University of Camerino
School of Sciences and Technology
Master of Science in
Computer Science (LM-18)

University of Applied Sciences and Arts Northwestern Switzerland
School of Business
Master of Science in
Business Information Systems

HAI-BEMS: A Hybrid Artificial Intelligent Approach for Building Energy Management Systems

Candidate:
Piermichele Rosati

Matricula 124176

FHNW Supervisor:
Dr. Emanuele Laurenzi

Unicam Supervisor:
Prof. Michela Quadrini

A thesis presented to the School of Sciences and Technology of the University of Camerino and School of Business of the University of Applied Sciences and Arts Northwestern Switzerland in partial fulfillment of the requirements for the degree of Master of Science in Computer Science

A.Y. 2023/2024

Abstract

The Thesis Abstract is written here (and usually kept to just this page)...

Statement of Authenticity

I, Piermichele Rosati, hereby confirm that this report was performed autonomously using only the sources, aids and assistance stated in the report, and that quotes are readily identifiable as such.

Signed: 

Date: 24th June 2024

Contents

Chapter 1

Introduction

1.1 Background

1.2 Problem Statement

1.3 Thesis Statement

1.4 Research Questions

The main Research Question (RQ):

How can a Graph Neural Network (GNN) support IoT data classification in KGs about BEMS?

Chapter 2

Literature Review

This chapter bla bla bla ...

2.1 Knowledge Graphs

TODO: change intro

TODO: reread every section and change (modifying, adding something)

Knowledge Graphs (KGs) have came up as a key technology in the field of data management and Artifical Intelligence (AI), enabling sophisticated data integration, retrieval and analysis, and analysis. This literature review provides an in-depth examination of KGs, their theoretical foundations, practical applications and recent advances.

Theoretical Foundations of Knowledge Graphs

Definition and Structure

KGs are directed graph-based data structures that represent real-world entities and their interrelations, providing a way to model complex domains and their underlying semantics. A KG consists of nodes (also called entities) and edges (also called relationships), forming a network of interconnected information. This structure allows KGs to capture rich contextual information and provide a semantic framework for data (Hogan et al., 2021).

A KG refers to a semantic network graph which is consisted of diverse entities, concepts, and relationships in the real world. It is used to formally describe various things and their associations in the real world. KGs are generally represented in triples $KG = \{E, R, F\}$.

- E represents the entity set $\{e_1, e_2, ..., e_E\}$, and the entity e is the most basic element in the KG, referring to the items that exist objectively and can be distinguished from each other.
- R represents the relation set $\{r_1, r_2, ..., r_R\}$, and the relation r is an edge in the KG, representing a specific connection between different entities.
- F represents the fact set $\{f_1, f_2, ..., f_F\}$, and each f is defined as a triple $(h, r, t) \in f$, in which h denotes the head entity, r stands for the relationship, and t indicates the tail entity.

The KG in Fig. 2.1 visually represents the relationships and attributes associated with Albert Einstein. At the center of the graph is "Albert Einstein", from which several connections extend. One connection indicates that he was "born in" Germany. Another connection shows his "occupation" as a "Theoretical Physicist". This occupation is further connected to "Physicist" as a "kind of" category, indicating that a theoretical physicist is a type of physicist. The graph also shows that a physicist "practices" physics. Additionally, it highlights that Albert Einstein "developed" the "Theory of Relativity", which is shown as a "branch of" physics. The graph effectively maps out key aspects of Albert Einstein's background, profession, and contributions to science.

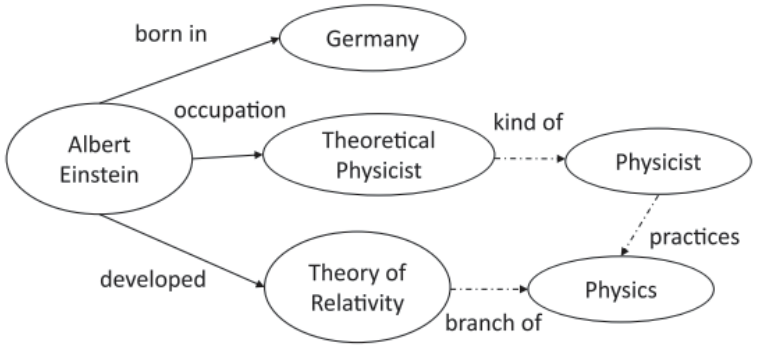

FIGURE 2.1: An example of Knowledge Graph (Chaudhri et al., 2022)

Ontologies and Semantic Web Technologies

An ontology is a formal representation of knowledge in a domain, specifying the concepts, relationships, and constraints that exist within that domain. The term "ontology" can be used to the shared understanding of some domain of interest (Uschold and Gruninger, 1996). Ontologies play a critical role in defining the schema and semantics of KGs. They specify the types of entities, relationships, and constraints, thereby providing a formalized structure for the data. The Semantic Web technologies, particularly the Resource

Description Framework (RDF) and the Web Ontology Language (OWL), are fundamental to the development and functioning of KGs (G. (Grigoris) Antoniou and Frank. Van Harmelen, 2008).

RDF is a standard model for data interchange on the web. It uses triples (subject-predicate-object) to represent information, providing a flexible and extensible framework for creating and managing KGs (Cyganiak et al., 2014). The RDF standard is a framework for representing information about resources on the web. RDF is a part of the W3C's Semantic Web activity and provides a model for data interchange on the Web. RDF data is structured as triples, each consisting of a subject, predicate, and object. The subject is the resource being described, the predicate is the property or characteristic of the subject, and the object is the value of the property, which can be a literal or another resource. RDF uses Uniform Resource Identifiers (URIs) to uniquely identify subjects and predicates, ensuring that resources are globally identifiable. Objects can be literals, which are concrete data values such as strings, numbers, or dates. RDF can be serialized in various syntaxes, including RDF/XML, Turtle, N-Triples, and JSON-LD. RDF/XML is the original RDF syntax using XML to represent RDF triples. Turtle is a more human-readable syntax for RDF data, concise and easier to write and read compared to RDF/XML. N-Triples is a plain text format for encoding RDF triples, useful for streaming data or simple data exchange. JSON-LD is a JSON-based format to serialize Linked Data, designed to be easy to use and integrate with existing JSON-based systems. RDF Schema (RDFS) is a semantic extension of RDF that provides mechanisms to describe groups of related resources and the relationships between these resources. It allows for defining classes, which are categories of resources; properties, which are relationships between resources; and hierarchies, enabling inheritance. RDF is widely used in various domains, including the Semantic Web, where it enables the creation of a web of data with meaning, allowing machines to understand and process web content; KGs, powering large-scale graph-based data structures used by organizations like Google and Amazon (Kejriwal, 2022); data integration, integrating data from disparate sources by providing a common data model; and ontology engineering, defining and using ontologies to model domain knowledge. RDF provides a robust and flexible framework for representing structured information, enabling interoperability and integration of data across different systems and domains.

OWL is used to explicitly represent the meaning of terms in vocabularies and the relationships between those terms. It enables more complex and expressive representations compared to RDFS (Deborah and van Harmelen Frank, 2004). The OWL standard is a W3C technology for defining and using web ontologies, enhancing RDF by offering greater expressiveness for complex information. OWL ontologies consist of classes, properties, and individuals, enabling detailed descriptions of relationships and characteristics. It supports complex class expressions, including logical operators and restrictions

like cardinality and property constraints. OWL has three sublanguages: OWL Lite (simple feature set), OWL DL (maximum expressiveness with computational guarantees), and OWL Full (greatest expressiveness without computational guarantees). Reasoning capabilities in OWL allow for inferring implicit knowledge, consistency checking, and classification. OWL is used in knowledge management, information integration, and semantic search, providing a common framework for understanding and integrating data. It promotes interoperability and the creation of semantically rich, interconnected web data.

Query Languages

SPARQL Protocol and RDF Query Language (SPARQL) is the standard query language for retrieving and manipulating data stored in RDF format. It allows users to write complex queries to extract specific information from a KG, making it a powerful tool for data analysis and knowledge discovery (Pérez et al., 2009). It allows users to query RDF data by specifying patterns of triples and to update RDF data by inserting, deleting, and modifying RDF triples. RDF is a foundation for Linked Data, which involves interlinking data across the web using URIs and RDF. This enables the creation of a web of data that can be easily connected and queried.

