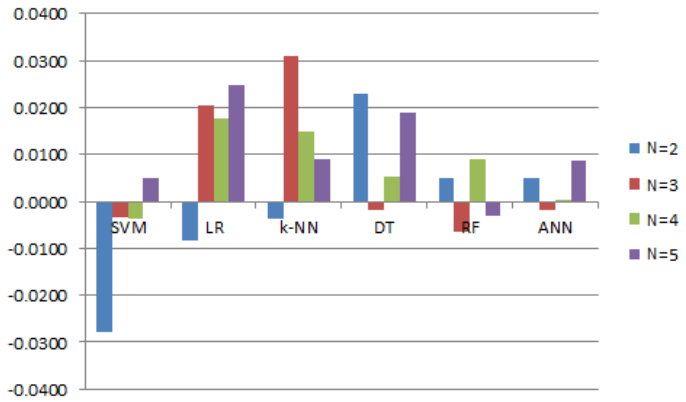


FIGURE 1. Differences in Macro-Accuracy (MAcc) averages between fully lower-cased and original-case texts for each N-gram size (N = 2 to 5) and for each of the six classification methods: SVM, LR, k-NN, DT, RF, and ANN. The values used to calculate the differences presented in the chart, were computed as follows: for each N-gram size, each classification method and each (letter) casing, several runs were executed (as described in Table 1 and in section 3.6) and the results were averaged.

particularly for $k > 3$ and for the k-NN and LR classifiers. However, in most cases, the difference remains below 0.02, which is negligible and likely falls within the range of normal variation due to hyperparameter tuning.

For ANN, the average difference between casing strategies is even smaller, approximately 0.009. Interestingly, at n-gram size $N = 5$, the trend reverses slightly, with original-case texts performing better, as further illustrated in Figure 4.

These findings suggest that letter casing has no consistent or substantial impact on classification performance across the evaluated methods. The minor differences observed are unlikely to affect practical outcomes or model selection decisions.

3.7.2. *The Impact of N-gram Size on Classification Performance.* Figure 2 presents the variation in of Macro-Accuracy (MAcc) with respect to the N-gram size parameter k (ranging from 2 to 5) for each of the six classification methods: SVM, LR, k-NN, DT, RF, and ANN.

It is important to note that in Figures 2e and 2f, the y-axis scale is adjusted to focus on the observed variations in MAcc, rather than spanning the full range from 0 to 1. This scaling facilitates a clearer visualization of performance trends as k changes.

A noticeable decline in MAcc is observed for RF (Figure 2e) as the N-gram size increases. Conversely, ANN shows a general upward trend in MAcc with larger N-gram dimensions. For SVM, the highest MAcc is achieved at $k = 4$, while k-NN attains its peak performance at $k = 2$. The remaining methods do not exhibit a clear monotonic trend in MAcc relative to the N-gram size.

3.7.3. *Best Results.* ANN achieved the best overall performance, with average accuracy and macro-accuracy (MAcc) values exceeding 0.9, as summarized in Table 2. Tables 3 and 4 present the average accuracy and MAcc for each model across N-gram sizes ($N = 2$ to 5), reported separately for original-case and lowercase texts.

Model	**Macro-Accuracy (avg)**	**Accuracy (avg)**
SVM	0.761	0.762
LR	0.701	0.725
k-NN	0.608	0.590
DT	0.632	0.631
RF	0.874	0.884
ANN	0.934	0.935

TABLE 2. The average of Macro-Accuracy and Accuracy scores obtained for all six classification methods (SVM, LR, k-NN, DT, RF, ANN).

ANN consistently outperformed other models, except at $k = 2$, where RF achieved superior results. This trend is illustrated graphically in Figure 3, which visualizes the MAcc values from the aforementioned tables.

For ANN, the highest performance was observed at $k = 4$ using original-case letters, and at $k = 5$ using lowercase letters—with lowercase slightly outperforming original case by 0.003 in MAcc. Conversely, RF achieved the best results at $k = 2$, with average macro-accuracy values of 0.914 (original case) and 0.919 (lowercase). Examining RF results further (Figures 2e) we observe a slight decrease in average accuracy as k increases, although MAcc values across different k are not clearly separated.

Notably, ANN achieved perfect classification (accuracy = 1.0) in one of the test runs. This raises the question of whether such perfect accuracy reflects a consistent pattern for that particular train-test split, or if it is an isolated occurrence. To address this, we conducted an extended series of 15 experiments using different random seeds:

$\{7, 17, 29, 31, 37, 41, 42, 43, 47, 53, 59, 67, 83, 101, 137\}$

for both original-case and lowercase texts, for that specific split . As shown in Figure 4, perfect accuracy was reached in four cases: twice for $random\_state = 42$ (in both casing conditions) and two additional times for original-case texts.

Model		**2-gram**	**3-gram**	**4-gram**	**5-gram**
SVM	MAcc avg	0.328	0.869	0.934	0.899
	Acc avg	0.309	0.874	0.933	0.911
LR	MAcc avg	0.502	0.791	0.795	0.744
	Acc avg	0.531	0.807	0.812	0.770
k-NN	MAcc avg	0.772	0.603	0.526	0.559
	Acc avg	0.765	0.589	0.502	0.535
DT	MAcc avg	0.635	0.575	0.666	0.676
	Acc avg	0.633	0.581	0.667	0.675
RF	MAcc avg	0.919	0.883	0.857	0.838
	Acc avg	0.916	0.893	0.869	0.857
ANN	MAcc avg	0.894	0.945	0.951	0.954
	Acc avg	0.899	0.944	0.951	0.952

TABLE 3. The Macro-Accuracy (MAcc) and Accuracy (Acc) scores obtained for different N-gram sizes ($N = 2$ to 5), using fully lowercased text, for all six classification methods (SVM, LR, k-NN, DT, RF, ANN). Values shown represent the average of the macro-accuracy scores computed over multiple experimental runs.

Model		**2-gram**	**3-gram**	**4-gram**	**5-gram**
SVM	MAcc avg	0.356	0.872	0.938	0.894
	Acc avg	0.348	0.877	0.938	0.909
LR	MAcc avg	0.510	0.771	0.777	0.719
	Acc avg	0.541	0.793	0.798	0.748
k-NN	MAcc avg	0.755	0.572	0.511	0.550
	Acc avg	0.768	0.550	0.483	0.526
DT	MAcc avg	0.611	0.577	0.660	0.657
	Acc avg	0.603	0.575	0.662	0.655
RF	MAcc avg	0.914	0.890	0.848	0.841
	Acc avg	0.912	0.898	0.863	0.863
ANN	MAcc avg	0.889	0.947	0.951	0.945
	Acc avg	0.895	0.943	0.950	0.948

TABLE 4. The Macro-Accuracy (MAcc) and Accuracy (Acc) scores obtained for different N-gram sizes ($N = 2$ to 5), using original case text, for all six classification methods (SVM, LR, k-NN, DT, RF, ANN). Values shown represent the average of the macro-accuracy scores computed over multiple experimental runs.

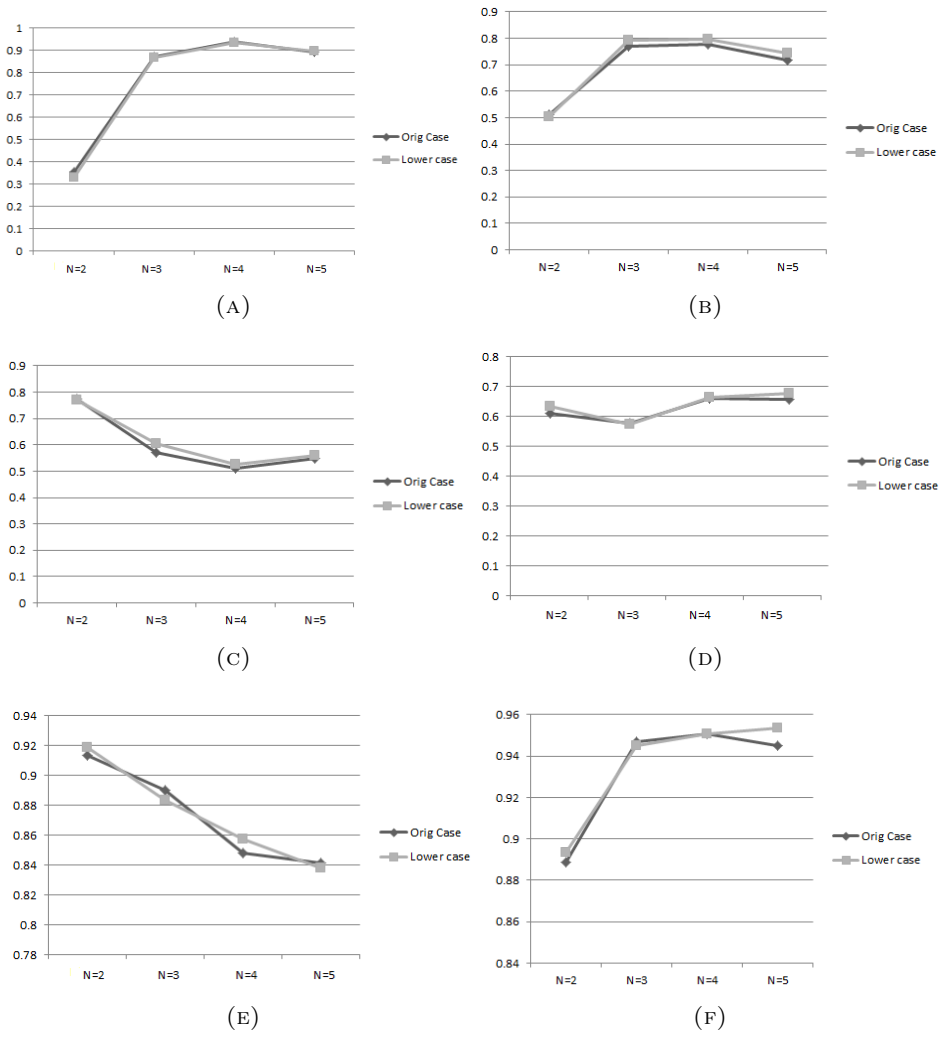
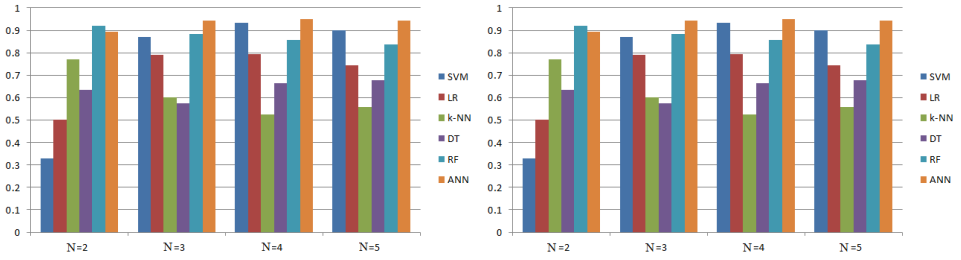


FIGURE 2. Variation of MAcc across different N-gram sizes ($N = 2$ to 5) for all six classification methods: (A) SVM, (B) LR, (C) k-NN, (D) DT, (E) RF, and (F) ANN. This analysis examines how varying the N-gram size affects the models' performance. Values shown represent the average of the macro-accuracy scores computed over multiple experimental runs.

The standard deviation of MAcc across these runs was approximately 0.02 (0.02039 for lowercase, 0.01938 for original case, and 0.02040 overall). No clear pattern emerged relating accuracy variation to the random seed. However,



(A) (B)

FIGURE 3. Comparison of Macro-Accuracy (MAcc) scores for N-gram with $N = 2$ to 5, for original case text (A), and fully lowercased text (B) and for all six classification methods (SVM, LR, k-NN, DT, RF, ANN). Values shown represent the average of the macro-accuracy scores computed over multiple experimental runs.

original-case texts tended to perform slightly better on average, with a 0.01 higher MAcc (0.974 vs. 0.962 for lowercase), reversing the trend observed in Tables 3 and 4. Given the small magnitude of these differences, and the reversed trend in the second set of experiments for ANN, we conclude that casing (original vs. lowercase) does not have a significant impact on classification performance.

3.8. **Comparison with Existing Results.** In the literature, the best reported accuracies for authorship attribution often exceed 95%. For example, Posadas et al. [16] combined word N-grams with the Doc2Vec method, achieving over 98% accuracy on some test sets. Similarly, Zhang et al. [25] explored character-level convolutional neural networks (CNNs) and reported a minimal error rate of 1.31% when using N-grams.

Reported accuracies vary widely depending on the dataset, feature selection, and classification methods, typically ranging from approximately 70% upwards.

Authorship attribution for Romanian is still relatively underexplored, although recent interest has been increasing. Notably, Niţu et al. [15] reported an F1-score of 0.87 on a 19-author dataset, improving to 0.95 on ROST, the same dataset used in our study. Also on ROST, Avram et al. [3] achieved a lowest overall error rate of 20.40% across several machine learning models, while Avram et al. [2] reported macro-accuracy up to 87% using a Romanian pretrained BERT model.

Our approach, employing an Artificial Neural Network (ANN), achieved an initial average macro-accuracy of 0.935 across five data splits, with one split

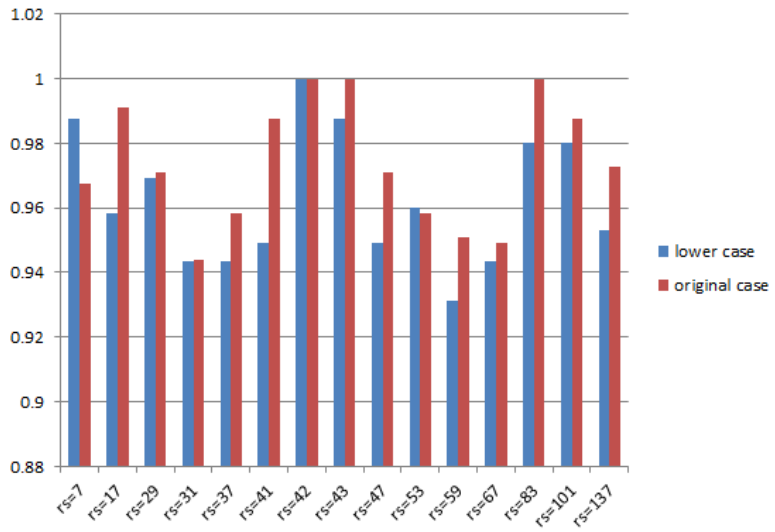


FIGURE 4. The Macro-Accuracy (MAcc) scores corresponding to each of the 15 experimental runs of the Artificial Neural Network (ANN) and for the one split that produced the highest classification accuracy in the previous experiments. **rs** denotes the *random_state* value used to initialize the ANN and shows the 15 distinct parameter settings.

yielding perfect classification (macro-accuracy = 1.0). To evaluate the consistency of this result, we extended the analysis to 15 additional runs with randomized seeds (Section 3.7.3). For original-case texts, the expanded trials demonstrated a mean macro-accuracy of 0.974 (with a standard deviation $\sigma \approx 0.02$), aligning with the 0.979 baseline from initial testing on the same split. Perfect classification occurred in four runs: once under lowercase transformation and three times with original casing. Notably, random seed 42 produced perfect classification under both text conditions.

These results demonstrate that our lightweight, character N-gram based approach with ANN matches or surpasses existing benchmarks for Romanian authorship attribution, offering a competitive alternative to more complex models such as BERT or transformer-based approaches.

4. Conclusions and Future Work

In this paper, we addressed the authorship attribution problem using a Romanian dataset previously employed in [2] and [3]. To our knowledge, this is the first study to apply character N-gram based methods for Romanian authorship attribution.

We evaluated six machine learning techniques: Support Vector Machine (SVM), Logistic Regression (LR), k-Nearest Neighbor (k-NN), Decision Trees (DT), Random Forest (RF), and Artificial Neural Networks (ANN). Among these, the ANN model achieved the best performance, including perfect classification in four out of thirty runs for 5-gram features.

For future work, we plan to extend our research in the direction of identifying significant character-level stylistic features. We also plan to extend our investigation by incorporating additional feature types such as word N-grams, part-of-speech tags, and other linguistic markers. These enhancements aim to further improve attribution accuracy. Given the good evaluation scores reported in some papers over the collection of texts we used, we also plan to develop a more challenging benchmark for Romanian and with a higher real-life relevance.

References

[1] Altman, N. S. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician 46*, 3 (1992), 175–185.

[2] Avram, S.-M. Bert-based authorship attribution on the romanian dataset called rost. *arXiv preprint arXiv:2301.12500* (2023).

[3] Avram, S.-M., and Oltean, M. A comparison of several ai techniques for authorship attribution on romanian texts. *Mathematics 10*, 23 (2022), 4589.

[4] Boser, B. E., Guyon, I. M., and Vapnik, V. N. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory* (1992), pp. 144–152.

[5] Dinu, L. P., Popescu, M., and Dinu, A. Authorship identification of romanian texts with controversial paternity. In *LREC* (2008).

[6] Fix, E., and Hodges, J. J. Discriminatory analysis: Non-parametric discrimination: Consistency properties. Tech. rep., USAF School of Aviation Medicine, 1951.

[7] Fix, E., and Hodges, J. J. Discriminatory analysis: Non-parametric discrimination: Small sample performance. Tech. rep., USAF School of Aviation Medicine, 1952.

[8] Houvardas, J., and Stamatatos, E. N-gram feature selection for authorship identification. In *Artificial Intelligence: Methodology, Systems, Applications* (2006).

[9] Howedi, F., and Mohd, M. Text classification for authorship attribution using naive bayes classifier with limited training data. *computer engineering and intelligent systems 5*, 4 (2014), 48–56.

[10] Kestemont, M. Function words in authorship attribution. from black magic to theory? In *Proceedings of the 3rd Workshop on Computational Linguistics for Literature (CLFL)* (2014), pp. 59–66.

[11] Kestemont, M., Tschuggnall, M., Stamatatos, E., Daelemans, W., Specht, G., Stein, B., and Potthast, M. Overview of the author identification task at pan-2018: cross-domain authorship attribution and style change detection. In *Working Notes Papers of the CLEF 2018 Evaluation Labs. Avignon, France, September 10-14, 2018/Cappellato, Linda [edit.]; et al.* (2018), pp. 1–25.

[12] Koppel, M., Schler, J., and Argamon, S. Computational methods in authorship attribution. *Journal of the American Society for information Science and Technology 60*, 1 (2009), 9–26.

[13] Misini, A., Canhasi, E., Kadriu, A., and Fetahi, E. Automatic authorship attribution in albanian texts. *Plos one 19*, 10 (2024), e0310057.

[14] Neal, T., Sundararajan, K., Fatima, A., Yan, Y., Xiang, Y., and Woodard, D. Surveying stylometry techniques and applications. *ACM Computing Surveys (CSuR) 50*, 6 (2017), 1–36.

[15] Nitu, M., and Dascalu, M. Authorship attribution in less-resourced languages: A hybrid transformer approach for romanian. *Applied Sciences 14*, 7 (2024), 2700.

[16] Posadas Durán, J., Gomez Adorno, H., Sidorov, G., Batyrshin, I., Pinto, D., and Chanona-Hernández, L. Application of the distributed document representation in the authorship attribution task for small corpora. *Soft Computing 21* (02 2017).

[17] Potthast, M., Barrón-Cedeno, A., Stein, B., and Rosso, P. Cross-language plagiarism detection. *Language Resources and Evaluation 45* (2011), 45–62.

[18] Quinlan, J. R. Induction of decision trees. *Machine learning 1*, 1 (1986), 81–106.

[19] Ramezani, R., Sheydaei, N., and Kahani, M. Evaluating the effects of textual features on authorship attribution accuracy. In *ICCKE 2013* (2013), pp. 108–113.

[20] Sapkota, U., Bethard, S., Montes, M., and Solorio, T. Not all character n-grams are created equal: A study in authorship attribution. In *Proceedings of the 2015 conference of the North American chapter of the association for computational linguistics: Human language technologies* (2015), pp. 93–102.

[21] Sari, Y., Vlachos, A., and Stevenson, M. Continuous n-gram representations for authorship attribution. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers* (Valencia, Spain, Apr. 2017), M. Lapata, P. Blunsom, and A. Koller, Eds., Association for Computational Linguistics, pp. 267–273.

