Masters of Data Science Dissertation

Rumor Detection using Graph Neural Networks and Attention Mechanisms

Matthew Haskins

Student ID: 24003606

Supervisor: Jin Hong

The University of Western Australia - Department of Mathematics and Computer Science

Contents

Chapter 1

Introduction

Neural networks have transformed how we model complex data, from images to text, by learning hierarchical representations. Early deep learning models such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs) proved effective for Euclidean-structured data (grids of pixels, sequential text). However, they struggled with *non-Euclidean* structured data (like graphs), which do not lie on regular grids ([1611.08097] Geometric deep learning: going beyond Euclidean data). Another breakthrough in deep learning was the introduction of *attention mechanisms*. An attention mechanism directs a model to "prioritize (or attend to) the most relevant parts of input data" (What is an attention mechanism? | IBM), effectively computing weights that indicate the importance of different input components (What is an attention mechanism? | IBM). First proposed by Bahdanau *et al.* (2014) to improve neural machine translation (What is an attention mechanism? | IBM), attention alleviated the burden on sequence models by allowing dynamic focus on parts of the sequence. This idea was later generalized in the Transformer architecture, which relies purely on self-attention layers without recurrence or convolution (What is an attention mechanism? | IBM), leading to state-of-the-art performance in many domains.

The success of attention in sequence modeling naturally inspired its adoption in other domains, including graphs. Meanwhile, a new subfield called **geometric deep learning** emerged, aiming to extend neural network de-

sign to non-Euclidean domains such as graphs and manifolds ([1611.08097] Geometric deep learning: going beyond Euclidean data). Social networks, knowledge graphs, and other relational data are inherently graph-structured, so they cannot be directly fed into CNNs or RNNs which assume grid or sequence input. Geometric deep learning provides a framework to handle such *structured* data by respecting their underlying geometry (e.g. graph connectivity). This paves the way for *graph neural networks (GNNs)* – models specifically designed to learn from graph data. In the following, we review neural network models leading up to attention-based GNNs, emphasizing **graph neural networks** and how attention mechanisms have been integrated into them. We then explore advanced GNN architectures for **rumor detection** on social media, a task that benefits from modeling the *propagation structure* of information. Throughout, we chart the progression of model architectures over time – from early neural rumor detectors to the latest graph-attentional models – and discuss their effectiveness in detecting rumors. Key challenges (e.g. scalability, real-time detection) and assumptions are highlighted, and we survey commonly used datasets (Twitter15, Twitter16, PHEME, etc.) in this research area. Finally, we outline where experimental results from our own application will be incorporated, discuss the application's utility in embedding engineering, and compare our approach with state-of-the-art models. Figures illustrating model architectures, data distributions, and attention visualizations are suggested in relevant sections for clarity.

(Figure 1 about here: Illustration of the evolution from traditional neural networks to attention mechanisms and geometric deep learning. For example, a timeline showing RNN/CNN → Attention → GNN → Graph Attention Networks.)

Chapter 2

Geometric Neural Networks

Core Concepts. *Geometric neural networks* (often synonymous with graph neural networks in practice) refer to neural models that operate on data with geometric structure, such as graphs. Unlike images or sequences, graphs have an arbitrary topology (nodes connected by edges) and no fixed ordering. GNNs generalize neural network operations to respect this topology. In essence, a GNN performs a form of *message passing* or neighborhood aggregation: each node iteratively updates its representation by combining information from its neighbors in the graph. This allows the network to learn node embeddings that capture both the node's features and the graph structure around it ([1609.02907] Semi-Supervised Classification with Graph Convolutional Networks). One foundational GNN model is the **Graph Convolutional Network (GCN)** by Kipf & Welling (2016) ([1609.02907] Semi-Supervised Classification with Graph Convolutional Networks). The GCN introduced a convolution-like operation on graphs defined via the spectral graph Laplacian, which the authors approximated with a simple linear neighborhood average. Formally, a GCN layer can be written as a matrix operation:

$$H^{(l+1)} = \sigma\Big(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}\,H^{(l)}W^{(l)}\Big),$$

where $H^{(l)}$ is the matrix of node features (or embeddings) at layer l (with $H^{(0)}$ being the initial input features), $\tilde{A} = A + I$ is the adjacency matrix of the graph with added self-loops, $\tilde{D}$ is the diagonal degree matrix of $\tilde{A}$, $W^{(l)}$ is

a trainable weight matrix, and σ is a non-linear activation (e.g. ReLU). This equation means: each node's new representation is computed as a weighted aggregate of itself and its neighbors' previous representations, normalized by neighbor counts. The result is that the learned embeddings "encode both local graph structure and features of nodes" ([1609.02907] Semi-Supervised Classification with Graph Convolutional Networks). GCNs are efficient (linear in the number of edges) and can be applied for node classification, link prediction, etc. on graph data.

One important distinction of GNNs from traditional networks is their ability to handle varying graph sizes and structures. They achieve *permutation invariance*: reordering node indices does not change the result, since computation is tied to the graph connectivity. Additionally, many GNNs can operate in both **transductive** settings (where the whole graph is given at training time) and **inductive** settings (where the model can generalize to new nodes or entirely new graphs). An example of an inductive GNN is **GraphSAGE** (Hamilton *et al.*, 2017), which introduced a sampling-based neighborhood aggregator ([1706.02216] Inductive Representation Learning on Large Graphs). Instead of using the full adjacency matrix, GraphSAGE *samples* a fixed number of neighbors for each node and then applies a chosen aggregate function (mean, sum, max, or even an RNN) to combine neighbor features. By *learning a function* to aggregate neighbor information, GraphSAGE can generate embeddings for previously unseen nodes or new subgraphs on the fly ([1706.02216] Inductive Representation Learning on Large Graphs). In other words, "instead of training individual embeddings for each node," GraphSAGE learns how to compute a node's embedding from its neighborhood ([1706.02216] Inductive Representation Learning on Large Graphs). This was a key step toward scaling GNNs to large, dynamic graphs (e.g. evolving social networks), and it highlighted that the choice of aggregator function is flexible – it could even be a learned function.

Graph Neural Networks vs. Traditional Neural Networks. Traditional neural nets like MLPs or CNNs assume a fixed input size or grid structure. If you feed a social network graph into a vanilla MLP, you'd have to vectorize the adjacency matrix or perform heavy feature engineering, los-

ing relational information. GNNs, by contrast, take the graph's adjacency as an inherent part of the model: each node's embedding is influenced by its connections. CNNs achieve translational weight sharing by sliding filters over a grid; analogously, GNNs achieve permutation-invariant weight sharing by applying the same aggregation function to each node's neighborhood. This means GNNs naturally respect the *invariances and symmetries* of graph data (invariance to node ordering, for example) ([1611.08097] Geometric deep learning: going beyond Euclidean data). Another difference is that graph data often have *variable* size (different number of nodes, each node with different degree), so GNNs must handle a variable receptive field. In CNNs, a pixel's neighbors are the fixed adjacent pixels; in a graph, a node could have 3 neighbors or 300. GNN layers are designed to handle this by summing or averaging over an arbitrary neighbor set. Because of these properties, GNNs are well-suited for structured data like social networks, citation networks, knowledge graphs, molecules, etc., where relationships between entities are as important as the entities themselves.

From a geometric perspective, one can view GNNs as a form of **geometric deep learning** that generalizes the notion of convolutions to graphs ([1611.08097] Geometric deep learning: going beyond Euclidean data). Just as CNNs leverage the *geometry* of images (the grid and translational symmetry) and Transformers leverage the *structure* of sequences with positional embeddings and attention, GNNs leverage the *geometry of graphs* – specifically, the connectivity and possibly spatial embedding of nodes. In social networks, this is crucial because information propagates through connections: *how* a rumor spreads (who shares it with whom, and in what pattern) can be as telling as *what* the content is. Geometric neural networks capture these propagation patterns by design.

(Figure 2 about here: Diagram comparing a CNN vs a GNN. For example, show a 2D image grid with a CNN filter vs a graph with a GNN neighborhood aggregation. Highlight that CNN uses fixed neighboring pixels, whereas GNN uses graph-defined neighbors.)

Chapter 3

Graph Attention Networks (GATs) and GATv2

One of the most significant advancements in GNN architecture was the incorporation of **attention mechanisms** into graph neural networks. The resulting model, the **Graph Attention Network (GAT)**, was introduced by Veličković *et al.* (2017) ([1710.10903] Graph Attention Networks). GAT leverages **masked self-attention** to learn weights for each neighbor in a graph convolution, rather than treating all neighbors uniformly (as GCN does with a simple average). In a standard GCN, a node blindly averages or sums over its neighbors' features, which means it cannot discriminate which neighbor is more important for the task at hand. GAT addresses this by computing an attention coefficient α_{ij} for each edge (from neighbor node j to target node i) that reflects the *relevance* of node j's features to node i. High-level, *each node attends to its neighbors* and weighs their contributions differently ([1710.10903] Graph Attention Networks). This was inspired by the success of attention in other domains, bringing the benefit of **adaptive weighting** to graph learning.

