Research Paper Recommendation System Using Natural Language Processing

Submitted in partial fulfilment of the requirements for the degree of

Bachelor of Technology

in

Computer Science & Engineering

by

Siddharth Sanjay Gandhi

19BCE0005

Under the guidance of

Dr. Akila Victor

School of Computer Science & Engineering

VIT Vellore

Dr. Felix Dietrich,

Department of Informatics

Technical University of Munich

VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

May 2023

DECLARATION

I hereby declare that the thesis entitled "**Research Paper Recommendation System with Natural Language Processing**" submitted by me, for the award of the degree of *Bachelor of Technology in Computer Science & Engineering* to VIT is a record of bonafide work carried out by me under the supervision of **Dr. Akila Victor** & **Dr. Felix Dietrich**.

I further declare that the work reported in this thesis has not been submitted and will not be submitted, either in part or in full, for the award of any other degree ordiploma in this institute or any other institute or university.

Place: Vellore

Date: 17th May 2023

Signed,

Siddharth Gandhi [*19BCE0005*]

CERTIFICATE

This is to certify that the thesis entitled "*Research Paper Recommendation System with Natural Language Processing*" submitted by **Siddharth Gandhi [19BCE0005]**, **School of Computer Science & Engineering**, **VIT**, for the award of the degree of *Bachelor of Technology in Computer Science & Engineering*, is a record of bonafide work carried out by him/her under my supervision during the period, 22.12.2022 to 19.05.2023, as per the VIT code of academic and research ethics.

The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma in this institute orany other institute or university. The thesis fulfills the requirements and regulations ofthe University and in my opinion, meets the necessary standards for submission.

Place: Vellore

Date: 17th May 2023

Signature of the Guide

Internal Examiner

External Examiner

Dr. Vairamuthu S

Head of Department

School of Computer Science & Engineering

Technische Universität München |
Boltzmannstraße 3 | D-85748 Garching b. München

Siddharth Gandhi
K-602 Indraprastha Towers
Near Drive-In Cinema, Memnagar
Ahmedabad, Gujarat, 380052
India

Phone: +919429210616
Email: ssgandhi602@gmail.com / siddharthsanjay.gandhi2019@vitstudent.ac.in

Garching, May 10, 2023

Confirmation of advisory position for Bachelor project: Siddharth Gandhi

To whom it may concern,

I am writing this letter as a formal confirmation that I advised Siddharth Gandhi (currently final year CSE student at VIT Vellore, Reg. No: 19BCE0005) on his Bachelor Thesis/Capstone Project, from January 2023 until May 2023. The title of the project is "Research Paper Reference Prediction with Natural Language Processing".

The objective of the project was to develop and evaluate a method to predict a reference list for a research paper, given its title and abstract. The work includes parsing a large volume of research papers in the field of machine learning, extracting their abstracts and reference lists, constructing a deep learning-based natural language encoding method, and construct a map from the encoded abstract to its (encoded) reference list.

Sincerely,

F. Dietrich

Dr. Felix Dietrich
Emmy Noether Research Group Leader
TUM Junior Fellow
Scientific Computing in Computer Science

Technical University of Munich
School of Computation, Information and Technology
Department of Informatics
Scientific Computing in Computer Science

Dr. Felix Dietrich
Boltzmannstr. 3
85748 Garching b. München

Tel. (089) 289 18 638
Fax (089) 289 18 607
felix.dietrich@tum.de
www.tum.de
www.fd-research.com

ACKNOWLEDGEMENTS

The project "*Research Paper Recommendation System with Natural Language Processing*" was made possible because of valuable inputs from everyone involved, directly or indirectly. I would like to thank both my advisors, **Dr. Akila Victor and Dr. Felix Dietrich**, for their invaluable guidance, support, and encouragement throughout the entire process of researching and writing this thesis. Their expertise and insightful comments have been instrumental in shaping my work and helping me to achieve my academic goals.

I would also like to thank **Vellore Institute of Technology**, for providing us with a flexible choice in the execution of the project and for providing me with an excellent academic environment and the necessary resources.

Finally, I am immensely grateful to my **parents** for their unwavering love, encouragement, and support throughout my academic journey. Their sacrifices and dedication have been the driving force behind my success, and I cannot thank them enough for everything they have done for me.

Place: Vellore
Date: 17th May 2023

Signed,
Siddharth Gandhi [*19BCE0005*]

ABSTRACT

The literature review is an essential part of the research process, as it helps researchers understand the current state of knowledge in their field and identify gaps that their research can address. However, the current review process with manual paper searching, can be time-consuming and labour-intensive. This is particularly true for researchers working in fields with large and rapidly-growing bodies of literature such as Medicine or Generative AI. To tackle this issue, we aim to build RefPred - a system that uses a citation-informed transformer (SPECTER) with a recommendation engine to recommend relevant papers to assist researchers in the review process. Specifically, given a new title/abstract, it should be able to predict the most relevant papers and sort them according to some metric (such as citation count or similarity score). For doing this, we create a dataset comprising thousands of research paper metadata, sourced from Semantic Scholar (S2), by crawling from the S2 API asynchronously and storing locally on a MongoDB database. We then use the citation-informed transformer model SPECTER to embed each paper, capturing its citation and semantic meaning simultaneously. Using this, we construct an embedding space of papers, which is used to build a recommendation engine based on KNN as a baseline to give relevant recommendations for a new paper. Finally, we propose a novel approach to use a feed-forward neural network to rerank the initial KNN candidates, resulting in 70% better Precision and Recall @ 20 scores on the test set over the baseline KNN approach.

