Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Bibliobuild

Bhushan Milind Borole

A Dissertation

Presented to the University of Dublin, Trinity College

in partial fulfilment of the requirements for the degree of

Master of Science in Computer Science (Intelligent Systems)

Supervisor: Yvette Graham

August 2022

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

Bhushan Milind Borole

August 19, 2022

Permission to Lend and/or Copy

I, the undersigned, agree that Trinity College Library may lend or copy this thesis upon request.

Bhushan Milind Borole

August 19, 2022

Bibliobuild

Bhushan Milind Borole, Master of Science in Computer Science

University of Dublin, Trinity College, 2022

Supervisor: Yvette Graham

While doing some research, one tends to get lost in finding papers related to the idea or any other paper/article. Since everything is digital and connected, there are humongous amounts of research papers or academic articles available at your fingertip. They are a source of data that can have many wide applications, which people have not worked upon in the past, but searching through them is a dilemma. This dissertation aims to reduce that task by using ML algorithms to find similarity between papers moreover, by generating a graph that will help in easy visualization of the same. The significant challenges faced were: execution time and visualizing a massive graph. Checking whether two papers are similar or not is based on their *Title* and *Abstract*. ACL Anthology dataset was used that had 77,000 research papers, out of which approximately 30,000 were in English and had both *Title* and *Abstract* fields. We find semantic similarities between their titles and abstract's using a rich machine learning model (i.e., BERT) and then apply weights to merge them and get a single similarity metric value. Then if the value crosses the threshold, it can be said that the given two papers are similar or share some similar ideas. Based on the dataset, a prototype graph was created, which showed similar papers properly, although with some inaccuracies. The ML models were correlated with the STS-B dataset, which gave a 0.84 correlation and another 0.9 correlation. There was no closed-source tool/data used.

Acknowledgments

My sincere thank you to my supervisor, Yvette Graham, for the valuable guidance, assistance, and providing the resources that I needed. I would also like to thank my family for their support.

Bhushan Milind Borole

University of Dublin, Trinity College
August 2022

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Introduction

When working on a project or carrying out research in a new area, one may require some background information regarding the topic or research, and that background information may be obtained by looking through various published papers/articles/journals. In academic writing, it is common to discuss multiple concepts that are related to one another or multiple points of view on the same topic. Finding such relevant papers may be facilitated by combining the papers and relating them to one another through the application of similarity metrics (in the form of a similarity graph).

A similarity graph is an undirected graph depicting similarities between various papers. Every research paper in the graph has its own node, and the connections between the nodes are what serve as evidence for the similarity between them. It is an undirected graph since there isn't a clear beginning or end; therefore, even if paper A and paper B are similar, the opposite is also true.

Various insights can be deciphered from the graph. Topic Modeling can be used to create clusters within the graphs that indicate papers that belong to a specific topic which can be helpful for faster searching of similar papers. It can also be used as a base on which a recommender system can be built. Researchers can search for the relevant papers using this graph and cite them in their papers, thus contributing to a statistical study called bibliometrics. It can also be used in plagiarism detection, searching a similar semantic paper also, and the recommendation of citations. Similarity graph can be merged with Citation

Network [10] into a powerful tool that can help researchers to continue with their research with extra information related to similar papers and other papers that have cited those similar papers at their fingertip.

1.2 Motivation

A similarity graph can be an ocean of information for researchers. However, manually compiling a similarity graph requires searching for a paper, locating papers similar to it, and then repeating the search for each one. The second step of displaying and interpreting the search results adds time, effort, and potential complications.

The objective of this dissertation was to automate the search for related research papers/articles utilizing cutting-edge machine learning algorithms and enhanced visualization capabilities, offering the researchers a rapid and straightforward application. As a result, the users of this project will be able to get papers similar to a specific paper and those similar to the papers found. This increases the user's capacity to explore a network, uncovering new articles, trends, and insights.

1.3 Research Objectives

The main objective while working on this dissertation was to find a way that can be used to check whether two academic articles or papers are similar or not. No machine learning model can be as accurate as a human brain in deciphering the similarity between papers or at any other task. However, it is also impossible for a human brain to read millions of papers and construct a graph based on them. Therefore ML models which have been trained on millions of data points and evaluated with the highest correlation were used.

The other objective was to generate a graph while keeping the computational cost at the lowest. Different ways of storing a graph were taken into consideration, and many tools were looked into which can plot a dense, complex graph. The adjacency list turned out to be reasonable compared to the Adjacency graph, as it helped in batch-wise processing and also helped in reducing the space complexity.

1.4 Key Terms

- *BERT*: Bidirectional Encoder Representations from Transformers, is a machine learning model used widely in the field of NLP.
- *NLP*: Natural Language Processing
- *ML*: Machine Learning
- *STS*: Semantic Textual Similarity

1.5 Structure of Dissertation

The current state of the art is outlined in Chapter 2, along with several visualization techniques and methodologies, as well as the components that go into creating citation networks. In Chapter 3, the application's design is described along with the justifications for the choices that were taken. Chapter 4 provides a breakdown of the steps used to create the application. Chapter 5 provides an evaluation of the application. Chapter 6 provides a summary of the project's findings.

Chapter 2

State of the Art

This chapter gives an overview of the current state of the art as well as a background on the obstacles and possibilities that come with building a similarity graph. The review begins with the dataset which will be used to build the network, which is by far the most critical component in any graph network. Then follows different ways of visualizing a dense graph, and then finally, closely related existing Machine Learning frameworks or tools that were studied in order to reach the goal.

2.1 Dataset

In order to create a graph that depicts similarities between papers, we need a dataset of research papers in order to compute how similar the papers are. There are various APIs available that can provide us with access to papers, or we can build one manually or download one off the internet. If the data is poor and less in amount, then it can affect the usefulness of the project.

2.1.1 Building a Dataset

There is a various large dataset that can be used for this project. Tensorflow's scientific-papers [11]. The datasets were retrieved from the ArXiv and PubMed OpenAccess libraries, respectively. arXiV, which is maintained and run by Cornell University, has also published a dataset [12] that contains 1.7 million articles. ACL Anthology [6] is an archive of 77,000

research papers or articles. There are also closed source datasets which are Google Scholar or Microsoft Academic Service (MAS).

2.2 Visualisation

Bibliobuild has a focus on users to help them read or discover research papers similar to ones they are currently working on. The humongous data can be visualized in various ways. Only the thing to keep in mind is that the visualization should be aesthetically pleasing and should be easily comprehensible by the user. There are multiple ways of visualizing a dense graph; some are open source libraries, whereas some are open source software tools, as well as different layouts and algorithms to make the graph aesthetic.

2.2.1 Tools helpful for Visualisation

ASK-GraphView [13] is a large-scale graph visualization system that addresses some of the most important issues researchers face - dependency on the system's hardware for, e.g., RAM, interacting with the graph, clustering. A client-server system that creates a hierarchy tree that considers the available random access memory (R), screen (S), and disk as three buffers of varying sizes and access characteristics. It supports up to 200,000 nodes. It also has a clustering feature that can be used without any requirement of additional hierarchical knowledge.

Gephi [14] is an open source tool for viewing graphs and analyzing dense networks. It speeds up the investigation by using a 3D render engine to show massive networks in real-time. Given that it is based on a multi-task paradigm, it can handle huge networks (i.e., those with more than 20,000 nodes), and it makes use of multi-core processors. Node design is customizable; in place of a traditional form, it might be a texture, a panel, or a picture. Real-time execution of highly configurable layout algorithms is possible on the graph window. For instance, in the Force Atlas algorithm, which is created by the Gephi team, real-time parameters include speed, gravity, repulsion, auto-stabilize, inertia, and size-adjust.

Computing similarity values and projection of these metrics into a 2D system are made at the cost of computation complexity. LargeVis [15] is a method that first builds a K-nearest neighbor graph from the data that is approximative in every way and then arranges the graph in a low-dimensional space. The stochastic gradient descent algorithm built with O(N) time

complexity tends to reduce the computational cost, and hence LargeVis tend to outperform state-of-the-art methods.

Graphic [16] is another such tool that is used to visualize or plot large complex datasets. It supports many input formats. It supports 3D rendering as other tools do. Graphia can generate and visualize graphs from numerical data tables and present the structures that arise. It may also be used to see and analyze data that has previously been graphed. As opposed to tools such as Gephi, it only supports one Force-Directed layout; moreover, for large graphs, the rendering options are not good. Visualizations for graphs having millions of edges are very cluttered and not aesthetic.

2.2.2 Algorithms and Layouts for Visualisation

There are various algorithms that are invented to make that plot look aesthetically pleasing to the eyes, which will reflect in getting good amount of information from the graph.

Spring-Electrical Model

Yifan Hu [17] states that one solution is to transform the graph drawing problem into the challenge of determining the minimum energy configuration of a physical system. The author discusses the Spring-Electrical Model [18, 19] and a MultiLevel Approach which is better for larger graphs. Fruchterman and Reingold [19] explain a modified version of the model as explained in [18]. The Fruchterman-Reingold layout is a force-directed layout technique that uses edges as springs to move vertexes closer or further apart in an attempt to establish an equilibrium that minimizes system energy. The main goal is to minimize energy. *Force-Directed algorithm*, an iterative algorithm, is used to reduce the model's energy by shifting the vertices in the direction of the forces acting on them.