Mechanism of GAT. In a GAT layer, every node i first linearly transforms its feature vector $h_i^{(l)}$ (from layer l) using a weight matrix W (shared for all nodes). Then, for each neighbor j of i, a *shared attention mechanism* (a small neural network) computes a scalar score e_{ij} that represents the *un-*

normalized importance of j's features to i. Typically, this attention network is a single-layer feedforward operation parameterized by a weight vector a (often called the attention vector). One common form is:

$$e_{ij} = \text{LeakyReLU}\Big(a^\top [\, W\, h_i \,\|\, W\, h_j \,]\Big),$$

where $[\,\cdot\|\cdot\,]$ denotes concatenation. Here $W\, h_i$ and $W\, h_j$ are the transformed features of nodes i and j, and $a^\top$ times the concatenation produces a single scalar which is passed through LeakyReLU nonlinearity. This e_{ij} can be seen as a *compatibility score* between node i and neighbor j's features. Next, we normalize these scores across all neighbors $N(i)$ of node i using a softmax:

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k \in N(i)} \exp(e_{ik})},$$

so that the attention coefficients α_{ij} sum to 1 over $j \in N(i)$. The coefficient α_{ij} represents the attention weight node i gives to neighbor j's features. Finally, the neighbor features are aggregated using these weights:

$$h_i^{(l+1)} = \sigma\Big(\sum_{j \in N(i)} \alpha_{ij}\, W\, h_j^{(l)} \Big),$$

where σ is an activation function. (In practice, one often includes the node itself $j = i$ in the neighborhood with a self-loop, so a node can attend to its own features as well.)

This mechanism allows the model to **learn** which neighbors are most influential for each node's representation, rather than assuming all neighbors contribute equally. Importantly, the attention computation is *shared* across all edges (the same W and a apply to all pairs), making it efficient and keeping the number of parameters independent of the graph size. By stacking multiple such layers, GAT can propagate information multiple hops away, while at each hop focusing on the most relevant signals. The original GAT paper showed that this approach "enables (implicitly) specifying different weights to different nodes in a neighborhood, *without requiring any costly matrix operations or prior knowledge of graph structure*" ([1710.10903]

Graph Attention Networks). In other words, GAT achieved benefits of adaptive weighting without the heavy spectral computations that earlier spectral methods needed (like computing eigenvectors of the graph Laplacian). GATs were demonstrated to achieve or match state-of-the-art results on node classification benchmarks (Cora, Citeseer, Pubmed citation networks, and a protein interaction graph) ([1710.10903] Graph Attention Networks), validating the power of attention on graphs.

Multi-head attention: GAT also borrowed the idea of multi-head attention from Transformers. Instead of one attention computation, GAT can employ K independent attention "heads" and then concatenate their outputs (or average them) to form the next layer's features. Each head attends to neighbors with a different learned weight vector a_k, so this can stabilize learning and allow the model to capture different types of relations. For example, one head might focus on structural importance of neighbors while another head focuses on similarity of content features, etc. In practice, multi-head attention improved GAT's performance ([1710.10903] Graph Attention Networks) ([1710.10903] Graph Attention Networks).

GAT vs GCN: The key improvement of GAT over a basic GCN is the ability to distinguish neighbors. A GCN's update can be seen as

$$h_i^{\text{GCN}} = \sigma\Big(\sum_{j \in N(i)} \frac{1}{\sqrt{d_i d_j}} W h_j \Big)$$

(where d_i is degree of i). The coefficients $\frac{1}{\sqrt{d_i d_j}}$ are fixed (coming from the normalization of A) and do not depend on the node features or the learning task. GAT replaces these fixed weights with learned α_{ij} that *do* depend on h_i and h_j. This makes the aggregation data-dependent and task-dependent. For instance, for a node representing a Twitter user, a GCN would average over all tweets/users connected to that user. A GAT could learn to give higher weight to credible users and lower weight to outliers or noise, if such weighting helps the end task (say, classifying the user's tweets as rumor vs non-rumor). The ability to focus on the most informative neighbors is especially useful in social network data, where not every connection is

equally informative – some retweets or replies might carry more indicative signals of veracity than others. As Veličković *et al.* note, GAT effectively "addresses the shortcomings of prior methods based on graph convolutions" by combining the strengths of neighborhood aggregation with the flexibility of attention ([1710.10903] Graph Attention Networks). Moreover, GAT, being a spatial/domain method, naturally works in an **inductive** manner too (it doesn't require the full graph's Fourier basis as spectral methods do), making it applicable to new nodes or evolving graphs ([1710.10903] Graph Attention Networks).

(Figure 3 about here: Diagram of a Graph Attention Network layer. For example, show a central node with several neighbor nodes, each edge annotated with an attention weight α_{ij}. Depict multiple attention heads in different colors converging on the node. This figure would illustrate how the central node aggregates neighbor information with learned weights.)

GATv2 – Improved Attention: After the initial success of GAT, researchers investigated its limitations. Brody *et al.* (2022) identified that the original GAT's attention mechanism has a constraint they term *static attention* ([2105.14491] How Attentive are Graph Attention Networks?). In GAT, the way the attention score e_{ij} is computed (with a dot product $a^\top[Wh_i \| Wh_j]$) means that the *ranking* of neighbor importance is actually independent of the specific node i's features in a certain sense. Technically, for a given attention head, the ordering of α_{ij} vs α_{ik} (for two neighbors j, k) doesn't depend on h_i after training – it's determined solely by the learned parameters and neighbor features. This is because Wh_i appears linearly in e_{ij} along with Wh_j. They showed that this leads to a less expressive model than one might hope: GAT cannot learn certain mappings that a fully general attention could. To overcome this, they proposed **GATv2**, described in *"How Attentive Are Graph Attention Networks?"* ([2105.14491] How Attentive are Graph Attention Networks?). GATv2 changes the attention scoring function to be *more expressive (dynamic).* In practice, they achieve this by allowing a richer function of h_i and h_j inside the attention. The GATv2

scoring formula is:

$$e_{ij} = a^{\top}\text{LeakyReLU}\Big(W[\,h_i \| h_j\,]\Big),$$

which looks similar but subtly fixes the issue. By applying the shared linear transformation W *to the concatenation of features* (rather than applying W to each and then concatenating), the ranking of attention scores can now depend on h_i in a non-linear way (GATv2 Explained | Papers With Code). The authors call this *dynamic attention*, since the importance of neighbor j truly depends on the specific context of node i. Intuitively, in GAT, the contribution of h_i to e_{ij} was via a dot product with part of the vector a, which made the relative ordering static once a and W are learned. In GATv2, h_i and h_j are fused through a LeakyReLU and then weighted, introducing an additional non-linearity that makes the attention scores more context-aware ([2105.14491] How Attentive are Graph Attention Networks?) (GATv2 Explained | Papers With Code). GATv2 has the same computational complexity and number of parameters as GAT, but is strictly more powerful in theory. Empirically, Brody *et al.* showed that GATv2 indeed outperforms GAT on numerous benchmark datasets while "matching their parametric costs" ([2105.14491] How Attentive are Graph Attention Networks?). In other words, one can generally swap a GAT layer with a GATv2 layer to get better performance without downside. Many modern libraries (PyTorch Geometric, DGL, etc.) now include GATv2 as a drop-in option.

Beyond GAT and GATv2, there is ongoing research blending transformers with graphs (graph transformers), using attention with edge features, and other sophisticated attention mechanisms (e.g. attention that considers multi-hop neighbors or global structure). However, GAT and GATv2 remain core architectures for attentive message-passing and have been directly applied or extended in many domains – including rumor detection, as we will see next.

Chapter 4

Model Architectures for Rumor Detection

Detecting rumors (or misinformation) on social media is a challenging task that benefits greatly from graph-based modeling. A rumor on Twitter typically starts with a **source post** (a tweet making a claim) and then spreads via replies, retweets, and quotes, forming a **propagation tree** or graph. Additionally, users interact and respond (sometimes supporting, sometimes denying), creating a conversational structure. Early research on automated rumor detection leveraged hand-crafted features of the source tweet, the users, and simple propagation statistics (Rumor Detection on Social Media with Bi-Directional Graph Convolutional Networks). This was followed by approaches using RNNs to process the *sequence* of posts over time (Rumor Detection on Social Media with Bi-Directional Graph Convolutional Networks). A notable milestone was the work of Ma *et al.* (2018), who introduced tree-structured recursive neural networks (RvNN) to model rumor propagation (Rumor Detection on Twitter with Tree-structured Recursive Neural Networks - ACL Anthology). They represented each rumor conversation as a tree (with the source tweet at the root and replies as children, etc.) and built two recursive neural nets: a **bottom-up tree RNN** (aggregating from leaves up to root) and a **top-down tree RNN** (propagating influence from root downwards) (Rumor Detection on Twitter with Tree-

structured Recursive Neural Networks - ACL Anthology). These models could learn representations of the entire propagation tree and achieved much better performance than flat classifiers, even showing "superior capacity on detecting rumors at very early stage" (Rumor Detection on Twitter with Tree-structured Recursive Neural Networks - ACL Anthology). The success of Ma's Tree-RvNN confirmed that exploiting the **propagation structure** is crucial for rumor detection.

