GRAPH-BASED FAKE NEWS DETECTION IN SOCIAL NETWORKS

FINAL PROJECT REPORT

Théau d'Audiffret
theau.daudiffret@student-cs.fr

Ibrahim Al Khalil Ridene
ibrahim-al-khalil.ridene@student-cs.fr

Ulysse Roux
ulysse.roux@student-cs.fr

Syphax Yesli
syphax.yesli@student-cs.fr

ABSTRACT

The rapid spread of misinformation on social media poses a growing societal threat, influencing public opinion, politics, and health. Traditional text-based fake news detection methods often fall short in capturing the dynamic, context-dependent nature of online discourse. In this work, we explore a graph-based approach to rumour detection by modeling the structure of tweet conversations as propagation trees. We investigate the effectiveness of Graph Neural Networks (GNNs) in leveraging both content features and interaction patterns, and compare their performance against classical Machine Learning models such as Random Forests. Our results highlight the potential of graph-structured learning for detecting misinformation in social networks.

1 INTRODUCTION AND MOTIVATION

The exponential rise of social media platforms has transformed how information is produced, disseminated, and consumed. However, this transformation has also amplified the spread of misinformation, or "fake news," which can undermine public trust, distort political discourse, and endanger public health. Traditional fake news detection methods often rely on analyzing the textual content of news articles or posts. While effective to some extent, these approaches face significant limitations, particularly due to the high linguistic similarity between real and fake news, as well as the adaptability of misinformation tactics.

As a result, recent research has shifted towards leveraging the structural and temporal aspects of information diffusion in social networks. Graph-based approaches, which model user interactions and content propagation as structured graphs or trees, offer a promising alternative. By capturing how information spreads and who engages with it, these methods can provide complementary signals to textual analysis.

In this project, we focus on rumour detection in Twitter conversations by modeling them as tree-structured graphs rooted at the original tweet. We investigate the extent to which the reply structure and user interaction patterns contribute to distinguishing between rumours and legitimate information. We compare the performance of Graph Neural Networks (GNNs), which natively operate on such structures, with a classical machine learning model to assess the advantages of structure-aware learning in misinformation detection.

Such methods have potential applications in real-time content moderation, early-warning systems for misinformation outbreaks, and public policy tools to assess the credibility of viral information during crises.

CONTENTS

2 PROBLEM DEFINITION

We consider the task of *rumour detection* in social media, specifically on conversation trees derived from Twitter discussions. The underlying data structure is a **tweet graph**, modeled as a rooted, directed tree $G = (V, E)$ where:

- Each node $v \in V$ corresponds to a tweet.
- The root node v_0 represents the *original tweet* that initiated the conversation.
- An edge $(v_i, v_j) \in E$ indicates that tweet v_j is a direct reply to tweet v_i.

Each node $v \in V$ is associated with a feature vector $\mathbf{x}_v \in \mathbb{R}^d$ encoding:

- User features (e.g., verified status, follower count).
- Tweet content features (e.g., encoded text).
- Tweet metadata (e.g., timestamp, number of likes, retweets).

Let $\mathcal{G} = \{G_1, \ldots, G_N\}$ denote a dataset of tweet graphs. Each graph G_i is labeled with a binary label $y_i \in \{0, 1\}$ indicating whether the root tweet is a rumour ($y_i = 1$) or not ($y_i = 0$). The goal is to learn a function $f : \mathcal{G} \to \{0, 1\}$ that maps an unseen tweet graph G to its corresponding rumour label $\hat{y} = f(G)$.

Objective. The objective is to learn f such that the classification error over unseen graphs is minimized. Formally, we aim to minimize the expected loss:

$$\mathbb{E}_{(G,y)\sim\mathcal{D}} \left[\ell(f(G), y)\right],$$

where $\mathcal{D}$ is the (unknown) distribution over tweet graphs and labels, and ℓ is a loss function such as binary cross-entropy or 0-1 loss.

Graph-Based Assumption. A key hypothesis in this work is that the *replying structure* of the conversation tree carries valuable signal for rumour detection. In particular, the pattern and depth of interactions, as well as the content of replies, may differ significantly between rumour and non-rumour threads. This motivates the use of Graph Neural Networks (GNNs) or tree-based models that exploit both node features and the graph topology.

Constraints.

- Each graph G is a rooted, directed tree with no cycles, and a single path from the root tweet to each reply.
- Node features $\mathbf{x}_v$ are heterogeneous, combining numerical (e.g., follower count), categorical (e.g., verified status), and textual (e.g., encoded tweet content) attributes.
- The graphs vary significantly in size, with a wide range in the number of tweets (nodes) and replies (edges).

Hardness.

- **High-dimensional and noisy input:** Tweets often contain informal language, abbreviations, or sarcasm, making textual features difficult to model.
- **Structural variability:** Conversation trees differ widely in depth, breadth, and branching patterns, increasing the complexity of graph-level modeling.
- **Class imbalance:** The dataset is heavily skewed toward non-rumour instances, making it harder for models to detect minority (rumour) cases reliably.
- **Multimodal feature fusion:** Effectively combining structural, textual, and user-based features within a unified model remains a non-trivial challenge.