[22] Stamatatos, E. A survey of modern authorship attribution methods. *Journal of the American Society for information Science and Technology 60*, 3 (2009), 538–556.

[23] Stamatatos, E. On the robustness of authorship attribution based on character n-gram features. *Journal of Law and Policy 21*, 2 (01 2013), 421–439.

[24] Wanwan, Z., and Jin, M. A review on authorship attribution in text mining. *Wiley Interdisciplinary Reviews: Computational Statistics 15* (04 2022).

[25] Zhang, X., Zhao, J., and LeCun, Y. Character-level convolutional networks for text classification. In *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds. Curran Associates, Inc., 2015, pp. 649–657.

[26] Zurada, J. M. Introduction to artificial neural systems, 1992.

Department of Computer Science, Babeş-Bolyai University, Cluj-Napoca, Romania

Email address: dana.lupsa@ubbcluj.ro

Email address: sanda.avram@ubbcluj.ro

Email address: radu.lupsa@ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LXX**, Number 1-2, 2025
DOI: 10.24193/subbi.2025.03

BERT-BASED AUTHORSHIP ATTRIBUTION ON THE ROMANIAN DATASET CALLED ROST

SANDA-MARIA AVRAM

Abstract. Having been around for decades, the problem of authorship attribution remains a current focus. Some of the more recent instruments used are the pre-trained language models, the most prevalent being BERT. Here we used such a model to detect the authorship of texts written in the Romanian language. The dataset used is highly unbalanced, i.e., significant differences in the number of texts per author, the sources from which the texts were collected, the period in which the authors lived and wrote these texts, the medium intended to be read (i.e., paper or online), and the type of writing (i.e., stories, short stories, fairy tales, novels, literary articles, and sketches). The results are better than expected, sometimes exceeding 87% macro-accuracy.

1. Introduction

The problem of automated Authorship Attribution (AA) has been studied for decades and remains highly relevant today. It is defined as the task of determining the author of an unknown text based on its textual characteristics [1]. We note that, while traditional approaches to AA often employed artificial intelligence (AI) methods using simple classifiers (such as linear SVMs or decision trees) and classical feature sets like bag-of-words or character n-grams [2, 3], the field has evolved in recent years. The adoption of deep neural networks in natural language processing (NLP) has led to their application in authorship identification as well. More recently, pre-trained language models—in particular BERT and GPT-2—have been used for fine-tuning and accuracy improvements in AA tasks [2, 4, 5].

Received by the editors: 25 June 2025.

2010 *Mathematics Subject Classification.* 68T50.

1998 *CR Categories and Descriptors.* I.2.7 [**Artificial Intelligence**]: Natural Language Processing – *Text Analysis*; I.2.6 [**Artificial Intelligence**]: Learning – *Induction*.

Key words and phrases. Authorship Attribution, BERT, ROST.

This paper investigates the effectiveness of a Romanian pre-trained BERT model for authorship attribution. We specifically address the challenges posed by dataset imbalance, diversity of text sources, and genre variability.

The paper is organized as follows:

Section 2: reviews related work, especially methods employing artificial intelligence for authorship attribution;
Section 3: details the dataset preparation and preprocessing steps for the Romanian BERT model;
Section 4: presents implementation details and experimental results;
Section 5: concludes with final remarks and future directions.

2. Related work

The strategies used for addressing the AA problem are multiple. Authors of [4] grouped the main approaches into 4 classes:

Ngram: -includes character n-grams, parts-of-speech and summary statistics as shown in [6, 7, 8, 9];
PPM: - uses Prediction by Partial Matching (PPM) compression model to build a character-based model for each author, with works presented in [10, 11];
BERT: - combines a BERT pretrained language model with a dense layer for classification, as in [12];
pALM: - the per-Author Language Model (pALM), also using BERT as described in [13].

Here, we will focus on the Ngram and BERT classes.

2.1. Ngram: AI methods on ROST dataset.

In [14] we introduced a dataset, named *ROmanian Stories and other Texts* (ROST), consisting of Romanian texts that are stories, short stories, fairy tales, novels, articles, and sketches. We collected 400 such texts of different lengths, ranging from 91 to 39195 words. We used multiple AI techniques for classifying the literary texts written by multiple authors by considering a limited number of speech parts (prepositions, adverbs, and conjunctions). The methods we used were Artificial Neural Networks, Support Vector Machines, Multi-Expression Programming, Decision Trees with C5.0, and k-Nearest Neighbour.

For the tests performed in [14], the 400 Romanian texts were divided into training (50%), validation (25%), and test (25%) sets as detailed in Table 1. Some of the 5 aforementioned methods required only training and test sets. In such cases, we concatenated the validation set to the training set.

A numerical representation of the dataset was built as vectors of the frequency of occurrence of the considered features. The considered features were

TABLE 1. List of authors; the number of texts and their distribution on the training, validation, and test sets

#	**Author**	No. of texts	TrainSet size	ValidationSet size	TestSet size
0	Ion **Creangă**	**28**	14	7	7
1	Barbu Şt. **Delavrancea**	**44**	22	11	11
2	Mihai **Eminescu**	**27**	15	6	6
3	Nicolae **Filimon**	**34**	18	8	8
4	Emil **Gârleanu**	**43**	23	10	10
5	Petre **Ispirescu**	**40**	20	10	10
6	Mihai **Oltean**	**32**	16	8	8
7	Emilia **Plugaru**	**40**	20	10	10
8	Liviu **Rebreanu**	**60**	30	15	15
9	Ioan **Slavici**	**52**	26	13	13
	TOTAL	**400**	**204**	**98**	**98**

TABLE 2. Names used in [14] to refer to the different dataset representations and their shuffles.

#	**Designation**	**Features to Represent the Dataset**	**Shuffle**
1	**ROST-P-1**	prepositions	#1
2	**ROST-P-2**	prepositions	#2
3	**ROST-P-3**	prepositions	#3
4	**ROST-PA-1**	prepositions and adverbs	#1
5	**ROST-PA-2**	prepositions and adverbs	#2
6	**ROST-PA-3**	prepositions and adverbs	#3
7	**ROST-PAC-1**	prepositions, adverbs and conjunctions	#1
8	**ROST-PAC-2**	prepositions, adverbs and conjunctions	#2
9	**ROST-PAC-3**	prepositions, adverbs and conjunctions	#3

inflexible parts of speech (IPoS). Three different sets of IPoS were used. First, only prepositions were considered, then adverbs were added to this list, and finally, conjunctions were added as well. Therefore, three different representations of the dataset (of the 400 texts) were obtained. For each dataset representation (i.e., corresponding to a certain set of IPoS), the numerical vectors were shuffled and split into training, validation, and test sets as detailed in Table 1. This process (i.e., shuffle and split 50%–25%–25%) was repeated three times. We, therefore, obtained different dataset representations, which were referred to as described in Table 2.

All these representations of the dataset as vectors of the frequency of occurrence of the considered feature lists can be found as text files at reference [15].

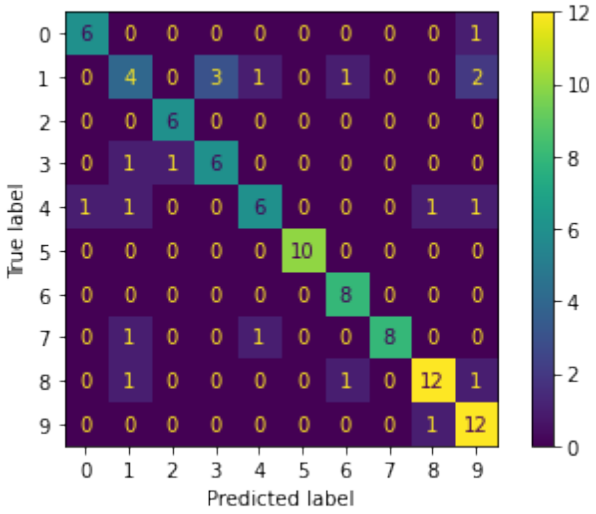


FIGURE 1. Confusion matrix on MEP's best results. The numbers from 0 to 9 are the codes given to our authors, as specified in the first column of Table 1.

These files contain feature-based numerical value representations for different texts on each line. The last column of these files contains numbers from 0 to 9 corresponding to the author, as specified in the first column of Table 1.

The best value obtained amongst all aforementioned AI methods was on ROST-PA-2 by using MEP, with a test error rate of 20.40%. This means that 20 out of 98 tests were misclassified. The obtained *Confusion Matrix* is depicted in Figure 1. The *macro-accuracy* value obtained was 80.94%. The Python code for building the *Confusion Matrix* and calculating the *macro-accuracy* value is provided at [16].

Bidirectional Encoder Representations from Transformers (BERT) is a language representation model designed to pretrain deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers [17]. It was first introduced in October 2018 by a team from Google and is now used as a machine learning framework for natural language processing (NLP) tasks (i.e., classification, entity recognition, question answering, etc.).

BERT is based on multiple techniques that preceded it, starting from *RNN* and *LSTMs* to *Attention* and *Transformers*. Starting from 2015, *Recurrent Neural Networks* (RNNs) were used for author identification [18]. However,

RNNs have a problem with remembering long-term dependencies (over ten sequences) [19]. *Long Short Term Memory* (LSTM) architecture [20] is a special kind of RNN, introduced to overcome this weakness of the traditional RNN.

RNN encoder-decoder [21] architectures were used mostly for translations. In this context, the encoder translates the input into a "codified" vector, which is then passed to the decoder, which translates it again into the output language. The input and output vectors do not need to have the same length. The challenge with the encoder-decoder approach is that it has to transform the input sentence into a fixed-length vector that becomes a bottleneck. Therefore, such a model has difficulty translating long sentences. The *attention* technique was introduced in [22, 23], through which instead of passing the last hidden state of the encoding stage, the encoder passes all the hidden states to the decoder. This allows access to parts of the source sentence that are relevant to predicting a target word. In 2017, the concept of *Multi-Head Self Attention* was introduced in the paper "Attention Is All You Need" [24]. This technique is called *transformers* and uses self-attention for language understanding, parallelization for better translation quality, and attention to integrate the relevance of a set of values (information) based on some keys and queries. Google proposed BERT [17] in 2018, which uses a bi-directional transformer. Due to its complexity, more data is required for training such models. Therefore, there are currently many pretrained models in different languages.

BERT was used for AA in [25], while the performance of BERT was compared with other techniques [18, 13, 4] for solving the AA problem. Work in [12] concludes that BERT is the highest-performing AA method.

2.2. **Comparing methods.** The multitude of AA approaches makes it difficult to have a unified view of the state-of-the-art results. In [4], authors highlight this challenge by noting significant differences in:

: Datasets
- size: small (CCAT50, CMCC, Guardian10), medium (IMDb62, Blogs50), and large (PAN20, Gutenberg);
- content: cross-topic ($\times_t$), cross-genre ($\times_g$);
- imbalance (imb): i.e., the standard deviation of the number of documents per author;
- topic confusion (as detailed in [6]).

: Performance metrics
- type: accuracy, F1, c@1, recall, precision, macro-accuracy, AUC, R@8, and others;
- computation: even for the same performance metrics, there were different ways of computing them.

: Methods

- the feature extraction method:
  - Feature-based: n-gram, summary statistics, co-occurrence graphs;
  - Embedding-based: char or word embedding, transformers;
  - Feature and embedding-based: BERT.

The work presented in [4] tries to address and "resolve" these differences, bringing everything to a common denominator by using *macro-accuracy* as a metric.

The overall accuracy, also known as *micro-accuracy* or *micro-averaged accuracy*, weights the accuracy for each sample (text) equally. The *macro-accuracy* metric, also known as *macro-averaged accuracy*, is regarded to be more accurate in the case of imbalanced datasets, as it is computed by weighting equally the accuracy for each class (author).

A formal description of computing micro- and macro-accuracy was provided by Jacob Tyo in [26] and is described next:

$$\begin{aligned} N &= \text{total number of texts} \\ M &= \text{number of authors} \\ N_i &= \text{number of texts for author } i \\ C_i &= \text{number of texts correctly predicted for author } i \\ \mathcal{A}_{micro} &= \text{micro-averaged accuracy} \\ \mathcal{A}_{macro} &= \text{macro-averaged accuracy} \end{aligned}$$

$$\mathcal{A}_{micro} = \frac{1}{N} \sum_{i=1}^{M} C_i$$

$$\mathcal{A}_{macro} = \frac{1}{M} \sum_{i=1}^{M} \frac{C_i}{N_i}$$

The results of the state of the art as presented in [14] are shown in Table 3.

TABLE 3. State of the art *macro-accuracy* of authorship attribution models. Information collected from [4] (Tables 1 and 3). *Name* is the name of the dataset; *No. docs* represents the number of documents in that dataset; *No. auth* represents the number of authors; *Content* indicates whether the documents are cross-topic ($\times_t$) or cross-genre ($\times_g$); *W/D* stands for *words per document*, representing the average length of documents; *imb* represents the *imbalance* of the dataset measured by the standard deviation of the number of documents per author.

Dataset						**Investigation Type**			
Name	**No. Docs**	**No. Auth**	**Content**	**W/D**	**Imb**	**Ngram**	**PPM**	**BERT**	**pALM**
CCAT50	5000	50	-	506	0	76.68	69.36	65.72	63.36
CMCC	756	21	$\times_t$ $\times_g$	601	0	86.51	62.30	60.32	54.76
Guardian10	444	13	$\times_t$ $\times_g$	1052	6.7	100	86.28	84.23	66.67
ROST	400	10	$\times_t$ $\times_g$	3355	10.45	80.94	–	–	–
IMDb62	62,000	62	-	349	2.6	98.81	95.90	98.80	-
Blogs50	66,000	50	-	122	553	72.28	72.16	74.95	-
PAN20	443,000	278,000	$\times_t$	3922	2.3	43.52	-	23.83	-
Gutenburg	29,000	4500	-	66,350	10.5	57.69	-	59.11	-

For the work presented in [14], we did not investigate how BERT would perform on ROST. Therefore, we will conduct this investigation here.

3. Prerequisite

Natural language processing (NLP) models, including those discussed in Section 2.1, as well as architectures such as LSTMs and CNNs, require input data to be represented as numerical vectors. This typically involves converting linguistic features, such as vocabulary terms and parts of speech, into numerical encodings. In contrast to traditional methods that depend on fixed word frequency counts or unique indices, BERT generates contextually informed word embeddings [27], whereby the numerical representation of a word dynamically varies according to its surrounding context and semantic meaning.

BERT is a pretrained language model that requires input data to conform to a specific format [27]. During pretraining, the model constructs a vocabulary comprising individual characters, subwords, and complete words that optimally represent the training corpus. The tokenizer initially attempts to match entire words within this vocabulary; if unsuccessful, it decomposes the word into the largest possible subword units present in the vocabulary. As a final fallback, the tokenizer segments the word into individual characters [28]. This tokenization strategy often increases the length of the original input sequence.

BERT processes input sequences with a maximum length of 512 tokens, of which the special start token ([CLS]) and end token ([SEP]) are predefined and mandatory components [27].

To accommodate these constraints, it was necessary to adjust the length of our texts to ensure compatibility with the model's input requirements. Through preliminary experiments testing sequence lengths of 91, 200, and 400 words, we observed optimal performance with inputs truncated to 200 words. Notably, 91 corresponds to the length of the shortest text in our dataset, which spans from 91 to 39195 words in length. This approach balances the trade-off between preserving sufficient contextual information and adhering to BERT's architectural limitations on input size.

We have set the maximum length of the input vector to 256 to accommodate the situations where some words are not in the pretrained model vocabulary and therefore, may be divided into subwords or even individual characters. Considering that some of the texts were written a couple of centuries ago, that might happen especially for words that are no longer used.

We kept the initial shuffles as described in Table 1 with the following changes:

- for each text that was longer than 200 words, we divided it into multiple texts of 200 words maximum;

TABLE 4. List of authors; the number of texts and their distribution on the training and test sets prepared for BERT

#	**Author**	No. of texts	TrainSet size shuffle 1, 2, 3	TestSet size shuffle 1, 2, 3
0	Ion **Creangă**	**520**	446,399,393	74,121,127
1	Barbu Şt. **Delavrancea**	**922**	706,595,690	216,327,232
2	Mihai **Eminescu**	**792**	550,467,447	242,325,345
3	Nicolae **Filimon**	**475**	389,326,278	86,149,197
4	Emil **Gârleanu**	**203**	169,146,160	34,57,43
5	Petre **Ispirescu**	**674**	489,469,482	185,205,192
6	Mihai **Oltean**	**106**	81,74,86	25,32,20
7	Emilia **Plugaru**	**463**	386,329,312	77,134,151
8	Liviu **Rebreanu**	**711**	535,523,546	176,188,165
9	Ioan **Slavici**	**1966**	1660,1467,1445	306,499,521
	TOTAL	**6832**	**5411,4795,4839**	**1421,2037,1993**

- we have only training and test sets, the validation sets being added to the training sets.

Thus, we obtained 3 shuffles of the dataset with the number of texts as detailed in Table 4.

4. TESTS

For our tests, we used Google Colab Pro[29], an online framework that offers different runtime environments to run code written in Python.

4.1. **Fine-tuning a pretrained Romanian BERT model.** We used the Romanian pretrained BERT model, described in [30] and available at [31], called: `dumitrescustefan/bert-base-romanian-cased-v1`.

For training and testing, we used the following classes from HuggingFace's Transformers [32] library:

AutoTokenizer: - to download the tokenizer associated to the pretrained Romanian model we chose;

AutoModelForSequenceClassification: - to download the model itself;

Trainer: and **TrainingArguments** - to fine-tune the chosen model for our dataset, namely ROST.

The preprocessing of the texts done by the tokenizer consists of splitting the input text into words (or parts of words, punctuation symbols, etc.), usually called tokens, which are then converted to IDs/numbers. To process all the

TABLE 5. Configuration parameters

Parameter	Value
architectures	BertForSequenceClassification
hidden_act	gelu
hidden_dropout_prob	0.1
hidden_size	768
initializer_range	0.02
intermediate_size	3072
layer_norm_eps	1e-12
num_attention_heads	12
num_hidden_layers	12
torch_dtype	float32
transformers_version	4.26.0
vocab_size	50000

TABLE 6. BERT on ROST parameters

Parameter	Value
Evaluation strategy	epoch
Num Epochs	3
Instantaneous batch size per device	8
Total train batch size (parallel, distributed, and accumulation)	8
Test Batch size	8
Gradient Accumulation steps	1
Learning rate	5e-05
Seed	42
Total optimization steps	2031, 1800, 1815
Number of trainable parameters	124449034
Tokenizer maximum length	256

input texts as a batch, one needs to pad them all to the same length or truncate them to the maximum length by specifying that to the tokenizer.

The configuration for our tests is detailed in Table 5.

For our training, the parameters used are presented in Table 6.

There are three different values for *Total optimization steps* corresponding to the three shuffles.

Some training performance metrics we obtained are presented in Table 7.

4.2. **Evaluation.** For evaluating the results, we used HuggingFace's Evaluate [32]. With this library, we computed the *micro-accuracy* metric (aka. *accuracy* or *micro-averaged accuracy*). The corresponding values for this metric,

TABLE 7. BERT on ROST training performance metrics

Parameter	**for shuffle 1**	**for shuffle 2**	**for shuffle 3**
train_runtime (in seconds)	261.1692	239.6519	241.3783
train_samples_per_second	62.155	60.025	60.142
train_steps_per_second	7.777	7.511	7.519
train_loss	0.255753	0.297084	0.381490

TABLE 8. BERT on ROST evaluation performance metrics

Parameter	**for shuffle 1**	**for shuffle 2**	**for shuffle 3**
eval_accuracy	0.864883	0.864997	0.849974
eval_runtime	6.0036	8.474	8.299
eval_samples_per_second	236.69	240.382	240.149
eval_steps_per_second	29.649	30.092	30.124

obtained for the three shuffles, are shown in Table 8 alongside other evaluation performance metrics.

We also used the sklearn.metrics module from the Scikit-learn [33] library. The classes used, pertaining to this module, were:

confusion_matrix: - to compute the confusion matrix;
ConfusionMatrixDisplay: - to generate a Confusion Matrix visualization as the ones displayed in Figures 2;
classification_report: - to obtain a report with main classification metrics;
accuracy_score: - to compute the accuracy classification score, also known as micro-accuracy or overall accuracy;
balanced_accuracy_score: - to compute the balanced accuracy, also known as macro-accuracy.

