Name	**Roll number**	**Paper**
Hyder Presswala	16010122151	1
Soumil Mukhopadhyay	16010122257	2
Akshad Kakad	16010122258	3

Paper 1: ***"Concept Drift Adaptation in Text Stream Mining"***

---

1) Introduction & Motivation

- **Focuses on adapting machine learning models to text data streams affected by concept drift.**

  Explanation:

  In text stream environments—such as social media, e-commerce reviews, or news feeds—the distribution of data changes over time. These shifts, referred to as *concept drift*, make it difficult for traditional static machine learning models to maintain accuracy over long periods. This paper highlights the growing need to develop adaptive models that can learn continuously and respond effectively to new patterns in the data without requiring full retraining. The motivation stems from the observation that as language usage evolves (e.g., changes in slang, trends, or sentiment), models need mechanisms to detect and adapt to these changes to ensure reliable text classification.

Mapped to CO2 – Analyze patterns in streaming data
Mapped to CO4 – Understand the concepts of information retrieval and

web search

---

2) Methodology Used

- **Applies a systematic literature review to evaluate 48 papers on text stream mining.**

Explanation:

The authors conducted a thorough systematic literature review (SLR) to assess recent contributions addressing concept drift in the context of text streams. They applied a structured protocol to search, screen, and analyse 48 research papers published between 2018 and 2024. These studies were then classified based on the types of concept drift handled (real, virtual, or semantic), the detection strategies (explicit or implicit), and the model update mechanisms (retraining, incremental learning, or ensemble adaptation). This SLR methodology allowed the researchers to derive insights into the prevailing trends, gaps, and challenges in adaptive text stream mining.

Mapped to CO2 – Analyze patterns in streaming data
Mapped to CO1 – Describe the fundamental issues and challenges of mining complex data

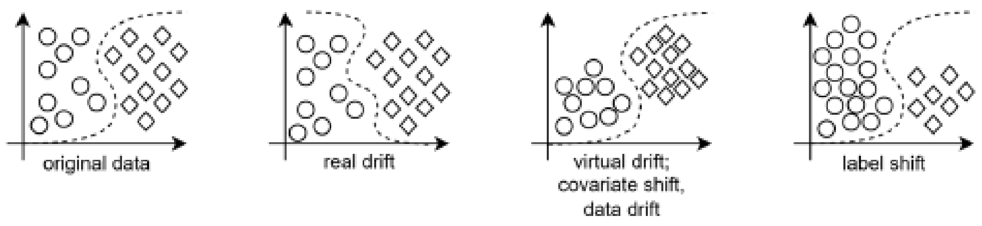


Fig. 2. Types of concept drift. Adapted from Gama et al. [62]. Each marker, i.e., circle, and diamond, represents an arbitrary class/label. Dashed lines correspond to the border between regions of classes.

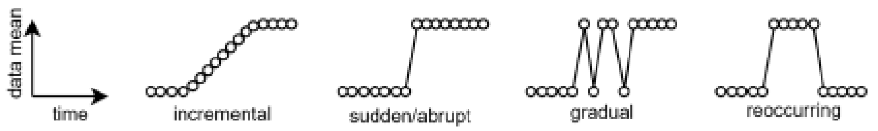


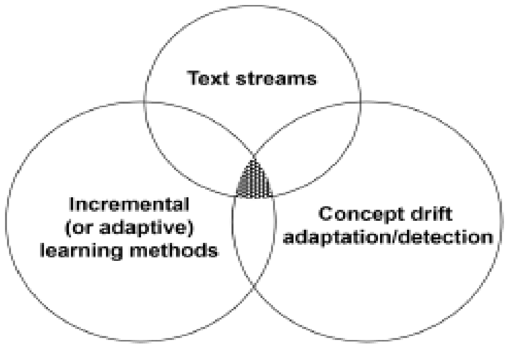
Fig. 3. Dynamics of concept drift over time. Adapted from Gama et al. [62].

- **Classification of methods based on text representation and update schemes.**

Explanation:

A significant part of the review focused on how different models represent textual data and how these representations adapt to drift. Techniques like TF-IDF, Word2Vec, and contextual embeddings (e.g., BERT) were assessed. Some models utilized fixed representations, while others employed adaptive representations that evolve with the data. The study emphasized the need for dynamic embeddings that can detect semantic drift, ensuring that classifiers stay relevant as meanings and associations between words change over time.