Given these challenges, the problem is a form of binary classification over trees, and requires models capable of learning from both content and structure.

3 Related Work

The problem of rumour detection in social media has been widely explored in the literature, traditionally framed as a binary text classification task using natural language processing (NLP) techniques. Early methods (1) focused on feature engineering from tweet metadata and content, leveraging classifiers such as SVMs or decision trees to distinguish credible from non-credible posts. While effective to some extent, these approaches struggled to generalize across events due to the high linguistic similarity between rumours and non-rumours.

To improve generalization, later studies introduced deep neural models for text encoding. Ma et al. (2) proposed using recurrent neural networks (RNNs) over temporal sequences of tweets in a thread, capturing how the content evolves over time. Similarly, Chen et al. (3) incorporated attention mechanisms to focus on key replies. However, these methods still treated tweets as linear sequences, overlooking the tree-like structure of online conversations.

Graph-based approaches have emerged to address this limitation. Wu et al. (4) modeled retweet cascades as graphs and applied recursive neural networks over the structure. More recent work by Bian et al. (5) proposed hierarchical Graph Neural Networks (GNNs) that exploit both textual features and propagation paths. GNNs such as GCN, GAT, and GIN have shown strong performance in misinformation detection by explicitly modeling the relational structure among posts (6; 7).

Our project builds upon this graph-centric paradigm by modeling Twitter conversation threads as directed trees and applying GNNs to capture both content and interaction patterns. Unlike prior work that often relies on synthetic or retweet-based graphs, we use actual reply structures derived from the PHEME dataset (8). Moreover, we extend prior methods by comparing the GNN-based models (e.g., GAT and GIN) with classical machine learning approaches trained on aggregated graph-level features, such as Random Forests and Gradient Boosting classifiers. This dual approach allows us to assess the relative contributions of node-level content and graph structure. The implementation of the Machine Learning approach in order to compair its performance with Deep Learning approach is crucial here since we have a small dataset (8) for training, validation and testing which could impact GNN performance.

In contrast to models that use only raw text, our GNN pipeline incorporates semantic embeddings (from RoBERTa), user metadata, and structural signals, demonstrating the effectiveness of combining multimodal features. To the best of our knowledge, few works have directly compared classical ML models and GNN-based classifiers under a shared experimental setup, which makes our work a valuable contribution in understanding the trade-offs between interpretability, performance, and scalability in rumour detection.

4 Methodology

We present our full pipeline for rumour detection, from dataset construction and preprocessing to modeling. The process includes feature engineering, selection, and stratified data splitting. We compare classical machine learning classifiers with deep graph-based models designed to capture both content and structure in tweet conversations.

4.1 Data Collection and Preprocessing

We use the PHEME dataset (8), which contains Twitter threads annotated for rumour veracity. Each thread is modeled as a tree-structured graph, with tweets as nodes and reply links as edges. For both ML and DL approaches, we extract the graph structure from `structure.json` and the labels from `annotation.json`, discarding unverified cases to retain a binary classification task.

In the ML pipeline, node-level metadata (user statistics, tweet metrics) and sentence embeddings (from MiniLM) are aggregated into graph-level features via mean pooling. In contrast, the DL approach uses RoBERTa-based embeddings to represent tweets and retains the full graph topology for GNN-based classification. This results in more expressive node-level representations suitable for structural learning.

4.2 Feature Engineering

Feature engineering is a crucial step for the machine learning (ML) approach, as traditional classifiers rely heavily on informative and well-crafted input features. In contrast, deep learning (DL) models such as Graph Neural Networks (GNNs) can learn hierarchical representations directly from raw data and graph structure, making manual feature design largely unnecessary. Therefore, this step was only applied to the ML pipeline.

For each graph in the dataset, we augmented node features with the following hand-designed structural attributes:

- **Link Prediction Heuristics (4 statistics $\times$ 5 methods):** For each node, we computed the *mean, sum, max,* and *standard deviation* of edge scores with its neighbors, using five common link prediction scores:
  - *Preferential Attachment (PA):* Based on the product of degrees of two nodes.
  - *Resource Allocation (RA):* Emphasizes the common neighbors of low degree.
  - *Adamic-Adar (AA):* We used a robust version that avoids division by zero.
  - *Jaccard Coefficient (JC):* Measures the overlap of neighbor sets.
  - *Second-Order Preferential Attachment (PA2):* A custom extension based on neighbors-of-neighbors' degrees.
- **Graph-Theoretic Features:** Captures topological roles of nodes within the graph:
  - *Node Degree:* Number of direct connections.
  - *Local Clustering Coefficient:* Measures triangle formation around the node.
  - *Betweenness Centrality:* Quantifies how often a node appears on shortest paths.
  - *HITS Hub and Authority Scores:* Reflect the influence and credibility of nodes.