To compute the confusion matrix, we provided the true and predicted classes (i.e., codes corresponding to the authors). The graphical visualizations of the confusion matrices corresponding to the results obtained for the 3 shuffles are depicted in Figure 2. The numbers from 0 to 9 are the codes given to our authors, as specified in the first column of Table 4.

Figure 2 shows a significant confusion reoccurring between Barbu Ștefănescu Delavrancea (1) and Liviu Rebreanu (8). The texts written by these authors occurred in almost the same period (i.e., 1884-1909 and 1908-1935, respectively).

For each class/author, the classification report provided various metrics:

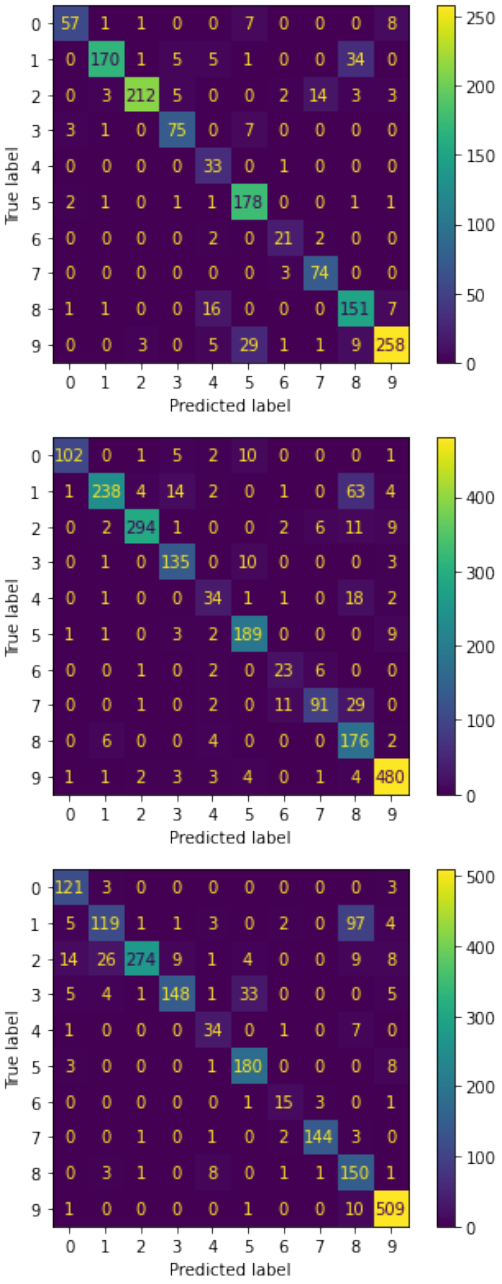


FIGURE 2. Confusion matrices for (a) shuffle 1, (b) shuffle 2, (c) shuffle 3

TABLE 9. Shuffle 1 results generated by sklearn.metrics's classification_report

Name	class	**precision**	**recall**	**f1-score**	support
Ion Creangă	0	0.905	0.770	0.832	74
Barbu Șt. Delavrancea	1	0.960	0.787	0.865	216
Mihai Eminescu	2	0.977	0.876	0.924	242
Nicolae Filimon	3	0.872	0.872	0.872	86
Emil Gârleanu	4	0.532	0.971	0.688	34
Petre Ispirescu	5	0.802	0.962	0.875	185
Mihai Oltean	6	0.750	0.840	0.792	25
Emilia Plugaru	7	0.813	0.961	0.881	77
Liviu Rebreanu	8	0.763	0.858	0.807	176
Ioan Slavici	9	0.931	0.843	0.885	306
	accuracy			0.865	1421
	macro avg	0.831	**0.874**	0.842	1421
	weighted avg	0.882	0.865	0.868	1421

: *Precision*—the number of correctly attributed authors divided by the number of instances when the algorithm identified the attribution as correct;
: *Recall (Sensitivity)*—the number of correctly attributed authors divided by the number of test texts belonging to that author;
: *F1-score*—a weighted harmonic mean of the *Precision* and *Recall*.

The classification results for the 3 shuffles are shown in Tables 9, 10, and 11.

A graphical representation of the results obtained per class and detailed in Tables 9, 10, and 11 is presented in Figure 3.

As can be seen, the worst results over the three shuffles are obtained consistently by Emil Gârleanu (4), Mihai Oltean (6), and Liviu Rebreanu (8), while the best results are obtained by Mihai Eminescu (2), Emilia Plugaru (7), and Petre Ispirescu (5).

Table 12 presents the best micro- and macro-accuracies obtained on the three shuffles.

The best overall accuracy is 0.864997, and it is obtained on the second shuffle. That is interesting, as in the investigations performed in [14] by using other AI techniques, also the second shuffle obtained the best results. This metric is also referred to as *micro-accuracy*, treating each sample (or, in this case, each text fragment) as equally important.

A more representative metric is the *macro-accuracy* as it computes the accuracy by treating each author equally. The best macro-accuracy obtained is 0.874031, and it is obtained on the first shuffle.

TABLE 10. Shuffle 2 results generated by sklearn.metrics's classification_report

Name	class	**precision**	**recall**	**f1-score**	support
Ion Creangă	0	0.971	0.843	0.903	121
Barbu Șt. Delavrancea	1	0.952	0.728	0.825	327
Mihai Eminescu	2	0.970	0.905	0.936	325
Nicolae Filimon	3	0.839	0.906	0.871	149
Emil Gârleanu	4	0.667	0.596	0.630	57
Petre Ispirescu	5	0.883	0.922	0.902	205
Mihai Oltean	6	0.605	0.719	0.657	32
Emilia Plugaru	7	0.875	0.679	0.765	134
Liviu Rebreanu	8	0.585	0.936	0.720	188
Ioan Slavici	9	0.941	0.962	0.951	499
	accuracy			0.865	2037
	macro avg	0.829	**0.820**	0.816	2037
	weighted avg	0.886	0.865	0.868	2037

TABLE 11. Shuffle 3 results generated by sklearn.metrics's classification_report

Name	class	**precision**	**recall**	**f1-score**	support
Ion Creangă	0	0.807	0.953	0.874	127
Barbu Șt. Delavrancea	1	0.768	0.513	0.615	232
Mihai Eminescu	2	0.986	0.794	0.880	345
Nicolae Filimon	3	0.937	0.751	0.834	197
Emil Gârleanu	4	0.694	0.791	0.739	43
Petre Ispirescu	5	0.822	0.938	0.876	192
Mihai Oltean	6	0.714	0.750	0.732	20
Emilia Plugaru	7	0.973	0.954	0.963	151
Liviu Rebreanu	8	0.543	0.909	0.680	165
Ioan Slavici	9	0.944	0.977	0.960	521
	accuracy			0.850	1993
	macro avg	0.819	**0.833**	0.815	1993
	weighted avg	0.871	0.850	0.850	1993

5. Conclusion and further work

In this paper, we continued our investigation started in [14], addressing the authorship attribution problem on a Romanian dataset using BERT. In order to accommodate BERT's input size limitations, we segmented texts into blocks of 200 tokens/words. This approach led to a more balanced dataset than in [14], as the texts have approximately the same length. However, other

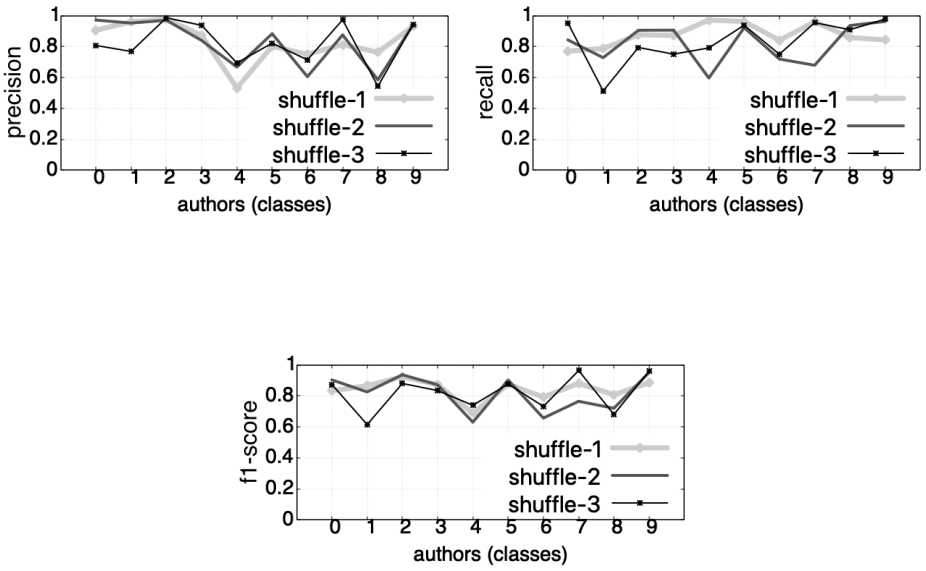


FIGURE 3. Graphical representations of results from Tables 9-11 for (a) precision, (b) recall, (c) f1-score

TABLE 12. Micro- and macro-accuracies

accuracy	**for shuffle 1**	**for shuffle 2**	**for shuffle 3**
micro-accuracy	0.864883	0.864997	0.849974
macro-accuracy	0.874031	0.819585	0.832905

sources of variability—such as the number of texts per author, the provenance of the texts, the historical periods during which the authors wrote, the intended reading medium (paper or online), and the range of genres (stories, short stories, fairy tales, novels, literary articles, and sketches)—remained.

The best result obtained in [14] was 80.94%. The best results achieved here with BERT reached 85.90%. However, due to changes in dataset segmentation made to meet BERT's processing requirements, results are not directly comparable across approaches. For a fair comparison, previous methods (including Artificial Neural Networks, Support Vector Machines, Multi-Expression Programming, Decision Trees with C5.0, and k-Nearest Neighbour) should be re-evaluated on the segmented (200-token) texts.

Our segmentation approach—dividing texts into fixed-size blocks—may disrupt sentence integrity and context. This could potentially impact BERT's performance. In future work, we plan to explore alternative segmentation strategies, such as overlapping sliding windows or segmenting at sentence boundaries, to preserve semantic coherence.

Additionally, we plan to further investigate methods such as Prediction by Partial Matching (PPM) as mentioned in Section 2. Future directions also include: (i) investigating BERT's attention maps to better understand model decision-making for Romanian authorship attribution, (ii) utilizing data augmentation techniques to address dataset imbalance, and (iii) combining BERT embeddings with other stylometric features, such as IPoS, to improve attribution accuracy.

References

[1] Wilson Alves de Oliveira Jr, Edson Justino, and Luiz Sá de Oliveira. Comparing compression models for authorship attribution. *Forensic science international*, 228(1-3):100–104, 2013.

[2] Mike Kestemont, Enrique Manjavacas, Ilia Markov, Janek Bevendorff, Matti Wiegmann, Efstathios Stamatatos, Benno Stein, and Martin Potthast. Overview of the cross-domain authorship verification task at pan 2021. In *CLEF (Working Notes)*, 2021.

[3] Mike Kestemont, Michael Tschuggnall, Efstathios Stamatatos, Walter Daelemans, Günther Specht, Benno Stein, and Martin Potthast. Overview of the author identification task at pan-2018: cross-domain authorship attribution and style change detection. In *Working Notes Papers of the CLEF 2018 Evaluation Labs. Avignon, France, September 10-14, 2018/Cappellato, Linda [edit.]; et al.*, pages 1–25, 2018.

[4] Jacob Tyo, Bhuwan Dhingra, and Zachary C Lipton. On the state of the art in authorship attribution and authorship verification. *arXiv preprint arXiv:2209.06869*, 2022.

[5] Georgios Barlas and Efstathios Stamatatos. A transfer learning approach to cross-domain authorship attribution. *Evolving Systems*, 12(3):625–643, 2021.

[6] Malik Altakrori, Jackie Chi Kit Cheung, and Benjamin C. M. Fung. The topic confusion task: A novel evaluation scenario for authorship attribution. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 4242–4256, Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics.

[7] Benjamin Murauer and Günther Specht. Developing a benchmark for reducing data bias in authorship attribution. In *Proceedings of the 2nd Workshop on Evaluation and Comparison of NLP Systems*, pages 179–188, Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics.

[8] Sebastian Bischoff, Niklas Deckers, Marcel Schliebs, Ben Thies, Matthias Hagen, Efstathios Stamatatos, Benno Stein, and Martin Potthast. The importance of suppressing domain style in authorship analysis. *arXiv preprint arXiv:2005.14714*, 2020.

[9] Efstathios Stamatatos. Masking topic-related information to enhance authorship attribution. *Journal of the Association for Information Science and Technology*, 69(3):461–473, 2018.

[10] Tempestt Neal, Kalaivani Sundararajan, Aneez Fatima, Yiming Yan, Yingfei Xiang, and Damon Woodard. Surveying stylometry techniques and applications. *ACM Computing Surveys (CSuR)*, 50(6):1–36, 2017.

[11] Oren Halvani and Lukas Graner. Cross-domain authorship attribution based on compression. *Working Notes of CLEF*, 2018.

[12] Maël Fabien, Esau Villatoro-Tello, Petr Motlicek, and Shantipriya Parida. BertAA : BERT fine-tuning for authorship attribution. In *Proceedings of the 17th International Conference on Natural Language Processing (ICON)*, pages 127–137, Indian Institute of Technology Patna, Patna, India, December 2020. NLP Association of India (NLPAI).

[13] Georgios Barlas and Efstathios Stamatatos. Cross-domain authorship attribution using pre-trained language models. In *IFIP International Conference on Artificial Intelligence Applications and Innovations*, pages 255–266. Springer, 2020.

[14] Sanda-Maria Avram and Mihai Oltean. A comparison of several ai techniques for authorship attribution on romanian texts. *Mathematics*, 10(23):1–35, 2022.

[15] Sanda-Maria Avram. Rost (romanian stories and other texts), 2022.

[16] Sanda Avram. Computing macro-accuracy of mep results on rost, 2023. Software available at `https://github.com/sanda-avram/ROST-source-code/blob/main/ROST_withMEPX.ipynb`.

[17] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[18] Douglas Bagnall. Author identification using multi-headed recurrent neural networks. *arXiv preprint arXiv:1506.04891*, 2015.

[19] Jianfeng Zhang, Yan Zhu, Xiaoping Zhang, Ming Ye, and Jinzhong Yang. Developing a long short-term memory (lstm) based model for predicting water table depth in agricultural areas. *Journal of hydrology*, 561:918–929, 2018.

[20] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[21] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27, 2014.

[22] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[23] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

[24] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[25] Andrei Manolache, Florin Brad, Elena Burceanu, Antonio Barbalau, Radu Ionescu, and Marius Popescu. Transferring bert-like transformers' knowledge for authorship verification. *arXiv preprint arXiv:2112.05125*, 2021.

[26] Jacob Tyo. Computing the macro-accuracy, January 2023. personal communication.

[27] Chris McCormick. Bert word embeddings tutorial, 2019. Available at `http://mccormickml.com/2019/05/14/BERT-word-embeddings-tutorial/`.

[28] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[29] Ekaba Bisong and Ekaba Bisong. Google colaboratory. *Building machine learning and deep learning models on google cloud platform: a comprehensive guide for beginners*, pages 59–64, 2019.

[30] Stefan Dumitrescu, Andrei-Marius Avram, and Sampo Pyysalo. The birth of Romanian BERT. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4324–4328, Online, November 2020. Association for Computational Linguistics.

[31] Stefan Dumitrescu, Andrei-Marius Avram, and Sampo Pyysalo. bert-base-romanian-cased-v1, 2023. Available at `https://huggingface.co/dumitrescustefan/bert-base-romanian-cased-v1`.

[32] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, October 2020. Association for Computational Linguistics.

[33] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Department of Computer Science, Babeş-Bolyai University, Cluj-Napoca, Romania

Email address: `sanda.avram@ubbcluj.ro`

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LXX**, Number 1-2, 2025
DOI: 10.24193/subbi.2025.04

WORD AND PUNCTUATION N-GRAM FEATURES IN ROMANIAN AUTHORSHIP ATTRIBUTION

DANA LUPŞA AND RADU LUPŞA

Abstract. This study addresses the problem of authorship attribution for Romanian texts, focusing on the use of N-gram features with an emphasis on semantic-independent representations. While character N-grams have been previously studied, this work extends the exploration to word and part-of-speech (POS) N-grams, as well as combinations involving punctuation, closed-class words, and filtered content words. Using the ROST corpus, we evaluate six supervised learning algorithms, with results averaged over multiple runs to ensure robustness. Our experiments show that Artificial Neural Networks (ANN) consistently achieve the highest performance, with word-based unigrams enhanced by punctuation reaching an average macro-accuracy of 0.93. Importantly, semantically independent features, such as closed-class words and POS replacements for nouns and verbs, yield small further improvements. These findings highlight the effectiveness of carefully designed N-gram features for Romanian AA and suggest that semantic-independent representations can complement traditional lexical approaches.

1. Introduction

Authorship attribution (**AA**) is the task of determining the author of a text based on its linguistic and stylistic properties. It has applications in fields such as digital humanities, plagiarism detection, forensic linguistics, and information security. A central challenge in AA is to identify features that capture the stylistic signature of an author while minimizing the influence of content and topic. Among the most widely used features are N-grams, which

Received by the editors: 20 September 2025.
2010 *Mathematics Subject Classification.* 68T50.
1998 *CR Categories and Descriptors.* code [**Artificial Intelligence**]: Natural Language Processing – *Text Analysis*; code [**Artificial Intelligence**]: Learning – *Induction*.
Key words and phrases. Authorship Attribution, Machine Learning, N-gram features.

encode recurring patterns at the level of characters, words, or part-of-speech (POS) tags.

While N-grams have been extensively studied in high-resource languages such as English, their use in Romanian AA remains relatively underexplored. The Romanian language presents additional challenges due to its rich morphology and limited availability of annotated corpora. Existing research on the ROST dataset has shown promising results with both traditional machine learning methods and modern neural approaches.

In this study, we provide a systematic evaluation of word and POS N-gram features, including their integration with punctuation, on a Romanian corpus. We introduce semantic-independent N-gram representations, showing how filtering out or replacing content words can enhance attribution performance. We demonstrate that ANN models achieve strong results, even with simple word unigrams plus punctuation, while semantically independent bigram features provide additional improvements. By focusing on both classical and semantic-independent feature sets, this study highlights effective strategies for authorship attribution in Romanian.

2. N-grams for authorship attribution

In authorship attribution, **N-grams** are widely used features due to their ability to capture stylistic and syntactic patterns unique to each author [10, 2, 16, 15, 17, 9, 7]. The most common types of N-gram features used in this task are character N-grams, word N-grams and POS N-grams. It is considered that for **short texts**, character N-grams are typically more effective, while character and POS N-grams offer higher robustness. Although N-grams are widely used in languages like English, it's been relatively understudied for Romanian texts. An initial exploratory study, based on character N-grams, was presented in [11].

Character N-grams are sequences of N consecutive characters. They are expected to capture style irrespective of word boundaries, by identifying spelling habits, punctuation, and affixation patterns. Most important advantages are that they are language-independent and robust to noise.

Word N-grams are sequences of N consecutive words. They capture lexical collocations; can reveal frequent phrases and idioms, but they are sensitive to topic variation. They can identify an author based on the themes he approaches (semantic preference), while the aim is to find features of a specific style.

POS tag N-grams are sequences of part-of-speech (POS) tags. They focuses on (syntactic) text structure and try to capture morpho-syntactic habits. A disadvantage is that, if POS tagging is performed automatically, it can introduce errors.

3. Features with no semantic information for authorship attribution

Excluding semantic information in authorship attribution is often desirable, because we want to capture the stylistic signature of the author rather than the topic or meaning of the text. If semantic content dominates, the model may just learn *what* the author writes about, not *how* they write. Strategies to exclude semantic information include:

- Use function words or closed-class words
- Use POS tag sequences, that is: convert text into part-of-speech (POS) sequences, then extract N-grams. It is an attempt to reflect syntactic structure, not semantics.
- Mask or replace content words: replace all nouns, verbs with placeholders, and keep only structural patterns (POS or function words).