Mapped to CO4 – Understand the concepts of information retrieval and web search


Text streams
Incremental (or adaptive) learning methods
Concept drift adaptation/detection

---

3) **Performance Evaluation and Results**
   - **Analyses models using drift-specific metrics like detection delay and classification accuracy.**

Explanation:

The paper examined how existing approaches are evaluated, highlighting the metrics used to measure their effectiveness. These include detection delay (how quickly the model detects drift), false positives (incorrect drift detections), and classification accuracy (pre- and post-drift). Adaptive learning methods, especially those using ensembles that adjust weights or replace outdated models, demonstrated higher resilience to both abrupt and gradual drift. Additionally, methods incorporating explicit drift detection mechanisms like DDM or ADWIN achieved faster response times with fewer false alarms. However, many models failed to explicitly address semantic drift, which remains an important area for future improvement.

Mapped to CO2 – Analyze patterns in streaming data
Mapped to CO4 –Understand the concepts of information retrieval and web search

4) Comparison Analysis

- **Contrasts different strategies: explicit vs implicit detection, and incremental vs retraining.**

Explanation:

This section compares various approaches to drift detection and model updating. Explicit detection methods, such as statistical change detectors (e.g., ADWIN), offer transparency and direct control, but may introduce delay or false alarms. Implicit methods, on the other hand, rely on performance degradation cues within models, which may react slower but require fewer assumptions. Similarly, incremental learning and ensemble techniques offer smoother updates with minimal computation, whereas retraining methods, though potentially more accurate, are computationally expensive and impractical for real-time use. The review notes that very few models currently integrate mechanisms to detect and adapt to semantic drift, revealing a significant research opportunity.

Mapped to CO1 – Describe the fundamental issues and challenges of mining complex data

Mapped to CO2 –Analyze patterns in streaming data

5) Future Scope

- **Proposes deep learning, semantic modeling, and cloud deployment as future directions.**

Explanation:

The paper outlines several areas for advancing concept drift adaptation. First, the integration of deep learning architectures, such as transformer-based models (BERT, GPT), is encouraged for their ability to capture nuanced semantic changes in language. These models can offer more context-aware representations that improve with streaming inputs. Second, the authors suggest leveraging hierarchical clustering and semantic embedding tracking to better identify subtle concept shifts. Lastly, the paper emphasizes the need for scalable, cloud-native solutions that allow for distributed learning and real-time updates. The creation of benchmark datasets with annotated drift events is also proposed to enable consistent evaluation across models.

Mapped to CO1 –Describe the fundamental issues and challenges of miningcomplex data

Mapped to CO4 – Understand the concepts of information retrieval and web search

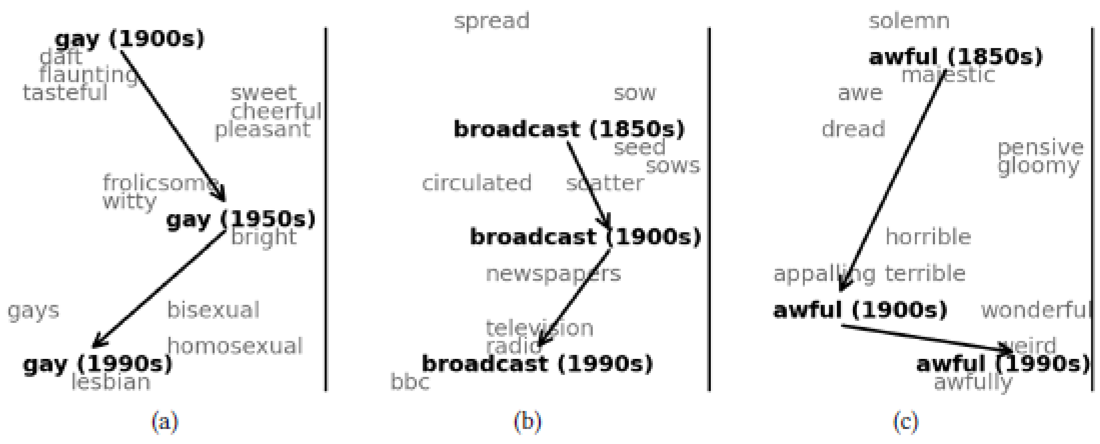


Fig. 4. Semantic shift across several decades or centuries. Adapted from Hamilton et al. [77].

Paper 2: ***"A Mining Frequent Itemsets Algorithm in Stream Data"***

---

1) Introduction & Motivation

- **Introduces SWFIUT-stream to improve mining of frequent itemsets in evolving data streams.**

Explanation:

This paper addresses the limitations of traditional frequent pattern mining algorithms like FP-Growth when applied to stream data. These algorithms are designed for static datasets and require multiple scans, which are impractical for real-time applications. SWFIUT-stream is introduced as an adaptive, sliding-window-based algorithm that mines frequent itemsets with minimal memory and computational overhead. The motivation is to handle dynamic and high-velocity data streams efficiently, by capturing the most relevant patterns in real time and adapting to evolving transaction behavior.