The algorithm devised in [19] models the graph as a system of springs between neighboring vertices which tend to pull the vertices together, at the same time, repulsive forces that exist push all vertices away from each other as shown in Fig 2.1, hence minimizing the overall energy given by the formula 2.1 will result in a readable layout.

$$Energy(x, K, C) = \sum_{i \in V} f^2(i, x, K, C) \tag{2.1}$$

where:

- x is a vector of coordinates, $x = \{x_i | i \in V\}$
- K is optimal
- C regulates the forces

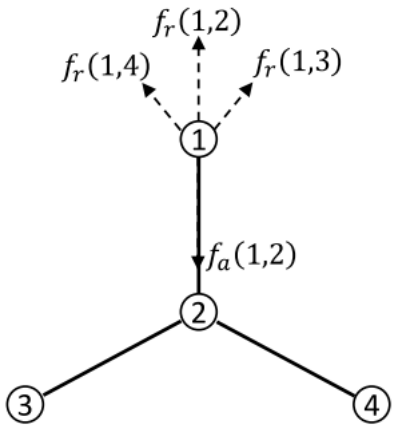

Figure 2.1: Spring Electrical
Note: Taken from [1]

Stress and Strain Model

In [17] the author also talks about Stress and Strain model, which overcomes the limitations of Spring-Electrical model, which is when the edges have fixed length. The stress model implies that springs link all pairs of network vertices, with the ideal spring length equal to the edge length. The goal is the same as in Spring-Electrical Model, to minimize the Energy which can be done using the iterative Force-Directed Algorithm in [19] or using Stress Majorization [20].

$$Energy = \sum_{i \neq j} w_{ij}(||x_i - x_j|| - d_{ij})^2 \tag{2.2}$$

where:

- d_{ij} is the distance between i^{th} and j^{th} vertex

- w is the weight factor

Strain Model is nothing but as same as Multidimensional Scaling (MDS) [21], which is used to convert "information about the pairwise 'distances' among a group of n items or persons" into a point configuration translated onto an abstract Cartesian space. In most cases, the ideal distance between all pairs of vertices must be computed, which necessitates an all-pair shortest path computation, which is why MDS is not good for large graphs.

2.3 Graphing

Graphs are a common way of denoting relationships between different parties or nodes in our case. The nodes in a graph represent items, such as research papers/articles in a network. The edges connect the nodes that exhibit relationships, such as similarities between two papers. Graphs are the most common way to illustrate relationships between parties, and a visual representation is easily understood by the human brain. If a graph has clusters, it can be helped in pattern recognition. Many different kinds of graphs serve different purposes.

2.3.1 Issues arising in graphing

While working with graphs, one major thing to keep in mind is that, how to store a graph. How is a graph stored can affect the space and time complexity of your entire program. There are five basic formats or structures in which a graph is stored - Adjacency Matrix, Adjacency List, Edge List, Incident List, Incident Matrix. All the 5 structures are compared in [22].

Adjacency matrix and Adjacency list are the most commonly used data structures for graphs with fewer nodes which is less complex. The key factors considered in the assessments were the algorithms' efficiency and space needs. Adjacency matrix, although it stores data for edges, it also stores a value of 0 where the edge is not present; hence a lot of space gets wasted. Adjacency List is considered to be the best for this dissertation because it can be used to search similar papers in a fast and efficient way; it will take less time and space required as compared to other structures. This way, duplicate entries can also be removed.

2.4 Machine Learning Approaches

It is easier for humans to understand the real meaning behind something and compare any two documents, whereas it is difficult for a machine to do so. The authors state a few reasons for this in [23]:

- Focus is not maintained in a long document, it keeps shifting from topic to topic, therefore a general meaning cannot be made out of it.
- Textual structure differs from document to document; therefore analysis gets difficult.
- Background knowledge such as fundamental lexical knowledge and common sense knowledge is frequently overlooked throughout the document creation process.

Most research documents generally have a fixed format. The authors have derived a semantic profile that covers key topics - Target, Methodology, Domain, Style, Date, and Keywords. This semantic profile makes up a general semantic representation of academic publications, as well as the development of a semantic enrichment approach based on external knowledge resources. They have also developed a framework that can determine the semantic similarity between any two profiles, which uses external resources and ontologies to provide semantic meaning. Lastly, they have come up with a Joint-Word Embedding model for better word representations. This model is also combined with background knowledge.

In [24] the authors state that the semantics of a large document can be determined by combining the semantics of tiny text units. Much recent research has used this idea to determine the semantic similarity between bigger text chunks. As we know, there is a constant shifting of focus from topic to topic, making gathering the semantics of the entire document difficult. However, the authors believe that those topics in a document are coherent, and those correlations can be obtained by conducting a comprehensive analysis of various aspects of a document. Similar to semantic profile but different in structure, the authors represent the entire meaning of a document by an event known as Topic Event (TE). The construction of TE is done in a similar way as a semantic profile in [23], built on the article structure and employs several information fields such as research objectives, methodology, keywords, and domains to explain various aspects of the research endeavor. For increasing accuracy, the authors have also developed two ontologies - Research Style Ontology and Domain Ontology.

Various BERT Approaches

BERT (Bidirectional Encoder Representation from Transformers) [2] is a model developed by Google researchers for NLP tasks. It is based on Transformers [7] which does not rely on Recurrent Neural Networks (RNN) or Convolution Neural Networks (CNN). BERT is useful where unlabeled textual data comes into the picture. It has two steps: pre-training and fine-tuning, as shown in Fig 2.2.

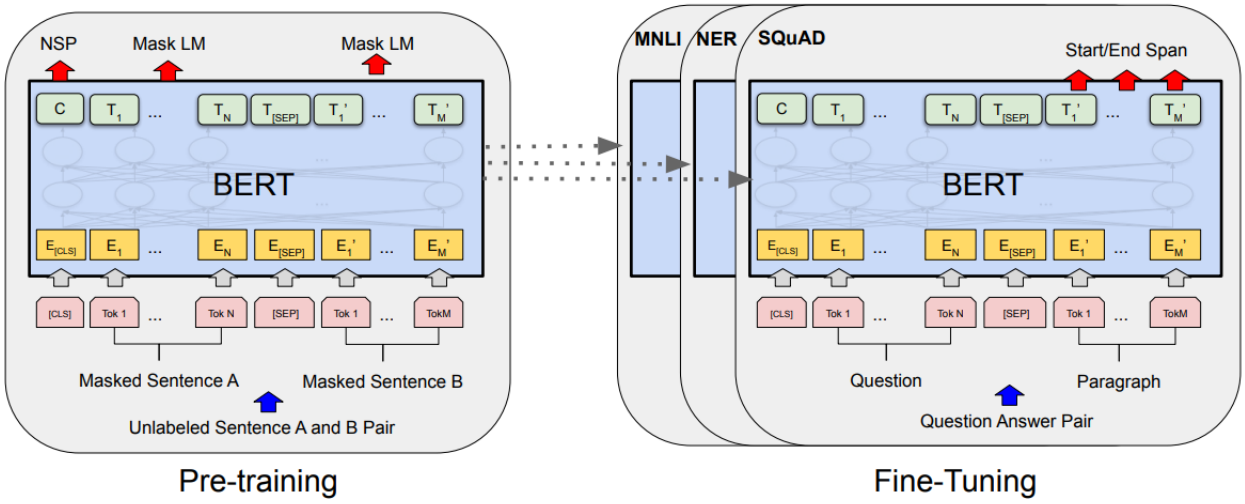

Figure 2.2: Overall pre-training and fine-tuning procedures for BERT
Note: Taken from [2]

The model is trained on unlabeled data across several pre-training tasks during pre-training. The BERT model is fine-tuned by first initializing it using the pre-trained parameters and then fine-tuning all of the parameters using labeled data from the downstream jobs.

Roberta [25] is a replication of the BERT model but outperforms all versions of BERT models. The authors have done simple and basic modifications over the original model, like training on longer sequences for a longer period of time, and they have increased the size of the batches, also increasing the volume of data to be trained on. There is a slightly different version of Roberta. The model 'stsb-roberta-large' is a variation which is trained using sentence-transformers CrossEncoder [26] class, and it is trained on STS-Benchmark data [27]. CrossEncoders are generally used where you want to compare two textual information or a list of pairs of texts. We send both phrases to the Transformer network at the same

time as shown in Fig 2.3. It then returns a number between 0 and 1, indicating how similar the input sentence pair is. CrossEncoder is more accurate than Bi-Encoder, in which we send phrases A and B to BERT separately, resulting in sentence embeddings. The cosine similarity of these sentence embeddings may then be compared.

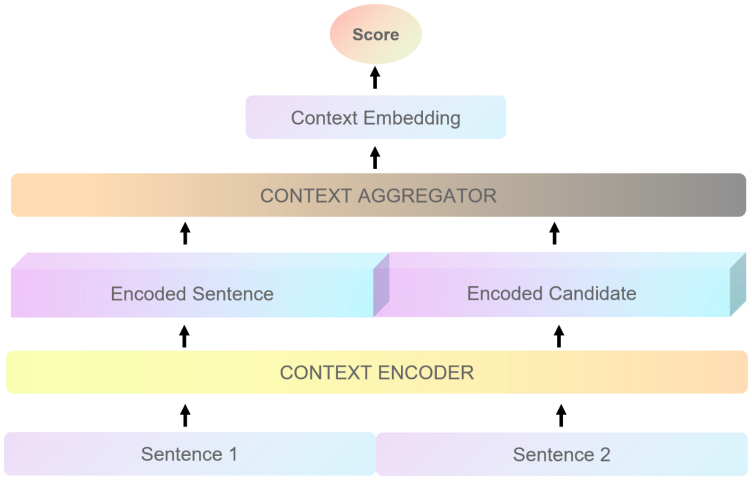

Figure 2.3: CrossEncoder
Note: Taken from [3]

The similarity between any two items in mathematical terms means how far they are apart in a Vector Space. The more the distance between them, the less similar they are and vice versa. There are many distance metrics developed - Edit Distance, Jaccard Distance, Cosine Distance. For dealing with texts, Cosine Similarity gives good results as compared to other metrics because it finds the cosine of the angle between two vectors rather than finding the distance between them. There may be a case where the distance is greater between two vectors because of the size, but the angle between them might be small. Cosine Similarity states that the smaller the angle, the more similar the vectors are. For using Cosine Similarity, the textual information should be converted into a vector-like representation that can be understood by the machine.