3.1. **Function words and closed class words for semantic information of a text.** Function words are words that have grammatical or relational meaning rather than lexical meaning. Unlike content words, which reflect the topic of a text, they are not topic-specific. Even if two texts discuss different subjects, the distribution of function words tends to reflect the writer's unconscious style. Because authors typically pay little conscious attention to how they use these words, their distributional patterns tend to be stable across different texts by the same writer, making them reliable stylistic markers.

By definition, closed-class words are word categories that form a fixed set in a language. While function words are closed-class, certain closed-class words are not function words, for example quantifiers like *some* or *many*. Also pronouns are excluded from the function word category because their primary role is to serve as nominal arguments. Closed-class words do not carry significant semantic information on their own. It's also worth noting that in some studies, the terms closed-class words and function words are treated interchangeably, as are open-class words and content words [4, 5, 13].

3.2. **Nouns and verbs for semantic information of a text.** In opposition with closed-class words, open-class words are nouns, verbs, adjectives, and adverbs. In general, verbs and nouns contribute more directly and specifically to the semantic theme of a text, than adjectives and adverbs. Nouns name entities, concepts, objects, people; they support the core ideas of a text. Verbs express actions, states, or processes; they give action and direction to the theme. Nouns and verbs are essential for understanding what the text is about. On the other hand, adverbs and adjectives are not so specific to the message of the text. The adjectives add details about qualities or characteristics of nouns

and they don't define the theme, they add nuance to it. Similarly, adverbs modify verbs, adjectives, or other adverbs, indicating manner, time, place, etc.

In this study, we extend the investigation of semantic-independent features by filtering out only nouns and verbs during feature extraction. To the best of our knowledge, this represents the first attempt to use only non-noun and non-verb words for Romanian authorship attribution.

4. Issues in Romanian authorship attribution

Authorship attribution for Romanian texts presents a unique set of challenges, largely due to the language's specific characteristics and the general limitations faced by research in less-resourced languages.

One of the most significant challenges is the lack of large, high-quality, and publicly available annotated datasets. While resources like the ROST corpus exist [2, 14], they are often smaller than what is available for high-resource languages like English, which can limit the effectiveness of data-hungry deep learning models. The ROST corpus is highly unbalanced, with a significant variation in the number of texts per author (ranging from 27 to 60) and a broad historical period (1850-2023). It can contain texts from various genres (e.g., novels, articles, short stories) exposing substantial variation in text length, from short texts with of 90 words to some exceeding 39000 words. These factors introduce significant challenges for model performance on unseen data.

Also, the high-quality Natural Language Processing (NLP) set of tools for Romanian is not as extensive as for major languages. This can hinder research that relies on detailed linguistic feature extraction.

Despite these challenges, recent research [1, 2, 3, 11, 14] has made significant progress in the field, by exploring approaches that combine traditional stylometric features with modern learning techniques, including fine-tuned BERT models for Romanian and achieving state-of-the-art performance.

5. The Experiments: Materials and Methods

5.1. **The dataset.** The ROST (ROmanian Stories and other Texts) dataset is a significant and publicly available corpus for the Romanian language, specifically designed for Authorship Attribution (AA). It contains 400 documents authored by 10 writers, but it is important to know that the dataset is highly unbalanced, and the texts are diverse in both genre and length.

Several research papers have used the ROST dataset to benchmark authorship attribution models, using different approaches, from traditional machine learning to modern deep learning models. Table 1 provides a short overview of the results, showing the results and improvements researchers made in Romanian AA over time.

Study	Methods / Features	Evaluation	Notes
Avram et al. [2] (2022)	ANN, MEP, k-NN, SVM, DT; inflexible parts of speech	MAcc: 80.94%	Introduced ROST dataset
Avram [1] (2023)	Romanian-specific BERT	MAcc: 87.40%	
Nitu et al. [14] (2024)	Hybrid Transformers; RoBERT	F1: 95% , Error: 4%	
Lupsa et al. [11] (2025)	SVM, LR, k-NN, DT, RF, ANN; character N-gram	MAcc: 95.1 %	several cases with perfect classification

TABLE 1. Short overview of previous studies on the ROST dataset.

5.1.1. *Data Preprocessing.* Following the work in [11], we used most of the normalization reported there, in order to work with standardized data. We made the diacritic standardization and punctuation unification; we replaced all digits with a special character (@). We also marked the beginning and ending of paragraphs with the dollar sign ($); this allows us to treat paragraph breaks as distinct tokens in the N-gram generation process. We also performed all experiments on both lowercase and original-cased text; while some differences were observed in the results, they were small and do not justify a separate analysis, as in [11].

5.1.2. *Features extraction.* To build the representations of the texts, we used word and Part-of-Speech (POS) N-grams. We extracted these features with the help of `spaCy`'s `ro_core_news_sm`. This is a pretrained `spaCy` model designed for Romanian language processing, that can split text into token words, punctuation, and symbols. The model also assigns POS tags, such as NOUN and VERB, which were needed for our analysis.

5.2. **Classification algorithms.** Authorship attribution has been addressed using numerous classification algorithms [15, 12, 2, 6]. Continuing the work in [11], our analysis utilized a suite of six supervised learning algorithms, implemented via the `scikit-learn` library: Decision Trees (DT), Random Forest (RF), k-Nearest Neighbor (k-NN), Logistic Regression (LR), Support Vector Machine (SVM), and Artificial Neural Networks (ANN).

Similar to the methodology used in [11], we repeated the model training and evaluation five times, with a train-test split of 80 to 20, varying the random seed of the ML models for each run, to reduce the impact of stochastic variation on our findings. This approach ensures that our findings are robust and not dependent on a specific random seed. We also did the experiments for lowercase and original-cased letters, in order to cover both methodologies.

For data processing and manipulation, we used `NumPy` and `pandas`, and `scikit-learn` for model training and evaluation, while the textual data was transformed into a numerical vector format using `TfidfVectorizer`, which is part of the `scikit-learn` library.

In this study, accuracy and macro-accuracy are the primary metrics used to assess performance. Accuracy (**Acc**) is the simplest metric and it is generally used for a quick, general view in case of balanced datasets. Macro-Accuracy (**MAcc**) is the average of the accuracy achieved on each class independently, and it should be used for imbalanced classes. All reported Acc and MAcc values represent averages over multiple runs for a specific selection of experiment set, and they are presented in a concise way, in order to highlight the most important of our findings.

6. Overall results and discussion

6.1. **One-type feature for N-grams.**

6.1.1. *Word N-grams.* To evaluate the impact of sequential word patterns for authorship attribution, we conducted experiments using word-based N-grams as features. In this setting, contiguous sequences of words of length N (with N ranging from 1 to 5) were extracted from the text corpus and used to construct the feature space.

A visual representation of the results (mean macro-accuracy scores) is provided in Figure 1A. Despite differences in how classifier performance changes with N, the figure indicates that the highest scores are generally obtained at $N = 1$, with a maximum value of 0.9377 achieved by ANN and $N = 1$. The results also show that, in four of the six classifiers, accuracy decreases as N grows.

6.1.2. *POS N-grams.* In addition to word-sequence N-grams representations, we conducted experiments using Part-of-Speech (POS) N-grams as features. In this approach, the original text was first tagged with POS labels, after which contiguous sequences of tags of length N (with N ranging from 1 to 5) were extracted to construct the feature space. By systematically varying the N-gram length, we aimed to assess how increasingly complex syntactic sequences influence classification performance. Comparatively, while word N-grams tend to capture richer lexical information, POS N-grams offer a complementary perspective by abstracting away from vocabulary-specific features.

A visual representation of these results is shown in Figure 1B. The best performance is observed for $N = 2$ to $N = 4$; however, all macro-accuracy values remain below 0.9. The figure also shows that the majority of classifiers achieved their best performance when considering sequences of length greater

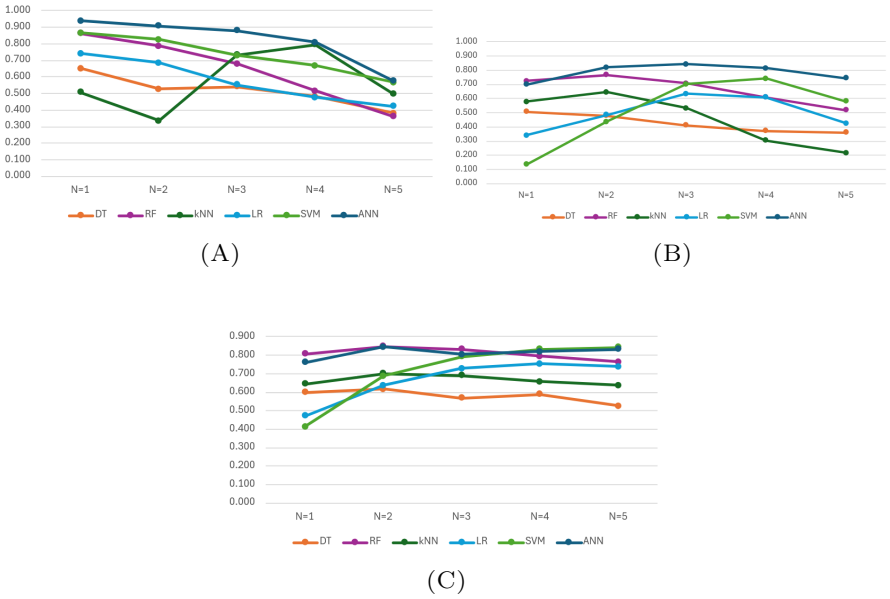


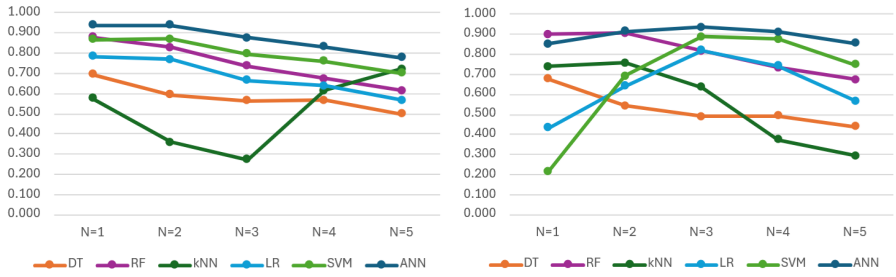
FIGURE 1. Results of classification experiments using exactly one feature type contributing to an N-gram. Used features are: only words in (A), only POS (part of speech) in (B), and only punctuation in (C). The figure shows the performance for each of the six classification methods: DT, RF, k-NN, LR, SVM, and ANN, when N varies from $N = 1$ to 5.

than 2, suggesting that contextual information beyond unigrams (syntactic structures and grammatical regularities) contributes significantly to classification accuracy.

6.1.3. *Punctuation N-grams.* We also conducted experiments in which punctuation marks were used exclusively as the basis for feature construction. In this setting, from the text corpus were extracted only punctuation symbols, and contiguous sequences of length N (with N ranging from 1 to 5) were extracted to form N-gram representations.

A visual representation of these results is presented in Figure 1C. Despite relying exclusively on punctuation-based features, the classification models achieve unexpectedly strong performance in certain cases, with maximum macro-accuracy values of 0.8453 for ANN at $N = 2$ and 0.8483 for RF at $N = 2$. While these findings highlight the potential discriminative power of punctuation, the achieved performance levels remain well below state-of-the-art performance levels.

6.2. **Inclusion of punctuation in feature sets.** Punctuation can encode subtle stylistic patterns [8, 9]. We evaluated the contribution of punctuation in context by adding it first to the basic word feature set and then to the POS feature set.



(A) (B)

FIGURE 2. Results of classification experiments when punctuation is added to N-grams. Used features are: (A) Words and punctuation; (B) POS and punctuation. The figure shows the performance for each of the six classification methods: DT, RF, k-NN, LR, SVM, and ANN, when N varies from $N = 1$ to 5.

Results for features combining words with punctuation are shown in figure 2A, while the corresponding results for word–punctuation features are presented in figure 2B. Reported values represent the mean macro-accuracy scores obtained across multiple experimental runs. We can see that the highest average macro-accuracy scores, slightly exceeding 0.93, are obtained with the ANN classifier. Specifically, values of 0.9392 and 0.9379 are achieved using word–punctuation features at $N = 1$ and $N = 2$, respectively, while a score of 0.9357 is obtained with POS–punctuation features at $N = 3$. We can also remark that, for word–punctuation features, most classifiers show no improvement for $N > 2$.

6.3. **Semantic-Independent N-gram Features.** Another series of experiments was made in order to evaluate the results obtained by using semantically independent features. We explored the effect of using closed-class words in authorship attribution, and also experimented with filtering out nouns and verbs.

6.3.1. *Closed-class words.* In our experiments, we did not use a predefined list of closed-class words, but we considered in this category all the words with POS being different than nouns, verbs, adjectives and adverbs. Non-word items, like punctuation and other symbols, are included in the feature set, for

this processing. We experimented with two distinct set of features based on closed-class words. First, we eliminated all open-class words from the feature set. After that, open-class words were replaced with their corresponding part-of-speech annotation.

An overview of the results is presented in the figures 3A and 3B. Reported values represent the mean macro-accuracy scores obtained across multiple experimental runs. We can see that the ANN classifier consistently achieved the highest performance across all values of N (from 1 to 5). The maximum performance achieved with closed-class features is obtained in the second configuration, and is reached with ANN and $N = 2$, having average MAcc value of 0.9446 and average Acc value of 0.9449. It is worth noting that there is one case for each of the two feature sets (see Table 2, sections named *closedclass* and *closedclass-pos*) in the classification experiments in which the ANN model attains perfect accuracy, meaning that all texts are correctly assigned to their corresponding authors.

6.3.2. *Filtering out nouns and verbs.* Nouns and verbs are generally regarded as the most informative lexical categories in a text. To investigate their role in authorship attribution, we designed two distinct feature sets based on their manipulation. In the first case, all nouns and verbs were entirely removed from the feature space. In the second case, nouns and verbs were replaced by their corresponding POS annotation, thereby abstracting their lexical identity while preserving structural information.

An overview of the results is presented in the figures 3C and 3D. We can see that ANN obtained the best results, for all values of N (from 1 to 5), as in the majority of the other experiments. The maximum performance achieved with filtering out the nouns and verbs is obtained in the second configuration, with ANN and $N = 2$, having average MAcc value of 0.9494 and Acc value of 0.9504. Results obtained by using ANN are also presented in Table 2, sections named *nonounverbs* and *nounverb-pos*. We should also mention that there is one case for each of the two feature sets (*nonounverbs* and *nounverb-pos*) in the classification experiments in which all the texts are correctly assigned to their authors (i.e. the accuracy is 1).

6.4. **Analysis of the results.** The results indicate that improvements in accuracy do not follow a correlated pattern across the six classifiers, suggesting that each model responds differently to the feature representations employed.

ANN consistently achieves the best results across the majority of experiments. Therefore, in the following, we focus on comparing the results obtained with ANN. A detailed summary of the results, including average, standard deviation, maximum values for accuracy and macro-accuracy values obtained,

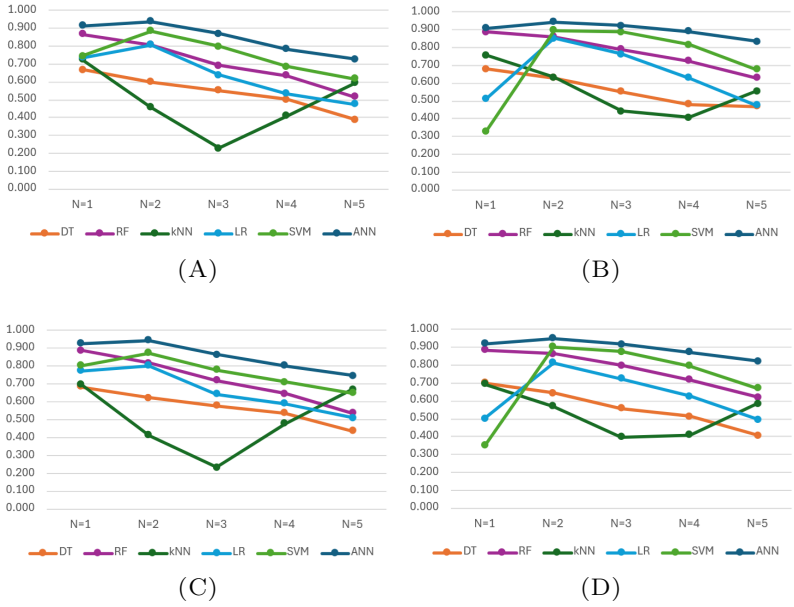


FIGURE 3. Results of classification experiments for N-grams formed by semantic-independent features. Used features are: (A) Closed class words and punctuation; (B) Closed class words, punctuation and POS tag for open class words. (C) Punctuations and all words different than nouns and verbs; (D) POS annotation for nouns and verbs and all other tokens (words and punctuation). The figure shows the performance for each of the six classification methods: DT, RF, k-NN, LR, SVM, and ANN, when N varies from $N = 1$ to 5.

is presented in Table 2. A brief visual summary of these results is presented in fig. 4.

The results obtained using the simple, classical word-based unigram features, including punctuation (see Section 6.2) yield an average macro-accuracy value of 0.9392. In our study on N-gram usage, if N-grams of order $N > 1$ fail to achieve better performance, it suggests that extending the classification to higher-order N-grams provides no additional benefit.

In Table 2, the MAcc value corresponding to word unigram features is marked with green, and all MAcc values exceeding this benchmark are highlighted in yellow for clarity. As shown in the table, approaches that rely on semantically independent features achieve higher macro-accuracy (MAcc) values. An interesting observation is that these improvements occur primarily at $N = 2$, whereas for $N = 1$ the MAcc values remain below 0.93—though still

Features		Acc			MAcc		
		mean	max	std	mean	max	std
word	N=1	0.9432	0.9753	0.0217	0.9392	0.9762	0.0249
	N=2	0.9442	0.9877	0.0195	0.9380	0.9929	0.0225
	N=3	0.8919	0.9383	0.0275	0.8772	0.9548	0.0315
	N=4	0.8506	0.9259	0.0280	0.8306	0.9025	0.0362
	N=5	0.7988	0.8395	0.0228	0.7789	0.8462	0.0338
closedclass	N=1	0.9170	0.9630	0.0246	0.9146	0.9735	0.0251
	N=2	0.9390	1.0000	0.0236	0.9381	1.0	0.0253
	N=3	0.8857	0.9383	0.0213	0.8698	0.9348	0.0301
	N=4	0.8067	0.8642	0.0355	0.7852	0.8624	0.0440
	N=5	0.7514	0.8519	0.0426	0.7265	0.8413	0.0433
closedclass-pos	N=1	0.9146	0.9630	0.0239	0.9100	0.9774	0.0298
	N=2	0.9449	1.0000	0.0261	0.9446	1.0	0.0278
	N=3	0.9272	0.9630	0.0257	0.9253	0.9786	0.0307
	N=4	0.8978	0.9630	0.0247	0.8908	0.9532	0.0274
	N=5	0.8556	0.9383	0.0405	0.8353	0.9276	0.0399
nonounverb	N=1	0.9314	0.9753	0.0227	0.9265	0.9762	0.0249
	N=2	0.9484	1.0000	0.0221	0.9457	1.0	0.0240
	N=3	0.8864	0.9259	0.0253	0.8657	0.9314	0.0358
	N=4	0.8306	0.8889	0.0233	0.8026	0.8869	0.0391
	N=5	0.7674	0.8519	0.0402	0.7457	0.8425	0.0486
nounverb-pos	N=1	0.9230	0.9630	0.0261	0.9204	0.9690	0.0307
	N=2	0.9504	1.0000	0.0244	0.9494	1.0	0.0263
	N=3	0.9215	0.9506	0.0255	0.9178	0.9607	0.0302
	N=4	0.8874	0.9506	0.0372	0.8746	0.9431	0.0416
	N=5	0.8442	0.9383	0.0450	0.8233	0.9122	0.0423
pos	N=1	0.8491	0.9259	0.0459	0.8514	0.9358	0.0485
	N=2	0.9217	0.9877	0.0268	0.9138	0.9917	0.0356
	N=3	0.9380	0.9753	0.0272	0.9357	0.9750	0.0292
	N=4	0.9168	0.9753	0.0377	0.9127	0.9774	0.0409
	N=5	0.8723	0.9630	0.0458	0.8552	0.9556	0.0433

TABLE 2. Performance results obtained by ANN for different N-gram feature configurations grouped by feature type. *word* and *pos* feature denote word and POS N-grams, including punctuation (as described in Section 6.2). *closedclass* refers to features restricted to closed-class words (as defined in Section 6.3.1), and *closedclass-pos* is for closed-class words combined with POS tags for all remaining items. *nonounverb* refers to experiments where all nouns and verbs were removed (see Section 6.3.2), while *nounverb-pos* denotes experiments in which nouns and verbs were retained but replaced with their corresponding POS tags.