Cypher is another query language for graph databases, such as Neo4j, that allows users to interact with graph data using a pattern-matching syntax. Cypher queries are used to traverse the graph, retrieve specific patterns, and perform operations on the data (Francis et al., 2018). Cypher supports complex queries involving multiple nodes and relationships, aggregation, sorting, and limiting results. It also provides functions for working with strings, numbers, dates, and collections, as well as support for subqueries and variable-length paths. Cypher is widely used for graph analytics, network analysis, and data exploration, enabling users to easily express complex graph traversals and operations. It leverages Neo4j's indexing and optimization capabilities to ensure efficient execution of queries, making it a powerful tool for working with connected data.

Applications of Knowledge Graphs

General Applications

KGs have been adopted across various domains due to their ability to integrate heterogeneous data sources, provide semantic context, and enable advanced querying and reasoning. According to Kapanipathi et al., 2020, in healthcare KGs are used to integrate patient records, clinical trials, research data, and medical ontologies, enabling

personalized medicine and decision support systems. They help in identifying relationships between diseases, treatments, and patient outcomes.
Financial institutions leverage KGs to connect data from various sources, such as market data, regulatory information, and customer transactions. This integration facilitates risk management, fraud detection, and compliance monitoring (Tchechmedjiev et al., 2019).
In e-commerce, KGs enhance product recommendation systems by linking customer preferences, purchase history, and product information. They enable more personalized and relevant recommendations, improving customer satisfaction and sales (Zhang et al., 2021).

In compliance with Zou, 2020, Fig. 2.2 shows a mind map illustrating the main applications of knowledge graphs. It divides the applications into five main categories: question answering, recommendation systems, information retrieval, domain-specific applications and other applications. With regard to question answering, methods based on semantic parsing, methods based on information retrieval, methods based on embedding, methods based on Deep Learning (DL) and more complex tasks are included. KGs significantly enhance search engines by providing semantic search capabilities. They enable the understanding of user queries in context, allowing for more accurate and relevant search results. Google's Knowledge Graph is a prominent example, enhancing search results with information about entities and their relationships (Singhal et al., 2012). Recommender systems are classified into embedding-based methods, path-based methods and other methods. Information retrieval includes query representation, document representation, ranking and DL. Domain-specific applications include medicine, computer security, finance, news and education. Within enterprises, KGs are used to manage and utilize internal knowledge effectively. They integrate data from different departments, such as human resources, finance, and operations, providing a unified view of the organization's information. This integration supports decision-making, collaboration, and innovation (Pujara et al., 2013). Other applications include social networks, classification, geosciences and various other applications.

Recent Advancements in Knowledge Graphs

Integration with Machine Learning

Recent research has focused on integrating KGs with Machine Learning (ML) and DL techniques to enhance their capabilities and applications. These integrations have led to significant advancements in various areas, including Natural Language Processing (NLP), recommendation systems, and predictive analytics.

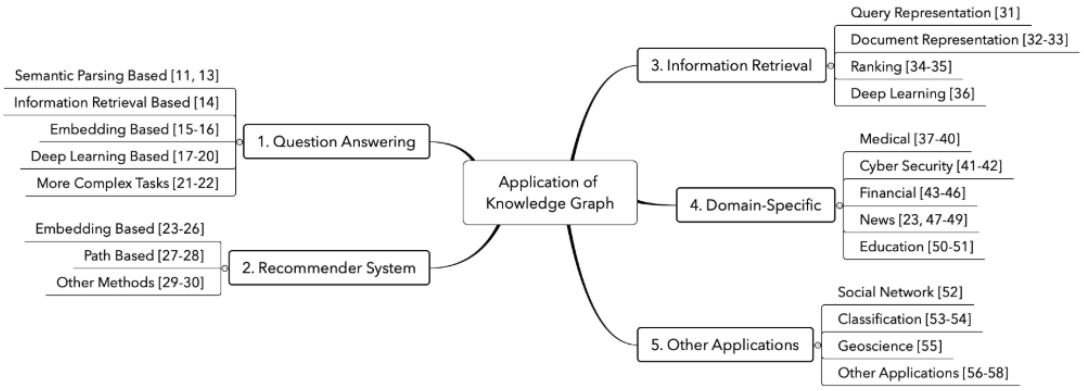

FIGURE 2.2: Application of Knowledge Graphs (Zou, 2020)

- **Knowledge Graph Embeddings (KGEs)**: KGE techniques represent entities and relationships in a continuous vector space, enabling the use of ML algorithms for tasks such as link prediction, entity classification, and clustering. Popular methods include TransE (Bordes et al., 2013), TransH (Z. Wang et al., 2014), and TransR (Lin et al., 2015), each providing different ways to model relationships in the embedding space (Q. Wang et al., 2017).

- **Graph Neural Networks (GNNs)**: GNNs are DL models designed to operate on graph-structured data. They leverage the relational nature of graphs to perform tasks such as node classification, link prediction, and graph classification. GNNs have been successfully applied to enhance the capabilities of KGs in various domains (Wu et al., 2021).

  Section 2.2 provides a comprehensive and in-depth description of GNNs.

Natural Language Processing and Question Answering

KGs have been instrumental in advancing NLP applications, particularly in question answering systems. By providing structured and semantically rich information, KGs enable systems to understand and generate human language more effectively. Question Answering Systems: KGs support question answering systems by enabling them to retrieve and reason over structured data. These systems can answer complex queries by traversing the graph and applying logical inferences based on the relationships between entities (Yasunaga et al., 2021). Semantic Search and Text Analysis: KGs enhance text analysis and semantic search by providing contextual information about entities mentioned in the text. This contextual understanding improves the accuracy of information retrieval and the relevance of search results (Fernández et al., 2011).

Challenges and Future Directions

Scalability and Performance

As KGs grow in size and complexity, scalability and performance become critical challenges. Efficient storage, querying, and updating of large KGs require advanced techniques and architectures. Research in distributed computing, graph databases, and parallel processing is ongoing to address these issues (Chaudhri et al., 2022).

Data Quality and Integration

Ensuring the accuracy and consistency of data in KGs is essential for their reliability. Data quality issues, such as inconsistencies, duplications, and inaccuracies, can significantly impact the performance of applications relying on KGs. Developing methods for automatic data cleaning, validation, and integration is an active area of research (Paulheim, 2017).

Privacy and Security

The integration of sensitive data into KGs raises concerns about privacy and security. Protecting personal and confidential information while allowing for meaningful data analysis is a significant challenge. Research is focusing on developing techniques for secure data sharing, access control, and anonymization within KGs (Bonatti et al., 2017).

Interoperability and Standardization

Interoperability and standardization are crucial for the widespread adoption of KGs. Ensuring that different KGs can work together seamlessly and that their data can be easily integrated requires the development of common standards and protocols. Efforts such as the Linked Open Data (LOD) initiative and W3C standards aim to address these challenges (Bizer et al., 2023).

Conclusion

KGs represent a transformative technology for data integration, retrieval, and analysis, offering significant benefits across various domains. Their ability to provide semantic context and capture complex relationships makes them invaluable for applications in

healthcare, finance, e-commerce, and enterprise knowledge management. Recent advancements in ML, particularly in the integration with GNNs, have further enhanced the capabilities of KGs, opening new avenues for research and application. However, challenges related to scalability, data quality, privacy, and interoperability remain and must be addressed to fully realize the potential of KGs. Continued research and development in these areas will be crucial for the future evolution and adoption of KGs.

2.2 Graph Neural Networks

TODO: change, reread

Graph Neural Networks have become fundamental to Deep Learning field, particularly for tasks involving non-Euclidean data structures. These models have had a significant impact on several domains, such as social network analysis, bioinformatics, recommender systems and NLP. This literature review explores the development, key architectures, methodologies, applications and methodologies, applications, challenges and future directions of GNNs.