These features were computed using NetworkX and concatenated to the original flattened features of each node. The result was a richer node representation that encodes both content and structural properties. Finally, a mean pooling operation was used to convert node-level features into graph-level vectors for use in ML classifiers. This transformation was implemented via a custom BaseTransform (AddNetworkFeatures) applied during dataset preparation.

4.3 Data Understanding

The dataset consists of conversation graphs extracted from the PHEME rumour detection dataset. After filtering out invalid samples (e.g., missing annotations or unverified labels), we obtained a total of 5,726 graph instances.

Class Distribution. Figure 1 shows the distribution of labels across the dataset. The data is highly imbalanced, with approximately 88.9% of the threads labeled as *nonrumour* and only 11.1% labeled as *rumour*. This class imbalance motivates the use of weighted loss functions to avoid biased predictions toward the majority class.

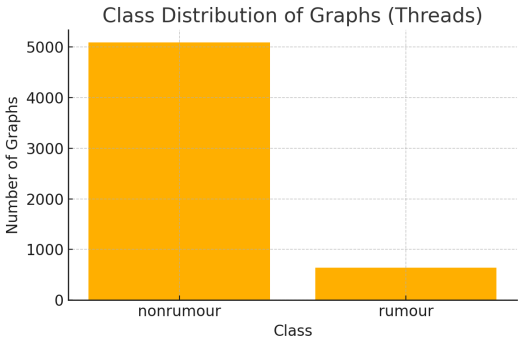

Figure 1: Class Distribution of Graphs (Threads)

Graph Structure. Each graph represents a tweet thread, with nodes corresponding to tweets and edges capturing reply relationships. Table 1 summarizes the structural statistics of the dataset. Graph sizes are highly variable: the number of nodes ranges from 1 to 346 and the number of edges from 0 to 345. On average, graphs have 16.32 nodes and 14.87 edges, with large standard deviations, reflecting the diversity of conversation scales on Twitter.

Node Feature Dimensionality. Each node is encoded with a feature vector whose dimensionality depends on the learning paradigm:

- For the **Machine Learning (ML)** pipeline, node features are constructed from tweet/user metadata and sentence embeddings, resulting in a vector of 426 dimensions.
- For the **Deep Learning (DL)** pipeline, RoBERTa embeddings of tweet content are used exclusively, producing 768-dimensional node features.

The full dataset statistics are summarized in Table 1.

Metric	Value
Average nodes per graph	16.32 ± 19.63
Average edges per graph	14.87 ± 18.98
Max nodes in a graph	346
Min nodes in a graph	1
Max edges in a graph	345
Min edges in a graph	0
Node features (ML approach)	426
Node features (DL approach)	768

Table 1: Graph dataset statistics and node feature dimensions

4.4 Input and Output Tensors

The input and output representations differ significantly between the machine learning (ML) and deep learning (DL) pipelines, primarily due to the different learning paradigms and architectural capabilities.

Machine Learning (ML) Pipeline. In the ML approach, standard classifiers such as Random Forest and SVM require fixed-size input vectors. Since the input is a graph, we must first convert the set of node-level features into a single graph-level representation. To achieve this, we apply **mean pooling** over the node embeddings within each graph:

$$\mathbf{x}_G = \frac{1}{|V|} \sum_{v \in V} \mathbf{x}_v,$$

where $\mathbf{x}_v$ is the feature vector of node v, and $\mathbf{x}_G$ is the resulting graph-level vector used as input to the classifier. The label $y_G \in \{0, 1\}$ indicates whether the root tweet is a rumour or not. This yields a dataset of (X, y) pairs, where:

- $X \in \mathbb{R}^{N \times d}$ is the feature matrix for N graphs with d-dimensional graph embeddings,
- $y \in \{0, 1\}^N$ is the vector of graph-level binary labels.

Deep Learning (DL) Pipeline. In contrast, the DL approach uses Graph Neural Networks (GNNs) such as GAT and GIN, which operate directly on node-level features and edge structures. Each graph is represented by a tuple:

$$(\mathbf{X}, \mathbf{A}, y_G),$$

where $\mathbf{X} \in \mathbb{R}^{|V| \times d}$ is the node feature matrix, $\mathbf{A}$ is the edge index tensor describing the graph connectivity, and y_G is the graph-level label. The GNN aggregates node information through message passing and performs **learnable global pooling** (e.g., mean pooling) to compute the final graph representation used for classification.

4.5 Feature Selection & Importance

In the context of machine learning, feature selection plays a crucial role in improving model performance, reducing overfitting, and accelerating training. Since classical ML models are sensitive to irrelevant or redundant features, a dedicated feature selection pipeline is applied. In contrast, deep learning models such as GNNs inherently perform end-to-end representation learning and are generally more robust to noisy or irrelevant features, reducing the necessity for manual feature pruning.