Keywords: Recommendation System, Transformers, Reference Prediction, SPECTER, Knowledge Graph, Semantic Similarity, Research Paper Embeddings

TABLE OF CONTENTS

ACKNOWLEDGEMENTS iv

ABSTRACT v

TABLE OF CONTENTS vi

LIST OF FIGURES viii

LIST OF TABLES ix

LIST OF ABBREVIATIONS x

1 **INTRODUCTION** 1

1.1 Motivation 1

1.2 Aim 1

1.3 Objectives 1

1.4 Background 2

1.4.1 Recommender Systems 2

1.4.2 Data Sources 5

2 **LITERATURE REVIEW** 7

2.1 Survey of Existing Work 7

2.2 Gaps Identified 13

3 **METHODOLOGY** 14

3.1 Architecture Diagram 14

3.2 Data Acquisition 15

3.3 SPECTER - Generating Paper Embeddings 17

3.3.1 Data Pre-processing 18

3.3.2 Training Model from Scratch 19

3.4 Recommender Engine 21

3.4.1 Evaluation 21

3.4.2 KNN Baseline 22

3.4.3 Neural Network Based Reranking of KNN Candidates 24

3.5 System Analysis & Design 32

3.5.1 Introduction 32

3.5.2 Requirement Analysis 33

4 RESULTS & DISCUSSION **38**

4.1 Data Acquisition & Parsing data from S2: 38

4.2 SPECTER Retraining 38

4.3 Neural Network-based Reranking 41

4.3.1 Exploratory Data Analysis 41

4.3.2 Generating Pairs for the Reranking Model 43

4.4 Web App 48

5 **CONCLUSION & FUTURE WORK** **49**

6 **REFERENCES** **51**

LIST OF FIGURES

Figure 1: System Architecture 14
Figure 2: Async Crawler Running 16
Figure 3: Papers stored in MongoDB database 17
Figure 4: Sample of the triplets used to generate the pickle files (5 = direct reference, 1 = reference of a reference, 0 = random) 19
Figure 5: Retraining the model on Paperspace Gradient with RTX 5000 GPU 20
Figure 6: Ball Tree Partitions and the corresponding search tree (Dolatshah 2015) 22
Figure 7: Code for getting embedding for a new paper, finding KNN for a given embedding and getting KNN paper IDs for a new title and abstract 24
Figure 8: Pseudo code for **PaperPairDataset** 26
Figure 9: Code for the **PaperPairModel** model 27
Figure 10: Code for the recommendations function 30
Figure 11: Code for evaluating the reranking model 31
Figure 12: 2D tSNE Visualization of 768-dimensional embedding vectors of both retrained & HF model (size & colour based on number of citations) 40
Figure 13: EDA of the paper data 41
Figure 14: Histograms of Max Precision @ 20 and Max Recall @ 20 for train, val and test paper ID sets. 42
Figure 15: Train & Val Loss vs Epochs for Best Model 47
Figure 16: Final Reranked Model implemented into a Flask Web App 48

LIST OF TABLES

TABLE 1: LITERATURE REVIEW MAJOR THEMES 12
TABLE 2: TIME TAKEN TO CRAWL & STORE PAPER METADATA LOCALLY (FROM S2 API) 38
TABLE 3: TRAIN, VAL, TEST SPLIT 38
TABLE 4: NUMBER OF GENERATED TRIPLETS FOR SPECTER 39
TABLE 5: COMPARISON OF RETRAINING SPECTER VS ORIGINAL PAPER 39
TABLE 6: INFORMATION ABOUT PREPARING +VE AND -VE PAIRS 43
TABLE 7: MODEL LOSS W.R.T. VARIOUS NEGATIVE DISTRACTOR SELECTION & SKIPPING FUTURE PAPERS 44
TABLE 8: ABLATION STUDY (AFTER 30 EPOCHS) 45
TABLE 9: HYPERPARAMETER OPTIMIZATION (AFTER 30 EPOCHS) 46
TABLE 10: BEST MODEL VS KNN BASELINE 47

LIST OF ABBREVIATIONS

KNN	K-Nearest Neighbours
S2	Semantic Scholar
API	Application Programming Interface
Val	Validation (as in Val dataset)
TF-IDF	Term Frequency - Inverse Document Frequency
t-SNE	t-distributed Stochastic Neighbour Embedding
RNN	Recurrent Neural Networks
GNN	Graph Neural Networks
BERT	Bidirectional Encoder Representations from Transformers
GPT	Generative Pre-trained Transformer
Async	Asynchronous
AI2	Allen Institute of Artificial Intelligence
L2	Euclidean Distance
CPU	Central Processing Unit
GPU	Graphical Processing Unit
HF	Hugging Face
Hparams	Hyperparameters
TUM	Technical University of Munich
LSI	Latent Semantic Indexing
SVD	Singular Value Decomposition
POS	Part Of Speech
MAG	Microsoft Academic Graph
NSP	Next Sentence Prediction
SOTA	State Of The Art

1 INTRODUCTION

1.1 Motivation

The rapid growth of research in fields such as artificial intelligence or medicine has made it increasingly difficult for researchers to keep up with the vast number of publications. This can make it challenging for researchers to identify the most relevant literature for their work, particularly when they are just starting a new project and may not be familiar with the relevant literature on that particular topic.

Additionally, identifying relevant literature for a research project can be time-consuming and labour-intensive, requiring researchers to spend significant time searching through databases and manually reviewing potentially relevant papers. This can be a significant barrier to productivity and may hinder the ability of researchers to make significant contributions to their field.

1.2 Aim

To address these problems, we propose to develop RefPred - a system to automate the literature review process. It should take as input a topic or abstract for a research project and use this information to predict which references are most relevant to the project. By automating this process, we aim to significantly streamline the literature review process and enable researchers to more efficiently identify relevant references, even if they are not familiar with the entire literature. This should allow researchers to focus on more high-value tasks, such as properly analysing this literature and working on their methodology.

1.3 Objectives

The specific objectives of this thesis are as follows:

- To create a dense dataset of research paper metadata by asynchronously crawling Semantic Scholar and storing the data in a MongoDB database.
- To generate SPECTER embeddings for the titles and abstracts of research papers to better capture their semantic relationships.

- To develop two recommendation approaches: basic KNN and a reranking model that refines the KNN recommendations.
- To evaluate the performance of the proposed system using precision and recall @ 20 as the performance metrics.