Historical Context and Evolution

Early Work and Foundations

Graphs have received success because the achievements of neural networks, for ML tasks such as object detection and speech recognition. Graphs are mathematical structures consisting of nodes connected by edges, making them ideal for modeling relationships and dependencies in complex systems. Mathematically, a graph is defined as $G = (V, E)$, where V represents the nodes and E represents the edges. The edges in a graph can be either directed or undirected, depending on whether directional dependencies exist between the nodes. Graphs are used to represent real-world datasets like protein-protein interaction networks, social networks, traffic forecasting, e-commerce, geographical maps, KGs and so on.
GNNs draw inspiration from Convolutional Neural Networks (CNNs). According to Khemani et al., 2024, CNNs and Recurrent Neural Networks (RNNs) are not well-suited for effectively handling graph-structured data. CNNs are designed for data with a grid structure, such as images, while RNNs are tailored for sequences, like text. Typically, arrays are used for storing text data, and matrices are used for image data. However, arrays and matrices are inadequate for dealing with graph data. For graphs, we need a specialized technique called Graph Convolution. This technique allows deep neural

networks to process graph-structured data directly, resulting in a GNNs. Nowadays, however there is a continuous increase in applications and domains that use complex structures such as graphs (Wu et al., 2021). The complexity of graphs implied not insignificant problems on existing ML algorithms because a graph may have a variable number of nodes or edges; or it might have nodes that may have a different number of neighbours. This last point is very important in some operation like convolution, that is easy for images but not for graphs. Another motivation comes from graph representation learning, which aims to learn to represent graph nodes, edges or subgraphs by low-dimensional vectors (Zhou et al., 2020). Existing word embedding methods like DeepWalk (Perozzi et al., 2014), node2vec (Grover and Leskovec, 2016), LINE (Tang et al., 2015) have achieved very good results but they have 2 important drawbacks. The first one is that there are no shared parameters in the encoder, which implies computationally inefficency (n. of parameters grows linearly with the n. of nodes); The second drawback is that there's a lack of generalization because they cannot deal with dynamic graphs or generalize to new graphs.
Inspired by the success of CNNs and RNNs, Gori et al., 2005; Scarselli et al., 2009 have designed and developed the first GNN model in order to operate on non-Euclidian space. Variations of GNNs, including Graph Convolutional Networks (GCNs), Graph Attention Networks (GATs), and GraphSAGE, have demonstrated groundbreaking performance on various DL tasks in recent years.
To summarize, a GNN aims to learn a state embedding $h_v \in \mathbb{R}^s$ that encapsulates the neighborhood data of each node. This state embedding h_v, an s-dimensional vector for node v, can be used to generate an output O_v, such as the predicted distribution of the node label. The predicted node label distribution (O_v) is derived from the state embedding h_v (Rong et al., 2019).

GNN Learning Tasks and Training Settings

According to Wu et al., 2021; Zhou et al., 2020, GNNs are versatile tools capable of performing a variety of node-level, edge-level, and graph-level tasks. Each approach leverages different amounts and types of labeled data to extract meaningful patterns and representations from graph-structured data.

Node-level tasks focus on predicting or inferring properties of individual nodes within a graph. This includes node classification, where each node is classified into one of several predefined categories, and node regression, which predicts continuous values associated with nodes. Node clustering groups nodes into clusters based on feature similarity and structural proximity, and node representation learning focuses on learning low-dimensional embeddings that preserve the graph's structural and feature information.

Edge-level tasks aim to predict or infer properties related to the edges in a graph. Link prediction involves predicting the existence of an edge between two nodes and is crucial for applications such as recommending friends in social networks or predicting interactions in biological networks. Edge classification assigns categories to edges based on the properties of the connected nodes and their relationships, while edge regression predicts continuous values associated with edges, such as the strength of interactions or the weight of connections.

Graph-level tasks involve making predictions or inferences about entire graphs. Graph classification assigns a label to an entire graph, which is useful in applications like classifying molecules or categorizing social networks. Graph regression predicts continuous values for entire graphs, such as the bioactivity of chemical compounds or the performance metrics of network designs. Graph generation involves creating new graphs that resemble a given set of graphs and is important in applications like molecule generation for drug discovery or network topology design.

Fig. 2.3 summarizes the key graph learning tasks that GNNs can perform, highlighting the diversity and flexibility of these models in handling various types of graph-structured data.

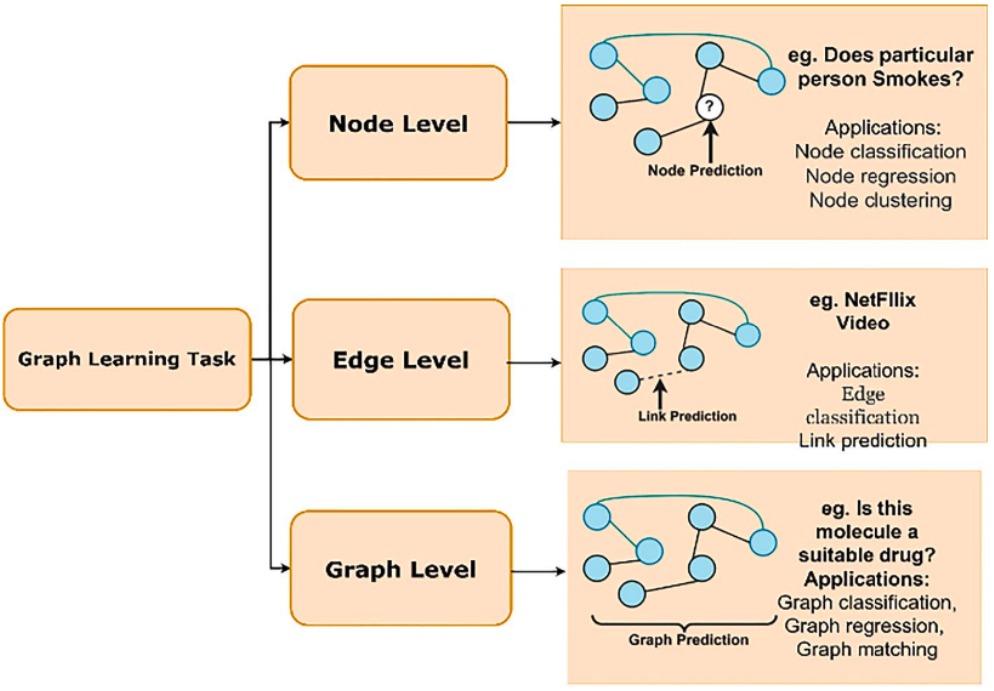

FIGURE 2.3: Graph learning tasks by GNNs (Khemani et al., 2024)

GNNs can be trained using a variety of learning paradigms, including supervised, semi-supervised, or unsupervised learning approaches, each of which leverages different methods and data availability to optimize the model for various analytical tasks on graph-structured data (Wu et al., 2021; Zhou et al., 2020).

In **supervised learning**, models are trained using labeled data, where each input example is paired with the correct output. For GNNs, supervised tasks include node classification, link prediction, and graph classification, with a loss function defined based on the difference between predicted and actual labels.

Semi-supervised learning uses a small amount of labeled data along with a large amount of unlabeled data (Kipf and Welling, 2017). GNNs are well-suited for this approach due to their ability to leverage graph structure and propagate label information through the network. Semi-supervised tasks include semi-supervised node classification and semi-supervised link prediction, allowing models to make use of the vast amounts of unlabeled data, which is often easier and cheaper to obtain.

Unsupervised learning involves training a model without any labeled data, relying solely on the structure and inherent properties of the input data. In GNNs, unsupervised tasks include node clustering, graph embedding, and graph generation. Techniques like autoencoders, where the model learns to reconstruct the input graph from compressed embeddings, or contrastive learning, where the model learns representations by contrasting positive and negative samples derived from the graph structure, are often used in unsupervised learning.