4.5.1 Low Variance Elimination

Features with near-zero variance provide little to no discriminative power, as they exhibit minimal variation across the dataset. Retaining such features introduces noise and can mislead learning algorithms. To mitigate this, we compute the standard deviation of each feature and eliminate those with zero variance:

- A total of **19 features** were found to have a standard deviation of exactly zero.
- These features remain constant across all graphs and hence carry no useful information for classification.
- After applying this filtering step, the number of retained features was reduced from 426 to **407**.

This step ensures that the subsequent feature selection stages operate on a more informative and compact feature space.

4.5.2 Filter methods

To further reduce the dimensionality of the dataset and retain only informative features, we apply a univariate filter method based on the Pearson correlation with the output labels. Specifically, we compute the Pearson correlation coefficient $\rho_{X,Y}$ between each feature X and the binary target variable Y, defined as:

$$\rho_{X,Y} = \frac{\text{cov}(X,Y)}{\sigma_X \sigma_Y} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

This coefficient measures the strength and direction of the linear relationship between a given feature and the target. Values close to 0 indicate weak or no linear correlation.

- Features were ranked by their absolute Pearson correlation with the output label.
- A threshold of ≥ 0.05 was selected to retain features with a minimum level of relevance to the classification task.
- As a result, the feature space was reduced from 407 to **268** features.

Figure 2 visualizes the sorted absolute correlation scores, demonstrating that a significant portion of the features have near-zero correlation, justifying their removal.

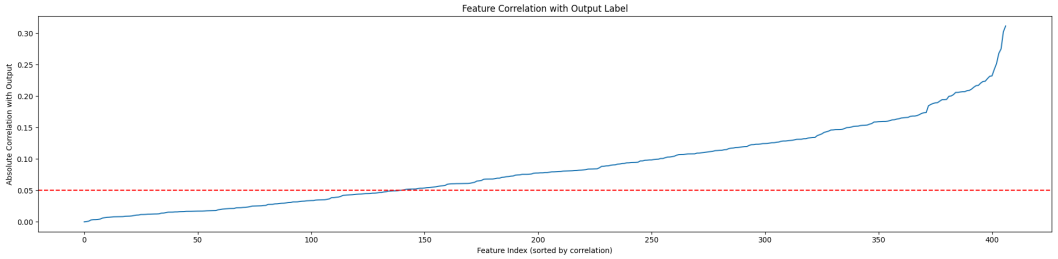

Figure 2: Feature correlation with output label (sorted by correlation)

4.5.3 MULTICOLLINEARITY

Multicollinearity refers to the presence of highly correlated features in the dataset, which can distort the learning process of many machine learning algorithms. It can lead to unstable parameter estimates and reduce model interpretability. To mitigate this, we performed a correlation-based clustering and pruning of redundant features.

We first computed the **pairwise correlation distance** between all features using the formula:

$$\text{Correlation Distance} = 1 - \text{Pearson Correlation}(x_i, x_j)$$

This results in a symmetric distance matrix where lower values indicate high similarity between features. Figure 3 shows this correlation distance matrix.

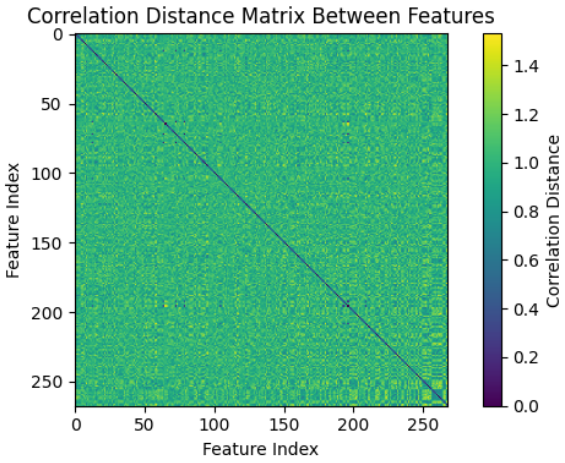

Figure 3: Correlation Distance Matrix Between Features

Next, we performed **hierarchical clustering** on the features using the average linkage method. The resulting dendrogram (Figure 4) visually depicts how features group into clusters based on similarity. We applied a threshold of $1 - 0.9 = 0.1$ to identify highly correlated clusters (correlation ≥ 0.9).