1.4 Background

The sheer volume of published papers necessitates the use of efficient recommendation systems to assist researchers in identifying relevant articles and prioritizing their reading lists. Existing recommendation systems can be broadly categorized into the following types:

1.4.1 Recommender Systems

- **Content-based Filtering**

Content-based filtering methods analyze the textual content of research papers, such as titles, abstracts, or full texts, to identify similarities and generate recommendations. These methods rely on extracting features from the documents and calculating the similarity between them. Here, we discuss some of the common techniques employed in content-based filtering:

a) **Term Frequency-Inverse Document Frequency (TF-IDF):** TF-IDF is a widely used technique in information retrieval and text mining. It calculates the importance of a term within a document and across a corpus. The term frequency (TF) measures the frequency of a term in a document, while the inverse document frequency (IDF) measures the importance of a term across the entire corpus. The product of TF and IDF results in a weight that reflects the significance of a term within a document and across the corpus. By representing documents as vectors of TF-IDF weights, the similarity between documents can be computed using distance measures such as cosine similarity.

b) **Latent Semantic Indexing (LSI):** LSI, also known as Latent Semantic Analysis (LSA), is another widely used technique in content-based filtering. It addresses the limitation of TF-IDF by capturing the latent semantic relationships between terms and documents. LSI applies singular value decomposition (SVD) on the term-document matrix to reduce its dimensionality, resulting in a lower-dimensional representation that captures the

underlying semantic structure. The similarity between documents can then be calculated in this lower-dimensional space.

Despite the usefulness of these methods in providing relevant recommendations based on textual similarities, they often fail to capture the deeper semantic relationships between papers. This limitation can be addressed by employing more advanced methods, such as deep learning techniques, to better capture the semantic content of research papers.

- **Collaborative Filtering**

Collaborative filtering methods generate recommendations based on the preferences or behaviour of users who have similar interests. These approaches can be user-based, item-based, or a hybrid of both. Collaborative filtering techniques can be classified as memory-based or model-based:

a) **Memory-based Collaborative Filtering:** This approach calculates the similarity between users or items using historical data, such as user-item rating matrices. In user-based collaborative filtering, recommendations are generated based on the preferences of similar users. In item-based collaborative filtering, recommendations are made by identifying items similar to those that the target user has previously interacted with or rated. Common similarity measures used in memory-based collaborative filtering include the Pearson correlation coefficient, cosine similarity, and Jaccard similarity.

b) **Model-based Collaborative Filtering:** This approach employs machine learning algorithms to learn patterns from historical data and generate recommendations. Techniques such as matrix factorization, clustering, and Bayesian networks have been applied in model-based collaborative filtering. One popular matrix factorization technique is singular value decomposition (SVD), which decomposes the user-item rating matrix into lower-dimensional user and item latent factor matrices. These latent factors can then be used to predict user preferences and generate recommendations.

While collaborative filtering can provide personalized recommendations, it suffers from the cold-start problem, where the lack of sufficient user interaction data leads to poor recommendations for new users or items. Additionally, collaborative filtering methods may not fully capture the content-based similarities between items, which can be addressed by incorporating content-based techniques or deep learning methods.

- **Graph-based Methods**

Graph-based methods model the relationships between papers, authors, and other entities in a network structure. Techniques such as citation analysis, co-authorship networks, and graph neural networks have been applied to generate recommendations. Some common graph-based methods include:

a) **Citation Analysis:** This approach models the relationships between papers based on their citation patterns. By analysing citation networks, researchers can identify highly cited papers, influential authors, and emerging research trends. Citation-based recommendations can be generated by identifying papers that are highly cited by or closely related to the target paper. However, citation analysis may not fully capture the semantic content of papers, as it relies solely on citation patterns.

b) **Co-authorship Networks:** Co-authorship networks model the relationships between authors based on their collaboration patterns. By analysing these networks, researchers can identify influential authors, potential collaborators, and research communities. Recommendations can be generated by identifying papers authored by collaborators or members of the same research community as the target user. However, co-authorship networks may not capture the content-based similarities between papers or the semantic relationships between topics.

c) **Graph Neural Networks (GNNs):** GNNs are a class of deep learning methods that learn to capture the complex patterns and relationships in graph-structured data. By modelling the relationships between papers, authors, and other entities in a graph structure, GNNs can learn powerful representations that capture both structural and content-based information. GNNs have been applied to various tasks in the context of research paper recommendations, such as link prediction, node classification, and clustering. However, GNNs can be computationally expensive, especially for large-scale datasets, and may require significant computational resources to train and deploy.

Despite the various approaches taken in existing research paper recommendation systems, there is still room for improvement, particularly in capturing the semantic relationships between papers and refining recommendations. Our proposed system addresses these limitations by using SPECTER embeddings to represent the semantic content of research papers more effectively and

introducing a reranking model to enhance the initial KNN recommendations. This combination of techniques promises to provide better quality recommendations, enabling researchers to identify relevant literature more efficiently.

1.4.2 Data Sources

In addition to the development of effective recommender systems, another crucial aspect of research paper recommendation is the collection of comprehensive and high-quality datasets. Several resources and datasets are available for collecting research paper metadata, including arXiv, Semantic Scholar, and other academic databases. In this section, we provide an overview of some popular resources for collecting research paper data:

- **arXiv**

arXiv[1] is a preprint repository maintained by Cornell University, which provides open access to over a million research papers in various disciplines, including physics, mathematics, computer science, and quantitative biology. The arXiv dataset on Kaggle[2] contains periodically updated metadata for more than 1.7 million research papers, including titles, abstracts, authors, categories, and citation information. This rich dataset can be used for various tasks in recommender systems, such as content-based filtering, citation analysis, and clustering.

- **Semantic Scholar (S2)**

Semantic Scholar [3] is a free, AI-powered research tool developed by the Allen Institute for AI. It indexes millions of research papers across various disciplines and provides features such as search, citation analysis, and author profiles. The Semantic Scholar Open Research Corpus[4] contains metadata for over 180 million research papers, including titles, abstracts, authors, venues, and citation information. This large-scale dataset can be used for various tasks in recommender systems, such as content-based filtering, collaborative filtering, and graph-based methods.