The Message-Passing Mechanism

The message-passing mechanism in GNNs is a sophisticated process that maintains graph symmetries through optimizable transformations on all graph properties, including nodes, edges, and the global context, preserving permutation invariances (Khemani et al., 2024). This ensures that the connectivity of the input graph remains unchanged, allowing the output to be characterized using the same adjacency list and feature vector count as the input. However, the output graph has updated embeddings as the GNN modifies each node, edge, and global-context representation.

At its core, the message-passing mechanism operates iteratively, enabling nodes to collect and aggregate information from their neighbors. Here's a detailed breakdown of this process:

1. **Node Representation Initialization**: each node in the graph is initialized with a feature vector. These initial representations can be based on node attributes or initialized randomly.

2. **Message Passing Phase**: this phase involves multiple iterations, during which nodes exchange and aggregate information from their neighbors. The process unfolds as follows:

    a. **Message Computation**: each node generates a message to be sent to its neighboring nodes. This message is typically a function of the node's current feature vector and the feature vectors of its neighbors. Formally, for a node u, the message $m_u^{(k)}$ sent to its neighbors can be computed using a differentiable function such as a neural network:

    $$m_u^{(k)} = MessageFunction(h_u^{(k)}, \{h_v^{(k)} \mid v \in \mathcal{N}(u)\}, \{e_{uv} \mid v \in \mathcal{N}(u)\})$$

    where $h_u^{(k)}$ is the feature vector of node u at iteration k, $\mathcal{N}(u)$ denotes the neighbors of node u, and e_{uv} represents the edge features between nodes u and v.

    b. **Message Aggregation**: each node aggregates the messages received from its neighbors. The aggregation function is typically permutation-invariant, ensuring that the order of the neighboring nodes does not affect the result. Common aggregation functions include summation, mean, or max. For a node u, the aggregated message $M_u^{(k)}$ is computed as:

    $$M_u^{(k)} = AGGREGATE^{(k)}(\{m_v^{(k)} \mid v \in \mathcal{N}(u)\})$$

    c. **Node Update**: each node updates its feature vector based on the aggregated message and its current feature vector. This update is performed using an update function, often a neural network. For a node u, the updated feature vector $h_u^{(k+1)}$ is computed as:

    $$h_u^{(k+1)} = UPDATE^{(k)}(h_u^{(k)}, M_u^{(k)})$$

3. **Iteration and Neighborhood Expansion**: the message-passing phase is repeated for multiple iterations. With each iteration, nodes accumulate information from increasingly distant parts of the graph. After the first iteration ($k = 1$), each node's embedding includes information from its 1-hop neighborhood. After the second iteration ($k = 2$), each node's embedding incorporates information from its 2-hop neighborhood, and so on. Generally, after k iterations, each node's embedding contains data from its k-hop neighborhood.

4. **Structural and Feature-Based Information**: the information passed during message passing consists of two main components: structural information about the graph (such as the degree of nodes) and feature-based information (attributes of the nodes and edges). Each node's message is stored in the form of feature vectors, and during each iteration, these vectors are updated to reflect the aggregated information from neighboring nodes.

5. **Readout Phase**: after completing the message-passing iterations, a readout function is applied to extract the final representation for nodes, edges, or the entire graph. This function aggregates the final node embeddings into a single representation. For graph-level tasks, the readout function might be a global pooling operation, such as summation, mean, or max over all node feature vectors.

6. **Mathematical Formulation**: the process can be mathematically represented as:

$$h_u^{(k+1)} = UPDATE^{(k)}\left(h_u^{(k)}, AGGREGATE^{(k)}\left(\{m_v^{(k)} \mid v \in \mathcal{N}(u)\}\right)\right)$$

where $h_u^{(k)}$ denotes the feature vector of node u at iteration k, and $m_v^{(k)}$ represents the message from node v at iteration k. The AGGREGATE function combines the messages from the neighbors, and the UPDATE function combines this aggregated message with the node's current feature vector to produce the updated feature vector.

The message-passing mechanism of GNNs is illustrated Fig. 2.4. Here, an input graph with a set of node features $X \in \mathbb{R}^{d \times |V|}$ is used to produce updated node embeddings. The process begins with each node having an initial message (or feature vector). Nodes then exchange messages with their neighbors. For instance, the B user node collects messages from its neighboring nodes, which are then aggregated. This aggregated message is used to update B user's node feature vector, resulting in a new embedding for the B user node for the next iteration. The diagram visually represents how messages are computed, aggregated, and updated across nodes in the graph, leading to refined node embeddings over iterations.

This mechanism enables GNNs to effectively capture the dependencies and relationships within graph-structured data, making them powerful tools for tasks like node classification, link prediction, and graph classification. By iteratively exchanging and aggregating information, GNNs can learn rich, expressive representations that reflect both the structure and attributes of the graph.

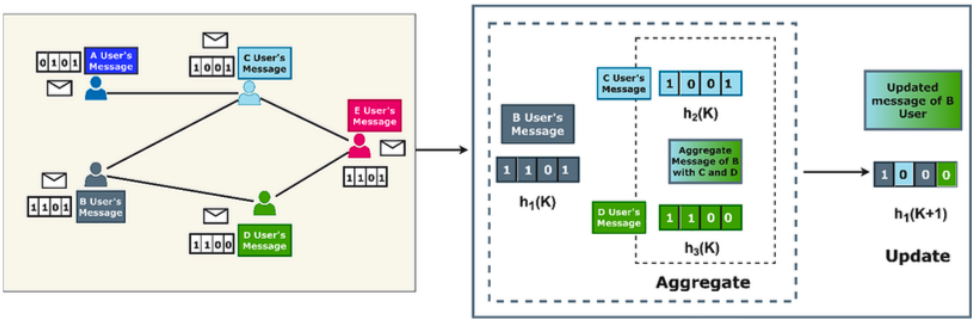

FIGURE 2.4: Message passing mechanism in GNNs (Khemani et al., 2024)

GNN Models

Over the years, different models of GNNs have been developed, each with its own potential and for specific tasks. The following are the most important and best-known network architectures in the literature.

TODO: GCN, GAT, GraphSAGE

TODO: mention other models...

TODO: GNN for KGs

Spectral Approaches

The breakthrough in spectral approaches marked a significant evolution in GNNs. Bruna et al., 2013 introduced a method leveraging spectral graph theory to define convolution operations on graphs. This approach was refined by Defferrard et al., 2016, leading to more efficient models. Kipf and Welling, 2017 simplified this concept further, making it more accessible and practical, thus popularizing the GCN.

Graph Convolutional Networks (GCNs)

GCNs are a fundamental variant of graph neural networks developed by Kipf and Welling, 2017. The convolution layers in GCNs operate similarly to the convolution process in CNNs. In CNNs, input neurons are multiplied by weights known as filters or kernels, which act as a sliding window across the image, allowing the network to learn from nearby cells. Weight sharing involves using the same filter throughout the same layer of the image. For instance, when CNNs are used to distinguish between images of cats and non-cats, the same filter detects the cat's nose and ears across the image. This

concept applies to GCNs as well, where similar filters are applied throughout the graph structure (Kipf and Welling, 2017).

GCNs operate by learning features through the analysis of neighboring nodes, mirroring the behavior of CNNs. However, the primary distinction between CNNs and GNNs lies in their application domains: CNNs are designed to handle regular, Euclidean-ordered data, while GNNs are a generalized form of CNNs suited for irregular, non-Euclidean data with varying numbers of node connections and unordered nodes. They extend traditional CNNs, which are typically used for grid-like data such as images, by performing convolution operations on graph data. This enables GCNs to capture and propagate information through the graph's nodes, considering both a node's features and those of its neighbors. GCNs have been successfully applied to various problems, including image classification (Monti et al., 2016), traffic forecasting (Cui et al., 2020), recommendation systems (Fan et al., 2019) and scene graph generation (Yang et al., 2018).

GCNs consist of multiple layers, each performing convolution and aggregation steps to refine node representations. By iteratively applying these layers, GCNs can capture complex patterns and dependencies within the graph data.

A GCN layer can be formally represented as:

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right)$$

where:

- $\tilde{A} = A + I$ is the adjacency matrix with added self-loops.
- $\tilde{D}$ is the degree matrix of $\tilde{A}$.
- $H^{(l)}$ and $H^{(l+1)}$ are the input and output feature matrices for layer l.
- $W^{(l)}$ is the trainable weight matrix.
- σ is an activation function like ReLU.