4.1 GNN-based Rumor Detection: Overview

Following the tree RNN approaches, researchers began applying graph neural networks to rumor data, often with even greater success. The advantage of GNNs is that they naturally accommodate the *global structure* of the propagation network and can integrate information from arbitrary hops. Moreover, GNNs can incorporate additional relations beyond the strict reply tree (for instance, user-user follow relationships or engagement patterns) by simply adding edges to the graph. By the end of the 2010s, we see the first GNN models designed for rumor detection tasks (Examining the Limitations of Computational Rumor Detection Models Trained on Static Datasets). These models view each rumor's conversational thread as a graph (or a tree, which is a special case of a graph) where nodes are posts (and sometimes users), and edges represent reply/retweet relationships or other interactions. The GNN then learns node (or graph) representations which are used to classify whether the source post is a rumor or not (or into fine-grained categories like true rumor, false rumor, etc.).

Several state-of-the-art architectures have been proposed. We will review a few influential ones: **Bi-GCN** which uses a bi-directional propagation graph, **Hierarchical GAT** which uses a claim-guided attention mechanism, and others like graph-based multimodal and adversarial models. We also highlight what each assumes and contributes.

(Figure 4 about here: Illustration of a typical rumor propagation graph for a tweet. For example, show a source tweet node at the center, with reply tweets as children, possibly forming a tree.

Indicate how information flows can be modeled top-down (from source) and bottom-up (from replies). Perhaps show supporting vs denying replies in different colors. This figure can be used to explain propagation vs dispersion.)

4.1.1 Bi-GCN: Bi-Directional Graph Convolutional Networks

One seminal GNN model for rumor detection is **Bi-GCN**, proposed by Bian *et al.* (2020) ([2001.06362] Rumor Detection on Social Media with Bi-Directional Graph Convolutional Networks). Bi-GCN was designed to capture two key aspects of rumor propagation: (1) the *deep propagation* from the source outward, and (2) the *wide dispersion* of a rumor as it branches out. Previous deep learning methods like the tree RvNN focused mostly on the propagation chain (depth), but "ignored the structures of wide dispersion" ([2001.06362] Rumor Detection on Social Media with Bi-Directional Graph Convolutional Networks) – meaning they didn't explicitly leverage how a rumor might branch into multiple conversation threads. Bi-GCN addresses this by constructing two directed versions of the propagation graph and applying GCNs on both. Specifically:

- It builds a **top-down propagation graph**, which is essentially the tree of replies starting from the source post (root). Edges direct outward from the source. A GCN on this graph will aggregate information from the source down to replies, mimicking how a rumor disseminates.
- It also builds an **opposite (bottom-up) graph**, reversing the direction (edges from replies toward the source). A GCN on this graph propagates information from the leaves up towards the root, capturing how the various reply threads converge back to the source.

The model then combines these two GCN outputs. By doing so, Bi-GCN "explores both characteristics by operating on both top-down and bottom-up propagation of rumors" ([2001.06362] Rumor Detection on Social Media

with Bi-Directional Graph Convolutional Networks). Intuitively, the top-down GCN learns the *causal* propagation patterns (how an initial rumor spawns responses), while the bottom-up GCN captures the *converging* evidence from all the responses (as if piecing together the overall picture from all replies) (Examining the Limitations of Computational Rumor Detection Models Trained on Static Datasets). Moreover, Bi-GCN reinforces the role of the source tweet by injecting the source's features into each GCN layer's representation ("the information from the source post is involved in each layer") ([2001.06362] Rumor Detection on Social Media with Bi-Directional Graph Convolutional Networks). This ensures that the source tweet's content (which often is the claim itself) remains influential throughout the graph computation, which they found to improve accuracy.

The results from Bi-GCN were impressive: on the benchmark Twitter15 and Twitter16 datasets (which have four classes: Non-rumor, True, False, Unverified), Bi-GCN significantly outperformed prior models. For example, it achieved **88.6%** accuracy on Twitter15 and **88.0%** on Twitter16, whereas prior best models (like SVM with tree kernels or the RvNN) were in the 73–75% range (Rumor Detection on Social Media with Bi-Directional Graph Convolutional Networks) (Rumor Detection on Social Media with Bi-Directional Graph Convolutional Networks). It also maintained high F1 scores across all rumor classes (Rumor Detection on Social Media with Bi-Directional Graph Convolutional Networks). Bi-GCN's ability to consider both the "propagation" and "dispersion" structure gave it a clear edge. An important practical note is that Bi-GCN, by using two GCNs, roughly doubles the computation compared to a single GCN, but since these graphs (conversation trees) are not huge (usually hundreds of nodes at most in Twitter15/16), it's still quite efficient. The authors emphasize that their approach can detect rumors early as well: even after the first few waves of propagation, the model can achieve high accuracy (Rumor Detection on Social Media with Bi-Directional Graph Convolutional Networks). In an online setting, one could run Bi-GCN incrementally as new replies come in, potentially improving its prediction over time. Bi-GCN's success demonstrated that GNNs are very suitable for rumor detection, inspiring several follow-up

works.

(Figure 5 about here: Architecture of Bi-GCN. Show the rumor propagation tree, one GCN processing it top-down (source to leaves) and another bottom-up (leaves to source). Indicate that their outputs are combined (maybe concatenated) before final classification. Also show the source node's feature being added to each layer.)

4.1.2 Claim-Guided Hierarchical GAT (HGAT)

Another state-of-the-art model is the **Claim-Guided Hierarchical Graph Attention Network** proposed by Lin *et al.* (2021) ([2110.04522] Rumor Detection on Twitter with Claim-Guided Hierarchical Graph Attention Networks). We refer to it as *Hierarchical GAT for Rumor Detection (HGATRD)* for brevity. This model was introduced to tackle two issues: (1) existing methods often *oversimplified* the conversation structure or assumed a tree, and (2) they did not adequately model the interactions among responses in context of the *central claim*. Lin *et al.* argue that **conversation threads** on Twitter provide valuable clues – users reply not just to the source but to each other, creating a reply network, and their stances (supporting, denying, questioning) relative to the claim can be exploited. Some previous approaches only considered a strict parent-child reply relation (like a tree), or conversely, flattened everything ignoring structure. HGAT takes a middle ground by representing the entire conversation as an **undirected interaction graph** ([2110.04522] Rumor Detection on Twitter with Claim-Guided Hierarchical Graph Attention Networks). In this graph, each post (the source or a reply) is a node, and edges indicate an interaction (e.g., a reply or a mention between two posts). By using an undirected graph, the model can capture both the reply hierarchy and sibling relationships (two replies to the same post can influence each other).

The **hierarchical** part of the model comes from how it processes this graph. The model is "claim-guided" – meaning it pays special attention to the *target claim* (the source tweet's content) when aggregating information.

The network has two levels of attention: one at the **post level** and one at a higher **conversation level**. Essentially, it first learns representations of each reply post by attending to the words in the post (using an internal attention or possibly a language model) and considering the context of the conversation. Then it uses a **graph attention network** (GAT) over the interaction graph, but with a twist: it incorporates the claim's representation to guide the attention weights. In other words, when computing attention for an edge between two posts, the model factors in how relevant those posts are to the original claim. This helps "attend over the posts that can semantically infer the target claim" ([2110.04522] Rumor Detection on Twitter with Claim-Guided Hierarchical Graph Attention Networks). For example, if a particular reply directly refutes the claim, the model should pay more attention to it (as it's highly informative about the claim's veracity) than to a tangential off-topic reply.

Because of this approach, the model effectively *reinforces useful interactions* (like exchanges that are indicative of a rumor being challenged or affirmed) and *down-weights irrelevant chatter* ([2110.04522] Rumor Detection on Twitter with Claim-Guided Hierarchical Graph Attention Networks). Lin *et al.* report that this hierarchical GAT significantly outperformed prior methods on Twitter15, Twitter16, and PHEME datasets ([2110.04522] Rumor Detection on Twitter with Claim-Guided Hierarchical Graph Attention Networks) – for instance, their *ClaHi-GAT* model (Claim-guided Hierarchical GAT) reached about **89.1%** accuracy on Twitter15 and **90.8%** on Twitter16 ([2110.04522] Rumor Detection on Twitter with Claim-Guided Hierarchical Graph Attention Networks) – which was a new state-of-the-art, slightly higher than Bi-GCN's results. Notably, they also demonstrated improved **early rumor detection** capabilities ([2110.04522] Rumor Detection on Twitter with Claim-Guided Hierarchical Graph Attention Networks): the model was better at making correct predictions when only a few replies had occurred. This is crucial for practical use, since catching a rumor early, before it goes viral, is the goal.