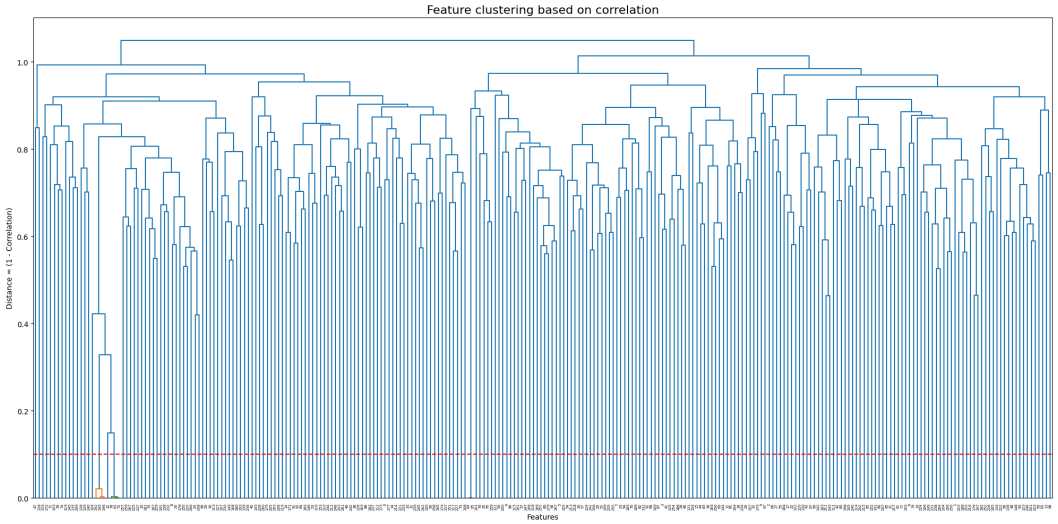

Figure 4: Feature Clustering Dendrogram Based on Correlation

From each cluster of correlated features, we retained only the first feature as a representative and discarded the rest. This ensures that no pair of features in the final set has a correlation above 0.9. As a result, the dimensionality was reduced from 268 to **263 features**, preserving diversity while eliminating redundancy.

This step is crucial for classical ML models sensitive to collinearity. In contrast, deep learning models (e.g., GNNs) often learn robust representations through end-to-end training and do not require such explicit decorrelation.

4.6 Train / Val / Test Split

To ensure robust evaluation while preserving the original class distribution, we performed a stratified split of the dataset into three parts: 70% training, 15% validation, and 15% testing. Stratification guarantees that the imbalance between nonrumour and rumour classes is consistently reflected across all sets. This is especially important due to the high skew in the dataset (approximately 89% nonrumour, 11% rumour).

Figure 5 illustrates the class distribution across the splits, confirming consistent proportions. Table 2 summarizes the exact number of samples per class and per set.

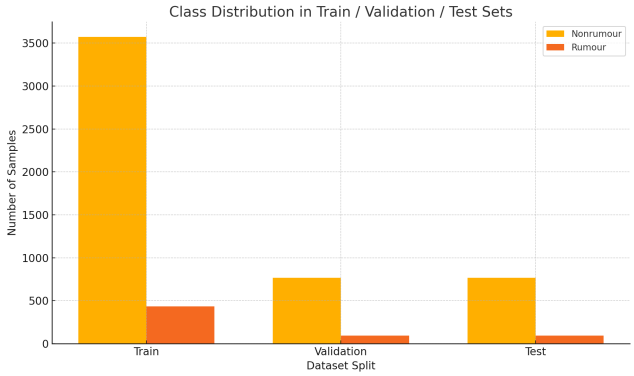

Figure 5: Class Distribution in Train / Validation / Test Sets

Set	Total	Nonrumour (0)	Rumour (1)
Training	4003	3558 (88.88%)	445 (11.12%)
Validation	859	763 (88.82%)	96 (11.18%)
Testing	858	763 (88.93%)	95 (11.07%)

Table 2: Stratified Dataset Split Statistics

4.7 Models

This section presents the different models used in our rumour detection pipeline. For the machine learning (ML) setup, we implemented three classical classifiers using handcrafted node-level features. For the deep learning (DL) setup, we experimented with multiple Graph Neural Network (GNN) architectures, ultimately selecting the Graph Attention Network (GAT) due to its superior performance.

4.7.1 Machine Learning Models

Random Forest. Random Forest (9) is an ensemble learning method that constructs multiple decision trees during training and outputs the majority vote for classification. Each tree is trained on a random subset of the data with a random subset of features:

$$\hat{y} = \text{mode}\left\{h_t(x)\right\}_{t=1}^{T},$$

where h_t is the prediction from the t-th tree.

Support Vector Machine (SVM). Support Vector Machine (10) finds the hyperplane that maximizes the margin between two classes:

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 \quad \text{s.t. } y_i(\mathbf{w}^\top x_i + b) \geq 1,$$

where $\mathbf{w}$ is the normal vector to the hyperplane, and $y_i \in \{-1, +1\}$ are the binary labels. We use a linear kernel, given the high dimensionality of the input features.

Gradient Boosting. Gradient Boosting (11) trains a sequence of weak learners (typically decision trees), each focusing on minimizing the residual errors of the ensemble:

$$F_t(x) = F_{t-1}(x) + \eta h_t(x),$$

where F_t is the ensemble at iteration t, $h_t(x)$ is the new learner, and η is the learning rate.