Mapped to CO2 –Analyze patterns in streaming data

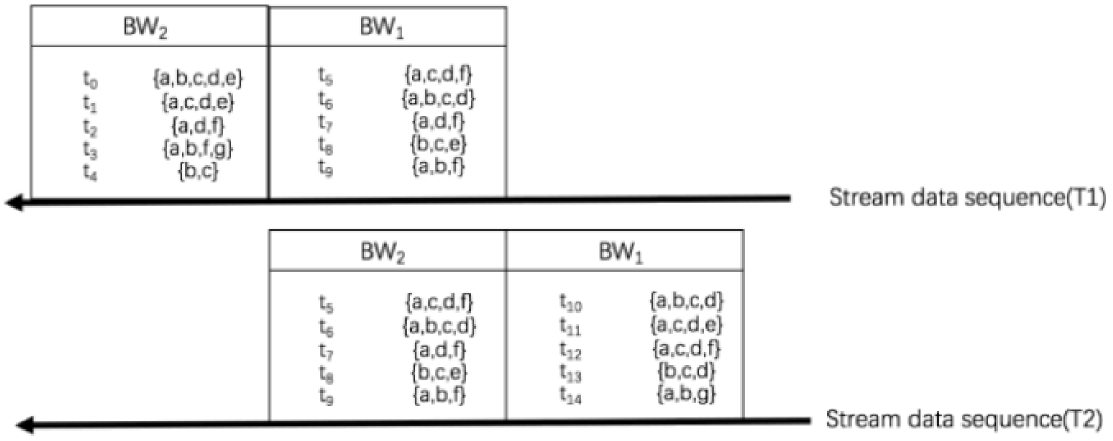


Figure 1. The sliding time window.

2) Methodology Used

- **Uses a sliding window framework where each window unit stores a fixed batch of transactions.**

  Explanation:

  The algorithm structures incoming stream data into fixed-size window units, which allows recent data to be processed while older transactions are phased out. This helps the model stay current and responsive to changing data patterns without overwhelming memory.

  Mapped to CO2 – Analyze patterns in streaming data

- **Applies a time decay factor (η) to reduce the weight of older transactions.**

  Explanation:

  The decay factor assigns exponentially decreasing weights to older transactions, ensuring that recent events have more influence on the discovery of frequent itemsets. This is crucial in adapting to concept drift, where older data becomes less relevant.

  Mapped to CO2 – Analyze patterns in streaming data

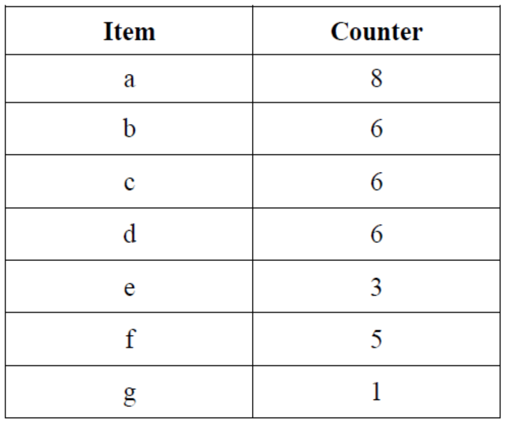
- **Maintains a 1-itemset counter table for efficient frequency tracking.**

  Explanation:

  A compact table tracks how many times each item has occurred in the sliding window. This lightweight structure enables rapid frequency calculation and supports real-time updates without needing to reprocess the entire window.

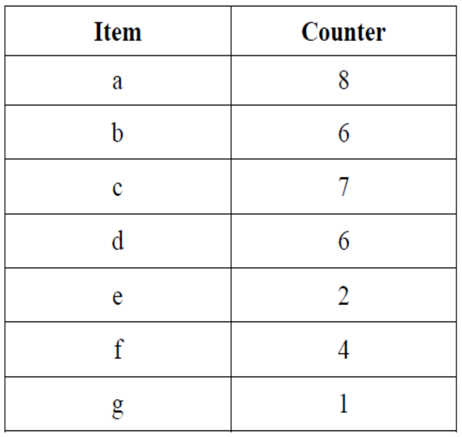
Mapped to CO2 –Analyze patterns in streaming data