One reason HGATRD excels at early detection is the claim-guided attention – even if few replies exist, the model is actively looking at how those

replies relate to the claim itself. If an early reply outright says "this is false, here's a source", the model can latch onto that. Traditional GCN or Bi-GCN might need a bit more evidence to accumulate. Additionally, by using an interaction graph (undirected), HGATRD does not force a specific propagation direction and can potentially utilize information even from *later* replies that appear earlier in the thread ordering (some datasets have replies not strictly chronological due to thread structure).

In summary, Hierarchical GAT brings the power of attention *within* the graph and aligns it with the semantic of the claim. It is a sophisticated architecture that pushes performance but at the cost of a more complex model. It likely uses a combination of text encoders (possibly BERT for the claim and replies) and two-tier attention (text-level and graph-level). This means it's heavier computationally than Bi-GCN (which only used GCN on relatively low-dimensional features). In terms of assumptions, HGATRD assumes we have the entire reply graph at once (like most methods) and that we can obtain meaningful semantic representations of the posts (they probably use pre-trained embeddings or a simple encoder). It also treats the conversation as undirected, which might ignore the temporal order of replies – however, they found this beneficial to maximize interaction info ([2110.04522] Rumor Detection on Twitter with Claim-Guided Hierarchical Graph Attention Networks). This implies that for their use case, *who replies to whom* and *what they say* mattered more than *when* they replied, at least for the final classification.

(Figure 6 about here: Schematic of the Claim-Guided Hierarchical GAT. Possibly two stages: (1) Represent each post (node) with its text features, perhaps with an attention over words (if applicable). (2) Construct the interaction graph of the conversation, and perform graph attention where the attention mechanism is influenced by the claim node. Highlight the claim node connected to others, perhaps with thicker edges to posts that are deemed relevant. This figure would illustrate the "claim-guided attention" concept.)

4.1.3 Other Notable Models

In addition to Bi-GCN and HGATRD, there have been other state-of-the-art models using GNNs for rumor detection:

- **Hierarchical Propagation Networks:** Some works combine tree-structured models and GNNs. For example, a *Tree-transformer* (with attention along tree edges) has been explored ([PDF] Debunking rumors on Twitter with tree transformer - InK@SMU.edu.sg), blending transformer-style attention with the propagation tree. This can be seen as an alternative way to inject attention in rumor propagation modeling.

- **Adversarial Learning on Graphs:** Yang *et al.* (2020) proposed a model with *Graph Structured Adversarial Learning* (Rumor Detection on Social Media with Graph Structured Adversarial Learning | IJCAI). They build a heterogeneous network including users, posts, and comments, and then simulate an attacker that tries to perturb the graph structure (e.g., by adding fake nodes or edges) to fool the rumor detector. The detector is trained simultaneously to resist these perturbations. This framework forces the GNN to learn more robust features that won't be swayed by small changes, thus improving generalization (Rumor Detection on Social Media with Graph Structured Adversarial Learning | IJCAI). The model achieved better results than prior methods on rumor datasets (Rumor Detection on Social Media with Graph Structured Adversarial Learning | IJCAI). Such adversarial robustness could be important if malicious actors try to manipulate the propagation patterns to evade detection.

- **Multi-modal and Knowledge-enhanced GNNs:** Rumors often come with images, or related web links, etc. Some recent methods extend GNNs to incorporate these. For instance, Wang *et al.* (2020) and Sun *et al.* (2021) introduced models that include visual nodes (for images) in the graph (Examining the Limitations of Computational Rumor Detection Models Trained on Static Datasets). The graph

might connect tweets to the images they share, allowing the GNN to spread information between textual and visual modalities. Similarly, if external knowledge (like fact-check databases or knowledge graphs) are available, those can be linked in. These multi-modal GNNs have shown improved performance especially on rumors where images are involved (e.g., a fake image being circulated) (Examining the Limitations of Computational Rumor Detection Models Trained on Static Datasets).

- **Contextual and Temporal Models:** Another line of work integrates user-specific or context-specific information. For example, the **DUCK** model by Tian *et al.* (2022) modeled *user networks* and *comment networks* separately and then combined them (Examining the Limitations of Computational Rumor Detection Models Trained on Static Datasets). Essentially, it built one graph of users (with edges if users interact or have certain relations) and another graph of posts, and used a dual GNN approach. DUCK achieved extremely high performance (reported 98% F1 on a Weibo dataset) by leveraging rich user attributes and network structure (Examining the Limitations of Computational Rumor Detection Models Trained on Static Datasets). The caveat (as we discuss later) is that such performance was measured on randomly split data and might not hold for truly unseen events. Temporal-order sensitive models attempt to perform *early detection*: e.g., they classify a rumor after a certain number of minutes or after k posts. Some methods explicitly evaluate at different time cutoffs ([2110.04522] Rumor Detection on Twitter with Claim-Guided Hierarchical Graph Attention Networks), and Hierarchical GAT we discussed did this, and others like Ma (2018) and Bi-GCN also reported early detection results. The challenge is to maintain good accuracy with limited propagation. Techniques like incremental GNN updates or combining streaming algorithms have been considered in recent research but remain challenging.

- **Heterogeneous Graph Networks:** Some approaches treat the rumor data as a heterogeneous graph with different types of nodes (tweet,

user, perhaps topic) and different types of edges (post–post reply, user–post, user–user follower). They then use specialized GNNs for heterogeneous data (like relational GCNs or meta-path based GNNs). An example is the use of user profiles: a user node connected to the tweets they wrote or shared allows propagation of credibility or historical behavior into the rumor classification. If a usually reliable user refutes a claim, that might be a strong signal. Models like PPC (Propagation, Path, Credibility) networks and others have dabbled in this direction, though complexity grows with more node types.

In summary, the progression in rumor detection models has gone from feature-based classifiers, to sequence models, to tree models, and now to various graph-based models with attention and additional bells and whistles. GNN-based architectures currently dominate the state-of-the-art in this space (Examining the Limitations of Computational Rumor Detection Models Trained on Static Datasets), thanks to their ability to naturally model propagation structure and jointly consider content, user, and network information. The best models often use a combination of techniques: graph neural networks for propagation, attention mechanisms for importance weighting, and sometimes external knowledge or adversarial training for robustness. Next, we will discuss some **challenges and assumptions** when applying these models, especially in real-world settings like live social media streams.

Chapter 5

Challenges and Assumptions

While GNN and attention-based models have proven highly effective for rumor detection in research settings, deploying them and obtaining robust performance comes with several challenges. Here we outline some key challenges and underlying assumptions:

1. Data Availability and Early Detection: A practical assumption in many studies is that the *propagation graph is largely available* by the time we classify a rumor. In reality, we often want to detect a rumor as early as possible – ideally when only a few reactions have occurred (because catching it early can limit its spread). Many GNN models, like Bi-GCN or HGATRD, use the full conversation thread, which means they achieve best accuracy only after the rumor has circulated for some time. If only 1–2 replies are present, the model has very little graph structure to leverage. Some models specifically evaluate *early rumor detection* by truncating the input (e.g., only first n replies or first t minutes) ([2110.04522] Rumor Detection on Twitter with Claim-Guided Hierarchical Graph Attention Networks). For instance, Ma's Tree-RvNN and Lin's Hierarchical GAT both demonstrated better-than-chance performance even with minimal input (Rumor Detection on Twitter with Tree-structured Recursive Neural Networks - ACL Anthology) ([2110.04522] Rumor Detection on Twitter with Claim-Guided Hierarchical Graph Attention Networks). Bi-GCN's results showed it maintained relatively high accuracy even "at a very early period after the source post"

(Rumor Detection on Social Media with Bi-Directional Graph Convolutional Networks). However, there is usually a trade-off: waiting for more propagation yields more evidence and higher confidence. Thus, a system must decide when it has seen "enough" to make a call. This challenge is not unique to GNNs, but any propagation-based model must grapple with it. One mitigation is to incorporate strong **content features** (like the text of the source post) so that even without any propagation, the model might detect an obvious false claim from text alone. Indeed, many models combine content and propagation for this reason.

2. Computational Complexity and Scalability: Graph neural networks, especially with attention, are more computationally intensive than simple classifiers. The GNNs we discussed operate on one rumor thread at a time (which might have tens or hundreds of nodes). In an outbreak of a rumor, or across a platform, there could be thousands of concurrent threads to analyze. Handling each with a GNN could become slow if not optimized. One challenge is that GNNs do a lot of matrix multiplications and neighbor lookups; on large graphs this can blow up, but in rumor detection the graphs are moderately sized. The bigger issue is perhaps the sheer number of events to analyze in real-time. For instance, if a platform like Twitter wants to run a rumor detector on every trending topic, it needs to process data continuously. Real-time applicability requires the model to be both fast and incremental. Some assumptions that researchers make are that the model can be run *offline* on static datasets, or that it can ingest the whole conversation at once. This ignores the streaming nature of social media. A real system might need to update the graph and output a new prediction every time a new reply comes in. Current GNN frameworks are not fully optimized for incremental updates (though some streaming GNN approaches exist). Another computational challenge is hyperparameter tuning and training – GNNs with attention have many parameters and can be slow to train. Techniques like neighbor sampling (GraphSAGE style) or limiting graph depth can help speed up training/inference.