4.7.2 Deep Learning Models

Graph Attention Network (GAT). Graph Attention Networks (12) extend standard message-passing GNNs by introducing an attention mechanism that assigns different weights to neighboring nodes:

$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}(\mathbf{a}^\top[\mathbf{W}\mathbf{h}_i \| \mathbf{W}\mathbf{h}_j])\right)}{\sum_{k\in\mathcal{N}(i)} \exp\left(\text{LeakyReLU}(\mathbf{a}^\top[\mathbf{W}\mathbf{h}_i \| \mathbf{W}\mathbf{h}_k])\right)},$$

$$\mathbf{h}'_i = \sigma\left(\sum_{j\in\mathcal{N}(i)} \alpha_{ij}\mathbf{W}\mathbf{h}_j\right),$$

where α_{ij} are learned attention coefficients, $\mathbf{W}$ is a shared linear transformation, and $\mathbf{a}$ is the attention weight vector. We compared GAT to other GNN architectures including GCN (13) and GIN (14), and retained GAT for its superior accuracy on the validation set.

5 Evaluation & Results

This section presents how we evaluated our models, the metrics used, the experimental settings, and the final results on the test data for both machine learning (ML) and deep learning (DL) approaches.

5.1 Evaluation Setup and Metrics Used

We evaluate model performance using accuracy, precision, recall, and F1-score. Given the class imbalance, the F1-score is prioritized as it better reflects performance on the minority (rumour) class by balancing precision and recall.

Throughout this section, we study both the training and validation performance during hyperparameter tuning, and we present final test set results for each approach (ML and DL). Our goal is to understand which models perform best overall, and particularly how well they detect rumours.

5.1.1 ML Approach

To optimize hyperparameters for the machine learning models, we employed **Bayesian Optimization** via the `Optuna` framework (15). Unlike exhaustive or grid search, Bayesian Optimization builds a probabilistic surrogate model of the objective function—typically a Gaussian Process or Tree-structured Parzen Estimator (TPE)—to model the relationship between hyperparameters and performance. It uses an acquisition function to balance *exploration* (trying diverse configurations) and *exploitation* (refining around promising regions).

Let $\mathcal{L}(\theta)$ denote the validation F1 score for a given hyperparameter configuration θ. The optimization objective is:

$$\theta^* = \arg\max_{\theta\in\Theta} \mathbb{E}[\mathcal{L}(\theta)]$$

This process iteratively refines its belief over the space Θ to efficiently approach the optimum θ^*. We executed 50 trials per model under a time constraint, targeting the maximization of the F1 score due to class imbalance. Each trial involved data normalization using `StandardScaler`, training the model, and computing the validation F1. The best-performing configuration was selected for final evaluation on the test set.

5.1.2 DL Approach

Due to computational constraints, we did not employ Bayesian hyperparameter optimization for the deep learning models. Instead, we manually tested several combinations of key hyperparameters — including hidden layer size, learning rate, batch size, and dropout — and retained the configuration yielding the best validation F1-score.

To address the significant class imbalance between rumours and non-rumours, we used a **weighted cross-entropy loss**, which assigns higher loss penalties to misclassifications of the minority class. The loss function is defined as:

$$\mathcal{L} = -\sum_{i=1}^{N} w_{y_i} \cdot \log(p_{y_i})$$

where N is the number of samples, w_{y_i} is the weight associated with the true class y_i, and p_{y_i} is the predicted probability for the correct label. The class weights were computed inversely proportional to class frequencies, following the formula:

$$w_c = \frac{1}{(\text{count}(c))^{\gamma}}$$

where γ is a smoothing factor (set to 0.6), which prevents extremely large weights for very rare classes.

Moreover, we employed a **scheduled learning rate strategy** to improve optimization stability and convergence:

- **Warm-up phase:** During the first 5% of training epochs, the learning rate was gradually increased linearly from a very small value up to the initial learning rate.
- **Step decay:** After warm-up, we applied a `StepLR` schedule, reducing the learning rate by a multiplicative factor $\gamma = 0.95$ every 10 epochs. This allows for larger updates during early training and finer adjustments later, facilitating convergence to a better local optimum.

This combination of weighted loss and learning rate scheduling proved crucial for achieving competitive performance in an imbalanced and complex setting.

5.2 Experimental Results

We now present the results of training and testing both ML and DL models. The following subsections detail the validation and final testing performance.

5.2.1 Training and Validation

ML Approach. We performed Bayesian hyperparameter optimization for three classifiers: Random Forest (RF), Support Vector Machine (SVM), and Gradient Boosting (GB). The optimization aimed to maximize the F1 score on the validation set, particularly focusing on the rumour class due to the strong class imbalance. The best validation F1 scores and corresponding hyperparameter configurations are summarized in Table 3.

Model	Best Validation F1	Best Hyperparameters
Random Forest	0.5442	`n_estimators=203, max_depth=6`
SVM	0.5938	`C=39.49, kernel=rbf, gamma=6.36e-4`
Gradient Boosting	0.5532	`n_estimators=148, max_depth=4, learning_rate=0.345`

Table 3: Best ML model validation performance and hyperparameters

Among the three models, the Support Vector Machine (SVM) achieved the highest F1 score (0.5938) on the validation set, suggesting that a non-linear decision boundary better separates rumours from non-rumours in the feature space. Gradient Boosting and Random Forest also performed reasonably but slightly worse. This shows that although ensemble tree-based methods capture complex interactions, SVMs can sometimes outperform them under high-dimensional but moderately sized data settings.