Table 1. 1-Itemset counter table data



Item	Counter
a	8
b	6
c	6
d	6
e	3
f	5
g	1

Table 5. 1-Itemset counter table data at T2 time



Item	Counter
a	8
b	6
c	7
d	6
e	2
f	4
g	1

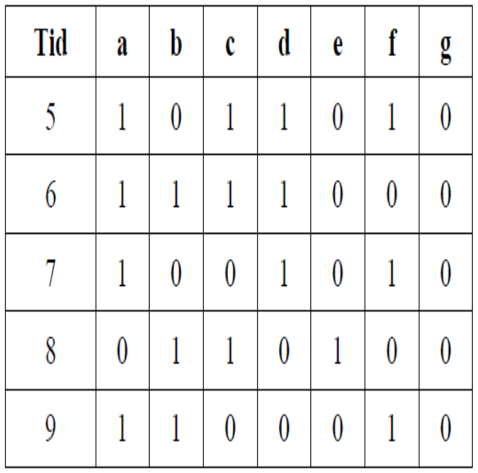
- **Implements a two-dimensional transaction table across window units.**

Explanation:

This matrix stores transactions and item presence in binary form (1 or 0). It allows efficient scanning and quick identification of frequent combinations, supporting high-throughput processing needs.

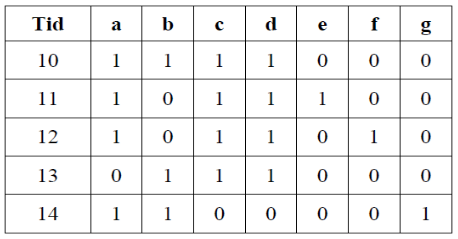
Mapped to CO2 –Analyze patterns in streaming data

Table 2. Data of BW_1 table



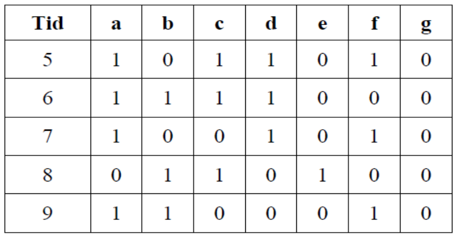
Tid	a	b	c	d	e	f	g
5	1	0	1	1	0	1	0
6	1	1	1	1	0	0	0
7	1	0	0	1	0	1	0
8	0	1	1	0	1	0	0
9	1	1	0	0	0	1	0

Table 3. Data of BW_1 table at T2 time



Tid	a	b	c	d	e	f	g
10	1	1	1	1	0	0	0
11	1	0	1	1	1	0	0
12	1	0	1	1	0	1	0
13	0	1	1	1	0	0	0
14	1	1	0	0	0	0	1

Table 4. Data of BW_2 table at T2 time



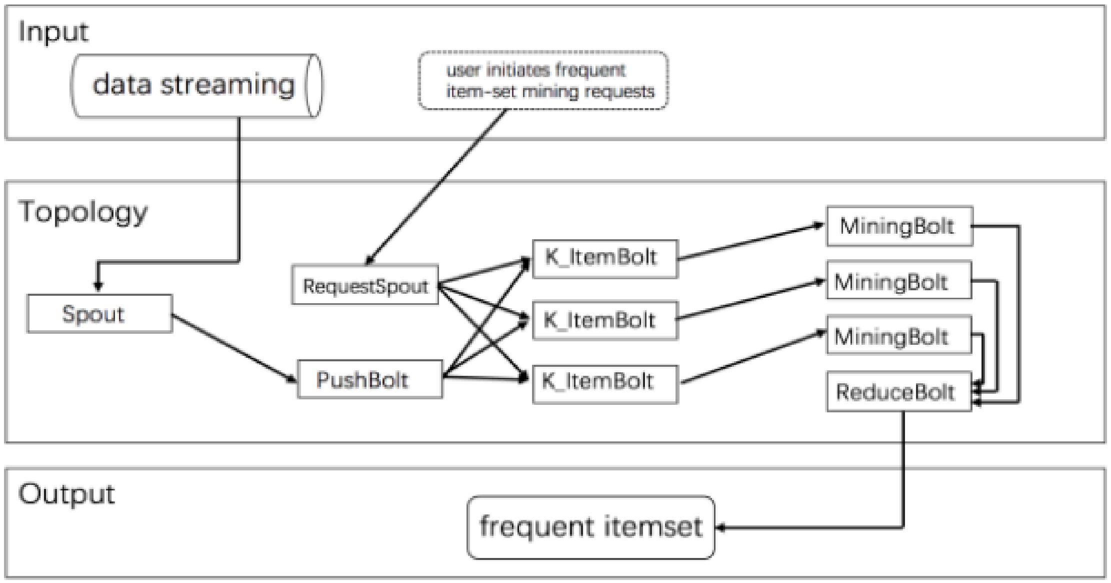
Tid	a	b	c	d	e	f	g
5	1	0	1	1	0	1	0
6	1	1	1	1	0	0	0
7	1	0	0	1	0	1	0
8	0	1	1	0	1	0	0
9	1	1	0	0	0	1	0