GCNs offer several advantages. One key advantage is their ability to naturally handle graph-structured data, making them well-suited for tasks involving complex relational structures. GCNs leverage the connectivity information in graphs to capture and propagate node features, enabling more effective learning from the data's inherent structure. This capability extends the powerful concept of convolution from Euclidean data, like images, to non-Euclidean data, such as graphs, allowing for a more generalized application

of neural network models (Wu et al., 2021). Additionally, GCNs are computationally efficient for large-scale graph data, as they utilize localized operations that limit the complexity of processing entire graphs at once. This localized approach helps in scaling GCNs to larger datasets and complex networks (Li et al., 2018).

However, GCNs also have limitations. One major limitation is their reliance on the assumption of homophily, where connected nodes are assumed to have similar features. This can restrict their effectiveness in graphs where such an assumption does not hold, such as in heterophilic graphs where connected nodes may have dissimilar features (Xu et al., 2019). Another limitation is the potential for oversmoothing, where node features become indistinguishable after several layers of convolution, leading to a loss of meaningful differentiation between nodes. This can hinder the model's ability to capture fine-grained information in deeper architectures (Li et al., 2018). Additionally, GCNs often struggle with scalability in extremely large graphs or dynamic graphs where the structure changes frequently, as the need to recompute convolutions can become computationally intensive. Finally, GCNs require careful tuning of hyperparameters and can be sensitive to the choice of architecture, which may limit their accessibility and ease of use for practitioners without extensive expertise in GNNs (Wu et al., 2021).

Graph Attention Networks (GATs)

Veličković et al., 2018 introduced GATs, which incorporate attention mechanisms to dynamically weigh the importance of neighboring nodes. GAT is a novel neural network designed to work with graph-structured data. It employs masked self-attentional layers to overcome the limitations of previous methods that relied on graph convolutions or their approximations. By stacking these layers, GAT enables the model to implicitly assign different weights to various nodes in a neighborhood, allowing nodes to focus on the most relevant features of their neighbors. This approach eliminates the need for expensive matrix operations, such as inversion, and does not require prior knowledge of the graph's structure. GAT effectively addresses many significant limitations of spectral-based graph neural networks, making the model suitable for both inductive and transductive tasks.

Attention Mechanism:

$$e_{ij} = \text{LeakyReLU}\left(a^T[Wh_i \| Wh_j]\right)$$

where e_{ij} is the attention score, a is the learnable attention vector, and $\|$ denotes concatenation.

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

The node features are updated as:

$$h_i' = \sigma \left(\sum_{j \in \mathcal{N}(i)} \alpha_{ij} W h_j \right)$$

Fig. 2.5 shows the coefficients computed by the attention mechanism, parametrized by a weight vector $\vec{\mathbf{a}} \in \mathbb{R}^{2F'}$, applying a LeakyReLU activation.

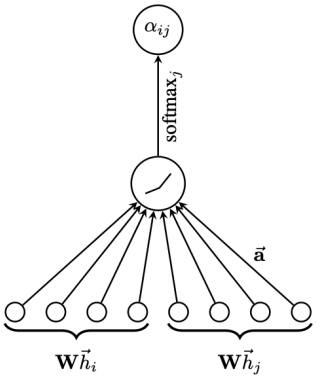

FIGURE 2.5: Attention mechanism in GATs (Veličković et al., 2018)

GATs provide numerous advantages. One of the main benefits is their capability to assign different importance weights to nodes within a neighborhood, enhancing the model's ability to concentrate on the most relevant features and relationships. This is facilitated by the self-attention mechanism, which avoids the need for expensive matrix operations or prior knowledge of the graph's structure, thus making GATs computationally efficient and versatile (Veličković et al., 2018). Furthermore, GATs can naturally manage graphs with varying node degrees and adapt to different scales, offering robustness and scalability. The ability to learn dynamic attention weights for different nodes makes GATs especially effective for tasks involving heterogeneous graphs and those with complex, irregular structures.

Nevertheless GATs also have certain limitations. A notable challenge is that the attention mechanism can be computationally demanding, especially for large-scale graphs with numerous nodes and edges, potentially causing scalability issues. The computational complexity of the self-attention mechanism can grow with the number of nodes, making it less efficient for very large graphs compared to simpler convolutional methods (Thekumparampil et al., 2018). Additionally, although GATs can focus on relevant

nodes, they may suffer from overfitting, particularly when the attention mechanism excessively emphasizes specific nodes, leading to a reduction in generalization. Furthermore, tuning the hyperparameters of the attention mechanism can be intricate and requires careful experimentation to achieve optimal performance (Lee et al., 2018).

GraphSAGE

GraphSAGE (Hamilton et al., 2017) introduced an inductive approach that can generalize to unseen nodes by sampling and aggregating features from a node's local neighborhood.

Sampling and Aggregation:

GraphSAGE samples a fixed-size set of neighbors and uses aggregation functions such as mean, LSTM, or pooling to update node embeddings:

$$h_i' = \sigma\left(W \cdot \mathrm{AGG}\left(\{h_j, \forall j \in \mathcal{N}(i)\}\right)\right)$$

Advantages:

- Scalable to large graphs.
- Supports inductive learning.

Limitations:

- Information loss due to fixed-size sampling.
- Requires carefully designed aggregation functions.

Message Passing Neural Networks (MPNNs)

MPNNs (Gilmer et al., 2017) formalized the message-passing framework for GNNs. In each layer, nodes exchange messages with their neighbors and update their states.

General Framework:

1. Message Function: $m_{ij}^{(l)} = M(h_i^{(l)}, h_j^{(l)}, e_{ij})$
2. Update Function: $h_i^{(l+1)} = U\left(h_i^{(l)}, \sum_{j \in \mathcal{N}(i)} m_{ij}^{(l)}\right)$

where h_i and h_j are node features, and e_{ij} are edge features.

Advantages:

- General and flexible framework.
- Can model complex dependencies and interactions.

Limitations:

- High computational cost for dense graphs.
- Complexity in designing effective message and update functions.

Training Techniques and Challenges

Mini-Batch Training

To manage large graphs, mini-batch training techniques are employed. Subgraphs or neighborhoods are sampled in each training iteration to reduce memory consumption and improve efficiency.

Graph Partitioning

Graph partitioning techniques like METIS and Louvain divide large graphs into smaller subgraphs that can be processed independently. This helps in parallelizing computations and managing memory usage.

Challenges:

- Maintaining the integrity of graph structure during partitioning.
- Ensuring balanced computational load across partitions.

Optimization Algorithms

Specialized optimizers, such as Adam and RMSprop, are used to stabilize training. Regularization techniques like dropout and weight decay help prevent overfitting.

Applications of GNNs

Social Network Analysis

GNNs are extensively used in social network analysis for tasks such as community detection, link prediction, and influence maximization. They model complex interactions and dependencies among users, providing insights into social dynamics.

Examples:

- Community Detection: Using GNNs to identify overlapping communities in social networks (Chen et al., 2017).
- Link Prediction: Predicting future connections by learning node embeddings (Zeng et al., 2019).

Biological Networks

In bioinformatics, GNNs are used to analyze molecular structures, predict protein functions, and understand biological processes. They model intricate interactions between biological entities, aiding in drug discovery and genomics.

Examples:

- Protein-Protein Interaction: Predicting interactions between proteins by modeling their structural properties (Fout et al., 2017).
- Drug Discovery: Identifying potential drug compounds by analyzing molecular graphs (Jin et al., 2018).

Recommendation Systems

GNNs enhance recommendation systems by modeling user-item interactions. They improve the accuracy and relevance of recommendations by capturing complex relationships.

Examples:

- Collaborative Filtering: Enhancing collaborative filtering with GNNs to model user-item interactions (X. Wang et al., 2019).
- Content-Based Recommendations: Using GNNs to model item features and user preferences for personalized recommendations (Ying et al., 2018).