---

1 https://arxiv.org/

2 https://www.kaggle.com/Cornell-University/arxiv

3 https://www.semanticscholar.org/

4 https://allenai.org/data/s2orc

- **Microsoft Academic Graph**

Microsoft Academic Graph (MAG)[5] is a large-scale, heterogeneous graph that contains information about academic papers, authors, institutions, journals, conferences, and fields of study. MAG includes metadata for over 200 million research papers, as well as citation and co-authorship information. This comprehensive dataset can be used for various tasks in recommender systems, such as content-based or collaborative filtering, and graph-based methods.

- **PubMed**

PubMed[6] is a free search engine maintained by the US National Library of Medicine that provides access to more than 30 million citations and abstracts from life science journals and online books. PubMed offers a comprehensive and up-to-date resource for collecting research paper metadata in the biomedical domain. The dataset can be used for various tasks in recommender systems, such as content-based filtering, collaborative filtering, and graph-based methods.

- **Web of Science**

Web of Science[7] is a subscription-based research database that provides access to more than 1.7 billion cited references and covers over 33,000 journals across various disciplines. Web of Science offers features such as search, citation analysis, and journal impact factors. This dataset can be used for various tasks in recommender systems, such as content-based filtering, collaborative filtering, and graph-based methods.

These datasets offer a wealth of research paper metadata that can be used to develop and evaluate recommender systems. Data collection methods, such as web scraping and APIs, can be employed to gather and store the required metadata from these resources. The choice of resource and data collection method depends on factors such as the domain, scale, and specific requirements of the recommender system being developed.

---

5 https://www.microsoft.com/en-us/research/project/microsoft-academic-graph/

6 https://pubmed.ncbi.nlm.nih.gov/

7 https://clarivate.com/webofsciencegroup/solutions/web-of-science/

2 LITERATURE REVIEW

2.1 Survey of Existing Work

(Nigram 2021), a prior work at TUM, presented the groundwork for the reference prediction task. The goal of the project was to make a word embedding space for thousands of open-access PDFs of scientific papers. It involved finding the TF-IDF feature vector for various paper texts, clustering them together, and visualizing the clusters with the t-SNE dimensionality reduction algorithm. Finally, new words can be embedded into the embedding space by using the nearest clustering algorithm (like K-Means). However, since TF-IDF only relies on the number of occurrences of various keywords in a text, it fails to capture the semantics and meaning of a given paper. Thus, better techniques like BERT for document embedding can be applied for improving document representation.

(Vaswani 2017), a seminal paper, presented a new model for machine translation called the Transformer. This model uses self-attention mechanisms instead of the traditional recurrence or convolutions, to weigh the importance of different parts of the input when making predictions. The model architecture consists of an encoder and a decoder. The encoder is made up of multiple layers of self-attention and feed-forward neural networks. The decoder is similar to the encoder but also includes an attention mechanism that allows it to look at the encoder’s output. It is able to process input in parallel, rather than sequentially as in RNNs, which greatly improves its efficiency and allows for faster training times. The attention function can be described as mapping a query and a set of key-value pairs to an output. The final score involves scaling the dot product of the attention mechanism by the square root of the dimension of the input, which helps to prevent the gradients from becoming too large during training. It also uses a technique called multi-head attention, which allows it to attend to different parts of the input simultaneously. This architecture achieves state-of-the-art performance on multiple machine translation benchmarks (WMT 2014 English-to-German and WMT 2014 English-to-French translation tasks) after training for as little as twelve hours on eight P100 GPUs. The transformer has since been used in many other natural language processing tasks.

(Devlin 2018) introduced BERT, which is designed to pre-train deep bidirectional representations from the unlabelled text by joint conditioning on both left and right context in all layers. BERT consisted of stacked transformer encoders and the paper proposes different layer numbers for different variations. It uses WordPiece embeddings with a 30,000 token vocabulary and is trained on 2 unsupervised tasks - Masked Language Modelling (predict the missing word(s) in a sentence, given the context of the remaining words) and Next Sentence Prediction (NSP - given a pair of sentences, predict whether the

second sentence is the next sentence in the text or not). The authors showed that BERT significantly outperforms previous state-of-the-art models on a wide range of natural language understanding benchmarks, including the GLUE and SQuAD datasets, and also used for other tasks such as question answering, textual entailment and so on. Using unmasked inputs to make the model bi-directional (instead of the autoregressive models of GPT) made BERT very effective for capturing context for sentence-level tasks. The authors also show that BERT can be fine-tuned for specific tasks using a smaller dataset and that fine-tuning the model on a task-specific dataset improves its performance even further.

(Beltagy 2019) - Previous Large Language Models (LLMs) like GPT or BERT were trained unsupervised on a large corpus of crowd-sourced data (such as Wikipedia), which significantly improved performance for many NLP tasks. However, a major gap was the scientific literature for which annotated data was difficult and expensive to collect. This paper aims at solving that exact problem, by training BERT on a large corpus of scientific text. They use SciVocab vocabulary, a derivative of WordPiece vocabulary specifically for scientific corpus and a random corpus of 1.14 million full-text papers from Semantic Scholar. They first fine-tune the BERT model using the standard fine-tuning procedure, where the model is trained on a task-specific dataset using the pre-trained weights as initialization. After fine-tuning, they evaluate the performance of the SciBERT model on several scientific text understanding tasks: named entity recognition (NER - identify and classify named entities such as genes and chemicals), part-of-speech tagging (POS), and citation intent classification (classify the intent of a citation in a scientific paper; e.g. whether it is used to provide background information or to support a claim made in the paper). The results of the evaluation show that the SciBERT model outperforms the BERT model on all of the scientific text understanding tasks and also on several other tasks such as text classification, question answering, and semantic similarity. The authors also did an ablation study showing that fine-tuning on scientific text allows it to perform better on scientific text understanding tasks than fine-tuning it on general text.

(Jeong 2020) proposes a citation recommendation system that utilizes BERT and Graph Convolutional Network (GCN) to improve the performance of citation recommendation. The model takes into account the context of the input paper when making recommendations for additional citations. The datasets used are a combination of ACL's Anthropology Network (AAN) and FullTextPeerRead (derived from Kaggle), both of which have well-organized bibliographic information. First, the authors pre-process the input data by constructing a citation graph from the input dataset. The graph is constructed by connecting papers that cite each other. Then, they use BERT to encode the entire input paper and the papers in the citation graph, which captures the semantic information of the papers. Next, the authors use GCN to learn the representations of the papers in the citation graph, which captures the structural information of the papers.