- **Runs on Apache Storm for real-time, distributed processing.**

Explanation:

The algorithm is built on Apache Storm, which supports distributed stream processing. This allows the model to scale across nodes and maintain low-latency updates even under large data volumes, ideal for enterprise-level applications.

Mapped to CO1 –Describe the fundamental issues and challenges of mining complex data



- **Enhances the classic FIUT algorithm to reduce redundant computations.**

Explanation:

The original FIUT method involved multiple passes over the data, increasing computational load. This improved version optimizes the process by updating only what's necessary, minimizing delays and supporting single-pass processing.

Mapped to CO2 – Analyze patterns in streaming data

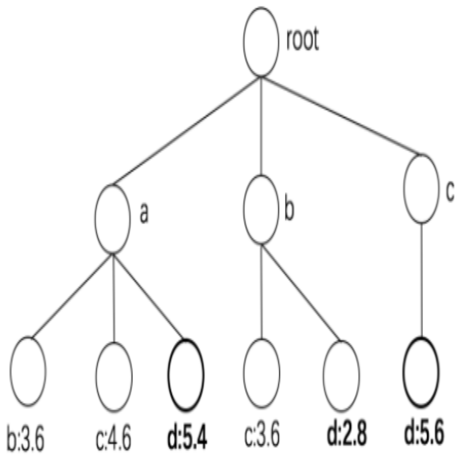


Figure 2. 2-FIU-tree.

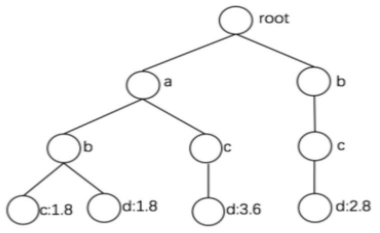


Figure 3. 3-FIU-tree.

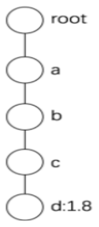


Figure 4. 4-FIU-tree.

3)Performance Evaluation and Results

- **SWFIUT-stream outperforms FP-Growth and other sliding window models in memory and speed.**

Explanation:

Experimental results showed that SWFIUT-stream significantly reduced both memory usage and processing time. It efficiently maintained up-to-date itemsets, adapted quickly to new data, and provided accurate support counts under varying stream conditions. These improvements made it well-suited for real-time analytical tasks like transaction monitoring or sensor data mining.

Mapped to CO2 –Analyze patterns in streaming data
Mapped to CO1 – Describe the fundamental issues and challenges of mining complex data

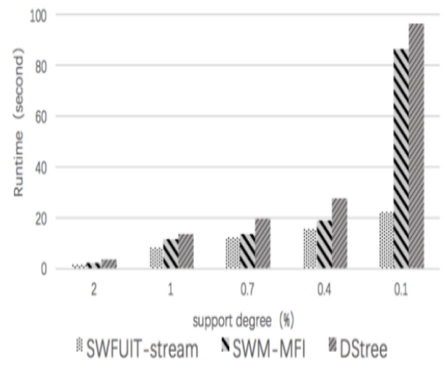


Figure 6. Time cost of each algorithm under different support degree levels.

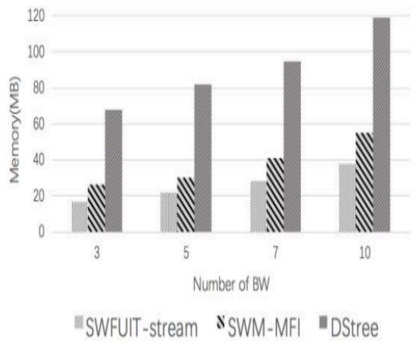


Figure 7. Memory cost of each algorithm under different number of window units.

4) Comparison Analysis

- **Compared with landmark and decay-based models, SWFIUT-stream offers better adaptability.**

  Explanation:

  The paper compared SWFIUT-stream to models using fixed windows (landmark models) or linear decay. The proposed method was superior in adapting to recent patterns, thanks to its weighted sliding window and distributed architecture. It also showed better real-time throughput and was less prone to memory overload.