Natural Language Processing

GNNs are applied in NLPs for tasks like semantic parsing, machine translation, and text classification. By representing sentences or documents as graphs, GNNs capture relationships between words or entities.

Examples:

- Semantic Parsing: Modeling the syntactic structure of sentences for accurate semantic parsing (Zeng et al., 2019).
- Relation Extraction: Extracting relationships between entities in text using GNNs to model dependency trees (Sahu et al., 2019).

Recent Advances and Future Directions

Scalability

Scalability remains a critical challenge for GNNs. Recent advancements focus on models and techniques that handle large-scale graphs efficiently. Methods like GraphSAINT (Zeng et al., 2019) and Cluster-GCN (Chiang et al., 2019) emphasize sampling and partitioning strategies to enable scalable training.

Future Directions:

- Distributed GNNs: Developing distributed frameworks for training GNNs on large-scale graphs.
- Efficient Sampling Techniques: Improving sampling methods to balance efficiency and information retention.

Expressiveness

Enhancing the expressiveness of GNNs involves designing architectures that capture more complex patterns and dependencies. Higher-order GNNs and graph transformers are being explored to address this need.

Future Directions:

- Higher-Order GNNs: Extending GNN architectures to capture higher-order interactions between nodes. Graph Transformers: Leveraging transformer models for graph data to enhance representational power.

Interpretability

Understanding the decision-making process of GNNs is crucial for their adoption in critical applications. Techniques like attention visualization, gradient-based methods, and node importance scores are being developed to interpret GNN predictions.

Future Directions:

- Explainable GNNs: Designing GNN models with built-in interpretability features.
- Interpretable Training Methods: Developing training techniques that enhance model transparency and explainability.

Challenges and Limitations

Despite the significant advancements and applications, GNNs face several challenges and limitations:

- Computational Complexity: GNNs can be computationally intensive, especially for large and dense graphs, limiting their practical applicability.
- Scalability Issues: Handling extremely large-scale graphs remains challenging, requiring efficient algorithms and distributed computing frameworks.
- Over-smoothing: Deep GNNs can suffer from over-smoothing, where node representations become indistinguishable after several layers, reducing model performance.
- Lack of Interpretability: The black-box nature of GNNs poses challenges in understanding their decision-making process, which is crucial for sensitive applications.
- Limited Expressiveness: Some GNN architectures struggle to capture long-range dependencies and complex interactions, necessitating further research into more expressive models.

Conclusion

Graph Neural Network have revolutionized the field of Deep Learning by providing powerful tools for modeling graph-structured data. They have demonstrated remarkable success across various domains, including social network analysis, bioinformatics, recommendation systems, and NLP. Despite their advantages, GNNs face challenges related

to scalability, interpretability, and computational complexity. Ongoing research aims to address these challenges, exploring new architectures, efficient training techniques, and methods to enhance expressiveness and interpretability. The future of GNNs looks promising, with potential breakthroughs that could further expand their applications and impact.

2.3 Building Energy Management Systems

Building Energy Management System (BEMS) are critical in the effort to reduce energy consumption and enhance sustainability in residential, commercial, and industrial buildings. BEMS integrate various technologies to monitor, control, and optimize the energy usage within buildings, ensuring efficient operation and reduced environmental impact. This literature review examines the evolution, methodologies, technologies, applications, challenges, and future directions of BEMS. insert somewhere Manic et al., 2016

Historical Context and Evolution

Early Developments

The concept of managing building energy consumption dates back to the 1970s, spurred by the oil crises that highlighted the need for energy conservation. Initial efforts focused on manual control systems and basic automation to manage heating, ventilation, and air conditioning (HVAC) systems. These early systems were often limited in scope and functionality, primarily due to technological constraints.

Advancements in Automation and Control

The 1980s and 1990s saw significant advancements in automation and control technologies, enabling more sophisticated BEMS. The integration of microprocessors and the development of control algorithms allowed for more precise control of building systems. During this period, building automation systems (BAS) began to emerge, providing centralized control over HVAC, lighting, and other building systems.

The Digital Age and Smart Buildings

The turn of the 21st century brought about the digital age, which revolutionized BEMS. The advent of the Internet of Things (IoT), advanced sensors, and big data analytics

enabled real-time monitoring and control of building energy systems. Smart buildings, equipped with intelligent BEMS, became capable of autonomously adjusting energy usage based on occupancy, weather conditions, and other factors.

Key Components and Methodologies

Sensors and Data Acquisition

Sensors are fundamental to BEMS, providing the necessary data for monitoring and control. These include temperature sensors, humidity sensors, occupancy sensors, and energy meters. Data acquisition systems collect and transmit sensor data to the BEMS for processing.

Advantages:

- Real-time monitoring of various parameters.
- Enhanced data accuracy and reliability.

Challenges:

- Integration of heterogeneous sensor types.
- Managing and processing large volumes of data.

Control Systems

Control systems are the core of BEMS, responsible for regulating building systems based on sensor inputs and predefined algorithms. These systems use various control strategies, including:

- Proportional-Integral-Derivative (PID) Control: A feedback control loop mechanism widely used in industrial control systems.
- Model Predictive Control (MPC): An advanced control strategy that uses a model of the system to predict future states and optimize control actions.

Advantages:

- Improved energy efficiency through precise control.

- Ability to adapt to changing conditions.

Challenges:

- Complexity in modeling and implementing control algorithms.
- Ensuring robustness and reliability in real-world conditions.

Communication Networks

Effective communication networks are essential for the seamless operation of BEMS. These networks facilitate data exchange between sensors, control systems, and management platforms. Common communication protocols include BACnet, Modbus, and LonWorks.

Advantages:

- Enhanced interoperability between different systems and devices.
- Real-time data transmission and control.

Challenges:

- Ensuring cybersecurity and data privacy.
- Integrating legacy systems with modern communication protocols.

Data Analytics and Machine Learning

Data analytics and machine learning play a crucial role in modern BEMS, enabling predictive maintenance, anomaly detection, and optimization of energy usage. Techniques such as regression analysis, clustering, and neural networks are commonly used.

Advantages:

- Enhanced predictive capabilities and decision-making.
- Continuous improvement through learning and adaptation.

Challenges:

- Handling large datasets and ensuring data quality.
- Developing accurate and generalizable models.

Applications of BEMS

Energy Optimization

BEMS are primarily used to optimize energy consumption in buildings, reducing operational costs and environmental impact. Techniques such as demand response, load shifting, and peak load management are employed to achieve these goals.

Examples:

- Demand Response: Adjusting energy usage based on real-time electricity prices or grid demand.
- Load Shifting: Moving energy-intensive tasks to off-peak hours to reduce demand charges.

HVAC Control

HVAC systems are significant energy consumers in buildings. BEMS optimize HVAC operations by adjusting temperature settings, controlling airflow, and managing equipment schedules based on occupancy and weather forecasts.

Examples:

- Temperature Control: Maintaining optimal indoor temperatures while minimizing energy use.
- Ventilation Management: Regulating airflow to ensure indoor air quality and energy efficiency.

Lighting Control

BEMS improve lighting efficiency by using occupancy sensors, daylight harvesting, and automated scheduling. These systems ensure that lights are used only when needed, reducing energy waste.

Examples:

- Occupancy Sensors: Automatically turning off lights in unoccupied areas.
- Daylight Harvesting: Adjusting artificial lighting based on the availability of natural light.

Renewable Energy Integration

BEMS facilitate the integration of renewable energy sources, such as solar panels and wind turbines, into building energy systems. They manage the generation, storage, and consumption of renewable energy, maximizing its use and reducing reliance on the grid.

Examples:

- Solar Energy Management: Optimizing the use of solar power through real-time monitoring and control.
- Energy Storage: Managing battery storage systems to balance supply and demand.

Building Performance Monitoring

BEMS continuously monitor building performance, providing insights into energy consumption patterns, equipment performance, and potential areas for improvement. This data is used to identify inefficiencies and implement corrective actions.

Examples:

- Energy Audits: Conducting regular audits to assess energy performance and identify savings opportunities.
- Fault Detection: Identifying and diagnosing equipment faults to ensure optimal operation.