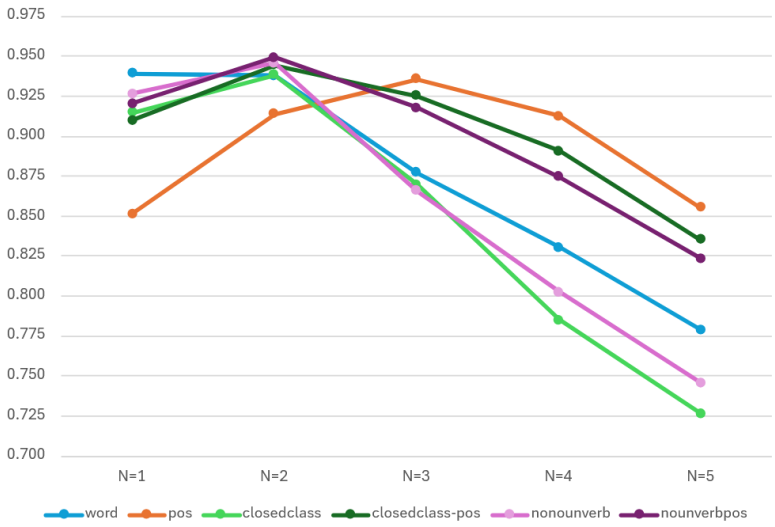


FIGURE 4. Macro-Accuracy averages for each N-gram size (N = 1 to 5) obtained by ANN, for the six feature selection methods: words with punctuation, part-of-speech (pos) with punctuation, features based on closed class words and features with no-nouns and no-verbs. The full set of results is reported in Table 2.

relatively high, exceeding 0.92. Also, the highest obtained value (0.9494) is close to the MAcc value of 0.951 reported in [11] (see Table 1).

Another interesting aspect is that, when the N-gram length increases beyond $N = 2$, i.e., for $N \geq 3$, the corresponding values consistently decrease, indicating reduced effectiveness of higher-order N-grams for most of feature selection cases. But feature representations using part-of-speech with punctuation information achieve better performance than the other feature selection when $N \geq 3$. This approach reaches a score of 0.9357 average MAcc for $N = 3$, and then decreases for higher values of N.

7. CONCLUSIONS

In this paper, we continued our investigation started in [11], addressing the authorship attribution problem on a Romanian dataset. To the best of our knowledge, this is the first study on Romanian that explores both word N-grams and POS N-grams, and also some of their combinations, focusing on N-grams of semantic independent features, as presented in section 6.3.

Our experiments show that Artificial Neural Networks (ANN) consistently achieve the highest performance. More than that, simple word-based unigram

features, enhanced with punctuation, reaching a macro-accuracy of 0.9392. Compared to this, feature representations designed to be semantically independent provide small improvements, particularly at the bigram level, with incorporating part-of-speech information for eliminated items, where they surpass the simple unigram scores and outperform the rest of results in our experiments. Higher-order N-grams ($N \geq 3$) show declining performance for most of the features. These findings highlight the effectiveness of carefully designed N-gram features for Romanian AA and suggest that semantic-independent representations can complement traditional lexical approaches.

References

[1] Avram, S.-M. Bert-based authorship attribution on the romanian dataset called rost. *arXiv preprint arXiv:2301.12500* (2023).

[2] Avram, S.-M., and Oltean, M. A comparison of several ai techniques for authorship attribution on romanian texts. *Mathematics 10*, 23 (2022), 4589.

[3] Briciu, A., Czibula, G., and Lupea, M. `AutoAt`: A deep autoencoder-based classification model for supervised authorship attribution. *Procedia Computer Science 192* (10 2021), 397–406.

[4] De Marneffe, M.-C., Nivre, J., and Zeman, D. Function words in universal dependencies. *Linguistic Analysis 43*, 3–4 (2024), 549–588.

[5] Drexler, E. Qnrs: Toward language for intelligent machines, 2021.

[6] He, X., Lashkari, A. H., Vombatkere, N., and Sharma, D. P. Authorship attribution methods, challenges, and future research directions: A comprehensive survey. *Information 15*, 3 (2024).

[7] Houvardas, J., and Stamatatos, E. N-gram feature selection for authorship identification. In *Artificial Intelligence: Methodology, Systems, Applications* (2006).

[8] Howedi, F., and Mohd, M. Text classification for authorship attribution using naive bayes classifier with limited training data. *Computer Engineering and Intelligent Systems 5* (2014), 48–56.

[9] Koppel, M., Schler, J., and Argamon, S. Computational methods in authorship attribution. *Journal of the American Society for information Science and Technology 60*, 1 (2009), 9–26.

[10] López-Anguita, R., Montejo-Ráez, A., and Díaz-Galiano, M. C. Complexity measures and pos n-grams for author identification in several languages: Sinai at pan@clef 2018. In *Conference and Labs of the Evaluation Forum* (2018).

[11] Lupsa, D., Avram, S.-M., and Lupsa, R. Oldies but goldies: The potential of character n-grams for romanian texts. *Studia Universitatis Babeș-Bolyai Informatica 70*, 1-2 (2025), 25–42.

[12] Misini, A., Kadriu, A., and Canhasi, E. A survey on authorship analysis tasks and techniques. *SEEU Review 17* (12 2022), 153–167.

[13] Niculescu, O., and Vasileanu, M. Prolongation in Romanian. In *Interspeech 2025* (2025), pp. 379–383.

[14] Nitu, M., and Dascalu, M. Authorship attribution in less-resourced languages: A hybrid transformer approach for romanian. *Applied Sciences 14*, 7 (2024), 2700.

[15] Stamatatos, E. A survey of modern authorship attribution methods. *Journal of the American Society for information Science and Technology 60*, 3 (2009), 538–556.

[16] Stamatatos, E. On the robustness of authorship attribution based on character n-gram features. *Journal of Law and Policy 21*, 2 (01 2013), 421–439.
[17] Wanwan, Z., and Jin, M. A review on authorship attribution in text mining. *Wiley Interdisciplinary Reviews: Computational Statistics 15* (04 2022).

Department of Computer Science, Babeş-Bolyai University, Cluj-Napoca, Romania
Email address: dana.lupsa@ubbcluj.ro
Email address: radu.lupsa@ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LXX**, Number 1-2, 2025
DOI: 10.24193/subbi.2025.05

THE G1 GAUSSIAN-TYPE RESOURCE ALLOCATION POLICY FOR VIRTUALIZED DATA CENTERS: THE SCALING PROBLEM AND VARIATION OF PARAMETERS

CORA CRĂCIUN

Abstract. Virtualized data centers use techniques such as resource consolidation to reduce the energy consumption and load balancing methods to improve performance. A previously proposed resource allocation policy of Gaussian type, named G1, has a behaviour in-between resource consolidation and load balancing. This policy was previously evaluated by simulation in a small data center configuration. This paper evaluates G1 for higher dimensional data centers with up to 400 hosts and 800 virtual machines, in order to establish if the policy scales with the dimension of the data center. G1 is compared with the First Fit heuristic by simulation for time-varying workloads. Metrics such as energy consumption, the mean number of active hosts, and the number of VMs migrations are calculated for different parametrizations of the score function of the G1 policy.

1. Introduction

Data centers use virtualization [1] as a way to manage the resources efficiently in dynamic conditions with time-varying workloads. The jobs are encapsulated in virtual machines (VMs) which are deployed on the physical resources and may be reconfigured or relocated. Through consolidation of the virtual machines on the hosts, the energy consumption of the data centers decreases. Heuristics such as First Fit (FF) and Best Fit (BF) proved efficient for workload consolidation [2, 3, 4, 5]. A drawback of the consolidation methods is that the virtual machines are packed too tightly on the physical

Received by the editors: 21 July 2025.

2010 *Mathematics Subject Classification.* 68M14, 68M20.

1998 *CR Categories and Descriptors.* C.2.4 [**Computer-Communication Networks**]: Distributed Systems – *Network operating systems*; C.4 [**Computer Systems Organization**]: Performance of Systems – *Design studies*.

Key words and phrases. G1 Gaussian-type policy, resource allocation, energy consumption, virtual machine migration.

hosts and therefore when the workload increases some virtual machines from the overloaded hosts should be migrated to other physical machines with spare resources [6, 7]. Virtual machine migrations, however, come with energy and performance overheads. The physical machines involved in the migration process consume more energy, the performance of the jobs encapsulated in the virtual machines decreases, and the network may become overused [8, 9, 10]. Unlike consolidation methods, load balancing techniques [11] aim to improve the performance of the data centers. The virtual machines are distributed on more hosts and afford better workload increases. When the workload is low, however, the physical machines may become underused and again the virtual machines are migrated and the empty hosts are switched off. The consolidation and load balancing techniques were combined with proportional sharing and Dynamic Voltage and Frequency Scaling procedures in order to address the performance and energy consumption problems [6, 12].

As a way to deal with the energy consumption - performance trade-off problem, we proposed previously two Gaussian-type resource allocation policies, G1 and G2, which pack the VMs on hosts less tightly than the greedy heuristics such as First Fit and Best Fit but more tightly than the load balancing procedures [13]. In the resource allocation process, the two policies maximize score functions of Gaussian shape and having some adjustable parameters. The G1 policy may be used in online conditions and chooses the hosts with the highest scores for the enqueued VMs. The G_1 score function depends on the used resources of the physical machines. The G2 policy, on the other hand, is more suitable for offline conditions and chooses the (VM,host) pairs that maximize a G_2 score function depending on the required resources by the VM and the available resources of the host. The single constraining resource for the VMs' allocation to the hosts was the CPU, but the Gaussian policies may be extended to other type of resources such as memory and disk space. The Gaussian-type policies were tested by simulation in a framework [14] built on the Haizea lease scheduler [15, 17, 18, 19, 20], for configurations with few resources, namely one with insufficient physical resources, (8 hosts, 40 VMs), and one with sufficient physical resources, (20 hosts, 40 VMs). The VMs had time-varying workloads, randomly generated in a given range, and were migrated when the hosts on which were deployed became overloaded. Metrics such as energy consumption, mean number of active hosts, VM migration number, flow time, and makespan were computed. The simulations showed that the G1 and G2 policies used more hosts than the FF and BF heuristics and their decreasing forms and thus consumed more energy, but reduced the number of virtual machines migrations, affording better the workload variations [13, 14, 16]. It should be noted that the reduction of the number of VMs

migrations was mainly achieved by a less tightly packing of the VMs with the Gaussian-type resource allocation policies. The migration of the VMs was performed with a VM migration policy, such as migrating the VMs with the least or highest CPU requirements.

This paper investigates the G1 policy in more detail. Using a new software implementation of the data center in Python, the G1 policy is tested for data centers with up to 400 hosts and 800 VMs. The energy consumption, the mean number of active hosts, and the VM migration number are computed for each data center configuration. Moreover, for a configuration with 300 hosts and 600 VMs, the simulations are performed for different parametrizations of the G_1 score function.

The paper is structured as follows. Next section reviews some investigations which aimed to reduce the number of VMs migrations in data centers. The section Computational details gives the expression of the G_1 score function and contains details about the software implementation. The section Results and discussion reports the computed metrics for the data centers of different sizes and for the G1 policy with various parametrizations. The final section concludes the current work.

2. Related work

Beside reducing the energy consumption or improving performance several studies tried to reduce the number of VMs migrations. In one of the algorithms used by the pMapper controller [21], Verma et al. utilized an incremental First Fit Decreasing resource allocation method combined with a VM migration policy that migrates the VMs with the smallest size, thus minimizing the power usage and reducing the VMs migrations. In [22], Ferreto et al. applied linear programming or heuristics for dynamic consolidation of the VMs and a migration controlled method which prevented to migrate the VMs with steady requirements. By performing in this way, the VMs migrations were reduced, with a small overhead in the number of used physical machines. Sandpiper [23] used algorithms to find the hotspots in data centers and to remove them by VMs operations such as VM migration, swapping, or resizing. The migration overhead was minimized by migrating the VMs with the highest volume-to-size ratio, from the physical machines with the highest volume to the least used physical machines. In order to reduce the energy consumption, the number of VMs migrations, and the SLA violations in heterogeneous data centers, Beloglazov et al. [5] combined a Power Aware Best Fit Decreasing VM packing method with a host overloading detection procedure based on thresholds and a Minimization of Migrations policy. The simulation experiments were performed using the CloudSim framework [24, 25]. In [26], Hermenier

et al. proposed a constraint programming procedure called BtrPlace in order to consolidate the VMs on the physical machines. When several constraints were not satisfied, a reconfiguration plan was generated, which enhanced with different strategies aimed to reduce the VMs relocations. Mastroianni et al. [27, 28] proposed a decentralized resource management method for Cloud data centers, called ecoCloud, in order to minimize the number of used physical machines, but without decreasing performance. The algorithm mapping the new or relocated VMs on servers solved a CPU and RAM-constraint Bin Packing Problem. The destination hosts were chosen probabilistically by performing Bernoulli trials [27]. Our Gaussian-type score functions resemble the probability functions used by ecoCloud for VM mapping and migration. Ashraf and Porres [29] proposed a multi-objective algorithm based on an ant colony system, aiming to reduce the number of used physical machines and the number of VMs migrations.

3. Computational details

A data center has a number N_H of homogeneous hosts and N_V virtual machines which must be mapped on physical machines for their deployment, based on their CPU requests. The data center has sufficient hosts for running all VMs. All initial VMs start processing simultaneously. Each VM has a trace of CPU requests lasting 40 minutes. The CPU requests are uniformly distributed numbers in the $[10, 40]$ range (in percents), rounded to the closest integer value. The CPU request of a VM changes at each 2 minutes. The VMs are mapped on hosts based on the G1 resource allocation policy or, for comparison, the FF heuristic. In time, because the VMs' workload changes, the hosts may become overused. In such a case, selected VMs are migrated to other hosts.

3.1. **VM scheduling.** The VMs are enqueued and are assigned to the hosts in their enqueuing order. When some VMs are suspended from the overloaded hosts, they are immediately assigned to other hosts which have spare resources. The time delays due to VMs' suspension, migration, and resumption on the new hosts are ignored.

3.2. **Resource allocation.** In current work, the G1 resource allocation policy is compared with the FF heuristic, known for its efficiency in reducing the energy consumption. The G1 policy uses a $G_1(U_{CPU})$ score function in order to map the VMs on hosts, where U_{CPU} is the CPU usage of a physical machine. The VMs are assigned in order to those physical machines which maximize the score function. A VM may be assigned to a host if its CPU requirements,

R_{CPU}, are less than the host's available resources, A_{CPU}. The total CPU capacity of a host, $T_{CPU} = U_{CPU} + A_{CPU}$, is 100%.

The $G_1(U_{CPU})$ function is the following Gaussian:

$$G_1(U_{CPU}) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left[-\frac{(U_{CPU} - \mu)^2}{2\sigma^2}\right],$$

where

$$\mu = \frac{Thr_L + Thr_H}{2}, \quad \sigma = \frac{Thr_H - Thr_L}{2\sqrt{2}\,\mathrm{erf}^{-1}(a)}.$$

erf^{-1} is the inverse error function.

The mean value, μ, of the two threshold values, Thr_L and Thr_H, is the middle point of the Gaussian. The threshold values fall in the $[0,\ T_{CPU}]$ interval and $Thr_H > Thr_L$. The parameter a represents the area between the threshold values, below the Gaussian, and belongs to the $(0, 1)$ interval.

3.3. **VMs suspension and migration.** A host is overloaded when $U_{CPU} > T_{CPU}$. The first trial when a host becomes overloaded is to suspend a single VM, with the smallest possible CPU request. If this is not possible in order to remove the overload then more VMs are suspended. The VMs of the overloaded host are sorted in increasing order of CPU requirements and the VMs are suspended in order until the overload is cleared. This process may lead to unnecessary suspensions. Then, the list of suspended VMs is scanned in decreasing order after CPU request and the VMs which still do not cause the overload are restored. This kind of suspension was used in reference [14]. The reason for suspending the VMs with smaller CPU requests is to reduce the overhead of their migration and to find easily spare resources on the hosts. The hosts are checked for overloading in the increasing order of their identifier. The VMs suspended from all overloaded hosts are sorted in increasing order of the initial host identifiers (the initial hosts are the hosts from which the VMs are migrated) and then after the VMs' identifier. Then, all simultaneously suspended VMs are mapped on other hosts with the resource allocation policy, with no time delay.

In current work only the overused hosts were subject to VMs suspensions. It is known that when the hosts are underused, all their VMs may be migrated and consolidated on other used hosts. The previously underused hosts, now freed, are switched off in order to reduce the energy consumption.

3.4. **The computed metrics.** The metrics computed which enabled us to compare the FF and G1 policies are the CPU energy consumption of the hosts, the mean number of active hosts, and the number of VMs migrations. The additional energy consumption of the hosts involved in migrations was neglected.

The CPU energy consumption may be computed as the following sum:

$$E = \sum_s P(\tau_s)(\tau_{s+1} - \tau_s),$$

where $P(\tau_s)$ is the power used by all physical machines at the scheduling time τ_s. The scheduling times were considered the times when the VMs' CPU requirements changed, namely 0, 2, ..., 38 minutes. For each host, the power was considered linearly dependent on the hosts' CPU utilization, U_{CPU}, according to the relation [30]:

$$P = P_{idle} + (P_{busy} - P_{idle})U_{CPU}.$$

P_{idle} is the power used when no VM is deployed on the host, while P_{busy} is the power when the host is fully utilized. The values used in computation were those from reference [5], $P_{busy} = 250$ W and $P_{idle} = 70\%P_{busy} = 175$ W. The energy consumption was computed in kilowatt hour.

The mean number of active hosts was computed by summing up the number of used hosts at all scheduling times and by dividing the result with the number of scheduling times. The number of VMs migrations is the total number of migrations experienced by the VMs in the makespan.

3.5. **Software design.** The software used in this work is a simplified implementation of the Haizea-based framework presented in [14], which may be used for higher number of VMs and hosts. Current software is not based on Haizea [15] or other available toolkit. The scheduling, resource allocation, and VM migration policies implemented in the software are identical with some of the policies from the Haizea-based framework. The reason for using a simplified implementation of the Haizea-based framework was to enable comparison of the current work with the results from [13, 16].

The CPU trace of each VM is implemented as a list containing the CPU requests of the workload, at each 2 minutes, for a total makespan of 40 minutes. The CPU requests are randomly generated between 10% and 40% as in references [13, 14, 16], again for direct comparison of present results with those from previous work. The VMs and hosts are implemented as Python classes. The VM class defines the VM identifier (ID), the state of the VM (not assigned to a host, assigned to a host, suspended for further migration), the current physical machine hosting the VM, the next host to which the VM is migrated, the CPU trace list with the workload requirements. The Host class defines the host identifier, the total, available, and used CPU of the host, the state of the host (switched off, busy, overloaded), the idle power usage, the total power usage, a list with power usage at each scheduling time, a list with the IDs of the current VMs deployed on the host, a list with the IDs of the

VMs suspended from the host, a function for updating the resources of the host.

The software creates a list with N_V VM objects and a list with N_H Host objects. Since the data center has enough physical machines, all VMs are mapped on hosts at the initial time, with the chosen resource allocation policy (FF, G1), and the hosts' resources are updated. At the other scheduling times, coinciding with the times of changing the VMs' CPU requests, the first operation is detection of overloaded hosts. Then one or more VMs are suspended from the overloaded hosts and the hosts' CPU resources are updated accordingly. The VMs to be migrated are mapped on other hosts with enough available resources, using the chosen resource allocation policy, and again the hosts' resources are updated. The scheduling, resource allocation, and VM migration policies used by the software are those described in the previous sections. At each scheduling time, the hosts' power usage and the number of used hosts are computed. After the processing of the VM CPU traces, the energy consumption, the mean number of active hosts, and the number of VMs migrations are reported.