  Mapped to CO2 – Analyze patterns in streaming data

---

5) Future Scope

- **Suggests extending the model to platforms like Spark/Hadoop and handling complex patterns.**

  Explanation:

  The authors propose expanding the algorithm's capabilities to platforms like Hadoop and Spark for even greater scalability. Future improvements include extending the mining process to handle sequences and time-series data, making the method applicable to advanced domains like bioinformatics, recommendation systems, and smart sensors. This direction also opens up integration with CO5 topics, such as multivariate time-series and sequence mining.

  Mapped to CO5 – Sequence, multivariate, and time-series mining
  Mapped to CO1 – Analyze patterns in multivariate time series data

---

Paper 3: *"Graph Neural Networks"*

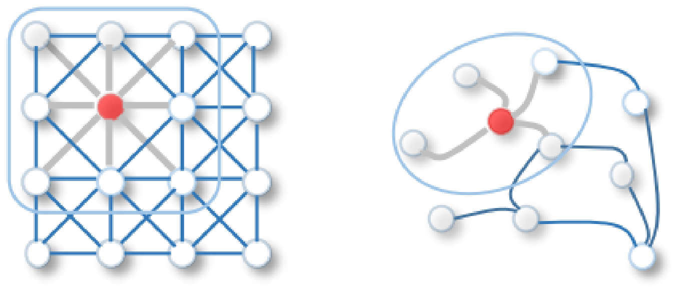
---

1) Introduction & Motivation

Key Point: The paper introduces Graph Neural Networks (GNNs) as a way to bring deep learning to complex, irregular data formats like graphs.

Explanation:
We're used to seeing deep learning work amazingly well on data like images or audio—things that follow a regular structure. But what happens when your data looks like a web of relationships? Think about social networks, road maps, citation networks, or even molecules. These don't follow neat rows and columns; they're graphs.

Traditional neural networks like CNNs or RNNs don't understand how to process graphs directly. That's where GNNs come in. They're designed to capture relationships and dependencies in this irregular graph structure. This paper sets the stage by showing how important GNNs have become and why we need a dedicated framework to understand and develop them.



(a) 2D Convolution. Analogous to a graph, each pixel in an image is taken as a node where neighbors are determined by the filter size. The 2D convolution takes the weighted average of pixel values of the red node along with its neighbors. The neighbors of a node are ordered and have a fixed size.

(b) Graph Convolution. To get a hidden representation of the red node, one simple solution of the graph convolutional operation is to take the average value of the node features of the red node along with its neighbors. Different from image data, the neighbors of a node are unordered and variable in size.

Mapped to CO2 – Analyze patterns in streaming data

---

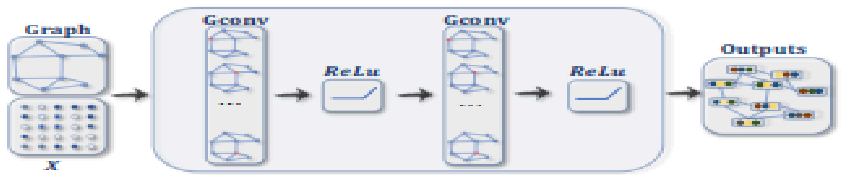
2) Methodology Used

Key Point: The paper breaks down GNNs into four main types and explains how they process graph data.

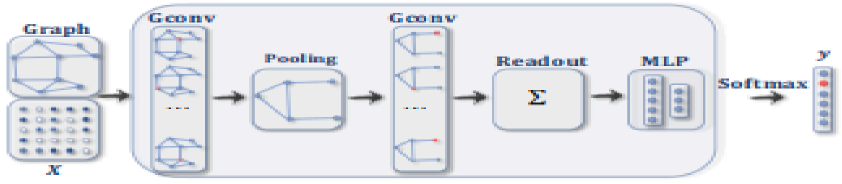
Explanation:
The authors offer a really helpful categorization of GNN models:

- **Recurrent GNNs (RecGNNs):** These work like RNNs—updating node information in cycles until things settle down.
- **Convolutional GNNs (ConvGNNs):** These mimic CNNs and extract node features layer by layer—either by using math-heavy spectral theory or by aggregating nearby node features.
- **Graph Autoencoders (GAEs):** Used mostly in unsupervised learning to compress graph data and reconstruct it—great for embeddings and anomaly detection.
- **Spatial-Temporal GNNs (STGNNs):** Tackle graph data that changes over time—think of traffic prediction, where both location and time matter.