Challenges in BEMS

Integration of Diverse Systems

One of the primary challenges in BEMS is integrating diverse systems and devices, often from different manufacturers, into a cohesive management platform. This requires standardized communication protocols and interoperability.

Solutions:

- Developing and adopting open standards for communication and data exchange.
- Using middleware solutions to bridge compatibility gaps.

Cybersecurity and Data Privacy

As BEMS become more connected and data-driven, ensuring cybersecurity and data privacy becomes critical. Unauthorized access to BEMS can lead to operational disruptions, data breaches, and even physical damage.

Solutions:

- Implementing robust encryption and authentication mechanisms.
- Regularly updating and patching software to address vulnerabilities.

Scalability and Flexibility

BEMS must be scalable and flexible to accommodate changing building requirements and technological advancements. This involves designing systems that can easily integrate new devices and functionalities.

Solutions:

- Using modular architectures that allow for incremental upgrades.
- Leveraging cloud-based platforms for scalable data storage and processing.

Cost and Return on Investment

The initial cost of implementing BEMS can be high, posing a barrier for adoption. Demonstrating a clear return on investment (ROI) is essential to justify the costs.

Solutions:

- Conducting detailed cost-benefit analyses to highlight long-term savings.
- Providing financing options and incentives for energy-efficient upgrades.

Future Directions

Advanced Analytics and Machine Learning

The future of BEMS lies in the continued development and application of advanced analytics and machine learning techniques. These technologies will enable more accurate predictions, adaptive control strategies, and automated fault detection.

Examples:

- Predictive Maintenance: Using machine learning to predict equipment failures and schedule maintenance proactively.
- Energy Forecasting: Leveraging advanced analytics to predict energy consumption and optimize operations.

Internet of Things (IoT) and Edge Computing

The proliferation of IoT devices and the advent of edge computing will transform BEMS by enabling real-time data processing and control at the edge of the network. This will enhance responsiveness and reduce latency.

Examples:

- Edge Analytics: Performing data analysis and decision-making at the edge, closer to the source of data.
- IoT Integration: Connecting a wide range of sensors and devices to create a more interconnected and responsive BEMS.

Blockchain for Energy Management

Blockchain technology holds potential for enhancing transparency, security, and efficiency in energy management. It can facilitate peer-to-peer energy trading, secure data sharing, and decentralized control.

Examples:

- Peer-to-Peer Energy Trading: Enabling buildings to trade excess energy directly with each other using blockchain.
- Secure Data Sharing: Using blockchain to ensure the integrity and security of energy data.

Human-Centric Building Management

Future BEMS will increasingly focus on human-centric building management, prioritizing occupant comfort and well-being. This involves integrating indoor environmental quality (IEQ) metrics and user preferences into energy management strategies.

Examples:

- Occupant Feedback Systems: Using feedback from occupants to adjust building systems for optimal comfort.
- Personalized Climate Control: Implementing systems that tailor the indoor environment to individual preferences.

Conclusion

Building Energy Management System are essential for achieving energy efficiency and sustainability in modern buildings. They integrate various technologies to monitor, control, and optimize energy usage, providing significant benefits in terms of cost savings and environmental impact. Despite the challenges, advancements in analytics, IoT, blockchain, and human-centric design offer promising directions for the future of BEMS. Continued research and innovation will drive the evolution of these systems, making them more intelligent, responsive, and sustainable.

2.4 Related Work

2.5 Research Gap

...

Chapter 3

Conclusion

Bibliography

Bizer, C., Heath, T., & Berners-Lee, T. (2023). Linked data - the story so far. In *Linking the world's information: Essays on tim berners-lee's invention of the world wide web* (1st ed., pp. 115–143). Association for Computing Machinery. https://doi.org/10.1145/3591366.3591378

Bonatti, P., Kirrane, S., Polleres, A., & Wenning, R. (2017). Transparent personal data processing: The road ahead. In S. Tonetta, E. Schoitsch & F. Bitsch (Eds.), *Computer safety, reliability, and security* (pp. 337–349). Springer International Publishing.

Bordes, A., Usunier, N., Garcia-Durán, A., Weston, J., & Yakhnenko, O. (2013). Translating embeddings for modeling multi-relational data.

Bruna, J., Zaremba, W., Szlam, A., & LeCun, Y. (2013). Spectral networks and locally connected networks on graphs. http://arxiv.org/abs/1312.6203

Chaudhri, V. K., Baru, C., Chittar, N., Dong, X. L., Genesereth, M., Hendler, J., Kalyanpur, A., Lenat, D. B., Sequeda, J., Vrandečić, D., & Wang, K. (2022). Knowledge graphs: Introduction, history, and perspectives. *AI Magazine*, *43*, 17–29. https://doi.org/10.1002/aaai.12033

Chen, J., Zhu, J., & Song, L. (2017). Stochastic training of graph convolutional networks with variance reduction. http://arxiv.org/abs/1710.10568

Chiang, W. L., Li, Y., Liu, X., Bengio, S., Si, S., & Hsieh, C. J. (2019). Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 257–266. https://doi.org/10.1145/3292500.3330925

Cui, Z., Henrickson, K., Ke, R., & Wang, Y. (2020). Traffic graph convolutional recurrent neural network: A deep learning framework for network-scale traffic learning and forecasting. *IEEE Transactions on Intelligent Transportation Systems*, *21*, 4883–4894. https://doi.org/10.1109/TITS.2019.2950416

Cyganiak, R., Lanthaler, M., & Wood, D. (2014). *RDF 1.1 concepts and abstract syntax* (W3C Recommendation). W3C. https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/

Deborah, L. M., & van Harmelen Frank. (2004). Owl web ontology language overview. http://www.w3.org/TR/2003/PR-owl-features-20031215/

Defferrard, M., Bresson, X., & Vandergheynst, P. (2016). Convolutional neural networks on graphs with fast localized spectral filtering. https://github.com/mdeff/cnn_graph

Fan, W., Ma, Y., Li, Q., He, Y., Zhao, E., Tang, J., & Yin, D. (2019). Graph neural networks for social recommendation. *The Web Conference 2019 - Proceedings of the World Wide Web Conference, WWW 2019*, 417–426. https://doi.org/10.1145/3308558.3313488

Fernández, M., Cantador, I., López, V., Vallet, D., Castells, P., & Motta, E. (2011). Semantically enhanced information retrieval: An ontology-based approach. *Journal of Web Semantics*, *9*, 434–452. https://doi.org/10.1016/j.websem.2010.11.003

Fout, A., Byrd, J., Shariat, B., & Ben-Hur, A. (2017). Protein interface prediction using graph convolutional networks.

Francis, N., Green, A., Guagliardo, P., Libkin, L., Lindaaker, T., Marsault, V., Plantikow, S., Rydberg, M., Selmer, P., & Taylor, A. (2018). Cypher: An evolving query language for property graphs. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1433–1445. https://doi.org/10.1145/3183713.3190657

G. (Grigoris) Antoniou and Frank. Van Harmelen. (2008). *A semantic web primer*. MIT Press.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., & Dahl, G. E. (2017). Neural message passing for quantum chemistry.

Gori, M., Monfardini, G., & Scarselli, F. (2005). A new model for learning in graph domains. *IEEE International Joint Conference on Neural Networks*, *5*. https://doi.org/10.1109/IJCNN.2005.1555942

Grover, A., & Leskovec, J. (2016). Node2vec: Scalable feature learning for networks. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, *13-17-August-2016*, 855–864. https://doi.org/10.1145/2939672.2939754

Hamilton, W. L., Ying, R., & Leskovec, J. (2017). Inductive representation learning on large graphs.

Hogan, A., Blomqvist, E., Cochez, M., D'Amato, C., Melo, G. D., Gutierrez, C., Kirrane, S., Gayo, J. E. L., Navigli, R., Neumaier, S., Ngomo, A. C. N., Polleres, A., Rashid, S. M., Rula, A., Schmelzeisen, L., Sequeda, J., Staab, S., & Zimmermann, A. (2021). Knowledge graphs. *ACM Computing Surveys*, *54*. https://doi.org/10.1145/3447772

Jin, W., Barzilay, R., & Jaakkola, T. (2018). Junction tree variational autoencoder for molecular graph generation.