The GCN is trained to propagate the representations of the papers along the edges of the citation graph, which allows the model to take into account the context of the input paper when making recommendations. The authors then combine the representations learned by BERT and GCN to make citation recommendations. They evaluate the model on a dataset of academic papers and their associated citations using rank-aware metrics like MRR (Mean Reciprocal Rank), MAP (Mean Average Precision) and Recall @ K. The model outperforms existing citation recommendation models in terms of accuracy and diversity of recommendations.

(Bhagavatula 2018) proposes a global content-based citation recommendation system that takes an entire research paper as input and gives recommendations (instead of local recommendations which only take a few sentences to recommend). Since the number of related works to an entire paper can be large, the authors propose a 2-step method: first is a fast recall-oriented candidate selection phase and second is a feature-rich precision-oriented ranking phase. Broadly, the first stage filters out all unrelated articles with low recall scores and the second stage find the nearest neighbours in the document embedding space to generate rankings for a given query paper. They evaluated their models with MRR and F1@20 on DBLP, PubMed and OpenCorpus datasets where they achieved SOTA results in the first two, even without the use of metadata (like authors, venue, or journal).

(Cohan 2020) introduces SPECTER, a system to generate document-level embedding of scientific documents based on pre-training a Transformer on a powerful signal of document-level relatedness: the citation graph. The paper argues that previous embedding models for scientific corpora (like SciBERT) are trained for intra-document purposes (like understanding the contents of individual papers) and not on inter-document metrics (like citation dependencies). It uses the title and abstract (which encapsulate the semantic meaning of a paper) of around 178K papers from the Semantic Scholar Corpus and trains it based on a custom loss function of citation-based pretraining objective. They also introduce the SciDocs evaluation framework for various tasks related to scientific literature such as paper topic classification, citation prediction, and reference recommendation. SPECTER was substantially better than other models like SciBERT, ELMo, and SentBERT when evaluated on SciDocs.

(Lo 2019) introduces S2ORC, which is a corpus of 81.1M English scientific papers from a range of academic disciplines from medicine to philosophy. It consists of rich metadata (title, abstract, authors, venue, journal) for all papers along with resolved bibliographic references. It also contains 8.1M full-text (parsed with ScienceParse and GROBID) open-access papers for research involving the entire contents of papers. It has significantly more and better-organized data than the previous datasets of PubMed and AAN. To evaluate the metadata quality, the authors pretrained a BERT model on S2ORC and compared it against

other SOTA datasets in various domains for a variety of different tasks (like dependency parsing and text classification). The S2ORC-SciBERT model was comparable to all of the datasets proving a rich and accurate variety of metadata for a multitude of domains. The pipeline used to construct the dataset CORD-19 (literature specific to COVID-19) and the Semantic Scholar Academic Graph API.

(Singh 2022) introduces SciRepEval, the first comprehensive benchmark for training and evaluating scientific document representations. It includes 25 challenging and realistic tasks, 11 of which are new, across four formats: classification, regression, ranking and search. They investigate whether existing document representation methods can generalize to a highly diverse set of tasks (SOTA models struggle to generalize), whether training on multiple tasks can improve document representation (surprisingly it doesn't), and if task-format-specific representations can improve generalization (yes!). They conclude that learning separate document representations (Burges 2005) for each of the four tasks is substantially better than trying to learn a single representation for all tasks (generalization).

(Burges 2005) Recognizing the shortcomings of traditional approaches to document ranking (modelling the score of each document independently) the authors proposed RankNet, a model that calculates the target probabilities between any two documents for a given query, serving as a single training record. The model was architected as a neural network function, where the output for each document, defined as $\boldsymbol{oi} = \boldsymbol{f}(\boldsymbol{xi})$ and $\boldsymbol{oj} = \boldsymbol{f}(\boldsymbol{xj})$, was passed through a logistic function to transform it to a probability range of [0,1]. A unique aspect of RankNet is the way it updates its weights. Unlike typical neural networks, RankNet processes each pair of documents as one training record, passing both through the same weights of the network to calculate $\boldsymbol{oi}$ & $\boldsymbol{oj}$, which are then used to compute the gradient and update the weights. This approach is a departure from the typical feedforward neural network process, and it uses a cross-entropy cost function to calculate the cost $\boldsymbol{Cij}$ for a pair of documents $\boldsymbol{di}$ and $\boldsymbol{dj}$. The paper provided the mathematical foundation for this novel approach to document ranking, which became a key stepping stone in the development of more advanced ranking algorithms.

Note: The next 3 papers were suggested by this very project in the end after training. We simply fed in our title and abstract and reviewed the results to find some more relevant papers.

(Ebesu 2017) introduces "Neural Citation Network (NCN)", a novel citation recommendation system that uses a flexible encoder-decoder architecture. The encoder in the NCN leverages a max time delay neural network (TDNN) to robustly represent citation context, while the decoder, a recurrent neural network (RNN), determines the best paper to recommend based on this representation and the paper's title. The NCN also includes an attention mechanism and author networks to further refine its recommendations. The system was evaluated on the large-scale CiteSeer dataset, where it demonstrated significant

improvements over existing methods. The experiments were conducted on the RefSeer dataset. The paper also notes that NCN outperforms all baselines on every metric by 13-16%. Specifically, it performed better than the Citation Translation Model (CTM), TDNN-to-RNN, and RNN-to-RNN models.

(Narechania 2021) presents VITALITY, a system aimed at promoting the serendipitous discovery of relevant academic literature using transformer language models and visual analytics. The system enables users to find semantically similar papers in a document-level embedding space given a list of input papers or a working abstract. VITALITY visualizes this embedding space as an interactive 2-D scatterplot using dimension reduction techniques. Additionally, it summarizes metadata such as keywords and co-authors and allows users to save and export papers for use in a literature review. The authors contribute data from 38 popular data visualization publication venues, along with open-source scrapers for the research community to expand the list of supported venues. VITALITY is evaluated through qualitative findings, which suggest that it can be a promising complementary technique for conducting academic literature reviews. The initial prototype focuses on the data visualization field, but the open-source system and scraper framework enable expansion to other venues and academic communities. VITALITY has the potential to enhance existing literature review practices by addressing the challenge of identifying relevant literature that may use different terminology, thus bridging the gap in academic literature searches and aiding in the exploration of new topics.