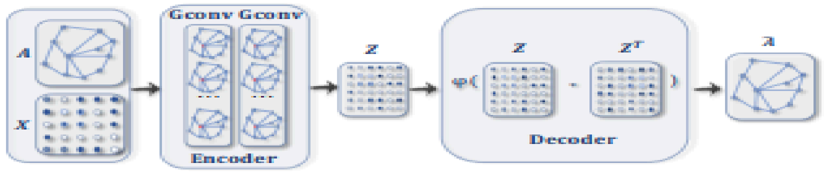
Besides that, the paper also dives into how pooling and attention work in GNNs. These parts help models focus on important nodes and reduce data size—kind of like zooming out to see the big picture.



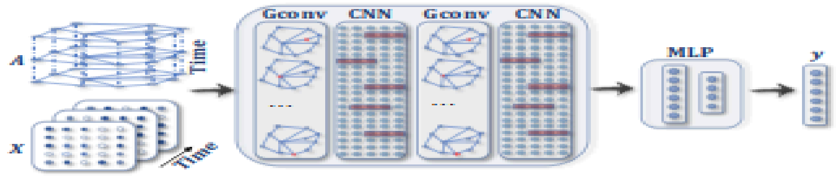
(a) A ConvGNN with multiple graph convolutional layers. A graph convolutional layer encapsulates each node's hidden representation by aggregating feature information from its neighbors. After feature aggregation, a non-linear transformation is applied to the resulted outputs. By stacking multiple layers, the final hidden representation of each node receives messages from a further neighborhood.



(b) A ConvGNN with pooling and readout layers for graph classification [21]. A graph convolutional layer is followed by a pooling layer to coarsen a graph into sub-graphs so that node representations on coarsened graphs represent higher graph-level representations. A readout layer summarizes the final graph representation by taking the sum/mean of hidden representations of sub-graphs.



(c) A GAE for network embedding [61]. The encoder uses graph convolutional layers to get a network embedding for each node. The decoder computes the pair-wise distance given network embeddings. After applying a non-linear activation function, the decoder reconstructs the graph adjacency matrix. The network is trained by minimizing the discrepancy between the real adjacency matrix and the reconstructed adjacency matrix.



(d) A STGNN for spatial-temporal graph forecasting [74]. A graph convolutional layer is followed by a 1D-CNN layer. The graph convolutional layer operates on A and $X^{(t)}$ to capture the spatial dependency, while the 1D-CNN layer slides over X along the time axis to capture the temporal dependency. The output layer is a linear transformation, generating a prediction for each node, such as its future value at the next time step.

Mapped to CO2 – Analyze patterns in streaming data

🔹 **3) Performance Evaluation and Results**

Key Point: The paper evaluates how GNNs perform across different datasets and how they handle computation and memory challenges.

Explanation:
To measure how good GNNs are, researchers often use standard datasets like Cora or Pubmed (citation networks) and look at metrics like classification accuracy or memory usage. The paper reviews various GNNs and compares how well they do in terms of speed, memory efficiency, and adaptability to large datasets.

What's interesting is that models like **GAT (Graph Attention Network)** and **GraphSAGE** don't just give better accuracy—they're more efficient too. They scale better, work faster, and adapt to new data more easily. On the other hand, the paper also points out that some models might perform better in theory but struggle in real-world cases due to memory or processing constraints.

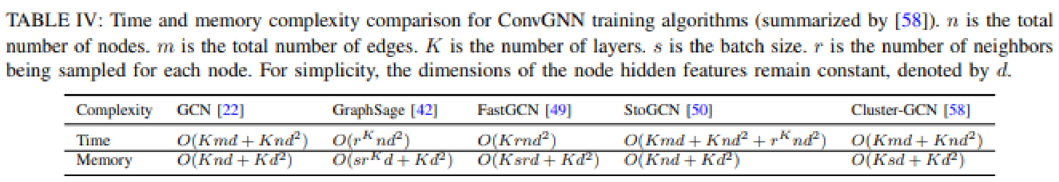


TABLE IV: Time and memory complexity comparison for ConvGNN training algorithms (summarized by [58]). n is the total number of nodes. m is the total number of edges. K is the number of layers. s is the batch size. r is the number of neighbors being sampled for each node. For simplicity, the dimensions of the node hidden features remain constant, denoted by d.