DL Approach. For the deep learning (DL) pipeline, we manually tuned the GAT model without automatic hyperparameter search, due to computational resource constraints. The training and validation loss curves, along with the learning rate evolution across epochs, are illustrated in Figure 6.

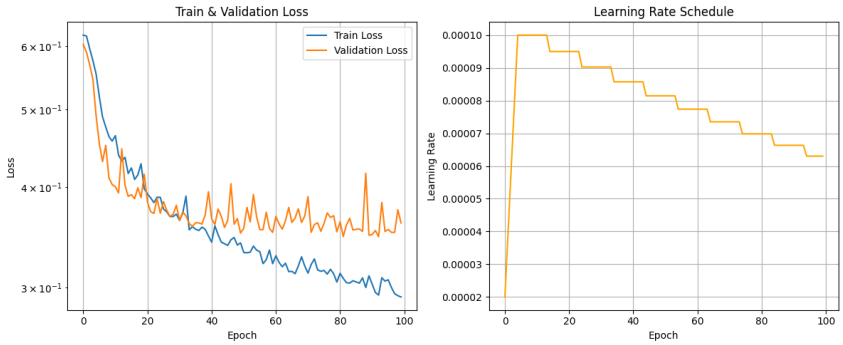

Figure 6: Training and validation loss curves (left) and learning rate schedule (right) for the GAT model

The loss curves show a consistent decrease for both training and validation sets, indicating stable convergence and absence of overfitting. Initially, the learning rate is increased linearly during the warm-up phase to avoid instability at early steps, then gradually decreased every 10 epochs following a step decay strategy. This helped refine the model during later epochs. Notably, the validation loss reached a stable low level after approximately 10–15 epochs, suggesting that the GAT model effectively learned discriminative patterns for rumour detection without significant overfitting.

5.2.2 Testing

ML Approach. After selecting the best hyperparameters through validation, we retrained each ML model (Random Forest, SVM, and Gradient Boosting) on the combined training and validation sets (85% of the data) and evaluated them on the held-out 15% test set. This ensures that the models will also take advantage of the information contained in the validation set, as well as a performed testing completely on unseen data, providing an unbiased estimate of generalization performance.

The resulting confusion matrix for the best-performing ML model (SVM) is shown in Figure 7.

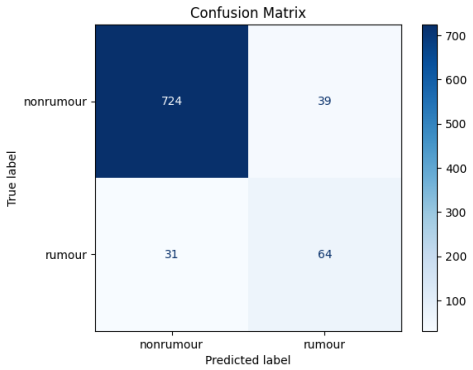

Figure 7: Confusion matrix for the best ML model (SVM) on the test set

Quantitative test results for all ML models are summarized in Table 4.

Model	Test Accuracy	Test F1 (Rumour)
Random Forest	0.9219	0.5563
SVM	0.9184	**0.6465**
Gradient Boosting	**0.9289**	0.5850

Table 4: ML model performance on the test set

Although Gradient Boosting achieved the highest overall accuracy (92.89%), the SVM provided the best F1-score (0.6465) for the rumour class, which was our primary interest due to the severe class imbalance. This indicates that the SVM is better at capturing minority class (rumour) instances without significantly compromising the overall model accuracy. The confusion matrix confirms this, showing a better balance between true positive and false negative rates for rumours.

DL Approach. The GAT model was similarly evaluated on the test set using the best-found hyperparameter configuration. The confusion matrix for the GAT model is shown in Figure 8.

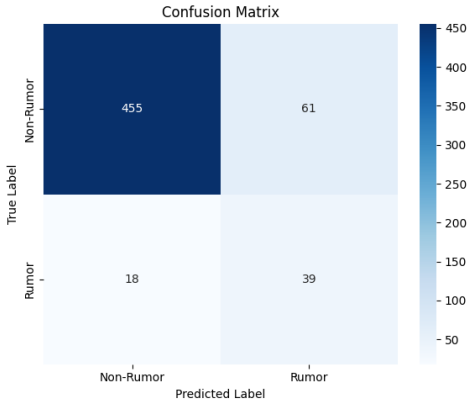

Figure 8: Confusion matrix for the GAT model on the test set

The detailed test set performance metrics are summarized in Table 5.