SPECTER [28] is a model which is developed for such purposes. In a research paper, *Title* and *Abstract* are enough to convey the general idea about the paper. SPECTER takes in the *Title* and *Abstract* of a paper and converts it into a vector representation. Internally SPECTER uses SciBERT [29] which is an enhanced version of BERT, trained on a large corpus of scientific text. Once we have the vector representations of two papers, we can

use Cosine Similarity to find how similar those two vectors are and hence how similar those papers are.

LSTM

Long Short-Term Memory (LSTM) [30] networks are recurrent neural networks as shown in Fig 2.4 that can learn order dependency in sequence prediction challenges. Long-term dependence concerns are deliberately avoided with LSTMs. Long-term memory is basically their default habit; it is not something they have to work hard to learn. This is a necessary behavior in complicated problem areas like as machine translation, speech recognition, and others.

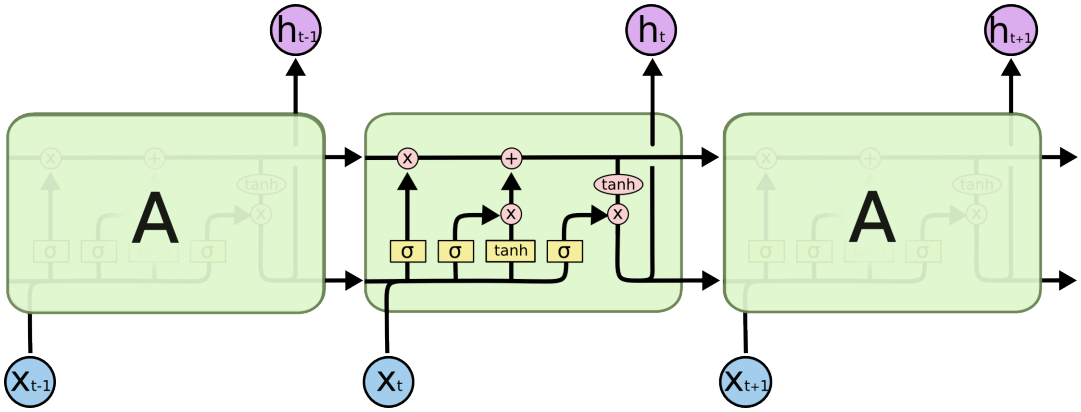

Figure 2.4: Repeating module in LSTM
Note: Taken from [4]

Manhattan LSTM (MaLSTM) [5] is an adaptation of the original LSTM. Siamese networks are those that contain two or more identical sub-networks. Siamese networks appear to perform well on similarity tests, and have been employed for tasks such as phrase semantic similarity, detecting counterfeit signatures, and many more. The architecture of MaLSTM is shown in Fig 2.5.

There are two networks, LSTMa and LSTMb, that each process one of the phrases in a given pair, however in this study, we only look at siamese architectures with linked weights, such that $LSTM_a = LSTM_b$. The model uses an LSTM to read in word-vectors representing each input sentence and employs its final hidden state as a vector representation for each

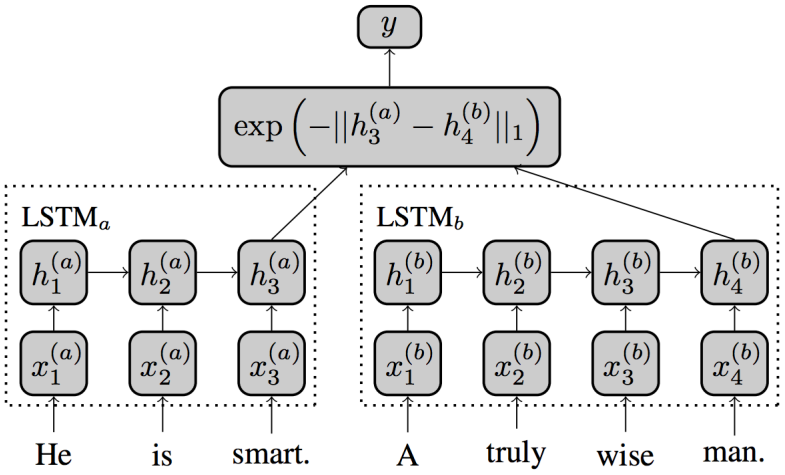

Figure 2.5: MaLSTM Architecture
Note: Taken from [5]

sentence. Subsequently, the similarity between these representations is used as a predictor of semantic similarity. The authors have names the model Manhattan LSTM, since they found using Manhattan Distance as the similarity metric increases the accuracy as compared with Cosine Similarity.

2.5 Closely-Related Projects

STriP Net - Semantic Similarity of Scientific Papers (S3P) Network

STriP Net [31] is a project whose main motive is to plot a graph of papers who are semantically similar to each other. This project leverages the used of SPECTER as explained in Section 2.4. The title and abstract are combined using a [SEP] token and the entire thing is encoded. Once the vector representations are created, then a cosine similarity matrix is matrix which is computed after computing the cosine similarity between each vector with all the other vectors. This gives an Adjacency Matrix, which can be easily plotted.

BERTopic [32] is used for Topic Modeling, which in the end is used to cluster similar topic related papers, which is very helpful in analysis of the graph. Supervised, (semi-)supervised, and dynamic topic modeling are all supported by BERTopic. BERTopic builds document embeddings using pre-trained transformer-based language models, clusters these embeddings,

and then uses the class-based TF-IDF technique to generate topic representations. BERTopic creates coherent topics and stays competitive over a wide range of benchmarks incorporating both traditional models and the more current clustering approach to topic modeling.

NetworkX [33] is a python based framework used for network analysis. NetworkX graph nodes may be any Python object, and edges can carry arbitrary data; this flexibility makes NetworkX perfect for describing networks seen in a variety of scientific subjects. PyVis [34] is a Python package that allows you to see and interact with network graphs in Jupyter notebooks or as a standalone web application. Pyvis is built on the strong and mature VisJS JavaScript framework, which enables quick and responsive interactions while abstracting away low-level JavaScript and HTML.

2.6 Conclusion

The above sections in this chapter focused on the similar work which has been done by authors on finding similarity between documents, layout of a graph, how to make a graph look aesthetically pleasing using various models and algorithms. The literature review gave critical insights that helped improve the application's performance, usefulness, and design, as well as prevent hazards.

Chapter 3

Design

3.1 Introduction

This chapter outlines the design, structure and the flow of the application, from collection and storing data, to visualising the similarity graph in the user interface. It is organized chronologically, with each successive segment building on the previous one.

3.2 ACL Anthology Dataset

This project requires a source where the papers can be obtained from. Instead of using open-source API's which would lead to more time consuming process including managing network latency, requests and quota, a dataset was used. Using an API involves responding to another entity, which does not provide additional control and usability over the data.

ACL Anthology's dataset which hosts 77,000 papers and keeps continuously growing was used. The dataset was downloaded on 24^{th} of May, 2022.

3.3 Database

Following the choice to employ a dataset, the next critical design decision was how to store it. Since we were going to apply Machine Learning methods to compare the similarity, we didn't require any traditional SQL or No-SQL database. Even various graph databases like Neo4j, GraphQL were not required.

Such a storing structure was required which was developed for easy data interchange, since the data present in ACL Anthology is in XML as shown in Fig 3.1, so instead of relying on ACL's repository for the data, it was decided to gather all the data at once and store in a particular format. JSON (Javascript Object Notation) uses less bytes for transit and it is built for easier and faster data interchange. Moreover, it is very easy to parse JSON as the parsers are less complex to understand.

```
<?xml version="1.0" encoding="UTF-8" ?>
<collection id="P18">
  <volume id="3">
    <meta>
      <booktitle>Proceedings of <fixed-case>ACL</fixed-case> 2018, Student Research Workshop</booktitle>
      <editor><first>Vered</first><last>Shwartz</last></editor>
      <url>P18-3</url>
    </meta>
    <frontmatter>
      <url>P18-3000</url>
      <!-- ... -->
    </frontmatter>
    <paper id="1">
      <title>Towards Opinion Summarization of Customer Reviews</title>
      <author><first>Samuel</first><last>Pecar</last></author>
      <url>P18-3001</url>
      <!-- ... -->
    </paper>
    <paper id="2">
      <title>Sampling Informative Training Data for <fixed-case>RNN</fixed-case> Language Models</title>
      <author><first>Jared</first><last>Fernandez</last></author>
      <author><first>Doug</first><last>Downey</last></author>
      <url>P18-3002</url>
      <!-- ... -->
    </paper>
    <paper id="3">
      <title>Learning-based Composite Metrics for Improved Caption Evaluation</title>
      <author><first>Naeha</first><last>Sharif</last></author>
      <author><first>Lyndon</first><last>White</last></author>
      <author><first>Mohammed</first><last>Bennamoun</last></author>
      <author><first>Syed Afaq</first><last>Ali Shah</last></author>
      <url>P18-3003</url>
      <!-- ... -->
    </paper>
    <!-- ...  -->
  </volume>
</collection>
```

Figure 3.1: Authoritative XML Format
Note: Taken from [6]

3.4 Service

The data should be imported from the ACL Anthology's repository and stored in JSON format for computing similarities and converting it into a graph. For visualizing the graph, since the edges are in millions, a tool had to be decided on which will be efficient and easy to use.

3.4.1 Data Operations

Python is the language that is widely used when it comes to Machine Learning and handling data, and it is easy to write and easily understandable by anyone who reads it. There are ready-made libraries and frameworks which have been created by developers for developers.

A python script was created to handle the conversion of data from XML to JSON, and once data is extracted, it will be passed to the machine learning libraries for computing the similarities. The similarity values from the ML model will be stored and passed to Gephi for visualization.