(Portenoy 2022) introduces Bridger, a system designed to facilitate the discovery of novel and valuable scholars and their work, aiming to counteract the information "filter bubbles" that arise from isolated silos of scientific research and information overload. Bridger constructs a faceted representation of authors based on information from their papers and inferred author personas, which enables the identification of commonalities and contrasts between scientists, thus balancing relevance and novelty. The system includes "slices" of a user's papers, allowing them to find authors who match the user only on a subset of their papers and on certain facets within those papers. In studies with computer science researchers, the facet-based approach helps users discover authors whose work is considered more interesting and novel compared to a relevance-focused baseline representing state-of-the-art retrieval of scientific papers. The authors demonstrate that Bridger connects authors from more distant communities in terms of publication venues, citation links, and co-authorship social ties.

Table 1: Literature Review Major Themes

Paper	Major Theme
(Nigram 2021)	This paper presents the groundwork for the reference prediction task by creating a word embedding space for scientific papers using TF-IDF and t-SNE. It discusses the limitations of TF-IDF and the potential for BERT to improve document representation.
(Vaswani 2017)	Introduces the Transformer model for machine translation, which uses self-attention mechanisms and parallel processing. It achieves state-of-the-art performance on multiple translation benchmarks and has since been used in many other NLP tasks.
(Devlin 2018)	Presents BERT, a model that pre-trains deep bidirectional representations from unlabelled text. It outperforms previous models on various NLP benchmarks and can be fine-tuned for specific tasks using smaller datasets.
(Beltagy 2019)	Proposes SciBERT, a BERT model trained on a large corpus of scientific text. It outperforms the original BERT model on scientific text understanding tasks and several other NLP tasks.
(Jeong 2020)	Proposes a citation recommendation system using BERT and Graph Convolutional Network (GCN) that captures both semantic and structural information of academic papers. The model outperforms existing citation recommendation models in terms of accuracy and diversity of recommendations.
(Bhagavatula 2018)	Introduces a global content-based citation recommendation system with a two-step method involving candidate selection and precision-oriented ranking. The model is evaluated on DBLP, PubMed, and OpenCorpus datasets and achieves state-of-the-art results in the first two datasets, even without the use of metadata.
(Cohan 2020)	Presents SPECTER, a system generating document-level embeddings of scientific documents by pre-training a Transformer on citation graphs. The paper introduces the SciDocs evaluation framework for tasks related to scientific literature, with SPECTER outperforming other models like SciBERT, ELMo, and SentBERT when evaluated on SciDocs.
(Lo 2019)	Introduces the S2ORC corpus, containing 81.1M English scientific papers with rich metadata. It provides more and better-organized data than previous datasets and is used to pretrain a BERT model for evaluation.
(Singh 2022)	Presents SciRepEval, a comprehensive benchmark for training and evaluating scientific document representations. It investigates the generalization of existing methods and the benefits of task-format-specific representations.
(Burges 2005)	Introduces RankNet, a novel approach to document ranking using a neural network function. It calculates target probabilities between documents for a given query and serves as a stepping stone in the development of advanced ranking algorithms.
(Ebesu 2017)	Introduces Neural Citation Network (NCN), a citation recommendation system using an encoder-decoder architecture with attention mechanisms & author networks. Results in improvements over existing methods on CiteSeer dataset.
(Narechania 2021)	Presents VITALITY, a system for serendipitous discovery of relevant academic literature using transformer models & visual analytics. Enables users to find semantically similar papers & visualize the document-level embedding space.
(Portenoy 2022)	Introduces Bridger, a system for discovering novel scholars and their work by constructing a faceted representation of authors. It helps users identify commonalities and contrasts between scientists, balancing relevance and novelty.

2.2 Gaps Identified

1. **Limitations of TF-IDF:** The prior work by (Nigram 2021) relied on TF-IDF for creating word embeddings, which fails to capture the semantics and meaning of a given paper. Our work utilizes SPECTER, a citation-informed transformer model, to create more effective embeddings.
2. **Improved document representation:** BERT and SciBERT are powerful models for document representation, but they have limitations when it comes to inter-document metrics like citation dependencies. Our work with SPECTER addresses this gap by focusing on both citation and semantic meanings of papers (also tries to address the generalization issue by (Singh 2022)).
3. **Efficient recommendation engine:** Many existing citation recommendation systems struggle with recall and precision scores. By constructing an embedding space of papers and utilizing KNN as a baseline, our work improves the performance of the recommendation engine.
4. **Neural network reranking:** To further enhance the performance of the recommendation system, we also propose a novel approach of using a neural network to rerank the initial KNN candidates, resulting in a significant improvement in Precision and Recall @ 20 scores compared to the baseline KNN approach with SPECTER embeddings.
5. **Balancing relevance and novelty:** Addressing the balance between relevance and novelty, as introduced by (Portenoy 2022), our work proposes a novel approach of using a neural network to rerank the initial KNN candidates. This approach is expected to improve the precision and recall scores on the test set, ensuring the recommendation system provides a mix of relevant and novel research papers.