3. Overfitting and Temporal Generalization: It has been observed that many rumor detection models risk overfitting to *specific events or time*

periods. If you randomly split a dataset into train and test, you might get artificially high performance because the same rumor (or very similar content) might appear in both sets ([2023]) ([2023]). For example, the PHEME dataset contains multiple tweets from the same news event; a random split can put some in train and some in test, making the task easier (the model can just learn the event's signature). A more realistic evaluation is a *chronological split* – training on earlier rumors and testing on later, unseen events ([2023]). Studies have shown that model performance drops significantly under such conditions, due to **temporal concept drift** ([2023]). Essentially, new rumors differ from old ones in topic, language, and propagation patterns. A challenge then is to build models that generalize to *emerging rumors*. This might require training on very diverse data and perhaps incorporating temporal adaptation. The assumption often made (in research papers) is that the distribution is roughly i.i.d. or at least that events in test were seen in train. This can be a weak point – a model that scored 90% on historical data might perform much worse on a brand new type of rumor. Solutions might include continual learning, using pre-trained language models that can handle new topics, and of course evaluating in a time-split manner to set realistic expectations (Examining the Limitations of Computational Rumor Detection Models Trained on Static Datasets). The latest research encourages using chronological splits for rumor detection evaluation ([2023]) to avoid optimistic results that won't hold in practice.

4. Feature Engineering and Embeddings: Although GNNs reduce the need for handcrafted network features, they still rely on good **node features** to start with. In rumor detection, what are the node features? Common choices are: textual embeddings of the post (e.g. averaging word vectors or using BERT), user features (e.g. user credibility score, follower count, account age), or temporal features (time since source tweet). Choosing and constructing these features is a form of *embedding engineering*. For instance, using a powerful language model embedding for each tweet can greatly enhance performance (because the model can understand the content better). Many earlier works used simple Bag-of-Words or TF-IDF vectors for text due to efficiency, but newer works leverage BERT or RoBERTa embeddings

for each tweet. The trade-off is computational cost – BERT embeddings are expensive to compute for thousands of tweets in real-time. A possible assumption in academic work is that one can use pre-computed BERT embeddings or even fine-tune BERT during training. In a real system, doing that on the fly might be infeasible, so one might cache user embeddings or limit to smaller models. Similarly, incorporating user profile features might require collecting additional data which may not always be accessible (privacy concerns or API limits). So, while feature engineering is less manual than before, there are still design choices: e.g., should we include a one-hot encoding of the source tweet's topic? Should we add sentiment features for each post? Each additional feature could help but adds complexity and potential noise.

5. Assumptions about Graph Structure: Most rumor GNN models assume the structure is a tree or a near-tree (few cycles). This holds for reply graphs in Twitter (which are indeed tree-structured by conversation). But if one were to incorporate retweet graphs or quote-tweets, those can introduce a more complex graph (retweets form a cascades that can converge if two users retweet each other's retweets, etc.). Some models simplify this by treating all retweets as direct children of the source (which is not exactly true but easier to model). Additionally, if we integrate user-user follow relationships, the graph becomes multiplex – potentially one huge connected component of the social network. To keep things tractable, models often treat each rumor's conversation in isolation, ignoring connections between different rumors or users across rumors. In reality, the same user might participate in multiple rumor threads, and a model could potentially benefit from knowing that (e.g., a user who spread past rumors might be more likely to spread the current one). However, incorporating that would make the graph enormous (linking all rumors via users). So the common assumption is that each rumor instance is independent and one can model its propagation graph separately. This is reasonable for evaluation (datasets are typically split by event), but it's an approximation of the real world.

6. Computational Resources and Real-Time Use: The use of attention (e.g., GAT) in GNNs improves accuracy but at a cost. A standard

GCN aggregates by a simple sum or average – this can be done by efficient sparse matrix multiplications. A GAT, on the other hand, requires computing e_{ij} for each edge and doing a softmax, which can be slower especially on CPU. If one needs to run a rumor detector in real-time (say, a service scanning tweets), using a GAT vs a GCN might be the difference between processing 100 events per second vs 20 per second. A challenge is to balance complexity and speed. One could assume high-performance GPUs are available which makes it fine, but not every deployment will have that. There's also a memory aspect: some models (like hierarchical ones) hold a lot in memory (text embeddings, graph structures). If deployed on edge or on devices, that might be too heavy.

7. Evaluation Metrics and Objectives: Rumor detection is often formulated as a classification problem (true rumor, false rumor, etc.). But in practice, one might be more interested in *prioritizing likely false rumors for human fact-checkers*. This is more like a risk detection than a strict classification. The models we have assume ground truth labels and optimize cross-entropy or similar. However, there are nuances: unverified rumors (those not yet confirmed true or false) are a category in some datasets. A model might learn to classify "unverified" as a separate class, but in application, distinguishing unverified vs false might be less important than distinguishing non-rumor vs rumor. Some works simplify the task to binary (rumor vs non-rumor). Others keep four classes. Depending on the end use, one might want to adjust the objective (e.g., high recall for catching any potential rumor). Many papers report accuracy or macro-F1 across classes, implicitly assuming all classes equally important. In reality, a false rumor might be more critical to catch than a true rumor (which is basically true information incorrectly flagged). So one might adjust thresholds or treat it as a ranking problem. These considerations are often outside the scope of architecture and more in how the system is used, but they are worth noting as challenges in translating research to practice.

In conclusion, the advanced GNN models come with assumptions of having a well-formed graph, enough data, and often operate in a static, offline way on benchmark datasets. Breaking these assumptions – dealing with par-

tial/incremental data, ensuring scalability, and generalizing to new rumors – is where ongoing research efforts lie. Our own application will specifically address some of these points by focusing on **embedding engineering** and efficiency, which we believe can improve real-time applicability without sacrificing accuracy (more on this in a later section).

Chapter 6

Datasets for Rumor Detection

Research on rumor detection has converged around a few key benchmark datasets, primarily collected from social media platforms where misinformation spreads. We will discuss the most commonly used datasets, with an emphasis on **Twitter** (Twitter15, Twitter16) and **PHEME**, as requested. We will also mention other relevant datasets such as Weibo (for Chinese rumors) and emerging datasets related to COVID-19 and fake news.

Twitter15 & Twitter16: These two datasets were introduced by Ma *et al.* (2017) and have become standard benchmarks ([2023]). They were collected from Twitter in 2015 and 2016 respectively, focusing on rumors that were tracked and verified by rumor debunking sites. Each dataset contains a set of **source tweets** (each initiating a thread about a topic/claim) along with the **conversation threads** (replies/retweets) and a **label** for the veracity of the rumor. The Twitter15 dataset has **1,490 source tweets** and Twitter16 has **818 source tweets** ([2023]). Each source tweet (event) is annotated into one of four classes:

- *Non-Rumor (NR):* the event is true or not a rumor at all (factual news).
- *False Rumor (FR):* the event is a false rumor.
- *True Rumor (TR):* the event is a true rumor (i.e., initially unclear but later verified true).

- *Unverified Rumor (UR):* the truth value remained unverified at the time of labeling.

These fine-grained labels make the task a four-class classification. In Twitter15, the distribution is roughly equal among the four classes ($\sim$ 370 each) ([2110.04522] Rumor Detection on Twitter with Claim-Guided Hierarchical Graph Attention Networks). Similarly for Twitter16, around 200 each. Each source tweet comes with a thread of tweets; some threads can be quite extensive (dozens of replies). The average number of comments (replies) in Twitter15 is about 22, and in Twitter16 about 16 (Examining the Limitations of Computational Rumor Detection Models Trained on Static Datasets), though the range varies (some rumors went viral, others not as much). The content is all in English (Twitter). These datasets also include user profile metadata and timing information which some models use as additional features (Examining the Limitations of Computational Rumor Detection Models Trained on Static Datasets).

(Visualization suggestion: A table or bar chart could be shown comparing Twitter15 vs Twitter16 vs PHEME statistics. For example, number of events, class distribution, average thread length, etc. This helps illustrate dataset characteristics. E.g., Figure: Dataset statistics for Twitter15, Twitter16, PHEME – showing count of rumors/non-rumors, etc.)

PHEME: The PHEME dataset (2017) is another widely used benchmark ([2023]). It was collected by Zubiaga *et al.* and others as part of the PHEME project. PHEME contains data from **9 real-world breaking news events** (such as the Ferguson unrest, Charlie Hebdo shooting, Ottawa shooting, etc.) ([2023]). Within those events, tweets were collected and annotated as rumor or non-rumor. In total, PHEME includes **5,802 tweets** (considered source tweets or rumor-centric tweets) with **two labels: Rumor or Non-rumor** ([2023]). Specifically, there are **1,972 rumors** and **3,830 non-rumors** in the dataset ([2023]). This is essentially a binary classification task. The conversations (replies) to those tweets are also included, forming threads. PHEME is different from Twitter15/16 in a few ways:

- It is *event-based*. The data is grouped by big events, which introduces topical homogeneity (all rumors from one event are about similar subject).
- It is binary labeled (no true/false/uncertain distinction, just whether it's a rumor or not).
- The dataset is slightly larger in total posts and has longer threads on average for some events (e.g., a big event might spark many discussions).
- It includes both English tweets and possibly some non-English if they were in those events (though majority are English since events were global news).