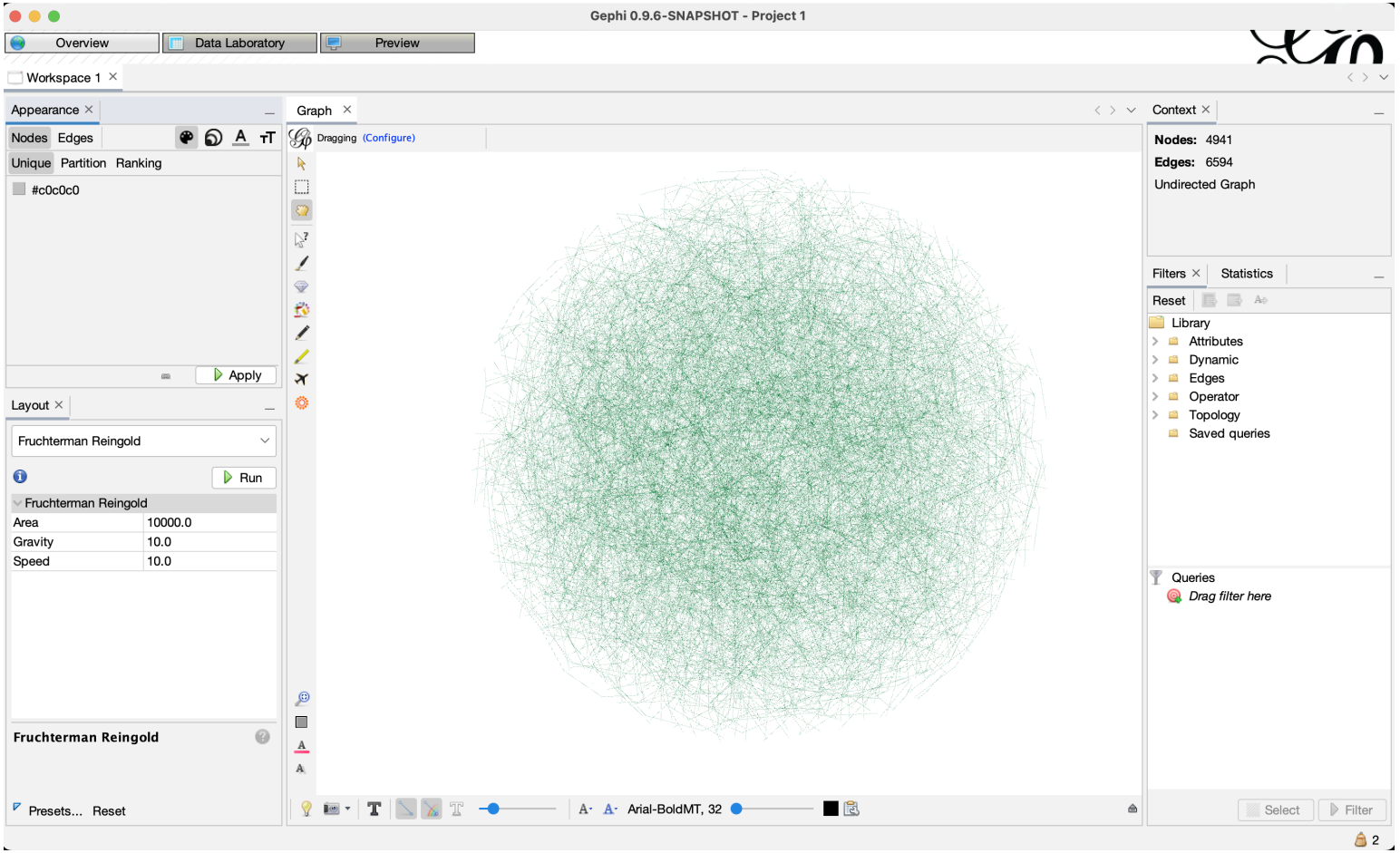

Figure 3.2: Sample UI of Gephi showing a complex undirected graph with 4941 nodes and 6594 edges.

3.4.2 User Interface

There were two ways of showing the graph to the user - using Python's inbuilt libraries like NetworkX and Plotly or using some existing tools. It was decided to go for a tool because of a few reasons:

- There is a limit to the interaction between the graph and user if libraries are used because libraries are written in Python, and it is an interpreted language. It makes the computations slow.
- Libraries are limited to scale, whereas tools such as Gephi, are built for visualizing large complex graphs.
- Graphs are not aesthetic enough when created using libraries as compared to that of using tools.

Fig 3.2 shows how a graph is being visualized in Gephi.

3.5 Similarity Graph Design

As discussed in Section 2.3.1, the graph will be stored as an adjacency list because it allows batch-wise processing and also takes up less space as compared to adjacency matrix.

For visualizing the similarity graph, there were many tools as seen in Section 2.2, which can be used. After trying out some of them and based on performances and different visualizing parameters, it was decided that Gephi was the best choice. There were a few reasons Gephi stood above some other tools.

- It is highly scalable, as it supports up to 2 million edges.
- It has many different layout configurations available.
- It is actively managed by developers who have developed different plugins for different use cases.

The research papers/articles would be represented in the form of nodes, and an edge between two nodes will represent that those two papers are similar. Graphs may illustrate a variety of data properties without using words, which saves space. Color-coding and scaling the

size of the nodes and edges, for example. Because the graph could display a huge number of articles, the user could indicate the significance of the papers using node size, making it simpler to identify papers of interest in the network.

3.6 Machine Learning Model

3.6.1 RoBERTa

There were many different BERT models to choose from, but out of those RoBERTa [25] (Robustly Optimized BERT Training Approach) was the most accurate. This approach emphasizes the significance of previously neglected design choices while also raising concerns about the source of recently reported gains. The authors have implied that the original BERT model was not up to the standard of training and have also made a few changes to it to make it more accurate:

- Longer training with more data.
- They have used long sequences and trained the model on it.
- The masking pattern on the data is being changed dynamically, whereas in BERT it is static.
- The intention of the next sequence prediction is removed.

Architecture

RoBERTa uses the Transformer architecture shown in Fig 3.3 with L number of layers. This general design is followed by the Transformer, which employs stacked self-attention and point-wise, completely linked layers for both the encoder and decoder, as seen in the left and right portions of Fig 3.3.

Encoder: Each layer of an Encoder stack comprises two sub-layers. The first is a multi-head self-attention mechanism, while the second is a basic, position-wise completely linked feed-forward network.

Decoder: The decoder inserts a third sub-layer keeping the two layers of Encoder as it is, which conducts multi-head attention over the encoder stack's output, in addition to the two sub-layers in each encoder layer.

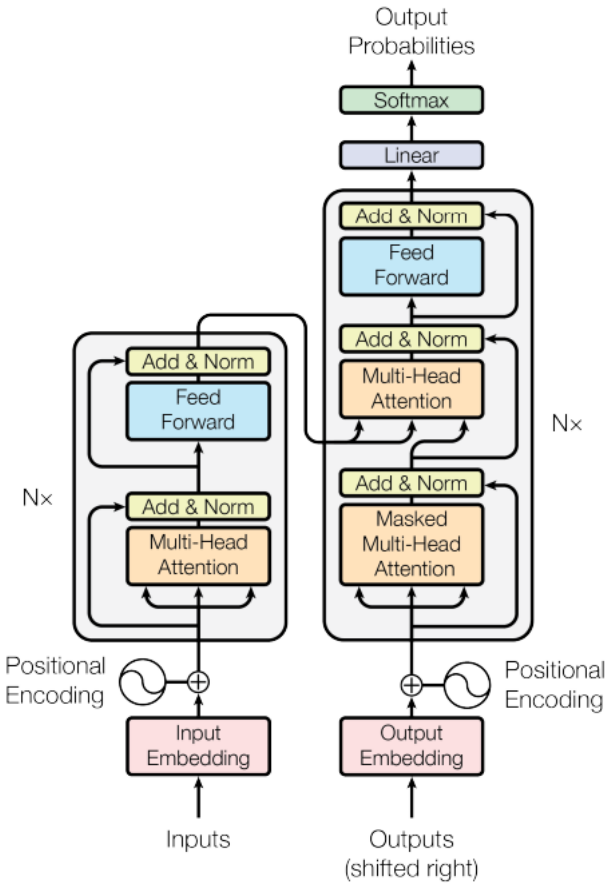

Figure 3.3: Transformer Architecture
Note: Taken from [7]

Cross-Encoder Architecture

Cross-Encoder [35] architecture (a more detailed diagram) shown in Fig 3.4 are used for sentence pair similarity or classification tasks. We send both phrases to the Transformer network at the same time. It then returns a number between 0 and 1, indicating how similar the input sentence pair is.

Cross-Encoders is be utilized if you wish to score a specific group of sentence pairings. For example, suppose you have 100 sequence pairings and wish to calculate a similarity score for each pair. This makes it a perfect choice for this dissertation since we are computing similarities between pairs of research papers and then plotting them as a graph. Cross-

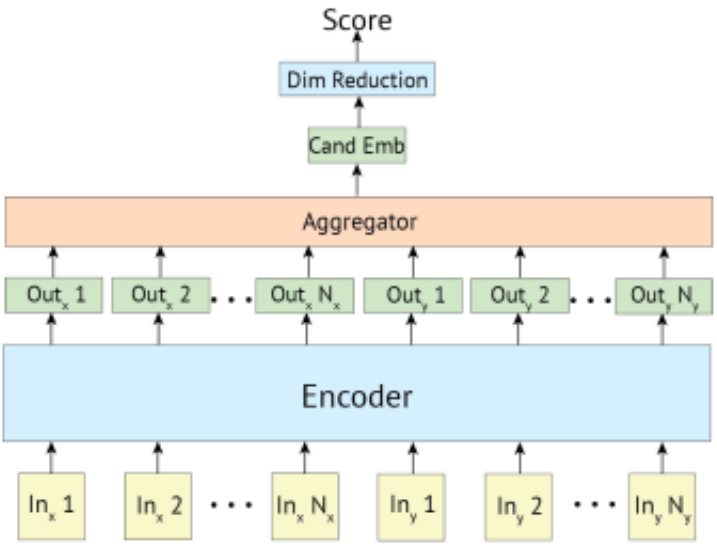

Figure 3.4: Detailed Cross-Encoder Architecture
Note: Taken from [8]

encoders, which undertake complete (cross) self-attention across a given input and label candidate and achieve substantially greater accuracies than Bi-encoder [36].