4. Results and discussion

4.1. **The scaling problem.** In order to investigate if the G1 resource allocation policy scales with the dimension of the data center, different simulation experiments were performed for the following configurations: (50 hosts, 100 VMs), (100 hosts, 200 VMs), (200 hosts, 400 VMs), (300 hosts, 600 VMs), and (400 hosts, 800 VMs). The number of hosts was chosen such that to have enough physical resources for VMs with CPU requirements in the [10,40] range. 20 simulation experiments, with different VM CPU traces, were performed for each data center configuration. Each experiment was performed for the FF heuristic and G1 policy, with the same VM CPU trace. The parameters for the G_1 score function were $Thr_L = 40$, $Thr_H = 80$, and $a = 0.8$, as in references [13, 14, 16]. Figure 1 presents the results of the 20 simulation experiments, in boxplot representation, for the three metrics computed in this paper, while Table 1 contains the relative contrasts of the mean values of the metrics for G1 compared with FF. A positive value in the table indicates that the metric is higher for G1 than for FF and a negative value corresponds to a lower value for G1 than for FF. As can be seen in Figure 1 the number of used hosts is significantly smaller than that provisioned, for both policies. The energy consumption when using the G1 policy is higher than for FF in the limit of 5%, for all data center configurations. This additional energy consumption for G1 comes from using a higher number of hosts. The VM migration number, on the other hand, is significantly lower (in the limit of 25%) for G1

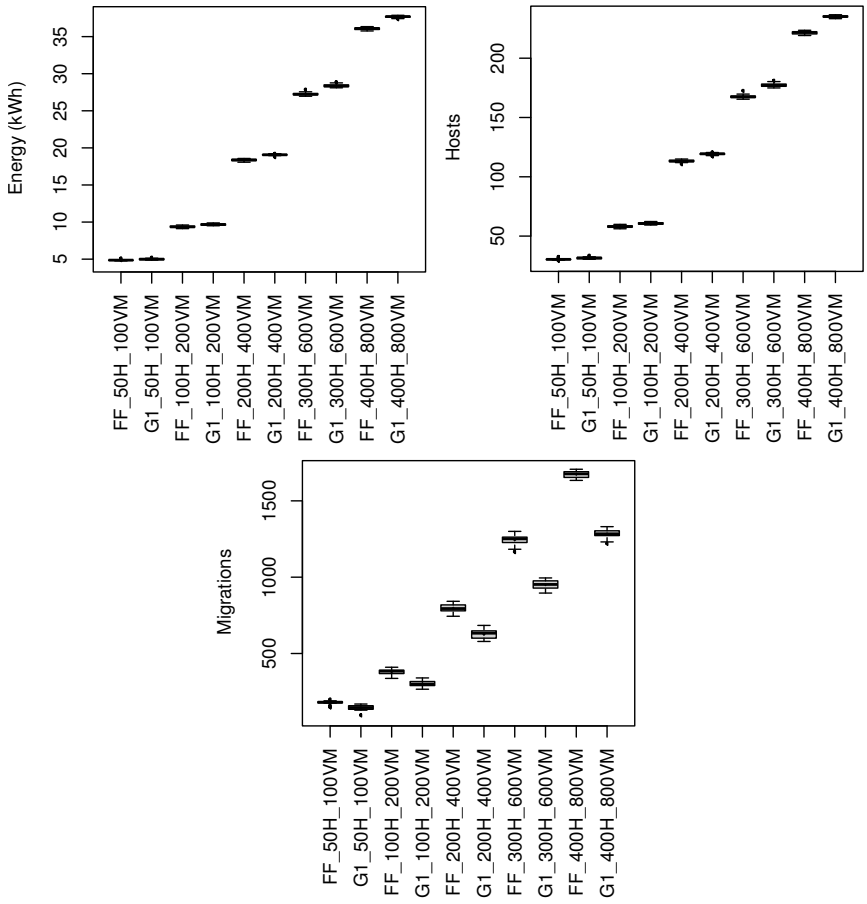


FIGURE 1. Boxplot representation [31] of the consumed energy, the mean number of active hosts, and the VM migration number for the G1 and FF policies, in data centers with different number of hosts and VMs. Simulation conditions: 40 min long VM CPU traces, 20 simulation experiments.

compared to FF. The relative values for the energy, mean number of active hosts, and VM migration number in absolute value increase slightly with the dimension of the data center. G1 reduces the number of VMs migrations with an energy consumption overhead, packing the VMs on hosts less tightly than the FF heuristic.

TABLE 1. The energy consumption, the mean number of active hosts, and the VM migration number relative contrasts (in percents), calculated from the mean values on 20 simulation experiments, for the indicated G1_H_VM policies, with the parameters $a = 0.8$, $\mu = 60$, and $\Delta Thr = 40$, with respect to FF_H_VM policies

Compared policies	θ^E_{rel}	θ^H_{rel}	θ^{NM}_{rel}
G1_50H_100VM vs. FF_50H_100VM	2.5	3.5	-19.1
G1_100H_200VM vs. FF_100H_200VM	3.2	4.4	-20.6
G1_200H_400VM vs. FF_200H_400VM	3.9	5.4	-21.8
G1_300H_600VM vs. FF_300H_600VM	4.1	5.8	-23.7
G1_400H_800VM vs. FF_400H_800VM	4.4	6.2	-23.4

4.2. **Variation of parameters for the G_1 score function.** For the (300 hosts, 600 VMs) data center configuration, simulation experiments were performed for different parametrizations of the G_1 score function. The Gaussian function was centered at the values $\mu = 40$, 50, 60, and 70, the distance between the thresholds was $\Delta Thr = 40$, and the a parameter had the values 0.4, 0.6, 0.8, and 0.95. Figure 2 presents the energy, the mean number of active hosts, and the VM migration number metrics for 20 experiments with the FF and G1_a_μ policies. Table 2 shows the relative contrasts of the mean values of the metrics for G1 with respect to FF. For energy the relative contrasts are in the limit of 7.5%, for the mean number of active hosts the relative contrasts are in the limit of 10%, while for the number of migrations the relative contrasts are in the limit of 38%. The a parameter changes the metrics only slightly. For instance the relative contrast for energy for G1_0.4_40 versus FF is 7%, while for G1_0.95_40 versus FF is 6.8%. For the number of migrations, the relative contrast for G1_0.4_40 versus FF is -37.0% and for G1_0.95_40 versus FF is -37.3%. The μ parameter and thus the values of the two thresholds, on the other hand, change significantly the relative contrasts of the metrics. When the μ value increases, the relative contrasts of G1 versus FF decrease, so G1 resembles more the FF heuristic. For instance, the relative contrast of the energy for G1_0.6_40 versus FF is 7.1%, but only 1.7% for G1_0.6_70 versus FF. The relative contrasts for the VM migration number for the two cases are -37.6% and -12.8%, respectively. It should be noted that at a VM migration number relative contrast of about -12%, the energy overhead of G1 is below 2%.

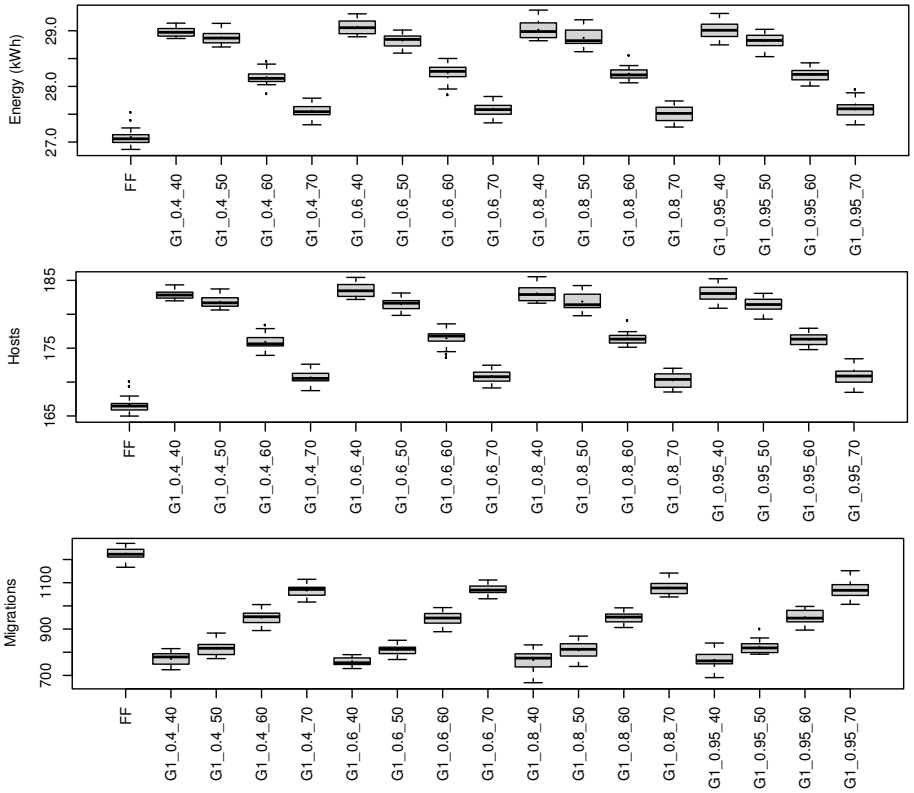


FIGURE 2. Boxplot representation [31] of the energy consumption, the mean number of active hosts, and the VM migration number for FF and G1_a_μ policies with different parameters, in data centers with 300 hosts and 600 VMs. Simulation conditions: 40 min long VM CPU traces, 20 simulation experiments.

5. Conclusions

This paper investigated in more detail a previously defined Gaussian-type resource allocation policy for virtualized data centers, named G1. The policy was compared with the First Fit heuristic by simulation, from the point of view of energy consumption, mean number of used active hosts, and number of VMs migrations performed. Simulation experiments with time-varying workloads were performed for different dimensions of the virtualized data center. The G1 policy scales with the dimension of the data center, showing the same trend of reducing significantly the number of VMs migrations, with a moderate

TABLE 2. The energy consumption, the mean number of active hosts, and the VM migration number relative contrasts (in percents), calculated from the mean values on 20 simulation experiments, for the indicated G1_a_μ policies with respect to the First Fit policy, in a data center with 300 hosts and 600 VMs

Policy	θ^E_{rel}	θ^H_{rel}	θ^{NM}_{rel}
G1_0.4_40	7.0	9.7	-37.0
G1_0.4_50	6.3	8.8	-33.5
G1_0.4_60	4.1	5.7	-23.5
G1_0.4_70	1.8	2.5	-13.5
G1_0.6_40	7.1	9.9	-37.6
G1_0.6_50	6.4	9.0	-34.1
G1_0.6_60	4.0	5.6	-22.5
G1_0.6_70	1.7	2.4	-12.8
G1_0.8_40	6.9	9.6	-37.6
G1_0.8_50	6.4	9.0	-34.5
G1_0.8_60	4.1	5.7	-22.3
G1_0.8_70	1.7	2.3	-13.0
G1_0.95_40	6.8	9.5	-37.3
G1_0.95_50	6.3	8.7	-33.6
G1_0.95_60	4.1	5.7	-22.0
G1_0.95_70	1.5	2.1	-11.9

energy consumption overhead. In the resource allocation process, the G1 policy maximizes a Gaussian score function depending on three parameters, two thresholds (or the mean of these thresholds and the distance between them) and an area parameter. Simulations with different parametrizations of the score function showed that the mean value of the thresholds affects the values of the metrics (energy, number of hosts, number of VMs migrations) significantly, while the area parameter has a negligible effect.

REFERENCES

[1] Gerald J Popek and Robert P Goldberg. Formal requirements for virtualizable third generation architectures. *Communications of the ACM*, 17(7):412–421, 1974.

[2] Laura Grit, David Irwin, Aydan Yumerefendi, and Jeff Chase. Virtual machine hosting for networked clusters: building the foundations for "autonomic" orchestration. In *Proceedings of the First International Workshop on Virtualization Technology in Distributed Computing (VTDC)*, pp. 7, 2006.

[3] Michael Cardosa, Madhukar R Korupolu, and Aameek Singh. Shares and utilities based power consolidation in virtualized server environments. In *Proceedings of the 11th*

IFIP/IEEE International Conference on Symposium on Integrated Network Management (IM), pp. 327–334, 2009.

[4] Mark Stillwell, Frédéric Vivien, and Henri Casanova. Virtual machine resource allocation for service hosting on heterogeneous distributed platforms. In *Proceedings of the 2012 IEEE 26th International Parallel & Distributed Processing Symposium (IPDPS)*, pp. 786–797, 2012.

[5] Anton Beloglazov, Jemal Abawajy, and Rajkumar Buyya. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future Generation Computer Systems*, 28(5):755–768, 2012.

[6] Anton Beloglazov, Rajkumar Buyya, Young C Lee, and Albert Zomaya. A taxonomy and survey of energy-efficient data centers and cloud computing systems. *Advances in Computers*, 82:47–111, 2011.

[7] Anton Beloglazov and Rajkumar Buyya. Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in Cloud data centers. *Concurrency and Computation: Practice & Experience*, 24(13):1397–1420, 2012.

[8] William Voorsluys, James Broberg, Srikumar Venugopal, and Rajkumar Buyya. Cost of virtual machine live migration in Clouds: a performance evaluation. In *Proceedings of the 1st International Conference on Cloud Computing (CloudCom)*, pp. 254–265, 2009.

[9] Anja Strunk and Waltenegus Dargie. Does live migration of virtual machines cost energy?. In *Proceedings of the 27th IEEE International Conference on Advanced Information Networking and Applications (AINA)*, pp. 514–521, 2013.

[10] Jinho Hwang, Sai Zeng, Frederick y Wu, and Timothy Wood. A component-based performance comparison of four hypervisors. In *Proceedings of the 2013 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pp. 269–276, 2013.

[11] Minxian Xu, Wenhong Tian, and Rajkumar Buyya. A survey on load balancing algorithms for virtual machines placement in cloud computing. *Concurrency and Computation: Practice & Experience*, 29(12):e4123, 2017.

[12] Kyong H Kim, Anton Beloglazov, and Rajkumar Buyya. Power-aware provisioning of virtual machines for real-time Cloud services. *Concurrency and Computation: Practice & Experience*, 23(13):1491–1505, 2011.

[13] Cora Crăciun and Ioan Salomie. Gaussian-type resource allocation policies for virtualized data centers. *Studia Universitatis Babeş-Bolyai, Informatica*, LXI(2):94–109, 2016.

[14] Cora Crăciun and Ioan Salomie. A filter-based dynamic resource management framework for virtualized data centers. *Studia Universitatis Babeş-Bolyai, Informatica*, LXII(1):32–48, 2017.

[15] Haizea. `http://haizea.cs.uchicago.edu/`.

[16] Cora Crăciun and Ioan Salomie. Bayesian analysis of resource allocation policies in data centers in terms of virtual machine migrations. In *Proceedings of 2017 13th IEEE International Conference on Intelligent Computer Communication and Processing (ICCP)*, pp. 511–518, 2017.

[17] Borja Sotomayor, Kate Keahey, and Ian Foster. Combining batch execution and leasing using virtual machines. In *Proceedings of the 17th International Symposium on High Performance Distributed Computing (HPDC)*, pp. 87–96, 2008.

[18] Borja Sotomayor, Rubén S Montero, Ignacio M Llorente, and Ian Foster. An open source solution for virtual infrastructure management in private and hybrid clouds. *IEEE Internet Computing, Special Issue on Cloud Computing*, 2009.

[19] Borja Sotomayor, Rubén S Montero, Ignacio M Llorente, and Ian Foster. Resource leasing and the art of suspending virtual machines. In *Proceedings of the 11th IEEE International Conference on High Performance Computing and Communications (HPCC)*, pp. 59–68, 2009.

[20] Borja Sotomayor Basilio. Provisioning computational resources using virtual machines and leases. PhD Dissertation, University of Chicago, Illinois, USA, 2010.

[21] Akshat Verma, Puneet Ahuja, and Anindya Neogi. pMapper: power and migration cost aware application placement in virtualized systems. In *Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware (Middleware)*, pp. 243–264, 2008.

[22] Tiago C Ferreto, Marco AS Netto, Rodrigo N Calheiros, and César AF De Rose. Server consolidation with migration control for virtualized data centers. *Future Generation Computer Systems*, 27:1027–1034, 2011.

[23] Timothy Wood, Prashant Shenoy, Arun Venkataramani, and Mazin Yousif. Sandpiper: Black-box and gray-box resource management for virtual machines. *Computer Networks*, 53(17):2923–2938, 2009.

[24] Rodrigo N Calheiros, Rajiv Ranjan, Anton Beloglazov, César AF De Rose, and Rajkumar Buyya. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Journal of Software: Practice and Experience*, 41(1):23–50, 2011.

[25] CloudSim. `http://www.cloudbus.org/cloudsim/`.

[26] Fabien Hermenier, Julia Lawall, and Gilles Muller. BtrPlace: A flexible consolidation manager for highly available applications. *IEEE Transactions on Dependable and Secure Computing*, 10(5):273–286, 2013.

[27] Carlo Mastroianni, Michela Meo, and Giuseppe Papuzzo. Probabilistic consolidation of virtual machines in self-organizing cloud data centers. *IEEE Transactions on Cloud Computing*, 1(2):215–228, 2013.

[28] Carlo Mastroianni, Michela Meo, and Giuseppe Papuzzo. Self-economy in cloud data centers: Statistical assignment and migration of virtual machines. In *Proceedings of the 17th International Conference on Parallel Processing (Euro-Par)*, pp. 407–418, 2011.

[29] Adnan Ashraf and Ivan Porres. Multi-objective dynamic virtual machine consolidation in the cloud using ant colony system. *International Journal of Parallel, Emergent and Distributed Systems*, 33(1):103–120, 2018.

[30] Xiaobo Fan, Wolf-Dietrich Weber, and Luiz A Barroso. Power provisioning for a warehouse-sized computer. In *Proceedings of the 34th annual International Symposium on Computer architecture (ISCA)*, pp. 13–23, 2007.

[31] The R Project for Statistical Computing. `https://www.r-project.org/`.

Faculty of Physics, Babeş-Bolyai University, Cluj-Napoca, Romania
Email address: `cora.craciun@ubbcluj.ro`

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LXX**, Number 1-2, 2025
DOI: 10.24193/subbi.2025.06

MONEY LAUNDERING DETECTION USING GRAPH NEURAL NETWORKS ENHANCED WITH AUTOENCODER COMPONENTS

TUDOR-IONUŢ GRAMA

Abstract. The paper addresses the topic of detecting money laundering operations in transaction data represented as graph data-structures. We propose the integration of autoencoder components in Graph Neural Networks(GNN) architectures, in order to incorporate a reconstruction step in the traditional edge classification problem and enhance model quality based upon the usage of reconstruction errors.

We show that enhancing GNNs with autoencoder components improves the predictive performance of money laundering detection, on data represented as homogeneous graphs. Additionally, the Shapley value is computed in order to gain further insight into the most important features from distinguishing normal and fraudulent activities.

1. Introduction

With the latest technological achievements, there has also been a significant increase in the complexity of fraudulent financial activities, as well as in the ability of criminal parties to render such efforts much more difficult to trace. *Money Laundering* is the process of making illicit funds hard or impossible to distinguish from those acquired by legal means, and causes damages in the billions, potentially even trillions [4]. Identifying such processes as accurately and early as possible is paramount for limiting both the resulting damages, as well as for putting a stop to the ability of organized crime to be able to acquire finances. Money Laundering is a complex process of obfuscating the origin of illicitly obtained funds by employing sequences of transactions or bank transfers, that make tracing the source of the money very difficult and

Received by the editors: 4 June 2025.

2010 *Mathematics Subject Classification.* 68T05, 68T99.

1998 *CR Categories and Descriptors.* I.2.6 [**Artificial Intelligence**]: Learning – *Connectionism and neural nets*; E.1 [**Data**]: Data structures – *Graphs and networks*.

Key words and phrases. Money laundering, Graph neural networks, Autoencoders, Explainability.

time-consuming. Traditional approaches for money laundering detection are typically rule-based systems [22], which can be prone to a high degree of false positives [16]. Recently, machine learning and deep learning driven approaches have seen significant development [9, 1]. Of particular note are developments in *graph neural networks* (GNNs), which manage to effectively depict complex networks of financial transactions [10], due to their ability of handling relational data, represented as graphs, which are a suitable representation for financial transactions.