3 METHODOLOGY

3.1 Architecture Diagram

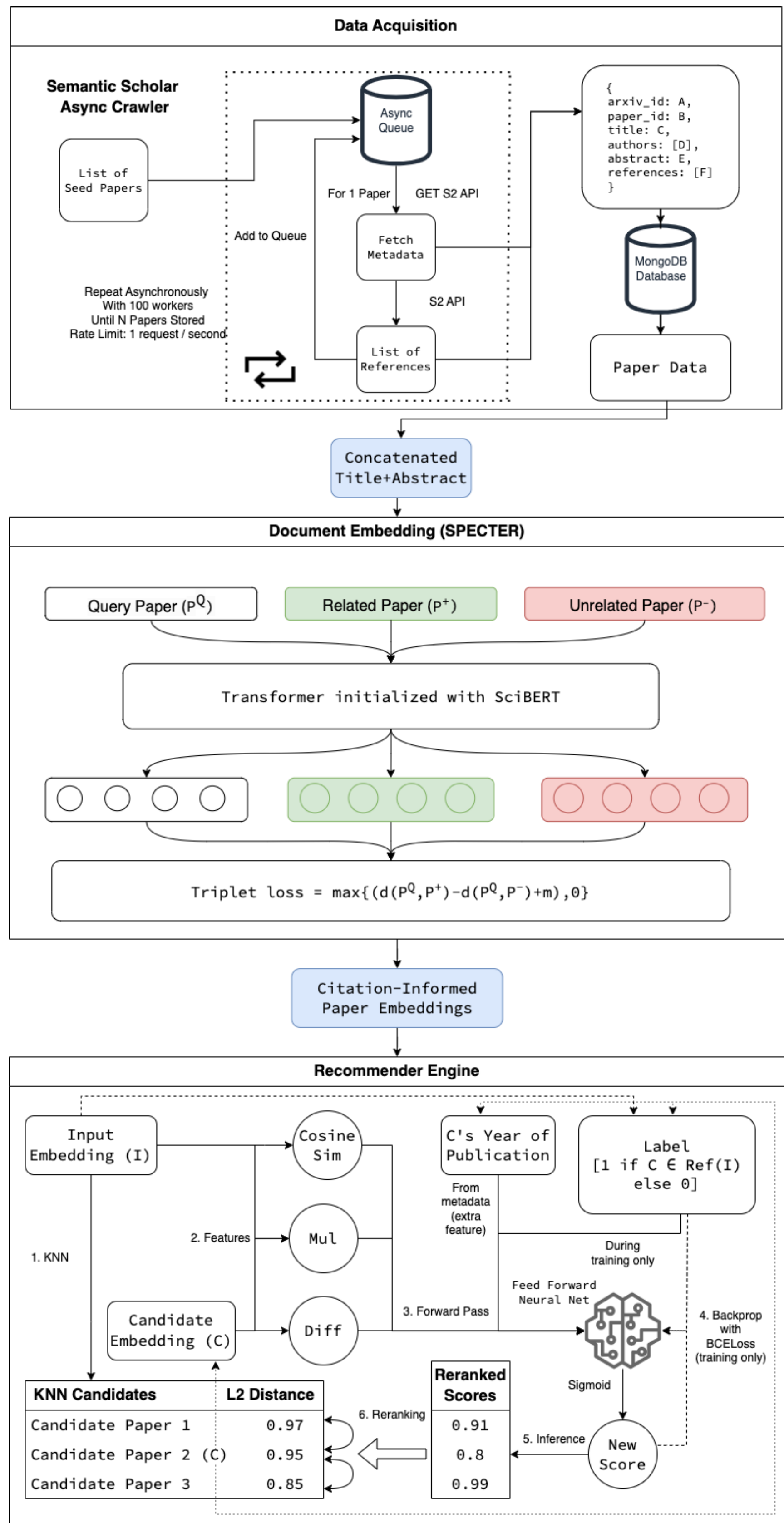


Figure 1: System Architecture

Overview of the system:

As mentioned in Figure 1, the system broadly consists of 3 stages:

a) Acquiring the data for thousands of research papers and storing them in a database.

b) Embedding each paper's data into an embedding in a way which captures both its semantic meaning and citation graphs to create an embedding space.

c) Finally, the recommendation engine which will use the embedding space to find relevant recommendations for a new title/abstract.

All three sections are elaborated below.

3.2 Data Acquisition

The first step in building the automated research paper recommendation system is to acquire a dataset of research papers that will be used to train the embedding model and recommendation engine. The dataset should contain metadata about the papers such as title, abstract, year of publication, citation count, and references. There are datasets available for this (such as Semantic Scholar's S2ORC dataset), however, they are too large (~500 GBs) which would require a lot of compute, and sampling randomly from it might not ensure a dense enough citation graph to learn from.

Hence, we have decided individually form a dataset using the Semantic Scholar API by starting with some initial seed papers, some of which are highly referenced papers (like the influential 'Attention is all you need' paper). We then built an async crawler on the API to speed up the data acquisition process as it allows for non-blocking I/O, which means that the system can continue processing requests while waiting for responses from the server. Async is preferred because multiprocessing will have too much overhead in creating and switching between processes while multithreading is not efficient because of Python's Global Interpreter Lock (GIL), meaning only one thread can effectively operate at a given time.

The data acquisition process works as follows:

a) Start with initial seed papers and add them to a queue.

b) Deque papers one by one and for each paper fetch the references using the S2 API.

c) Construct the JSON object for the current paper with metadata and references and store the JSON object to a MongoDB database.

d) Add the references to the queue.

e) Repeat b) to d) until *N* papers are stored asynchronously with 100 workers.

We also rate-limited the system to 100 requests/seconds to ensure that the system can continue to crawl papers over a longer period without running into problems. Finally, we also implemented a retry handling mechanism (up to 3 times per paper) to handle cases when the response is corrupted during transmission.