Because the input context and candidate label are concurrently encoded to create a final representation, the Cross-encoder enables for rich interactions. In [26] the authors have shown that Cross-Encoders have much better accuracy as compared to Bi-Encoder.

RoBERTa can be trained with Cross-Encoder class for calculating semantic similarity between two sentences [37].

3.7 Overview of the system

This the overview of the entire approach as shown in Fig 3.5 which is divided in 3 parts.

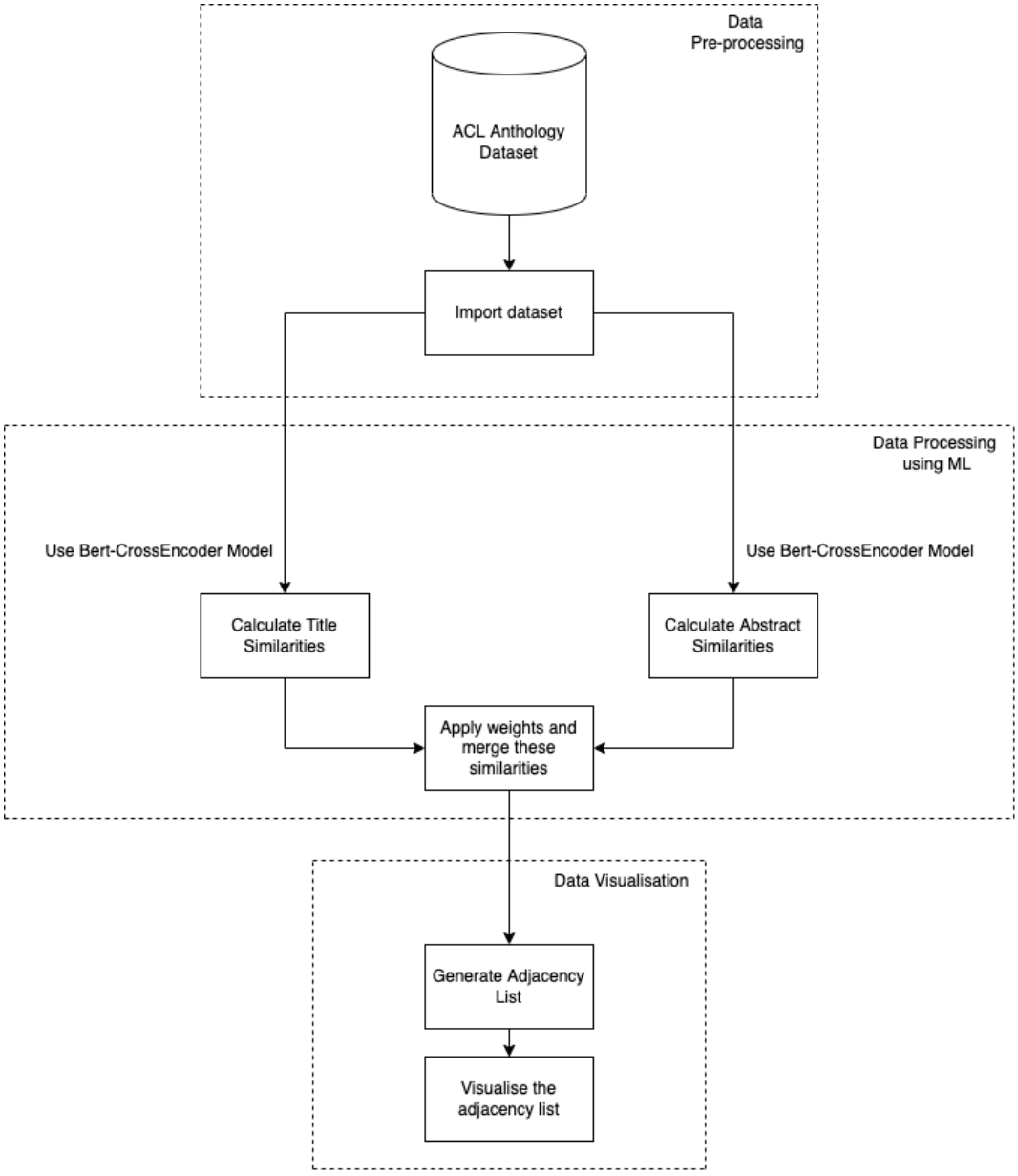

Figure 3.5: Flow diagram of the approach

Data Pre-processing

This block is where the dataset is collected from the ACL Anthology's repository and converted from XML format to JSON format.

Data Processing using ML

In this step, we calculate the title and abstract similarities using RoBERTa, and we merge them to get a final score.

Data Visualisation

In this step, we create an Adjacency List after pruning edges by using thresholding from the similarity scores received for visualization, and we import that into Gephi.

3.8 Conclusion

This section highlights the tools and models which will be used. It also explains a high level overview of the entire flow of the system. In the next section we will see a detailed implementation of the same.

Chapter 4

Implementation

4.1 Introduction

This chapter describes in detail the steps involved in developing the application. It all starts with gathering and storing data. Section 4.2 explains how the data is being processed and ready to be given as input to the ML models. The next Section, 4.3, is how the similarities are calculated and merged to obtain a single score for a pair of papers. Once you obtain the result, it is passed to the visualization tool.

4.2 Data Pre-Processing

As discussed in Section 3.3 the XML data should be converted into JSON to make the further process easy. Also, only those papers are considered in English and whose abstract is present in the XML since both fields are required to calculate the similarity between papers.

4.2.1 Data Extraction and Conversion

There is a field *langcode* present in the XML which states the language of the paper. For English the code is *eng*. But for most of the papers, this field is None, hence relying on this field was not an option. In a paper's title, for, e.g., if the paper is in Chinese, the title contained a phrase *[In Chinese]* similarly for other languages. Hence to check whether the paper is in English or not, the title had to be checked whether it had the phrase *[In* or not. If

this phrase is present, that means the paper is not in English, and hence it can be discarded.

Source Code 4.1: Data Conversion

```
1  anthology = Anthology(importdir='../data')
2
3  papers = defaultdict(dict)
4  for id_, paper in anthology.papers.items():
5      log.debug("export_anthology: processing paper '{}'".format(id_))
6      data = paper.as_dict()
7      title = paper.get_title("html")
8      if "[In" not in title: # Checks if the paper is English or not
9          if "xml_abstract" in data: #Checks if the paper has abstract
10             data["abstract"] = paper.get_abstract("html")
11             del data["xml_abstract"]
12             papers[paper.collection_id][paper.full_id] = data
13
14 with open('../../papers.json', 'w') as f:
15     json.dump(papers, f, indent=4)
```

In Code 4.1 at Line 1, we load the data using class Anthology(), which is an inbuilt class, and specify the path of the XML data. It returns an object of a class dictionary, which contains all the papers. On-Line 8 and 9, we check whether the paper is in English or not and whether the abstract is present or not. All the papers are stored in the paper's dictionary, and once all the papers are collected, they are dumped into JSON files, as shown in Line 14 and 15.

4.3 Data Processing using ML

As discussed in Section 3.6 we are going to use RoBERTa model trained with Cross-Encoder architecture. But the JSON data directly cannot be passed to the model, it needs to be `List[List[str]]`, where `str` means String. The format is a list of multiple lists, where the nested list consists of a pair of strings that need to be compared.

Source Code 4.2: Generating Pairs of papers

```
1  with open("papers.json", "r") as f:
2      papers = json.load(f)
3
4  keys = list(papers.keys())
5  combinations_keys = list(itertools.combinations(keys, 2))
6
7  combination_titles = [
8      [papers[k1]["title"], papers[k2]["title"]] for k1, k2 in combinations_keys
9  ]
10 combination_abstract = [
11     [papers[k1]["abstract"], papers[k2]["abstract"]] for k1, k2 in
       ↪  combinations_keys
12 ]
```

To perform this operation, we are going to use the itertools [38] library, which is inbuilt inside python, and has the combinations() function as shown in Line 5 in Code 4.2. The function takes in two parameters, the first one is an iterable (list of papers in our case), and the second is the count of combinations made, which is 2 in our case. We have made pairs of the IDs of the papers, and now we will just access the title and abstract using the `papers` dictionary, which is loaded from the JSON file. All the possible unique combinations of titles and abstract are stored in the variables `combination_titles` and `combination_abstract`, respectively.

Once we have the variables, we will use those as an input to the ML model as shown in Code 4.3.

Source Code 4.3: Generating Similarities

```
from sentence_transformers import CrossEncoder

model = CrossEncoder("cross-encoder/stsb-roberta-base", device="cuda")

sim_titles = model.predict(combination_titles, show_progress_bar=True)
sim_abstract = model.predict(combination_abstract, show_progress_bar=True)
```

Firstly we import the `CrossEncoder` class, and we initialize it with the name of the model which is `"cross-encoder/stsb-roberta-base"`, which loads the RoBERTa model trained with Cross-Encoder architecture. We have also passed `device="cuda"`, which means it will use the GPU (Graphical Processing Unit) of the system to calculate the similarities. The model has a `predict()` function which takes in one default argument which is the list of pairs, which we generated in 4.2, and many other optional arguments. The batch size is 32, which means at a time 32 pairs are scored, which increases the efficiency of the system. The parameter `show_progress=True`, which means it will show a progress bar that will tell how much job is done, and what the ETA is. The similarities of the pairs of titles and abstract are stored in `sim_titles` and `sim_abstract` respectively.