This work showcases GNNs with directed multigraphs, as well as various other adaptations which are either necessary for accurately depicting financial transactions or enhance predictive performance. Financial operations can encompass multiple transactions which can occur between the same entities over time, often time in both directions. The multigraph structure thus more dependably preserves the temporal and directional aspects of transactions and can serve as a better baseline for future developments. The main goal is to develop a robust manner of detecting instances of money laundering, that, at the same time is agnostic with respect to the transaction method, as well as being generalizable to a large deal of auxiliary factors, such as currency, financial institutions, or whether the transactions go beyond national boundaries. This should serve to ensure robustness for the complexities specific to the domain in real-life scenarios and improve the detection of such illicit operations, no matter the mechanisms involved.

To the best of our knowledge, there are no other studies in the literature that discuss the enhancement of GNN architectures with autoencoder components for the purpose of detecting money laundering. Our contribution consists of using autoencoder components in order to reconstruct edge attributes based on the embeddings of the nodes defining a transaction. The reconstruction error is used as part of the training loop being integrated in a binary cross entropy function which uses the edge attributes as well as the mean square error pertaining to the edge attribute reconstruction for the classification step. The output of the autoencoder is used to better differentiate transactions involved in money laundering that are not part of known or well defined patterns. Second, we integrate explainability and feature analysis using `SHapley Additive exPlanations` (SHAP) [14], in order to better understand what structural and substantive features of financial data are most telling of whether a transaction is normal or fraudulent. To conclude, three main research questions will be further investigated:

RQ1 Does integrating autoencoder components into the edge classification problem improve the predictive performance of money laundering detection?

RQ2 What is the impact of integrating autoencoder components in a GNN-based architecture for the task of detecting illicit transactions in directed heterogeneous multigraphs?

RQ3 What are the most significant features for detecting each type of transaction as provided by the SHAP explainability approach?

The rest of the paper has the following structure. Section 2 reviews existing machine learning approaches for money laundering detection. Section 3 details our methodology used in the study. The experimental results are presented and discussed in Section 4. Section 5 presents the application of SHAP values to provide interpretability for transaction classification, while the conclusions and some potential directions for future work are outlined in Section 6.

2. Literature review on machine learning approaches for money laundering detection

Weber et al. [23] employ GNNs for anti money laundering in bitcoin transactions. The researchers implement an architecture using two-layered graph convolutional networks. The authors name this Skip-GCN, incorporating a skip connection between intermediate embeddings and input features, and also experiment with EvolveGCN, a temporal model that creates separate GCN (Graph Convolutional Network) layers for each time step connected through a recurrent neural network. They show that the graph-based approaches outperform traditional methods.

GNN-based approaches have also been employed on real-life data. Johannessen and Jullum use a GNN architecture in order to detect money laundering in a network of transaction from the largest Norwegian bank. One variant explored by the researchers include a novel approach, by concatenating the node embeddings from the aggregation of messages and passing the resulting vector through a single-layer perceptron, which outputs a new representation. The model achieves impressive results on the detection of top 1% most suspicious customers, with two thirds of the ones identified by the model being actually suspicious. This is noteworthy especially due to the severe class imbalance, under 0.5% of the customers being considered suspicious [10].

Egressy et al. [5] introduce key adaptations to standard GNNs in order to render them a better fit for graphs of transactions. They implement port numbering (unique labels for each node), EgoIDs (boolean marker which allows a node to detect whether it is part of a cycle) and reverse message passing, as well as edge updates, for some of the model variants. In their experiments, reverse message passing provides the largest single performance boost for most tasks, Port numbering is shown to be essential in detecting two patterns that are common in fraudulent financial transactions. EgoIDs, coupled with the

other two adaptations are stated to improve the detection of cycle patterns, which are likewise common in fraudulent activities, up to a depth of 6.

There are other, non-GNN based approaches that have also shown promising results. Kumar et al have tested various machine learning models on data of bank loans. Support vector machines shows the best precision score, obtaining 100%, while also achieving a high accuracy score and a high f1 score, of 83% and 90% respectively. In the same study, random forest has also been shown to display competitive performance [12].

3. Methodology

This section introduces the methodology used in our study, in order to implement our proposed architectures. We describe the dataset and the pre-processing steps necessary for the data, as well as the methods used towards the implementation of the used models and the performance evaluation.

3.1. **Machine learning models used.** *Autoencoders*(AEs) are a class of neural networks that correspond to the category of self-supervised learning machine learning models which learn to reconstruct the given input data. An AE is composed of two parts which are connected by two layers. The first segment is an encoder which learns to compress the data to a latent space. If we consider the input space to have n dimensions and the desired size to be m dimensions, then we can formalize the process as such: $f : \mathbb{R}^n \to \mathbb{R}^m$. The decoder learns the reverse process $g : \mathbb{R}^m \to \mathbb{R}^n$, which teaches the model how to reconstruct the input data based on lower dimensional latent representations.

GNNs are models that are specifically adapted in order to handle data structured in a graph like format, of nodes connected by edges [26]. The neural networks typically handle particular forms of data depending on the graph structure. We can consider graphs to be either directed or undirected, depending on the presence or absence of direction at edge level, respectively homogeneous or heterogeneous. In homogeneous graphs, all the edges and nodes have the same types, where in heterogeneous graphs, multiple types can exist simultaneously.

The most common approach for developing GNNs is through the Message Passing Neural Network Framework (MPNN), first defined by Gilmer et al. [6]. Developments of GNN architectures that include autoencoder components are somewhat rare. Nevertheless, there are some notable research results. Kipf and Welling present two different approaches, one being a graph autoencoder (GAE) and the other a variational adaptation of the previous algorithm [11]. An innovative approach is proposed by Tang, namely Graph Auto-Encoder Via Neighborhood Wasserstein Reconstruction [20]. The researchers identify a shortcoming of normal autoencoders, that being their tendency to focus

on direct edges between nodes, while losing otherwise relevant topological information of the graph.

3.2. **Overview.** We aim to represent financial transactions in a graph structure, with a node being a particular account and the edges being transactions between said accounts. As there can exist multiple transactions between the same two accounts, the underlying dataset is represented as a directed multigraph. The data has been produced by the AMLWorld model from IBM [2]. The data generation is done using a multi-agent based approach, in which there are both agents that are simulating normal citizen behavior, as well as a minority of criminals, which generate the fraudulent transactions.

In the literature, multiple typical patterns have been observed. These patterns were first introduced by Suzumura and Kanezashi [19], and can be described as follows. A `fan-out` pattern (further denoted as fan_out) represents a fraudulent node being connected by outgoing edges to other non-fraudulent nodes. A `fan-in` pattern (further denoted as fan_in) is similar, but in this case the fraudulent node has incoming directed edges from normal nodes.

A message is created by every node based on the internal features of the node, the features of the edge caring the message, as well as the features corresponding to the connected nodes. Nodes aggregate incoming messages from their neighbors, using one aggregator, or multiple in conjunction. Afterwards each node updates its internal state, aggregating its initial state as well as the states of its neighbors.

The way we convert a traditional graph structure into a multigraph is using the approach defined by Egressy et al. [5], meaning we combine the use of port numbering, reverse message passing and EgoIDs. Port numbering consists in adding a label to the edges between nodes, the label being a tuple containing a number uniquely identifying the start node (the node sending the messages) and the end node (the node receiving the messages). Reverse message passing is implemented by using a separate layer for message-passing, for the incoming edges and another layer for outcoming edges. This makes nodes in MPNN able to differentiate incoming and outgoing messages. Finally, EgoIDs are used in order to ease the detection of cycles in graphs.

3.3. **Proposed architectures.** We propose two different architectures, one tailored for simpler, homogeneous graph data, as well as one with multiple adaptations and capability of handling heterogeneous data.

For the *first model*, we begin with a feature embedding layer, which projects the node and edge features into the latent space, in order to ensure the same dimensions for both set of features. The second part of the architecture is what we call the GNN Encoder. As in the base architecture of Graph Isomorphism

Network (GIN), this component is tasked with updating node embeddings using the information gathered from the local neighborhoods. We use the GINE variant of GIN (GIN with Edge features), which is an adaptation of the latter, capable of integrating edge features and has shown strong ability in differentiating graph structures. We use two layers of GINE in this component, in order to make the aggregation of information able to occur using data from neighboring nodes separated by up to two edges. The third component is a decoder that aims to reconstruct edge attributes from the node embeddings. The idea behind this approach is that the nodes would be more inclined to preserve and encode features that are more important for defining the edges in their close proximity. Second, it allows us to identify edges that display anomalous behavior, which is of interest, as we presume that fraudulent edges have some, albeit variable, specifics that ultimately make their representations deviate from expected patterns. The last component is a classifier, that combines the node embeddings, edge features and the reconstruction error for classification of transactions. The structure of the classifier is a three-layered Multi-Layer Perceptron (MLP), with two dropout layers with a considerable final dropout rate of 29%.

Below we showcase the core training loop of the model, as conducted for the results detailed in Table 3, without any other adaptations, such as those defined in Egressy et al. [5]:

The *second* architecture is similar to the first in many aspects, but there are noteworthy differences, which we will focus on further below. The feature embedding layer is enhanced in order to be able to handle both homogeneous and heterogeneous graph structures, as according to the approach in Egressy et al. [5]. The GNN Encoder has two other adaptations, besides the use of PNAConv (Principal Neighborhood Aggregation Convolution) instead of GINEConv, namely, compatibility with reverse message passing, by aggregating information both from incoming and outgoing messages, as well as optional edge updates. Otherwise, we also use two layers of convolutions, in order to keep the same depth when aggregating node features. The decoder structure is slightly altered, in the sense that the architecture is now symmetric, with two processing layers. The classifier respects the original structure. Below we showcase the training loop of the PNA-based, autoencoder-enhanced GNN. For the purposes of the results shown in the Table 3, we have implemented all the adaptations described in Egressy et al. [5], as well as the edge updates mechanism. The baseline architecture is called Multi-PNA, which refers to a PNA architecture enhanced with EgoIDs, reverse message passing and port numbering.

Input: $G = (X, E, A, Y)$ - graph with node features, edges, edge features, and labels; T - number of epochs; L - number of GNN layers
for *epoch* $= 1$ ***to*** T **do**
  for *batch* (X_b, E_b, A_b, Y_b) ***in*** G **do**
    $A_{orig} \leftarrow A_b$.clone() `// original edge attributes`
    $H \leftarrow \text{NodeEmbed}(X_b)$ `// node embedding`
    $A_{emb} \leftarrow \text{EdgeEmbed}(A_b)$ `// edge embedding`
    `// GIN Encoder with residual connections`
    for *layer* $i = 1$ ***to*** L **do**
      $H \leftarrow (H + \text{ReLU}(\text{BN}_i(\text{GINEConv}_i(H, E_b, A_{emb}))))/2$ `// batch normalized computations`
    end
    $Z \leftarrow \text{Linear}_{enc}(H)$ `// project to latent space`
    $Z_{src}, Z_{dst} \leftarrow Z[E_b[0]], Z[E_b[1]]$ `// source/dest embeddings`
    `// decoding edge attributes`
    $\hat{A} \leftarrow \text{DecoderMLP}([Z_{src} \| Z_{dst}])$
    `// Per-edge reconstruction error`
    $\epsilon \leftarrow \text{mean}((\hat{A} - A_{orig})^2, \text{dim} = 1)$
    `// features augmented with reconstruction error`
    $F \leftarrow [Z_{src} \| Z_{dst} \| A_{orig} \| \epsilon]$
    $\hat{Y} \leftarrow \text{Classifier}(F)$
    $\mathcal{L} \leftarrow \text{CrossEntropy}(\hat{Y}, Y_b)$
    Backpropagate $\mathcal{L}$ and update model parameters
  end
end

Algorithm 1: GIN-based GNN with autoencoder enhancement

3.4. **Training and testing.** The models were trained on the training portion of the `Small HI` dataset, more precisely, the first 60% of the timestamps, validated on the next 20%, and tested on the final 20%. The segmentation was done based on the temporal ordering of the transactions. For all cases, the training has been conducted using a batch size of 8192, over 100 epochs.

As mentioned in Section 3.3, both models use two GNNs layers, we further employ [100, 100] neighbor sampling for both created models, in order to keep our approach as similar as the one of Egressy et al. [5] Both models utilize a weighted binary cross-entropy loss, with a weight of 1 for the normal transactions and 7.1 for the laundering transactions, for the GNN component. We have used the Adam optimizer and set learning rate to 0.0006 for both the GIN based architecture as well as the PNA based architecture. We have opted for a lower learning rate in order to mitigate the risk of the models overfitting on the patterns identified in the training data, which would be exacerbated by the use of reconstruction error. In addition, we use hidden sizes of 64 for the GIN based architecture, and 20 for the PNA based one, as well as latent dimensions of 32 and 12, respectively. For both architectures, the decoders also use a dropout rate of 0.08.

Input: $G = (X, E, A, Y)$ - graph with node features, edges, edge attributes, labels; T - number of epochs; L - number of GNN layers

for *epoch* $= 1$ ***to*** T **do**
  for *batch* (X_b, E_b, A_b, Y_b) ***in*** G **do**
    $E_{fwd} \leftarrow E_b$ `// forward edges`
    $E_{rev} \leftarrow E_b$.flip(0) `// reverse edges`
    $A_{orig} \leftarrow A_b$.clone() `// original attributes`
    $H \leftarrow \text{NodeEmbed}(X_b)$ `// node embeddings`
    $A_{fwd} \leftarrow \text{EdgeEmbed}(A_b)$ `// forward edge embeddings`
    $A_{rev} \leftarrow \text{EdgeEmbed}(A_b)$ `// reverse edge embeddings`
    for *layer* $i = 1$ ***to*** L **do**
      $H_{fwd} \leftarrow \text{ReLU}(\text{BN}_i^{fwd}(\text{PNA}_i^{fwd}(H, E_{fwd}, A_{fwd})))$ `// batch normalized computations`
      $H_{rev} \leftarrow \text{ReLU}(\text{BN}_i^{rev}(\text{PNA}_i^{rev}(H, E_{rev}, A_{rev})))$
      $H \leftarrow (H + H_{fwd} + H_{rev})/3$ `// residual aggregation`
      $A_{fwd} \leftarrow A_{fwd} + \text{EdgeMLP}_i([H[E_b[0]] \| H[E_b[1]] \| A_{fwd}])/2$
      $A_{rev} \leftarrow A_{rev} + \text{EdgeMLP}_i([H[E_{rev}[0]] \| H[E_{rev}[1]] \| A_{rev}])/2$
    end
    `// encoding to latent space`
    $Z \leftarrow \text{Linear}_{enc}(H)$
    $Z_{src}, Z_{dst} \leftarrow Z[E_b[0]], Z[E_b[1]]$
    `// decoding edge attributes`
    $D \leftarrow \text{ReLU}(\text{Linear}_{expand}([Z_{src} \| Z_{dst}]))$
    for *layer* $j = 1$ ***to*** L **do**
      $D \leftarrow \text{Dropout}(\text{BN}_j(\text{ReLU}(\text{Linear}_j(D))))$
    end
    $\hat{A} \leftarrow \text{Linear}_{final}(D)$ `// reconstructed edge attributes`
    `// Reconstruction error as feature`
    $\epsilon \leftarrow \text{mean}((\hat{A} - A_{orig})^2, \text{dim} = 1)$ `// per-edge MSE`
    $F \leftarrow [Z_{src} \| Z_{dst} \| A_{orig} \| \epsilon]$
    $\hat{Y} \leftarrow \text{Classifier}(F)$ `// features augmented with reconstruction error`
    $\mathcal{L} \leftarrow \text{CrossEntropy}(\hat{Y}, Y_b)$
    Backpropagate $\mathcal{L}$ and update parameters
  end
end

Algorithm 2: Multi-PNA Autoencoder for Anti-Money Laundering Detection

The AE components, in both cases, provide a reconstruction error calculated using mean square error. The edge attributes are reconstructed based upon the node embeddings. The reconstruction error is calculated per feature and then it is averaged across features. For the Multi-PNA architecture, we also take into consideration the node embeddings coming from the reverse message passing layers when conducting reconstruction. In the end, besides the node and edge attributes, we also use the reconstruction error as part of the classification step

In order to assess model performance, we measure the `F1`-score of the minority class at the end of the last training epoch.

4. Experimental evaluation

This section presents the experiments enacted in order to answer the aforementioned research questions, namely whether autoencoder components are beneficial for detecting fraudulent transactions in GNN architectures and if these effects remain valid for a heterogeneous graph. The experimental evaluation has been carried out using the methodology presented in Section 3.

We mention that part of the baseline models from the Egressy et al. [5] paper have been tested during this study, in order to validate their performance and determine where the defined approach is valid and a potentially worthwhile starting point of further study.

4.1. **Dataset.** We have chosen the `Small HI` (Higher Illicit) transaction subset of the dataset. We consider the Higher Illicit transaction set to be more suitable in comparison to the Lower Illicit set. While this subset is less representative of real-life scenarios, the ratio of transactions involved in laundering being just slightly over 0.1, we consider that less illicit transactions would considerably hinder the capabilities of the model to learn the patterns associated with the minority class. This subset of data contains information for a primary period of 10 simulated days, with around 5 million transactions, realized by 515 thousand bank accounts.

Out of all the transactions in the main period, only approximately 3600 are illicit. Table 1 presents the statistics for the small cohorts of the data, both higher and lower illicit sets. The dataset includes a secondary period of 8 days, succeeding the primary period, with a much smaller number of accounts and transactions.

Statistics	Higher Illicit	Lower Illicit
Number of Days Spanned	10 + 8	10 +7
Number of Bank Accounts	515K +716	705K + 201k
Number of Transactions	5M + 1.1K	7M +283K
Number of Laundering Transactions	3.6K +655	4.0K + 435
Laundering Rate (1 per N Trans)	981	1942

Table 1. Statistics for the 'Small' subset of the dataset.

The initial features of the dataset are explained in the Table 2

When the data is processed for training and inference, the timestamp column, payment currency, receiving currency and payment format columns are encoded using label encoding. The columns 'to bank' and 'to account' are

Feature	Description
Timestamp	Timestamp of the transaction in YYYY/MM/DD HH:MM Format
From Bank	Numeric ID of the bank the transaction is sent from
Account	Hexadecimal ID of the account on the sending end of a transaction (The initial feature name is identical to the other Account name, later gets changed to 'From Account')
To Bank	Numeric ID of the bank the transaction is sent to
Account	Hexadecimal ID of the account on the receiving end of a transaction (Later changed to 'To Account')
Amount Received	Monetary amount received by 'To account'
Receiving Currency	Currency of the amount received by 'To Account'
Amount Paid	Monetary amount send to the 'To account'
Payment Currency	Currency of the amount sent by 'From Account'
Payment Format	How the transaction was conducted (Wire transfer, cheque)
Is Laundering	Boolean value of whether the transaction is money laundering or not

TABLE 2. Feature explanation

turned into a singular column, by appending the value of the latter to the former. This is similarly done for the 'from bank' and 'from account' columns. Finally, an EdgeID is created, which is an integer value that uniquely identifies every transaction.

The data that is inputted into the model, in the case of the two architectures, is identical, both in the case of the models which implement directed homogeneous multigraphs and those which implement directed heterogeneous multigraphs. The features used are the following:'Timestamp', 'Amount Received', 'Received Currency', 'Payment Format'. The only changes between heterogeneous and homogeneous multigraphs are those pertaining to the edge index, which needs to be flipped when handled by the reverse message passing layer, as well as the port numbers, if port numbering is enabled. The main differentiator between the layers is which messages are handled by which layer, namely homogeneous implementations only use the data from incoming messages, the edge features of incoming edges, in order to update a node's internal representation, while in heterogeneous graphs, the outgoing messages are also used, together with the features of the outgoing edges and handled by the reverse message passing layer. This implementation is used in order to remain consistent with the implementations of Egressy et. al.[5]. For example, given the following example transaction:

2019/01/01 00:22,800319940,8004ED620,808519790,872ABC810,120.92,US Dollar,120.92,US Dollar,Credit Card,0

The final features kept would be the Timestamp (2019/01/01 00:22), Amount Received (120.92),Received Currency (US Dollar) and the Payment Format (Credit Card). The Is_Laundering (0) value is used as the target variables.