PHEME is often used to evaluate a model's ability to generalize across events. For instance, one evaluation is *leave-one-event-out*, training on 8 events and testing on the held-out event ([2023]), to see how the model handles an unseen topic. This is challenging due to the earlier mentioned concept drift. In such splits, performance typically drops compared to random splits.

Weibo (Chinese rumor dataset): Aside from English data, there is a notable dataset from **Weibo** (a Chinese microblog platform). Ma *et al.* (2016) collected a Weibo rumor dataset with **4,664 posts** (events) ([2023]). These are labeled as **true or false** (and perhaps non-rumor; some versions include non-rumor). According to one source, it has 2,313 rumors and 2,351 non-rumors ([2023]), making it balanced. The data includes the original post and the subsequent comments and re-posts, similar to Twitter threads. Weibo threads can be very large – on average, older Weibo rumors had hundreds of comments (the statistic in one study was $\sim$ 800 comments on average for a rumor on the Weibo16 dataset) (Examining the Limitations of Computational Rumor Detection Models Trained on Static Datasets). This is because Weibo allows very deep commenting and re-sharing. This means the propagation trees in Weibo data are much bigger, posing challenges for models (but also providing more information). Weibo data is written in Chinese, so it tests language-generalization of models (one needs Chinese

text processing). Some recent works include both an English dataset and the Weibo dataset to show effectiveness in both languages ([PDF] Improving Rumor Detection with User Comments - Shujun Li). Techniques like using multilingual BERT or language-agnostic features would be needed for that.

Emerging and Other Datasets: Beyond these, there are a few others:

- **CoAID (Cui & Lee 2020):** A COVID-19 misinformation dataset containing news articles and social media posts. It includes tweets related to COVID-19 fake news. DUCK (2022) used CoAID as one of testbeds (Examining the Limitations of Computational Rumor Detection Models Trained on Static Datasets). It's not purely a rumor propagation dataset, but a mix of data sources about COVID.

- **FakeNewsNet:** This is a collection for fake news detection, with two sub-datasets (GossipCop and PolitiFact) containing news articles and the social context (who shared them on Twitter). While focused on fake news (which is a broader category), it includes tweet propagation graphs that can be seen as rumor propagation. However, the style is slightly different (centered on news articles).

- **BuzzFeed/PolitiFact (rumor verification):** Some earlier work used datasets where each item is a news claim with associated tweets. They often require stance classification from replies. Not as common as the above for GNN models.

- **Emergent, TwitterEn, etc.:** There are a few smaller datasets (TwitterEn by Enayet & El-Beltagy, etc.) used in stance detection and rumor verification shared tasks. These are typically smaller and not graph-structured in the same way (maybe just collections of tweets labeled true/false).

- **Sun et al. Multi-modal (2021):** The "Sun-MM" dataset mentioned in one paper (Examining the Limitations of Computational Rumor Detection Models Trained on Static Datasets) appears to be a multi-modal rumor dataset with images. Possibly it's a subset of

Weibo or Twitter rumors that include images. It's less commonly used.

- **Pheme+COVID mixed sets:** With the pandemic, there have been new datasets combining COVID rumors from Twitter (e.g., Panacea dataset, etc.), but these are relatively new and not yet as established.

For our purposes, **Twitter15, Twitter16, and PHEME** are central. They provide a good benchmark trio: two fine-grained multi-class datasets and one binary event-based dataset. Many papers report results on all three ([2110.04522] Rumor Detection on Twitter with Claim-Guided Hierarchical Graph Attention Networks). Typically, models are trained and evaluated on Twitter15 and 16 separately, and sometimes on PHEME (either a random split or a cross-event split). The performance on Twitter15/16 is usually higher (since classes are balanced and content is more mixed), whereas PHEME can be harder if using leave-one-event-out.

When evaluating, researchers often use metrics like Accuracy and F1-score (macro-averaged or per class). For example, Bi-GCN reported accuracy $\sim$ 0.88 on both Twitter15/16 (Rumor Detection on Social Media with Bi-Directional Graph Convolutional Networks); Hierarchical GAT reported 0.89/0.91 ([2110.04522] Rumor Detection on Twitter with Claim-Guided Hierarchical Graph Attention Networks); and on PHEME, Hierarchical GAT got about 0.859 accuracy ([2110.04522] Rumor Detection on Twitter with Claim-Guided Hierarchical Graph Attention Networks), whereas earlier methods might be around 0.80–0.85 on PHEME. These numbers help gauge progress.

It's worth noting that in the **unverified rumor detection scenario**, the datasets have that as a class but it's a bit ambiguous – unverified means the truth isn't known, so how useful is it to classify something as unverified? It's basically saying "we're not sure." Some models struggle with that class because it's not clearly separable (it might look like a rumor but without resolution).

(Visualization suggestion: A figure could be included that visualizes an example from a dataset. E.g., "Figure: Example prop-

agation graph from Twitter16 – showing a source tweet (claimed event) and replies colored by stance or truth label." Or a pie chart of the class distribution in Twitter15 and Twitter16 highlighting the four categories.)

Now that we have an overview of datasets, we will indicate where in our document the **Experiments** will be described, which will include results on these datasets using our implemented models.

Chapter 7

Experiments and Results (Placeholder)

This section will be filled with details of our experiments. We will describe the experimental setup, training procedure, and present the results of our application on the aforementioned datasets. This includes performance metrics on Twitter15, Twitter16, and PHEME, comparisons to baseline models, and analysis of attention weights or case studies. We will also include tables or charts of the results. (For now, this is a placeholder indicating where the experiment details and findings should be inserted.)

(Figure 7 about here: Placeholder for a results table or chart. For example, a table comparing Accuracy/F1 of our model vs. baseline models on Twitter15, Twitter16, PHEME. Or a chart showing performance vs. time for early detection.)

Chapter 8

Application Utility and Embedding Engineering

In this section, we discuss the purpose and design of our **application** in the context of rumor detection. Our application is built with a focus on **embedding engineering** – that is, carefully designing and utilizing feature representations (embeddings) for the nodes in the rumor propagation graph to maximize the GNN's effectiveness. The motivation behind this is as follows: While many prior models propose new network architectures (GCN vs GAT vs hierarchical, etc.), we observed that the choice of input embeddings (textual features, user features, etc.) can significantly influence performance. Our application aims to simplify experimentation with different embedding configurations and ultimately improve rumor detection by feeding the GNN better information.

Purpose of the Application: The application serves as a **framework or pipeline** for rumor detection using GNNs. It allows ingestion of social media data (tweet threads, user metadata) and construction of a graph (propagation graph, possibly enriched with user nodes). It then provides modules to generate embeddings for each node:

- For *post nodes* (tweets): it can generate embeddings from various sources, e.g., TF-IDF vectors of the tweet text, pre-trained language model embeddings (BERT, RoBERTa), or even multi-modal embed-

dings if an image is attached. We place special emphasis on *text embeddings* because the language content of rumors often contains clues (certain words, phrases, or linguistic patterns can indicate misinformation).

- For *user nodes* (if included): it can produce an embedding from user profile features (like number of followers, verification status, etc.) or from the user's history of spreading rumors (if such data is available). In the simplest case, user features might be a vector of profile attributes or a learned embedding if we treat users as entities.

- For *structure*: while the structure is encoded by the graph edges, we might also include position embeddings (e.g., depth of the node in the reply tree, or time delay embeddings) to inform the model of temporal/structural context.

The application then feeds these embeddings into a GNN model (which could be a configurable choice: GCN, GAT, etc., but in our case we experiment with attention-based GNNs mostly). By isolating the embedding step, our framework makes it easy to plug in improved embeddings without changing the graph model. For instance, one can switch from using static GloVe vectors (Examining the Limitations of Computational Rumor Detection Models Trained on Static Datasets) to contextual BERT embeddings for the tweet content, and see how much the performance improves.

Role in Embedding Engineering: We call it *embedding engineering* because we actively design and test different representation methods. Traditional feature engineering might involve manually crafting features like "count of question marks in the tweet" or "sentiment score". Instead, we engineer the input in the representation space: for example, by including a 768-dimensional BERT encoding of the tweet, we implicitly give the model nuanced semantic information. If we include a user's credibility score in the embedding vector, we give the model a hint about the source reliability. Our application is set up to incorporate these seamlessly. This engineering is critical: a GNN can only propagate and mix the information that is present in

node features. If important cues (like a reply explicitly saying "this is fake") are not encoded in the initial features (perhaps because we used a bag-of-words that missed the nuance), the GNN might not catch it. So, we ensure rich embeddings that capture content, context, and user signals.