After obtaining the title and abstract similarities, we have to apply a weight parameter so that the similarities are combined. The equation is:

$$S = W * A_{sim} + (1 - W) * T_{sim} \tag{4.1}$$

where:

- A_{sim} is the abstract similarity
- T_{sim} is the title similarity
- W is the weight parameter applied to the abstract similarities

Source Code 4.4: Merging Similarities

```
def merge_mat(title_mat, abstract_mat, abstract_weight):
    title_weight = 1 - weight
    return np.add(title_mat * title_weight, abstract_mat * abstract_weight)

merged_sim = merge_mat(sim_titles, sim_abstract, 0.9)
```

The function `def merge_mat()` takes in the list of similarities - title and abstract, along with the weight which is to be applied to the abstract similarities, and returns the merged similarity as per the Equation 4.1. The output is stored in the variable `merged_sim`.
The next step is to apply a threshold to prune those edges which are not relevant. The similarity score ranged between 0 and 1, 0 being the least similar and 1 being the most similar.

Source Code 4.5: Thresholding

```
di = {}

for k, v in zip(combinations_keys, merged_sim):
    if v > 0.7:
        di[k] = 1

print(f"Total Edges: {len(di)}")
```

We create an empty dictionary `di`, and we simultaneously iterate through `combination_keys` and `merged_sim` while checking the score. The threshold value which is used is 0.7. If the score is above 0.7, then only the edge will be added in the dictionary `di`.

Here inside `di` we have an Edge list, but the optimal storing data structure as discussed in Section 2.3.1 is an adjacency list.

Source Code 4.6: Edge List to Adjacency List

```
1  from itertools import groupby
2  import csv
3
4  edges = di.keys()
5
6  with open("adj_list.csv", "w") as f:
7      for k, g in groupby(sorted(edges), lambda e: e[0]):
8          source = k
9          connections = [v[1] for v in g]
10         row = [source] + connections
11         writer = csv.writer(f)
12         writer.writerow(row)
```

In Line 5, we use the `groupby()` function of the `itertools` package. It create an iterator that takes input from the iterable and returns a sequence of keys and groups. A function that determines a key value for each individual element is the key. We write the Adjacency List in a specific format in a CSV file which is required by Gephi shown in Fig 4.1.

```
a;b
b;c;d
```

Figure 4.1: CSV Format

In the above figure, it represents a graph with 4 nodes a, b, c and d with 3 edges: $a \rightarrow b$, $b \rightarrow c$ and $b \rightarrow d$. Once we have the CSV, we can import that into Gephi very easily, and then Gephi will display the graph. Gephi automatically recognizes the format i.e. Adjacency List, and it plots the graph.

4.4 Results

The Figure 4.2 shows a graph of 50 nodes and 141 edges. The Figure 4.3 shows a graph of 500 nodes and 7770 edges. One can use a different layout and various filtering parameters to do a better understanding of the graph.

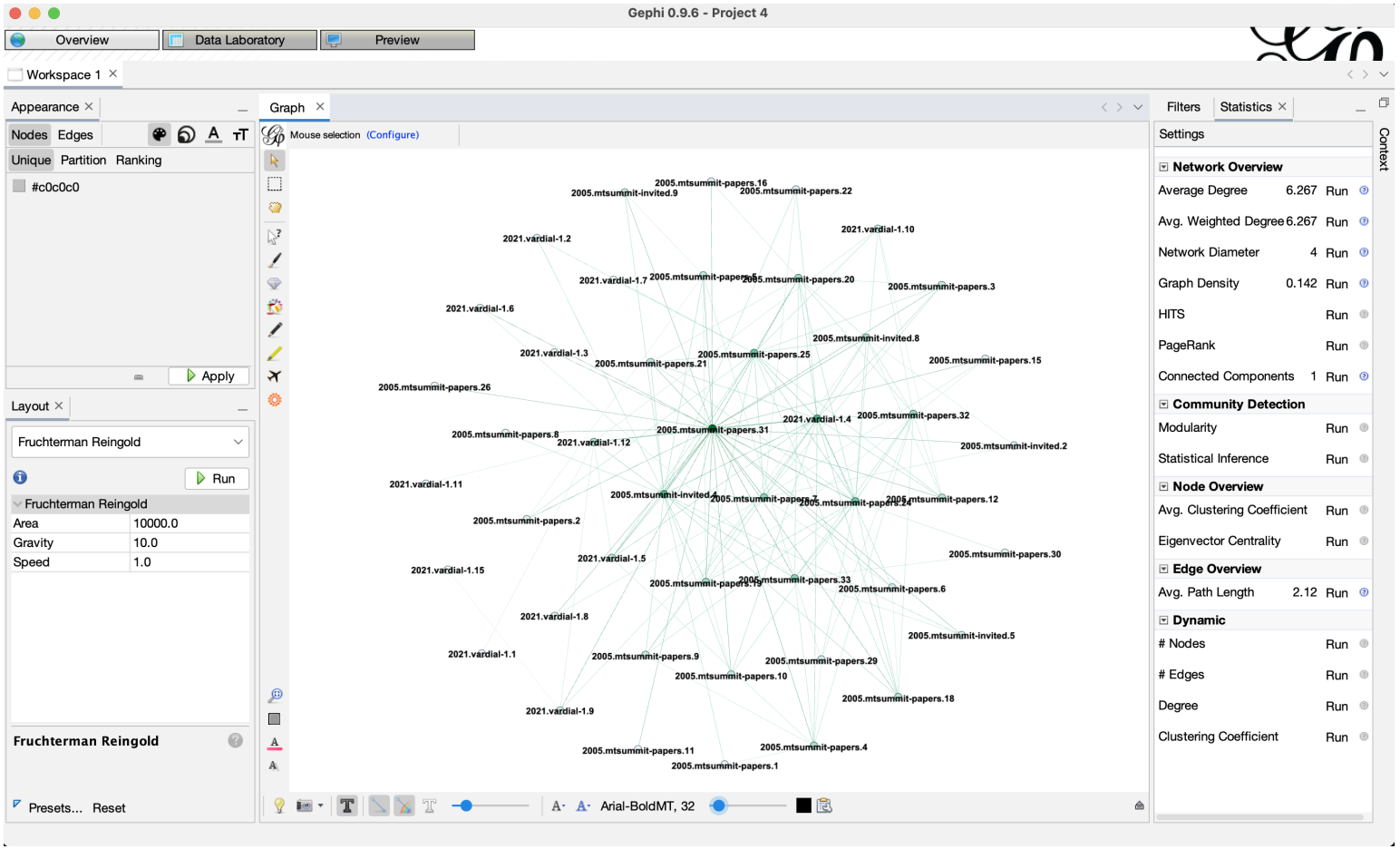

Figure 4.2: Similarity Graph with 50 nodes and 141 edges.

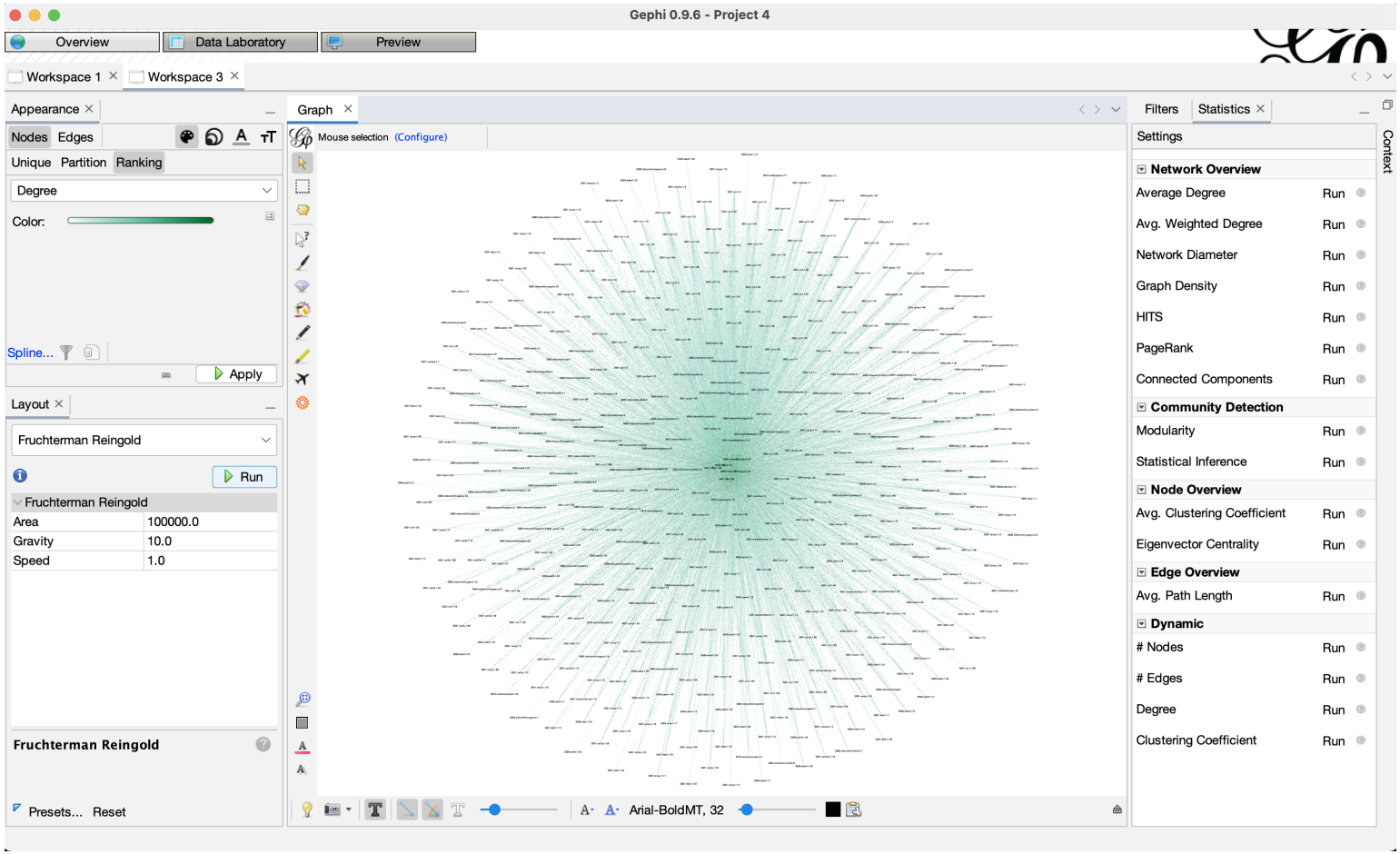

Figure 4.3: Similarity Graph with 500 nodes and 7770 edges.