If port numberings is enabled, unique port numbers are added as edge features. We shall consider the starting node to be 800319940-8004ED620, encoded as '0' (zero), and the receiving node to be 808519790-872ABC810, encoded as '1'. Therefore, the port numbering for this edge would be the tuple (0, 1). The edge index is likewise represented as a tuple, respecting the encoded format (0, 1). For node '1', in an homogeneous graph implementation, the edge features and internal node representation for node '0' would be used, as '1' is a receiving node and the transaction (0, 1) represents an incoming transaction for node '1'. Node '0' would not use this transaction to update its internal state, as it represents the sending node, of this transaction.

When we activate reverse message passing, the original message passing layer would keep the behavior of a directed homogeneous multigraph. The reverse message passing layer needs to reverse the edge index and port numbering tuple, thus becoming (1, 0). In the case of the reverse message passing layer, node '1' acts as the sender, and '0' as the receiver. By enabling reverse message passing, now both nodes are able to use all the receiving and outgoing transactions and messages, in order to update their internal state. The resulting structure represents the directed heterogeneous multigraphs.

4.2. **Results and discussion.** Table 3 comparatively presents the performance of baselines and our proposed architectures. The most extensive results in the literature, specifically on the used dataset, using part of the adaptations discussed are presented in the work of Egressy et al. [5]. We consider these models as baselines for our architectures, which are written as GIN+AE (GIN+ AutoEncoder) and Multi-PNA+EU (Edge Updates)+AE respectively. The experiments using the models GIN + ReverseMP + Ports, R-GCN and all the experiments prefixed with 'Multi-' handle the dataset as a directed heterogeneous multigraph, the others as a directed homogeneous multigraph. The tree-based model baseline use the dataset augmented with graph features. As Anti-Money Laundering systems are prone to false positive rates approaching 90%[15], due to the resulting negative effect on precision, we can assess that F1-scores are likely to remain reduced. To our knowledge, Multi-PNA+EU+AE represents the state of the art for money laundering detection on directed heterogeneous multigraphs and GIN+EgoIDs represents the state of the art on directed homogeneous multigraphs.

From Table 3, which consists of the average across five runs with different seeds, plus-minus the standard deviation, in case of the baselines, we observe that the architectures that exhibit all three of the adaptations mentioned

TABLE 3. Performance of baselines and proposed architectures (F1-score %).

Model	AML Small HI
LightGBM+GFs [2]	62.86 ± 0.25
XGBoost+GFs [2]	63.23 ± 0.17
GIN [24, 7])	28.70 ± 1.13
GIN+AE	34.98
PNA [21]	56.77 ± 2.41
GIN+EU [3]	47.73 ± 7.56
R-GCN [18]	41.78 ± 0.48
GIN+EgoIDs [25]	39.65 ± 4.73
GIN+Ports [17]	54.85 ± 0.89
GIN+ReverseMP [8] +Ports	46.79 ± 4.97
GIN+Ports	56.85 ± 2.64
+EgoIDs (Multi-GIN)	57.15 ± 4.99
Multi-GIN+EU	64.79 ± 1.22
Multi-PNA	64.59 ± 3.60
Multi-PNA+EU	68.16 ± 2.65
Multi-PNA+EU+AE	50.92

Egressy et al. [5], namely the EgoIDs, port numbering and Reverse Message Passing display the best performance. For the graph isomorphism network, port numbering shows the biggest performance impact, with an increase from the minority class `F1` score from 28.70 ± 1.13 to 54.85 ± 0.89. The second biggest impact comes from Edge updates, followed by Reverse Message Passing and EgoIDs, with the lowest impact. It is shown that combining all these adaptations render the comparably simpler GIN architecture to be very close in performance with the Multi-PNA Model.

Adding an autoencoder component on the normal GIN architecture has a positive impact on performance, with an increase of the evaluation metric to 34.98%, which is greater than the values exhibited at the tall end of a confidence interval given by the researchers. This tells us that autoencoder components can be beneficial in directed, homogeneous graphs.

However, in the case of heterogeneous graphs, we have observed a large drop in performance when compared to the equivalent baseline implementation (Multi-PNA+EU), obtaining a minority class `F1` of 50.92% We theorize that this might be due to the increased complexity of the resulting node embeddings, coupled with the higher volume and diversity of edges and their corresponding features, which might necessitate special adaptations in the reconstruction process, for stable performance. One approach that might result in potential improvements is to consider the inbound and outbound messages as two separate homogeneous graphs, strictly for the purpose of calculating reconstruction errors. Then, we calculate the reconstruction error separately

for each instance and add them both in the final classifier. This could lead to improved performance if the boost associated with the autoencoder component remains consistent across multiple homogeneous graphs. For the purpose of testing of our implementations, due to time and computational constraints, we had ran the experiments only once.

Therefore, the increase in performance exhibited in the case of the first model, when compared to the baseline renders, corroborated with the lower performance exhibited by the second model renders us able to answer **RQ1**: the performance improves by adding autoencoder components in the case of homogeneous graphs, and decays in the case of heterogeneous graphs. As an answer to **RQ2**, we may may that the performance decays when integrating autoencoder components, when compared to the baseline, at least in the case of our architecture. Nevertheless, further research and tests are required for definitive validation. We note that there are no other graph autoencoder architectures in the literature that deal with the problem of money laundering detection as an edge labeling task

5. Explainability

For answering **RQ3** we further investigate the application of SHAP values for providing interpretability for transaction classification.

SHAP values are based upon shapley values and provide a standardized approach for estimating the effect and the associated magnitude of the input features on the model output. The SHAP values are computed using explainer models and can be used in order to infer feature importance for all predictions as a whole, as well as individually. In the first case, they can provide an aggregate overview of the feature importance for the global predictive behavior of the model, more precisely showing the effect of the feature on the given predictions. This depiction can be rendered even more informative, with the use of beeswarm plots, which show how values of the feature influence the importance of the feature in prediction, or scatter plots, which can model interaction effects between feature values and their associated SHAP values. Second, SHAP values can be used to infer feature importance for individual predictions made by the model. This is relevant for explaining how the model reaches a result for a particular observation, and, likewise, we can use either waterfall plots or force plots for this end.

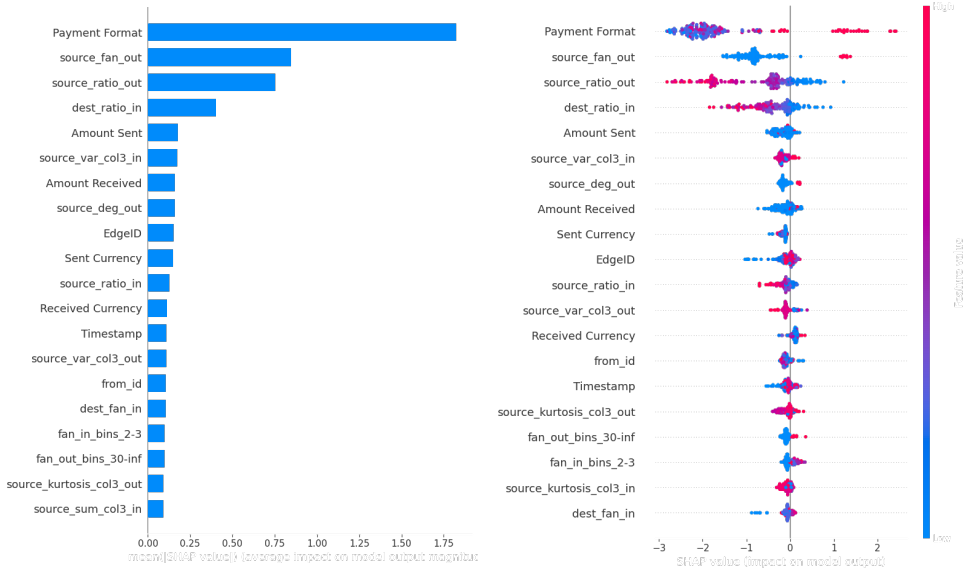
Due to the complexity of computing SHAP values for graph structure data, particularly for edges, we will first process the data in order to extract meaningful structural features, which will render the data more usable by more traditional classifiers. To this end, we will employ the Graph Feature Preprocessor module from the SNAPML library. The preprocessor computes two kinds of graph based features, which the researchers label as graph patterns

and graph vertex statistics. The first set of features consists of identifying for each transaction the kind of patterns it is implicated in. The second set of features is represented by features like the number of neighboring edges, both incoming and outgoing as well as the `fan`/`degree` ratio (the `fan` is the number of neighboring nodes and the `degree` is the number of incident edges), as well as `average`, `sum`, `minimum`, `maximum`, `median`, `var`, `skew`, and `kurtosis` of the selected raw features of the aforementioned three features. All the features shown in Figure 1 and Figure 2, written in snake-case represent augmentations computed with SNAPML. The total final number of features is 100.

The feature explanation has been computed by wrapping an XGBoost model using a `TreeExplainer` model from SHAP [13], which is an architecture that is adapted for SHAP values computation for tree-based models. XGBoost is an efficient implementation of gradient boosting machines that has seen widespread adoption in machine learning competitions.

The global and beeswarm feature importances are shown in Figure 1. Looking at the global feature importance (Figure 1(A)), we can see that four features show the greatest influence in the final result: (1) the most important predictor on whether a transaction is fraudulent or not is `Payment Format`; (2) the second feature refers to the number of neighboring nodes of a source node (`source_fan_out`); (3) the third feature represents the ratio between the `fan` and the *degree* for a starting node of the outgoing transactions; (4) the fourth feature is similar, but it instead is computed on the destination node.

We can see that, the number of the neighboring nodes of a source node is a strong predictor for fraudulent transactions. The correlation between different amounts sent and fraud does not seem to be very clearly defined, but low transaction amounts have a small tendency of lowering the risk of a transaction being fraudulent. Another insight we can gain from the plot is by looking at the EdgeID feature. The edges in the dataset are labeled using numeric ids, which are incremented by 1 for every subsequent transaction that is simulated. As the edges with low feature values (a lower ID) are shown to decrease the chance a feature is actually fraudulent, we can stipulate that the researchers have first created the normal transactions, and only afterwards created the bulk of fraudulent transactions. As the transactions with medium-valued EdgeIDs are shown to increase the probability of being a money laundering transaction (in this instance, meaning they are actually fraudulent), we can assume with reasonable certainty that the fraudulent transactions were simulated in a later step of the simulation, and late transactions act as filler, likely necessary for increased veracity. This is further corroborated by the `Timestamp` features, where we can see lower timestamps mostly corresponding to normal transactions.



(A) Global Feature Importance. (B) Beeswarm feature importance.

FIGURE 1. Feature importance.

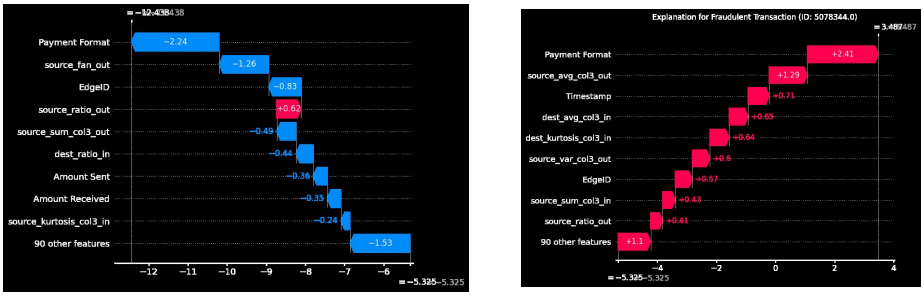
We can also stipulate that the transactions were generated in batches, as even for a small subset of samples (the first two hundred), the bee swarm plot and the influence of feature values remain mostly consistent with the previous case. The beeswarm plot illustrated in Figure 1(B) showcases this behavior.

TABLE 4. Transaction statistics per payment format

Payment	Not Laund.	Laund.	Total	Fraud %
ACH	596,314	4,483	600,797	0.75
Bitcoin	146,035	56	146,091	0.04
Cash	490,783	108	490,891	0.02
Cheque	1,864,007	324	1,864,331	0.02
Credit Card	1,323,118	206	1,323,324	0.02
Reinvestment	481,056	0	481,056	0.00
Wire	171,855	0	171,855	0.00

For a better comprehension of the importance of the features, we consider two sample transactions depicted in Figure 2 (a normal one, depicted in Figure 2(A) and a fraudulent one depicted in Figure 2(B)).

When looking at an individual **normal transaction** (Figure 2(A), we can see that most features make the sample less likely to be fraudulent, in this case, with the exception of the `fan/degree` ratio at the source node.



(A) Normal transaction - 298998. (B) Fraudulent transaction - 5078334

FIGURE 2. Sample transactions.

In order to better understand why these features have this particular impact, for this sample, on the model predictions, we analyzed the distributions of the nine most important features, according to global feature importance. We observed that the sample corresponds to a transaction conducted using reinvestments, that has a small, but above the mean `source_ratio_out`. We also observed that the transaction sent a large amount of money, well beyond the median but far below the mean, and the same holds for the amount received. Of note is the difference between the mean of the two, showing that the average received amount is much higher than the sent amount, signaling that monetary conversions into a weaker currency, than the starting one is particularly common. We can also see that the starting node of the transaction has 7 neighboring nodes, which is above the median, but well below the mean.

For the fraudulent transaction (Figure 2(B)), all the main features push the transaction towards being fraudulent. We observed that five of the nine most important features refer to statistical features of the timestamps (`col3` corresponds to the timestamp column). We note that the transaction was conducted using the ACH (Automated Clearing House) format, which was shown to have the highest percentage of fraudulent transactions 4. The transaction is also shown to have a small ratio of `fan/degree` for its starting node (has comparatively few transactions when compared to the number of implicated accounts.) The `source_fan_out` also shows us that the number of accounts connected to the source account is above the median, but well below the mean. For the end node, we can see the ratio is much closer to the mean and the median. Of note is also that the amount sent is only slightly larger than the median, but much lower than the mean. This suggests that transactions that are not particularly large, but still larger than most, are most likely to be part of fraudulent activity.

6. Conclusions

We have shown in this paper that enhancing GNN architectures with autoencoder components improves the predictive performance, when it comes to the money laundering detection tasks, on data represented as homogeneous graphs. This finding does not hold for heterogeneous graphs, at least when we consider the multi-PNA architecture, where we encountered a decrease in model performance. We theorize that this could indeed be due to the data differences, whose impact could be mitigated by utilizing different autoencoders for each type of data.

We can therefore state that, in regard to RQ1, autoencoder components can enhance the performance for homogeneous graph data. For heterogeneous data the performance decreases, more advanced implementations likely being necessary. This renders us able to answer RQ2, namely that the Multi-PNA architecture does not benefit from autoencoder components as implemented in our iteration. Regarding RQ3, we have shown that payment formats and the ratio of transactions to the participating accounts have a significant predictive capability in determining transactions involved in money laundering.

As the current research does not discuss the implementation of autoencoder components for all the model architectures discussed in the main paper, analyzing the impact of such adaptations to the other model variants will likely lead to valuable research results. Finally, in regards to explainability, developing SHAP adaptations for explainability on edge labeling tasks, directly applicable to GNNs would be a major improvement towards model interpretability. Otherwise, integrating new statistical features of transaction amounts, both for sent and received, as was done for the timestamps would likely also lead to valuable insights.

References

[1] Abal, R., Peña, L., and Soria Quijaite, J. Machine learning models for money laundering detection in financial institutions. a systematic literature review. In *Proceedings of the 22nd LACCEI International Multi-Conference for Engineering, Education, and Technology* (2024), pp. 1–10.

[2] Altman, E. R., Egressy, B., et al. Realistic Synthetic Financial Transactions for Anti-Money Laundering Models. In *Proceedings of NeurIPS 2023* (2023), pp. 1–24.

[3] Battaglia, P. W., Hamrick, J. B., et al. Relational inductive biases, deep learning, and graph networks. *CoRR abs/1806.01261* (2018), 1–40.

[4] Cassara, J. A. Countering international money laundering: Total failure is "only a decimal point away". Tech. rep., The FACT Coalition, Washington, DC, August 2017.

[5] Egressy, B., von Niederhäusern, L., et al. Provably Powerful GNNs for Directed Multigraphs. In *Proceedings of (AAAI-24)* (2024), AAAI Press, pp. 11838–11846.

[6] Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. *CoRR abs/1704.01212* (2017), 1–14.

[7] Hu, W., Liu, B., Gomes, J., Zitnik, M., Liang, P., Pande, V. S., and Leskovec, J. Pre-training graph neural networks. *CoRR abs/1905.12265* (2019), 1–22.
[8] Jaume, G., Nguyen, A., et al. edGNN: a Simple and Powerful GNN for Directed Labeled Graphs. *CoRR abs/1904.08745* (2019), 1–9.
[9] Jensen, R. I. T., and Iosifidis, A. Fighting money laundering with statistics and machine learning. *IEEE Access 11* (2023), 8889–8903.
[10] Johannessen, F., and Jullum, M. Finding money launderers using heterogeneous graph neural networks. *CoRR abs/2307.13499* (2023), 1–20.
[11] Kipf, T. N., and Welling, M. Variational graph auto-encoders. *CoRR abs/1611.07308* (2016), 1–3.
[12] Kumar, S., Ahmed, R., et al. Exploitation of machine learning algorithms for detecting financial crimes based on customers' behavior. *Sustainability 14*, 21 (2022), 1–24.
[13] Lundberg, S. M., Erion, G., et al. From local explanations to global understanding with explainable ai for trees. *Nature Machine Intelligence 2*, 1 (2020), 2522–5839.
[14] Lundberg, S. M., and Lee, S.-I. A Unified Approach to Interpreting Model Predictions. In *Adv Neural Inf Process Syst 30*. Curran Associates, Inc., 2017, pp. 4765–4774.
[15] Murphy, A., Robu, K., and Steinert, M. The investigator-centered approach to financial crime: Doing what matters. *McKinsey and Company* (June 2020). Accessed: March 8, 2025.
[16] Richardson, D., Williams, and Mikkelsen, D. Network analytics and the fight against money laundering. McKinsey and Company, 2019.
[17] Sato, R., Yamada, M., and Kashima, H. Approximation ratios of graph neural networks for combinatorial problems. *CoRR abs/1905.10261* (2019), 1–15.
[18] Schlichtkrull, M. S., Kipf, T. N., et al. Modeling Relational Data with Graph Convolutional Networks. In *Proceedings of ESWC 2018,* (2018), vol. 10843 of *Lecture Notes in Computer Science*, Springer, pp. 593–607.
[19] Suzumura, T., and Kanezashi, H. Anti-Money Laundering Datasets: InPlusLab anti-money laundering datadatasets. `http://github.com/IBM/AMLSim/`, 2021.
[20] Tang, M., Yang, C., and Li, P. Graph auto-encoder via neighborhood wasserstein reconstruction. *CoRR abs/2202.09025* (2022), 1–17.
[21] Velickovic, P., Fedus, W., Hamilton, W. L., Liò, P., Bengio, Y., and Hjelm, R. D. Deep graph infomax. *CoRR abs/1809.10341* (2018).
[22] Verhage, A. Supply and demand: anti-money laundering by the compliance industry. *Journal of Money Laundering Control 12* (10 2009), 371–391.
[23] Weber, M., Domeniconi, G., Chen, J., Weidele, D. K. I., Bellei, C., Robinson, T., and Leiserson, C. E. Anti-money laundering in bitcoin: Experimenting with graph convolutional networks for financial forensics. *CoRR abs/1908.02591* (2019), 1–7.
[24] Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How powerful are graph neural networks? *CoRR abs/1810.00826* (2018), 1–17.
[25] You, J., Gomes-Selman, J. M., Ying, R., and Leskovec, J. Identity-aware graph neural networks. *Proceedings of the AAAI Conference on Artificial Intelligence 35*, 12 (May 2021), 10737–10745.
[26] Zhou, J., Cui, G., Zhang, Z., Yang, C., Liu, Z., and Sun, M. Graph neural networks: A review of methods and applications. *CoRR abs/1812.08434* (2018), 57–81.

Department of Computer Science, Faculty of Mathematics and Computer Science, Babeş-Bolyai University, Cluj-Napoca, Romania
Email address: `tudor.grama@stud.ubbcluj.ro`