Anticipated Improvements: By improving embeddings, we anticipate several benefits over existing GNN frameworks:

- **Higher accuracy/F1:** Better text embeddings (especially from transformer-based language models) should allow the model to better differentiate a false rumor from a true one based on linguistic cues and context. Many older models trained word vectors on the small rumor dataset; we use pre-trained models that have seen a lot of language, which should generalize better to unseen phrasing.

- **Generalization to new topics:** A well-trained language model embedding can represent sentences about any topic in a consistent semantic space. This could help with the temporal generalization problem – our model might understand a new rumor's text better if, say, BERT has seen similar terms in general training. Thus, even if our GNN was trained on older data, the embeddings for a new COVID rumor might still place it near other conspiracy-like statements in embedding space, making detection easier. In contrast, a model without such embeddings might be lost with new jargon.

- **Reduced need for deep propagation:** If content embeddings are very informative, the GNN doesn't have to compensate by looking at many hops. For example, the claim tweet itself might be very likely fake (like "Breaking: The moon turned green tonight!"), and a strong content embedding might let the classifier deem it suspicious even with minimal propagation. This doesn't mean we ignore propagation, but it means the model can rely on both propagation and content more evenly.

- **Flexible architecture:** Our application can test multiple architectures (GCN vs GAT vs others) with the same embeddings to fairly

benchmark them. By holding the embeddings constant, we can see the genuine impact of the architecture. Conversely, we can take a strong architecture (say GATv2) and evaluate different embedding schemes to quantify their effect. This will be detailed in the experiments.

Another utility of the application is it provides **interpretability hooks**. With attention mechanisms, we can extract the attention weights to see which neighbors were most influential. Since we carefully feed in features, we can interpret, for example, that "the model paid attention to a reply from user X saying *'this is not true'* with weight 0.8, indicating it heavily used that reply to judge the rumor as false." We plan to visualize such attention distributions for insights. Our application might output these as part of a dashboard or log, aiding users or researchers in understanding the model's reasoning.

Comparison with Existing Frameworks: Existing GNN frameworks for rumor detection (if any publicly available) might not be flexible in terms of changing the input features easily – they might be hard-coded to use one-hot encodings or simple content features. Our tool is built to easily integrate with external NLP models and data sources. In a sense, it serves as a unifying platform to experiment with ideas from various papers:

- One can replicate Bi-GCN in it, and then try swapping the input from TF-IDF to BERT.
- One can replicate a hierarchical attention by configuring two layers and guided attention (though that might require custom coding, but the modular design helps).
- We can incorporate user graphs akin to DUCK by simply adding user nodes and edges in the input construction, and the same GNN part will handle it (with minor changes to encode different node types).
- The application could also be extended to new social media (with different data schema) by writing a new data parser but reusing the embedding+GNN pipeline.

Results expectation: We expect that with improved embeddings, our model will outperform many existing models that were evaluated on the same datasets. For example, if Bi-GCN (original) got 0.886 accuracy on Twitter15 with basic features, our version of a GAT or GATv2 with RoBERTa embeddings might push above 0.90 on the same split. The improvement comes not just from architecture but from richer initial information. In experiments, we anticipate seeing a jump in performance when switching on advanced embeddings. We also anticipate improved *early detection* metrics – e.g., at 5 replies in, maybe our model already hits a decent accuracy because even those few replies are encoded with a powerful language model (possibly catching negation, sarcasm, etc., better than before).

In summary, the application's utility lies in bridging the gap between **cutting-edge language understanding** and **graph-based propagation modeling**. By doing so, it aims to deliver a more powerful rumor detector that builds on existing GNN frameworks and enhances them. The results from our tests (to be inserted in the Experiments section) will illustrate the gains from this approach.

(Figure 8 about here: A flow diagram of our application's pipeline. E.g., input raw data → feature extraction (embedding engineering) → graph construction → GNN model → output prediction. Highlight where embedding choices come in (text encoder, user features).)

Chapter 9

Comparison with State-of-the-Art Models

Finally, we place our approach in context with other key state-of-the-art (SOTA) models in rumor detection, and summarize how they compare in terms of architecture and effectiveness. We've already discussed many of these models individually; here we'll explicitly compare their performance (as reported in literature) and their strengths/weaknesses:

Performance Benchmarks: In literature, the following are some top results on the common datasets:

- **Bi-GCN (Bian et al. 2020):** As noted, achieved $\sim 88.6\%$ accuracy on Twitter15 and $\sim 88.0\%$ on Twitter16 (Rumor Detection on Social Media with Bi-Directional Graph Convolutional Networks), and significantly improved early detection. It became a strong baseline for graph-based rumor detectors.

- **ClaHi-GAT (Lin et al. 2021):** Reported $\sim 89.1\%$ (Twitter15), $\sim 90.8\%$ (Twitter16), and $\sim 85.9\%$ on PHEME ([2110.04522] Rumor Detection on Twitter with Claim-Guided Hierarchical Graph Attention Networks), outperforming Bi-GCN and others. Particularly notable was its higher performance on Twitter16 and PHEME, and better early detection.

- **PPC/BERTRumor (Rao et al. 2021):** This was an approach that incorporated stance classification and pre-trained BERT. It's mentioned in context as using contextual info (Examining the Limitations of Computational Rumor Detection Models Trained on Static Datasets). While exact numbers vary, many such models got in the mid-to-high 80s on Twitter15/16. They showed that adding contextual (conversation) features to BERT improves over just BERT on the source tweet.

- **DUCK (Tian et al. 2022):** This model combining user and comment propagation networks achieved exceptionally high results on Weibo (98% F1) (Examining the Limitations of Computational Rumor Detection Models Trained on Static Datasets) and strong results on a COVID dataset (CoAID). On Twitter15/16, it's not directly reported in that snippet, but presumably also high. However, an important note (from the "limitations" paper (Examining the Limitations of Computational Rumor Detection Models Trained on Static Datasets)) is that these results are on random splits. When evaluated on chronological splits, performance can drop. So, DUCK might be near-perfect on Weibo16 random split, but on a realistic split it would be lower. Still, it underscores that incorporating user network helps a lot on that dataset.

- **GAT vs GCN Baselines:** One interesting comparison reported in some studies is between using a GCN vs a GAT on the same data. Generally, GAT outperforms GCN modestly because it can focus on important neighbors. For example, one could compare a plain graph convolution (like "UD-GCN" – undirected GCN) to a GAT on Twitter data. Lin et al. actually did an ablation: using directed trees in their model vs their full interaction graph, etc., showing that their attention model gave a boost of several percentage points ([2110.04522] Rumor Detection on Twitter with Claim-Guided Hierarchical Graph Attention Networks).

- **TreeLSTM (Ma et al. 2018):** For reference, their accuracy on Twitter15 was around 75% (Rumor Detection on Social Media with Bi-Directional Graph Convolutional Networks) (they reported significantly better than earlier baselines like SVM-TK which was $\sim$ 75% as well). So, the jump from 75% (TreeLSTM) to $\sim$ 88% (Bi-GCN) in two years is huge – mainly due to the power of GNN and possibly better features.

Key Components Comparison:

- **Use of Attention:** Among SOTA, models like ClaHi-GAT and Bi-GCN++ variants use attention. Bi-GCN in its original form did not use attention on the graph (just two GCNs). Later works either applied attention to the graph edges or at least to combining the two GCN outputs. Generally, attention models (GATs) have an edge in performance, as seen by GAT-based models topping leaderboards. Our approach also leverages attention (potentially GATv2), which aligns with this trend.

- **Use of Contextual Embeddings:** Recent SOTA all tend to use pretrained language model embeddings for text, whereas older ones might use TF-IDF or shallow embeddings. This is a bit implicit: e.g., Lin 2021 likely used BERT to encode the claim and replies (given the timeline, it's likely). Rao 2021 explicitly used BERT. The infusion of transformers for text content has boosted performance. Our approach does the same, which we expect will match or exceed SOTA on that front.

- **Graph Structure:** Some SOTA use just the reply tree (Bi-GCN, Ma's TreeLSTM), while others use more enriched graphs (Lin's undirected graph, DUCK's dual graphs including user relations, Yang 2020's heterogeneous graph with user nodes). There is evidence that adding more structure (undirected to allow sibling connections, or adding user-user edges) can help, but it also adds complexity. Lin 2021 found using the undirected version of threads helped over strictly

using directed trees ([2110.04522] Rumor Detection on Twitter with Claim-Guided Hierarchical Graph Attention Networks). DUCK found using user network helps a lot in Weibo where certain users are super-spreaders. Our approach can incorporate these ideas (though our initial experiments might stick to the propagation tree).

- **Robustness:** Adversarial training (Yang 2020) and temporal evaluation (Søgaard et al. 2021) are newer considerations. Not all SOTA address these. It's possible that a simpler model might outperform a complex one on *future data* if it overfits less. For instance, a robust GCN with regularization might beat a fancy hierarchical model on completely new topics. There's ongoing discussion in the community about this (Examining the Limitations of Computational Rumor Detection Models Trained on Static Datasets). We aim for a model that is not only high-performing on benchmarks but hopefully more robust (embedding-based approach might help in generalization, as argued).