4.5 Limitations

There are some limitations to this project, and it is mostly the hardware of the system. Prediction of the similarity scores happens fast, then the pairs to be compared are less, but as the number of papers increases, the inter-connections increase exponentially.

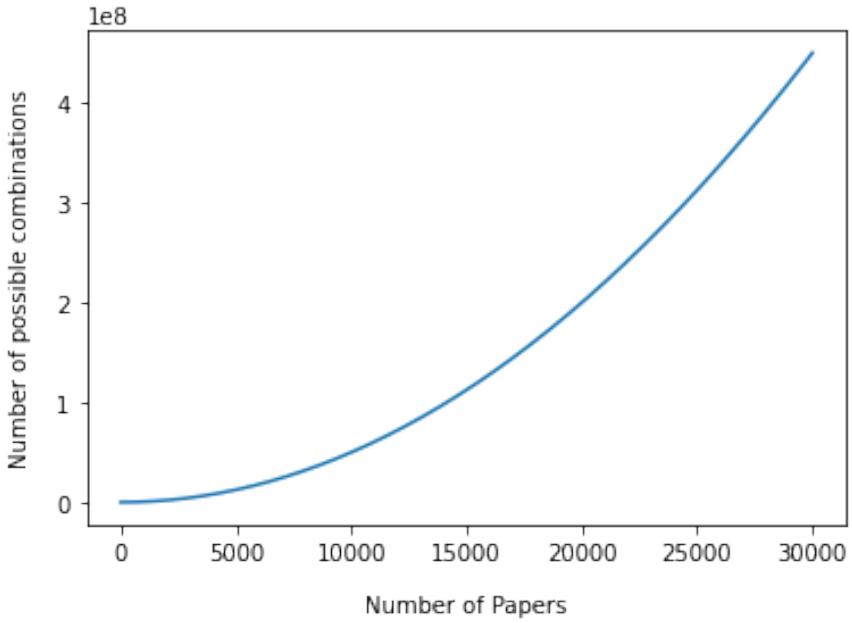

Figure 4.4: Exponential Growth of the Combinations

Fig 4.4 is a plot of the Number of Papers vs. a Possible number of unique pairs of papers, where if the number of paper increases, the combinations increases exponentially, which is also reflected in the graphs in Section 4.4. The 1e8 on the Y-axis represents the value in the order of 10^8. As the number of papers reaches 30,000, the number of combinations reaches 400 million. Thresholding turns out to be effective in pruning the edges, which are irrelevant, but any tool, which is created for visualizing graphs, cannot hold more than a million edges. It needs a huge amount of RAM, as well as processing power.

Chapter 5

Evaluation

This chapter explains all the methods that were taken into consideration the evaluate the ML model and the results to verify the output. The performance of the entire system will also be discussed here.

5.1 Evaluation of the ML Model

To evaluate any machine learning model, generally, it is compared against benchmark data. We have used STS-Benchmark [27] data to satisfy the purpose. The STS Benchmark is a collection of English datasets used in STS activities conducted between 2012 and 2017 under the SemEval framework. Text from news headlines, user forums, and picture descriptions are among the datasets used. The dataset consists of 5749 training labels and 1379 testing labels. The format of the data is shown in Fig 5.1. The format of the data is shown in Fig 5.1.

	sentence1	sentence2	similarity_score
0	A girl is styling her hair.	A girl is brushing her hair.	2.5
1	A group of men play soccer on the beach.	A group of boys are playing soccer on the beach.	3.6
2	One woman is measuring another woman's ankle.	A woman measures another woman's ankle.	5.0
3	A man is cutting up a cucumber.	A man is slicing a cucumber.	4.2
4	A man is playing a harp.	A man is playing a keyboard.	1.5

Figure 5.1: STS-Benchmark Data

Whether the two variables or two outputs have any relationship or not is done by calculating the correlation between them. The higher the correlation, the strong relationship between the two outputs. We perform the evaluation methods on the *test* labels of the dataset. The sample size is 1379. Then, we run the model on the STS-Benchmark data and compare the model's similarity scores and the score present in the data.

5.1.1 Pearson Correlation Coefficient

It is a technique for determining the linear correlation between two distinct data sets. In its most fundamental form, it is a normalized covariance measurement; hence the result is guaranteed to have a value between -1 and 1. Similar to covariance, the measure can only describe a linear correlation between variables and disregards many other conceivable interactions or connections.
The formula for Pearson's correlation is:

$$r_{X,Y} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{5.1}$$

where:

- n = sample size
- x_i, y_i = individual sample points at index i
- $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ sample mean

P-value

We have to define the null and alternate hypotheses. The hypothesis is as follows:

$$H_0 : \rho = 0$$
$$H_1 : \rho \neq 0$$

where:

- H_0 is the null hypothesis states that the correlation coefficient is 0 which means there is no significant result

- H_1 is an alternate hypothesis that states there is a significant result

When doing hypothesis testing, p-values are often employed to evaluate whether the null hypothesis should be rejected or not rejected. We calculate the p-value using the two-tailed test with $n-2$ degrees of freedom. A two-tailed test allocates half of your alpha to evaluating the statistical significance in one direction and half of your alpha to testing the statistical significance in the other direction if you are employing a significance level of 0.05. This indicates that your test statistic's distribution has 0.025 in each tail. No matter the direction of the association you hypothesize, by utilizing a two-tailed test, you are testing for the potential of the link in both ways.

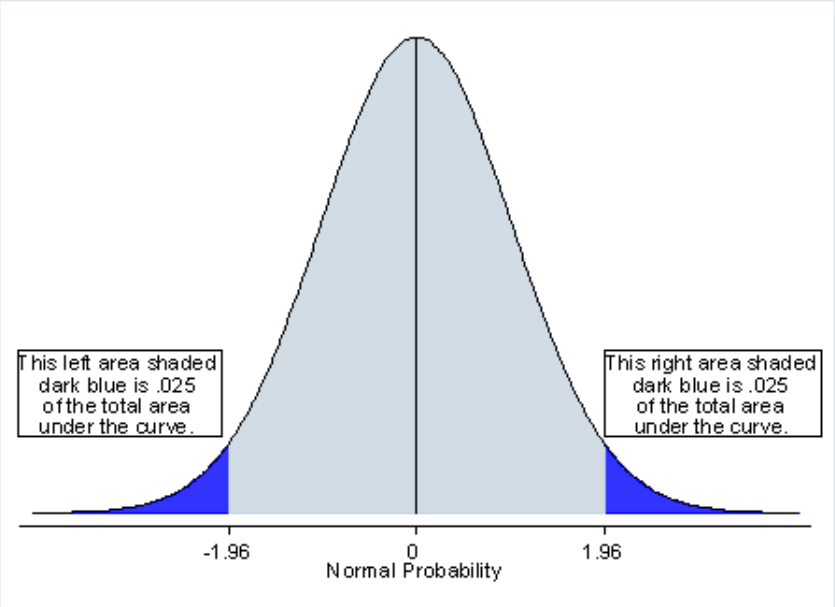

Figure 5.2: Two-Tailed Test
Note: Taken from [9]

In Fig 5.2 the area under the curve highlighted in red is 0.025 on both sides. If the mean is considerably more than x or significantly less than x, a two-tailed test will determine which is true. Suppose the test statistic falls inside the top 2.5 percent or bottom 2.5 percent of its probability distribution and has a p-value less than 0.05. In that case, the mean is deemed statistically different from x.

The formula of the p-value is as follows:

$$p = 2 - 2 * cdf_{t,d}(|t_{score}|) \tag{5.2}$$

where $cdf_{t,d}$ is the cumulative distributive function of the t-distribution with d degrees of freedom. The null hypothesis may be rejected if this probability is less than the standard threshold of 5% ($P \leq 0.05$). At this point, the correlation coefficient is deemed statistically significant.

ML Model	**Correlation**
RoBERTa Cross-Encoder	0.9041
Web-BERT	0.8456
SPECTER	0.6185

Table 5.1: Comparison of Pearson Correlation Coefficients for different ML Models

The Table 5.1 shows the corr elation coefficients of the different ML Models. Two more ML Models were used to calculate the similarities, all the p-values for 3 models were less than 0.05 hence the output can be considered. RoBERTa Cross-Encoder turned out to be the best model as compared to the rest.