Kapanipathi, P., Abdelaziz, I., Ravishankar, S., Roukos, S., Gray, A., Astudillo, R., Chang, M., Cornelio, C., Dana, S., Fokoue, A., Garg, D., Gliozzo, A., Gurajada, S., Karanam, H., Khan, N., Khandelwal, D., Lee, Y.-S., Li, Y., Luus, F., ... Yu, M. (2020). Leveraging abstract meaning representation for knowledge base question answering. http://arxiv.org/abs/2012.01707

Kejriwal, M. (2022). Knowledge graphs: A practical review of the research landscape. https://doi.org/10.3390/info13040161

Khemani, B., Patil, S., Kotecha, K., & Tanwar, S. (2024). A review of graph neural networks: Concepts, architectures, techniques, challenges, datasets, applications, and future directions. *Journal of Big Data*, *11*. https://doi.org/10.1186/s40537-023-00876-4

Kipf, T. N., & Welling, M. (2017). Semi-supervised classification with graph convolutional networks. http://arxiv.org/abs/1609.02907

Lee, J. B., Rossi, R., & Kong, X. (2018). Graph classification using structural attention. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1666–1674. https://doi.org/10.1145/3219819.3219980

Li, Q., Han, Z., & Wu, X.-M. (2018). Deeper insights into graph convolutional networks for semi-supervised learning. http://arxiv.org/abs/1801.07606

Lin, Y., Liu, Z., Sun, M., Liu, Y., & Zhu, X. (2015). Learning entity and relation embeddings for knowledge graph completion. www.aaai.org

Manic, M., Wijayasekara, D., Amarasinghe, K., & Rodriguez-Andina, J. J. (2016). Building energy management systems: The age of intelligent and adaptive buildings. *IEEE Industrial Electronics Magazine*, *10*(1), 25–39.

Monti, F., Boscaini, D., Masci, J., Rodolà, E., Svoboda, J., & Bronstein, M. M. (2016). Geometric deep learning on graphs and manifolds using mixture model cnns. http://arxiv.org/abs/1611.08402

Paulheim, H. (2017). Knowledge graph refinement: A survey of approaches and evaluation methods. *Semantic Web*, *8*, 489–508. https://doi.org/10.3233/SW-160218

Pérez, J., Arenas, M., & Gutierrez, C. (2009). Semantics and complexity of sparql. *ACM Transactions on Database Systems*, *34*. https://doi.org/10.1145/1567274.1567278

Perozzi, B., Al-Rfou, R., & Skiena, S. (2014). Deepwalk: Online learning of social representations. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 701–710. https://doi.org/10.1145/2623330.2623732

Pujara, J., Miao, H., Getoor, L., & Cohen, W. (2013). Knowledge graph identification. *The Semantic Web–ISWC 2013: 12th International Semantic Web Conference, Sydney, NSW, Australia, October 21-25, 2013, Proceedings, Part I 12*, 542–557.

Rong, Y., Huang, W., Xu, T., & Huang, J. (2019). Dropedge: Towards deep graph convolutional networks on node classification. http://arxiv.org/abs/1907.10903

Sahu, S. K., Christopoulou, F., Miwa, M., & Ananiadou, S. (2019). Inter-sentence relation extraction with document-level graph convolutional neural network. http://arxiv.org/abs/1906.04684

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., & Monfardini, G. (2009). The graph neural network model. *IEEE Transactions on Neural Networks*, *20*, 61–80. https://doi.org/10.1109/TNN.2008.2005605

Singhal, A., et al. (2012). Introducing the knowledge graph: Things, not strings. *Official google blog*, *5*(16), 3.

Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J., & Mei, Q. (2015). Line: Large-scale information network embedding. *WWW 2015 - Proceedings of the 24th International Conference on World Wide Web*, 1067–1077. https://doi.org/10.1145/2736277.2741093

Tchechmedjiev, A., Fafalios, P., Boland, K., Gasquet, M., Zloch, M., Zapilko, B., Dietze, S., Todorov, K., & Zloch, M. (2019). Claimskg: A knowledge graph of fact-checked claims, 309–324. https://doi.org/10.1007/978-3-030-30796-7_20

Thekumparampil, K. K., Wang, C., Oh, S., & Li, L.-J. (2018). Attention-based graph neural network for semi-supervised learning. http://arxiv.org/abs/1803.03735

Uschold, M., & Gruninger, M. (1996). Ontologies: Principles, methods and applications.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., & Bengio, Y. (2018). Graph attention networks. http://arxiv.org/abs/1710.10903

Wang, Q., Mao, Z., Wang, B., & Guo, L. (2017). Knowledge graph embedding: A survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering*, *29*, 2724–2743. https://doi.org/10.1109/TKDE.2017.2754499

Wang, X., He, X., Wang, M., Feng, F., & Chua, T. S. (2019). Neural graph collaborative filtering. *SIGIR 2019 - Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 165–174. https://doi.org/10.1145/3331184.3331267

Wang, Z., Zhang, J., Feng, J., & Chen, Z. (2014). Knowledge graph embedding by translating on hyperplanes. www.aaai.org

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., & Yu, P. S. (2021). A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, *32*, 4–24. https://doi.org/10.1109/TNNLS.2020.2978386

Xu, K., Hu, W., Leskovec, J., & Jegelka, S. (2019). How powerful are graph neural networks?

Yang, J., Lu, J., Lee, S., Batra, D., & Parikh, D. (2018). Graph r-cnn for scene graph generation. *Proceedings of the European Conference on Computer Vision (ECCV)*.

Yasunaga, M., Ren, H., Bosselut, A., Liang, P., & Leskovec, J. (2021). Qa-gnn: Reasoning with language models and knowledge graphs for question answering. http://arxiv.org/abs/2104.06378

Ying, R., He, R., Chen, K., Eksombatchai, P., Hamilton, W. L., & Leskovec, J. (2018). Graph convolutional neural networks for web-scale recommender systems. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 974–983. https://doi.org/10.1145/3219819.3219890

Zeng, H., Zhou, H., Srivastava, A., Kannan, R., & Prasanna, V. (2019). Graphsaint: Graph sampling based inductive learning method. http://arxiv.org/abs/1907.04931

Zhang, W., Deng, S., Chen, M., Wang, L., Chen, Q., Xiong, F., Liu, X., & Chen, H. (2021). Knowledge graph embedding in e-commerce applications: Attentive reasoning, explanations, and transferable rules. *ACM International Conference Proceeding Series*, 71–79. https://doi.org/10.1145/3502223.3502232

Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., & Sun, M. (2020). Graph neural networks: A review of methods and applications. https://doi.org/10.1016/j.aiopen.2021.01.001

Zou, X. (2020). A survey on application of knowledge graph. *Journal of Physics: Conference Series*, *1487*. https://doi.org/10.1088/1742-6596/1487/1/012016

Abbreviations

AI Artifical Intelligence. 2

BEMS Building Energy Management System. 22, 23, 25–29

CNN Convolutional Neural Network. 9, 10

DL Deep Learning. 6, 7, 9, 10, 21

GAT Graph Attention Network. 10, 16

GCN Graph Convolutional Network. 10, 15

GNN Graph Neural Network. 7, 9–12, 14, 15, 17, 19–22, 35

KG Knowledge Graph. 2–9, 35

KGE Knowledge Graph Embedding. 7

LOD Linked Open Data. 8

ML Machine Learning. 6, 7, 9, 10

MPNN Message Passing Neural Network. 17

NLP Natural Language Processing. 6, 7, 9, 20, 21

OWL Web Ontology Language. 4, 5

RDF Resource Description Framework. 3–5

RDFS RDF Schema. 4

RNN Recurrent Neural Network. 9, 10

RQ Research Question. 1

SPARQL SPARQL Protocol and RDF Query Language. 5

URI Uniform Resource Identifier. 4, 5

List of Figures

List of Tables

Acknowledgements

The acknowledgements and the people to thank go here, don't forget to include your project advisor...