Results of Our Model: While we haven't inserted the actual numbers yet, we anticipate our model (let's call it *Our GNN*) to be competitive. For example, suppose our GATv2-based model with embedding engineering achieves 0.91 on Twitter16 and 0.89 on Twitter15, it would slightly exceed ClaHi-GAT. On PHEME, maybe we get $\sim$ 0.87 accuracy, which is above 0.859 of ClaHi-GAT. These hypothetical numbers would place it at the state-of-the-art level. If we incorporate user features like DUCK, perhaps we'd boost further on certain datasets. We will compare such numbers in a table. The comparison will likely include:

- Our model vs Bi-GCN vs Hier-GAT vs RvNN vs possibly a text-only BERT baseline (for insight: e.g., just classify the source tweet with BERT gets maybe 70% – showing propagation gives the additional boost).

Literature Comparison: To ensure we are up-to-date: as of 2023, the state-of-the-art rumor detectors indeed are graph-based hybrids. No radically different approach (like pure transformer on concatenated tweets) has beaten

them, to our knowledge. GNNs with attention or knowledge are leading. One emerging direction is using *Graph Transformers* (applying full transformer architectures to the graph). Some papers have tried positional encodings on graphs to use a Transformer, but it's complex and not necessarily significantly better than GAT. Another direction is *COVID-specific models* that might incorporate epidemiological knowledge, but that's niche.

It's also useful to compare qualitatively:

- **Bi-GCN:** Simple, efficient, bidirectional propagation modeling. Weakness: no neighbor discrimination (all neighbors equal in GCN), no direct handling of irrelevant replies except via the second graph.
- **HGAT (Lin):** Complex, uses attention, claim guidance. Strength: great performance, early detection. Weakness: more parameters, might overfit if claim text wrongly guides (if all replies are off-topic maybe claim guidance confuses).
- **Our model:** We position it as taking the best of both – using attention (like GATv2) but also strong embeddings to ensure even a simpler architecture can excel. If we didn't implement hierarchical claim guidance, we might not explicitly handle irrelevant replies, but the attention mechanism itself can down-weight irrelevant neighbors to some extent.

Summary of Comparisons: We can summarize in a small table (in text form):

Model	**Architecture**	**Key Features**
Ma et al. (2018) TreeRNN	Top-down & bottom-up Tree LSTMs	Propagation only (no atte
Bian et al. (2020) Bi-GCN	2 GCNs (TD + BU)	Propagation + source boos
Lin et al. (2021) HGAT	Hierarchical GAT, claim-guided	Interaction graph, multi-le
DUCK (2022)	GCN on comment + user graphs	User features, context, mu
Our Model (2025)	GATv2 (attention GNN)	Enhanced embeddings (BE

(The above table is illustrative; actual experiment results will be inserted.)

From literature, it's clear that incorporating **attention** and **rich features** yields the best results. The gap between models like Bi-GCN and Hier-GAT shows the value of attention and better use of context. Additionally, the near-perfect result on Weibo by DUCK shows the importance of using *user networks and profile features* in certain contexts (Weibo has many users repeatedly spreading rumors, so user-level patterns emerge strongly – capturing that almost solves the problem for that dataset).

In our approach, if needed, we could integrate user features too. If we do, we expect similar boosts on datasets that have such signals. However, on Twitter15/16, user info is not as predictive as content/propagation because those datasets don't have as many repeated user behaviors (also they are smaller).

One more state-of-the-art worth mentioning: Some works have tried end-to-end stance + veracity prediction, where first each reply is classified as support/deny and then aggregated (the classic two-step approach by Zubiaga et al. 2018). End-to-end GNNs kind of fold that into one. But if we compare, those two-step approaches got lower performance typically than the newer GNNs.

Conclusion of Comparison: Our approach is aligned with the state-of-the-art trend: use graph structure + attention + transformer-based embeddings. We expect it to either match or set a new state-of-the-art on the given benchmarks. The true test will be generalization and efficiency, which we have designed for. By demonstrating competitive results and possibly better robustness, we contribute an approach that is not only effective but also practical.

(Figure 9 about here: perhaps a bar graph comparing model accuracies on a dataset, for visual comparison of SOTA. E.g., bars for TreeRNN, Bi-GCN, HGAT, OurModel on Twitter16. Or a line chart showing improvement over years.)

Chapter 10

Visualizations and Interpretability

Throughout this document, we have indicated points where figures should be included to aid understanding. Here we summarize and clarify the visualization strategy for a final report:

- **Model Architecture Diagrams:** We suggested multiple figures to illustrate how models like GAT, Bi-GCN, and Hierarchical GAT work. For instance, **Figure 3** (Graph Attention mechanism) would depict nodes and attention weights. **Figure 5** (Bi-GCN architecture) shows two parallel GCN flows. **Figure 6** (Hierarchical GAT) shows the two-level attention with the claim node. These diagrams help readers grasp the structural differences between models.
- **Dataset Distribution Visuals:** We proposed a figure (in the Datasets section) showing dataset statistics (maybe a table or bar chart). Visualizing class distribution (e.g., a bar for each class in Twitter15) or the number of events vs average thread length can contextualize the task difficulty (unbalanced data, etc.). For example, a bar chart for Twitter15 showing 374 Non-rumor, 370 False, 374 True, 372 Unverified ([2110.04522] Rumor Detection on Twitter with Claim-Guided Hierarchical Graph Attention Networks), vs Twitter16 with $\sim$ 200 each, vs PHEME with 1972 rumors and 3830 non-rumors ([2023]).

- **Attention Weight Visualization:** If possible, a figure could show an actual example of a rumor thread with attention weights from our model. For instance, **Figure X**: a small reply graph where the source tweet is in the center, and arrows to replies weighted by attention (thicker arrow = higher weight). This could illustrate how the model focused on, say, one particular reply that debunks the rumor. Such a figure provides interpretability. We can obtain these weights from our trained GAT model. Perhaps in experiments, we identify a case to visualize.

- **Performance Charts:** In the Experiments/Results section, a plot could show how accuracy improves as more tweets in the thread arrive (for early detection evaluation). For example, a line graph where the x-axis is the number of replies seen and the y-axis is accuracy, comparing Bi-GCN vs OurModel. If our model has a steeper rise, it shows better early detection. Another chart could be a simple bar graph comparing final F1 scores of different models. Visual presentation of results often makes the comparisons clearer than just text or tables.

- **Embedding Space Visualization:** Although not explicitly requested, one possible visualization is to use PCA or t-SNE to project the learned rumor vs non-rumor representations to see if they cluster. For example, after the GNN, do rumor events cluster separately from non-rumors in embedding space? This could illustrate the model's ability to separate classes. However, this may be extra.

Each figure is labeled and captioned so that readers can easily follow the discussion and understand the key aspects of the methodology and results.

Bibliography

[1] Bronstein, M. M., Bruna, J., LeCun, Y., Szlam, A., & Vandergheynst, P. (2017). Geometric deep learning: going beyond Euclidean data. *arXiv preprint arXiv:1611.08097*.

[2] IBM. What is an attention mechanism? Retrieved from `https://www.ibm.com/think/topics/attention-mechanism`.

[3] Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

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

[5] Hamilton, W. L., Ying, R., & Leskovec, J. (2017). Inductive representation learning on large graphs. *arXiv preprint arXiv:1706.02216*.

[6] Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., & Bengio, Y. (2017). Graph attention networks. *arXiv preprint arXiv:1710.10903*.

[7] Brody, S., et al. (2021). How attentive are graph attention networks? *arXiv preprint arXiv:2105.14491*.

[8] Bian, X., et al. (2020). Rumor Detection on Social Media with Bi-Directional Graph Convolutional Networks. *arXiv preprint arXiv:2001.06362*.

[9] Ma, J., et al. (2018). Rumor detection on Twitter with tree-structured recursive neural networks. In *Proceedings of ACL*.

[10] Lin, et al. (2021). Rumor Detection on Twitter with Claim-Guided Hierarchical Graph Attention Networks. *arXiv preprint arXiv:2110.04522*.

[11] Examining the Limitations of Computational Rumor Detection Models Trained on Static Datasets. *arXiv:2309.11576v2* (2023).

[12] Debunking Rumors on Twitter with Tree Transformer. InK@SMU.edu.sg (2021). Retrieved from `https://ink.library.smu.edu.sg/cgi/viewcontent.cgi?article=6602&context=sis_research`.

[13] Yang, et al. (2020). Rumor Detection on Social Media with Graph Structured Adversarial Learning. In *Proceedings of IJCAI 2020*.

[14] Li, S. (2022). Improving Rumor Detection with User Comments. Retrieved from `http://www.hooklee.com/Papers/TrustCom2022.pdf`.

[15] Tian, et al. (2022). DUCK: Rumour detection on social media with dual propagation networks. (2022).

[16] Rao, et al. (2021). BERTRumor: Exploiting semantic and sentiment information for rumor detection. (2021).

[17] Sun, et al. (2021). Multi-modal Rumor Detection with Graph Neural Networks. (2021).