Williams Test

We can see that difference in correlation for amongst the models is significant. However, the issue of whether an observed change in Pearson's r is statistically significant still remains. To determine if this discrepancy is reasonable or not, we will apply the Williams Test [39] to see whether the correlation between X_1 and X_3 is equal to the correlation between X_2 and X_3. In our case, X_3 is the similarity score present in the dataset and X_1 and X_2 are different scores obtained from the models. It is formulated [40] as:

$$t(n-3) = \frac{(r_{13} - r_{23})\sqrt{(n-1)(1+r_{12})}}{\sqrt{2K\frac{(n-1)}{n-3} + \frac{(r_{23}+r_{13})^2}{4}(1-r_{12})^3}} \tag{5.3}$$

where:

- r_{ij}: Pearson correlation between X_i and X_j
- n: population size
- $K = 1 - r_{12}^2 - r_{13}^2 - r_{23}^2 + 2r_{12}r_{13}r_{23}$

Since the Williams test considers the correlations between X1 and X2 (metric scores), it is more effective than the equivalent for independent samples. With everything else being equal, the test's statistical power increases with increasing metric score correlation.

X1	**X2**	**p-Value**
RoBERTa Cross-Encoder	SPECTER	< 0.01
Web-BERT	SPECTER	< 0.01
RoBERTa Cross-Encoder	Web-BERT	< 0.01

Table 5.2: p-Value after doing the Williams Test

Since all the p-values are less than 0.05, we can conclude that the correlation between different ML models are equivalent and justifiable.

5.1.2 Spearman's Rank

While Pearson's correlation evaluates linear connections, Spearman's correlation evaluates monotonic ones. The Spearman correlation between two variables equals the Pearson correlation between the rank values of those two variables (whether linear or not). Using the correlation coefficient, it is possible to summarize the strength and direction (positive or negative) of a link between two variables. The outcome will always fall in the range of 1 and -1. When observations rate the two variables similarly, the Spearman correlation between the two variables will be high; conversely, when observations rank the two variables differently, the Spearman correlation will be low. The formula is:

$$\rho = 1 - \frac{6 \sum d_i^2}{n(n^2 - 1)} \tag{5.4}$$

where:

- d_i is difference between the two ranks of each observation
- n is the number of observations

ML Model	**Correlation**
RoBERTa Cross-Encoder	0.9017
Web-BERT	0.8349
SPECTER	0.6125

Table 5.3: Comparison of Spearman Correlation Coefficients for different ML Models

The values of spearman coefficients and pearson coefficients have a very minimal difference. It concludes that the RoBERTa model trained with Cross-Encoder architecture is the best model amongst the rest.

Chapter 6

Conclusion

6.1 Conclusion

The Internet has made all the research papers or academic articles available at our fingertips, but which paper to refer to while doing research is what made easy by this dissertation. To facilitate the easy discovery of similar research papers related to a particular paper is what is achieved, hence making the extra work of searching redundant. The data used as the source came from the ACL Anthology. It was necessary to perform data preprocessing, during which the data were transformed from XML to JSON format. Python was used to develop the service, which takes in the data, applies the RoBERTa model, calculates the title and abstract similarities, and then combines the results by applying weights. Following the completion of the calculation of the similarity scores, the adjacency list is compiled. With the help of the program Gephi, the list can be represented in the shape of a graph. Gephi is a fairly straightforward program that is very easy to operate. Users are able to view network statistics and filter the information displayed on the graph. Users have the option of searching by either paper id, exact match, or partial match when searching.

Computing the similarity between the papers is the most time-consuming part as compared to the rest of the steps. The results, as shown in Section 4.4 are clean, and the graph can be visualized properly using different layout algorithms; also, search filters can be applied to have a more concise view.

6.2 Future Work

Currently, computing the similarities between papers consumes a lot of time. Another ML Model can be developed so that the calculation of the similarities can be made faster. In the future, there may be other more optimized graph visualization tools, which can hold up to a billion edges at once with much computing power.

A more aesthetic end-to-end application can be created instead of using a separate tool for visualization. The application will have inbuilt search functionalities with separate filters. It can also be loaded with ML Algorithms that can be applied to the graph, like clustering. Currently, it only shows papers similar to other papers; in the future, it can also be merged with the Citation Network [10] so that users can search more intuitively based on the citations, different authors, and many more The ability to search by subject would be a helpful improvement, and the resultant network would display the most significant publications for that topic and those expected to be significant. This search function might be used, for example, to decide which articles to read first while learning about a new area or to forecast its future.

Bibliography

[1] Y. Kashinskaya, E. Samosvat, and A. Artikov, "Spring-electrical models for link prediction," in *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, pp. 708–716, 2019.

[2] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[3] C. Vu, "Advance nlp model via transferring knowledge from cross-encoders to bi-encoders," Jan 2021.

[4] C. Olah, "Understanding lstm networks." `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`.

[5] J. Mueller and A. Thyagarajan, "Siamese recurrent architectures for learning sentence similarity," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, 2016.

[6] "Acl anthology," 2022. `https://aclanthology.org/`.

[7] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[8] S. Humeau, K. Shuster, M.-A. Lachaux, and J. Weston, "Poly-encoders: Transformer architectures and pre-training strategies for fast and accurate multi-sentence scoring," *arXiv preprint arXiv:1905.01969*, 2019.

[9] "What are the differences between one-tailed and two-tailed tests?." https://stats.oarc.ucla.edu/other/mult-pkg/faq/general/faq-what-are-the-differences-between-one-tailed-and-two-tailed-tests/.

[10] P. Gilsenan, "Bibliobuild: a citation network visualisation and exploration tool," 2021.

[11] A. Cohan, F. Dernoncourt, D. S. Kim, T. Bui, S. Kim, W. Chang, and N. Goharian, "A discourse-aware attention model for abstractive summarization of long documents," *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, 2018.

[12] C. University, "Arxiv dataset," Jul 2022.

[13] J. Abello, F. van Ham, and N. Krishnan, "Ask-graphview: A large scale graph visualization system," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 5, pp. 669–676, 2006.

[14] M. Bastian, S. Heymann, and M. Jacomy, "Gephi: an open source software for exploring and manipulating networks," in *Proceedings of the international AAAI conference on web and social media*, vol. 3, pp. 361–362, 2009.

[15] J. Tang, J. Liu, M. Zhang, and Q. Mei, "Visualizing large-scale and high-dimensional data," in *Proceedings of the 25th International Conference on World Wide Web*, International World Wide Web Conferences Steering Committee, apr 2016.

[16] T. C. Freeman, S. Horsewell, A. Patir, J. Harling-Lee, T. Regan, B. B. Shih, J. Prendergast, D. A. Hume, and T. Angus, "Graphia: A platform for the graph-based visualisation and analysis of complex data," *bioRxiv*, 2020.

[17] Y. Hu, "Algorithms for visualizing large networks," *Combinatorial Scientific Computing*, vol. 5, no. 3, pp. 180–186, 2011.

[18] P. Eades, "A heuristic for graph drawing," *Congressus numerantium*, vol. 42, pp. 149–160, 1984.

[19] T. M. Fruchterman and E. M. Reingold, "Graph drawing by force-directed placement," *Software: Practice and experience*, vol. 21, no. 11, pp. 1129–1164, 1991.

[20] E. R. Gansner, Y. Koren, and S. North, "Graph drawing by stress majorization," in *International Symposium on Graph Drawing*, pp. 239–250, Springer, 2004.

[21] W. S. Torgerson, "Multidimensional scaling: I. theory and method," *Psychometrika*, vol. 17, no. 4, pp. 401–419, 1952.

[22] C. Wang, S. Xu, L. Chen, and J. Li, "The application of graph algorithms: A reference mapping tool," in *2015 IEEE/ACIS 16th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, pp. 1–6, IEEE, 2015.

[23] M. Liu, B. Lang, Z. Gu, and A. Zeeshan, "Measuring similarity of academic articles with semantic profile and joint word embedding," *Tsinghua Science and Technology*, vol. 22, no. 6, pp. 619–632, 2017.

[24] M. Liu, B. Lang, and Z. Gu, "Calculating semantic similarity between academic articles using topic event and ontology," *arXiv preprint arXiv:1711.11508*, 2017.

[25] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[26] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*, Association for Computational Linguistics, 11 2019.

[27] "Sts-benchmark." `http://ixa2.si.ehu.eus/stswiki/index.php/STSbenchmark`.

[28] A. Cohan, S. Feldman, I. Beltagy, D. Downey, and D. S. Weld, "Specter: Document-level representation learning using citation-informed transformers," *arXiv preprint arXiv:2004.07180*, 2020.

[29] I. Beltagy, K. Lo, and A. Cohan, "Scibert: A pretrained language model for scientific text," *arXiv preprint arXiv:1903.10676*, 2019.

[30] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[31] M. S. Leo, "Strip net: Semantic similarity of scientific papers (s3p) network," Jan. 2022.

[32] M. Grootendorst, "Bertopic: Neural topic modeling with a class-based tf-idf procedure," *arXiv preprint arXiv:2203.05794*, 2022.

[33] A. Hagberg, P. Swart, and D. S Chult, "Exploring network structure, dynamics, and function using networkx," tech. rep., Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

[34] G. Perrone, J. Unpingco, and H.-m. Lu, "Network visualizations with pyvis and visjs," *arXiv preprint arXiv:2006.04951*, 2020.

[35] T. Wolf, V. Sanh, J. Chaumond, and C. Delangue, "Transfertransfo: A transfer learning approach for neural network based conversational agents," *arXiv preprint arXiv:1901.08149*, 2019.

[36] P.-E. Mazaré, S. Humeau, M. Raison, and A. Bordes, "Training millions of personalized dialogue agents," *arXiv preprint arXiv:1809.01984*, 2018.

[37] "Cross-encoder stsb-roberta-large." `https://huggingface.co/cross-encoder/stsb-roberta-large`.

[38] "Itertools." `https://docs.python.org/3/library/itertools.html`.

[39] E. J. Williams, *Regression analysis*, vol. 14. wiley, 1959.

[40] Y. Graham and T. Baldwin, "Testing for significance of increased correlation with human judgment," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 172–176, 2014.