Complexity	GCN [22]	GraphSage [42]	FastGCN [49]	StoGCN [50]	Cluster-GCN [58]
Time	$O(Kmd + Knd^2)$	$O(r^K nd^2)$	$O(Krnd^2)$	$O(Kmd + Knd^2 + r^K nd^2)$	$O(Kmd + Knd^2)$
Memory	$O(Knd + Kd^2)$	$O(sr^K d + Kd^2)$	$O(Ksrd + Kd^2)$	$O(Knd + Kd^2)$	$O(Ksd + Kd^2)$

Mapped to CO2 – Analyze patterns in streaming data
Mapped to CO1 – Describe the fundamental issues and challenges of mining complex data

---

🔹 4) Comparison Analysis

Key Point: A detailed look at how different GNNs stack up in terms of design, performance, and practicality.

Explanation:
The authors provide a well-rounded comparison across models. For instance:

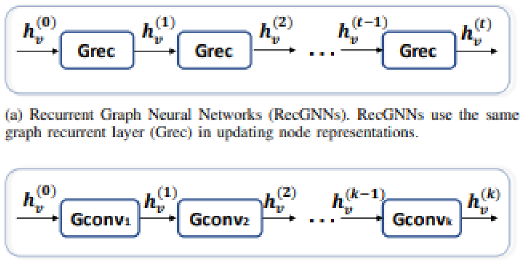
- **Spectral GNNs** use graph theory math (Laplacians), which makes them powerful but not very flexible.
- **Spatial GNNs** are much easier to apply to real data because they just aggregate neighborhood information.

There's also a comparison between **Recurrent GNNs** and **Convolutional GNNs**—RecGNNs use repetitive updates, while ConvGNNs stack layers like in CNNs. The latter is more efficient and easier to train.

Pooling methods are also compared:

- Simple ones like **mean/sum pooling** are fast.
- **SortPooling** and **DiffPool** can better capture structure but are more complex.

This section really shows that there's no one-size-fits-all GNN—it depends on what you care about: speed, memory, accuracy, or structure.



(a) Recurrent Graph Neural Networks (RecGNNs). RecGNNs use the same graph recurrent layer (Grec) in updating node representations.

(b) Convolutional Graph Neural Networks (ConvGNNs). ConvGNNs use a different graph convolutional layer (Gconv) in updating node representations.

Fig. 3: RecGNNs v.s. ConvGNNs

Mapped to CO2 – Analyze patterns in streaming data

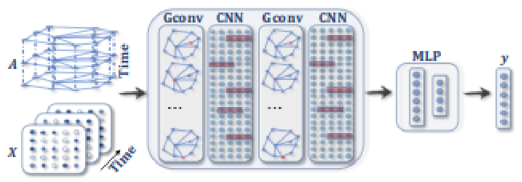
---

5) Future Scope

Key Point: The paper outlines where GNN research is headed, including scalability, deeper models, and dynamic graph handling.

Explanation:
Although GNNs have come a long way, there's still a lot of room for improvement:

1. **Depth Issues:** Just like deep CNNs, we want to build deeper GNNs. But there's a problem called over-smoothing, where all node features start to look the same after too many layers. Researchers are working on ways to fix this.
2. **Scalability:** Real-world graphs can be huge—think of social networks with billions of nodes. Current GNNs struggle to handle this scale.
3. **Dynamic Graphs:** Many systems (e.g., traffic, sensors, communication networks) are always changing. New models need to adapt to this constantly evolving structure.
4. **Heterogeneous Graphs:** Not all graphs are the same—some have different types of nodes and edges. Making GNNs work for such graphs is a key challenge.
5. **Interpretability:** Finally, we need models that not only perform well but also explain their decisions, especially in sensitive areas like finance or healthcare.



(d) A STGNN for spatial-temporal graph forecasting [74]. A graph convolutional layer is followed by a 1D-CNN layer. The graph convolutional layer operates on A and $X^{(t)}$ to capture the spatial dependency, while the 1D-CNN layer slides over X along the time axis to capture the temporal dependency. The output layer is a linear transformation, generating a prediction for each node, such as its future value at the next time step.

Fig. 2: Different graph neural network models built with graph convolutional layers. The term Gconv denotes a graph convolutional layer. The term MLP denotes a multi-layer perceptron. The term CNN denotes a standard convolutional layer.

Mapped to CO5 – Handle multivariate time-series and sequence mining

Links to papers:

- **Paper-1:https://drive.google.com/file/d/1frhyHPYiA3uIibATSBKQ5jEjrapbFG9Y/view?usp=sharing**
- **Paper-2: https://drive.google.com/file/d/1PIyJMp27aTMuqZtwMvVRO2h6rASChXMk/view?usp=sharing**
- **Paper-3: https://drive.google.com/file/d/1FmCHsZRhVq8VIizgX6jjLhnNmppZLGBL/view?usp=sharing**