Class	Precision	Recall	F1-score
Non-rumour	0.962	0.882	0.920
Rumour	0.390	0.684	0.497
Macro Avg	0.676	0.783	0.708
Accuracy		0.862	

Table 5: GAT model performance on the test set

The GAT model achieved a strong overall accuracy of 86.2%, but its performance on the rumour class remained moderate, with an F1-score of 0.497. The weighted loss function helped the model somewhat counterbalance the dataset skew, improving recall on the rumour class to 68.4%. However, precision for rumour detection (39%) remained relatively low, indicating a tendency to produce more false positives when predicting rumours. This reflects a typical challenge in graph-based social media rumor detection, where subtle structural and linguistic differences between real and fake content are difficult to distinguish.

Overall, while both ML and DL approaches demonstrated solid performance, ML models—especially SVM—offered better precision and F1 on rumours, whereas GAT provided stronger structural learning but faced precision challenges despite improved recall.

6 CONCLUSION

In this project, we addressed the challenging task of rumour detection in Twitter conversations by leveraging both classical machine learning (ML) and deep learning (DL) approaches. We modeled tweet threads as directed trees, extracted rich content and structural features, and formulated rumour detection as a graph-level binary classification problem.

For the ML pipeline, we engineered node-level and graph-level features, applied extensive feature selection techniques, and optimized model hyperparameters using Bayesian optimization with Optuna. Among Random Forest, SVM, and Gradient Boosting classifiers, the SVM model achieved the highest F1-score on the minority (rumour) class, highlighting the effectiveness of classical models when feature engineering is carefully designed.

In parallel, we explored a deep learning pipeline based on Graph Attention Networks (GATs), which natively process graph-structured data. Despite not applying extensive hyperparameter tuning due to computational constraints, the GAT model demonstrated competitive performance, especially in capturing the propagation dynamics within conversations. Class imbalance was addressed through a weighted loss function, and a scheduled learning rate policy helped stabilize convergence.

Our experiments demonstrated that while deep models can exploit relational structures, classical ML models remain strong contenders when supplied with well-crafted features, especially under limited supervision.

Future Work. Several promising directions can be explored to further advance this work:

- Incorporating user credibility signals (e.g., account age, historical behaviour) as additional features.
- Extending the GNN models with hierarchical or heterogeneous architectures (e.g., Heterogeneous Graph Neural Networks) to better capture different types of entities and relations.
- Leveraging large pretrained language models to enrich node representations dynamically during GNN training, instead of static embeddings.
- Applying self-training or semi-supervised learning techniques to leverage unlabelled threads and improve generalization.
- Exploring adversarial robustness, given that misinformation propagators may intentionally perturb tweet content or interaction structures.

Overall, our project highlights the potential of combining graph-based modeling and machine learning techniques for social media misinformation detection, offering a solid foundation for future research and real-world deployment.

REFERENCES

[1] Castillo, C., Mendoza, M., & Poblete, B. (2011). Information credibility on Twitter. *Proceedings of the 20th international conference on World Wide Web (WWW)*.

[2] Ma, J., Gao, W., Mitra, P., Kwon, S., Jansen, B. J., Wong, K. F., & Cha, M. (2016). Detecting rumors from microblogs with recurrent neural networks. *IJCAI*.

[3] Chen, G., Wu, S., Wang, F., Ge, Y., Song, Y., & Zhang, Y. (2017). Call attention to rumors: Deep attention based recurrent neural networks for early rumor detection. *PAKDD*.

[4] Wu, L., Morstatter, F., Carley, K. M., & Liu, H. (2018). Tracing fake-news footprints: Characterizing social media messages by how they propagate. *WSDM*.

[5] Bian, T., Zhou, X., Wang, Y., Zhang, X., & Jin, D. (2020). Rumor detection on social media with bi-directional graph convolutional networks. *AAAI*.

[6] Li, J., Dani, H., Hu, X., Tang, J., & Liu, H. (2020). Graph-based deep learning: A review. *Proceedings of the IEEE*.

[7] Monti, F., Frasca, F., Eynard, D., Mannion, D., & Bronstein, M. M. (2019). Fake news detection on social media using geometric deep learning. *arXiv preprint arXiv:1902.06673*.

[8] Kochkina, E., Liakata, M., & Zubiaga, A. (2017). PHEME dataset for Rumour Detection and Veracity Classification. *Figshare*. `https://figshare.com/articles/dataset/PHEME_dataset_for_Rumour_Detection_and_Veracity_Classification/6392078`

[9] Breiman, L. (2001). Random forests. *Machine learning*, 45(1), 5-32.

[10] Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20(3), 273-297.

[11] Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. *Annals of statistics*, 1189-1232.

[12] Veličković, P. et al. (2018). Graph Attention Networks. *ICLR*.

[13] Kipf, T. N., & Welling, M. (2016). Semi-supervised classification with graph convolutional networks. *arXiv:1609.02907*.

[14] Xu, K. et al. (2019). How Powerful are Graph Neural Networks?. *ICLR*.

[15] Akiba, T. et al. (2019). Optuna: A Next-generation Hyperparameter Optimization Framework. *KDD*.