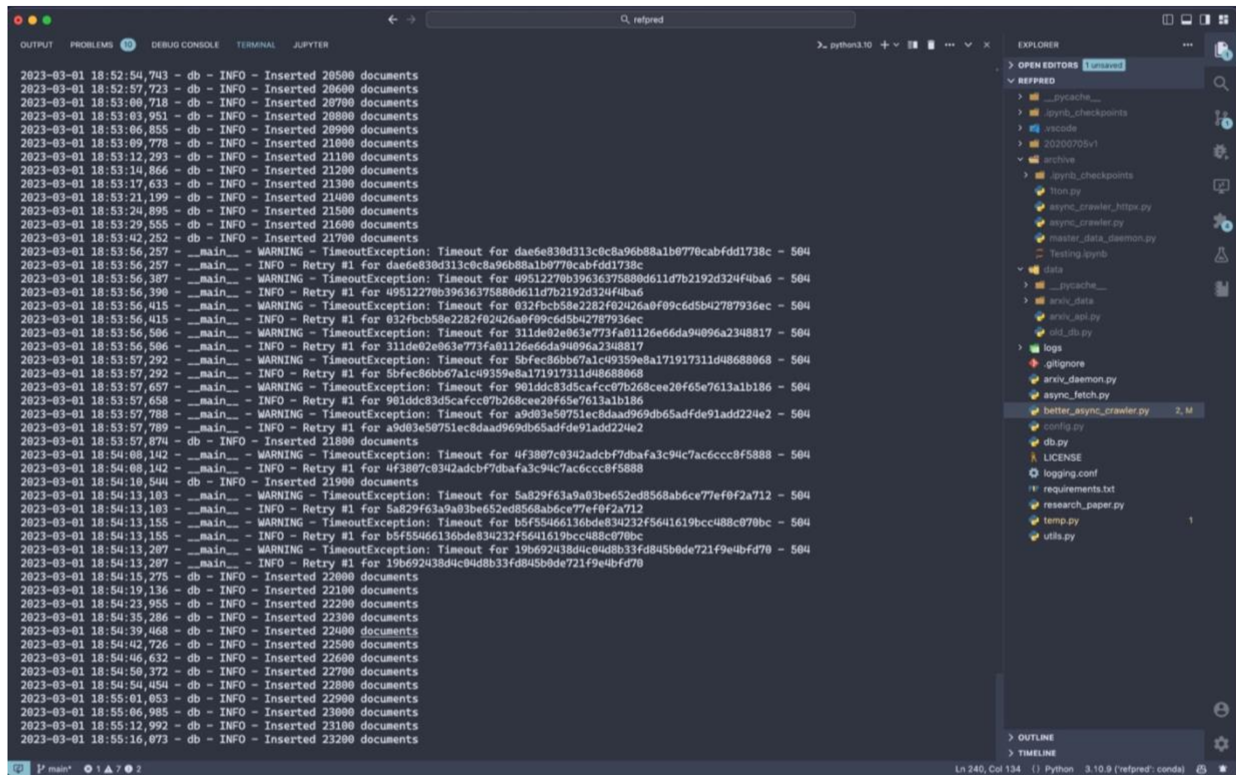


Figure 2: Async Crawler Running

We have successfully stored the data for 10,000 and 50,000 papers in the database which took around 12 minutes and 1 hour simultaneously. Going forward, we will be using the 10,000 papers for our experiments as it is faster to iterate upon.

This is how the database looked after crawling and storing 10K papers:

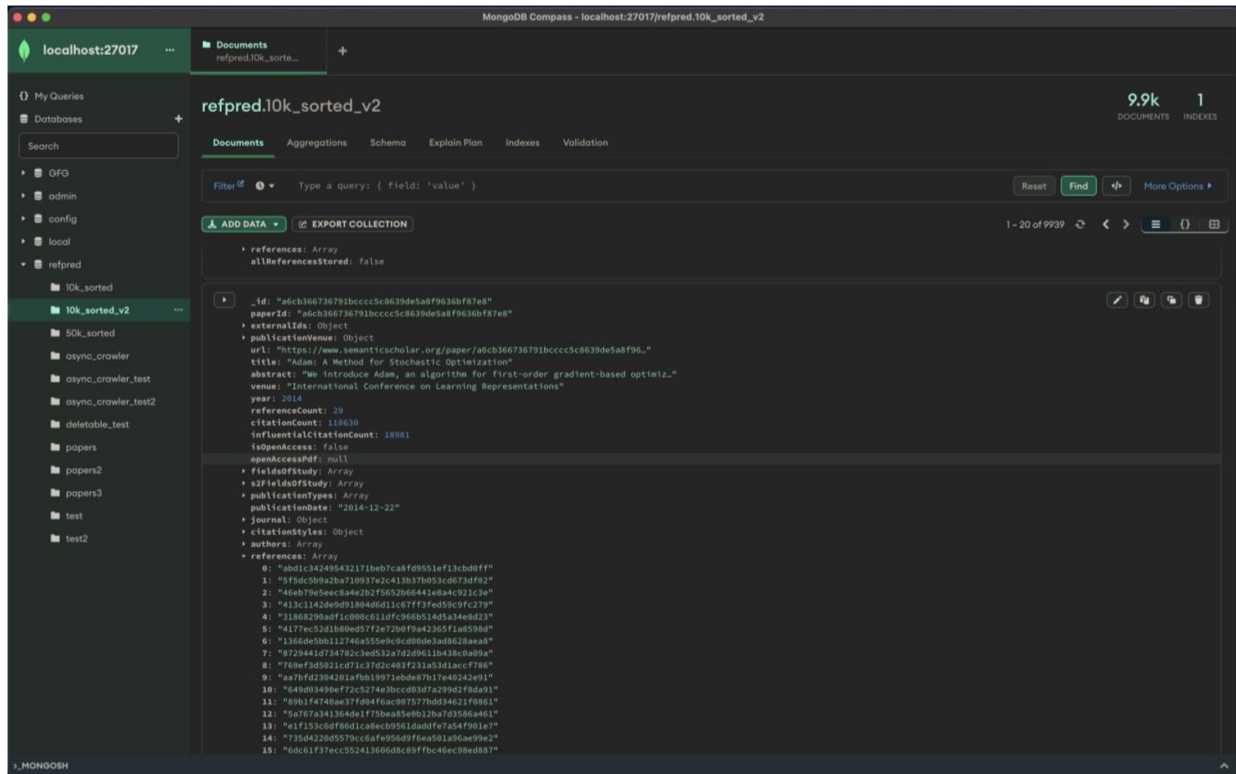


Figure 3: Papers stored in MongoDB database

3.3 SPECTER - Generating Paper Embeddings

Once the metadata is collected and pre-processed, the next step is to create a meaningful representation of each paper's content. This is achieved by embedding the title and abstract of each paper using the SPECTER model from AI2. SPECTER is a citation-informed transformer model that leverages both semantic and citation information to create rich, contextual embeddings.

The SPECTER model is initialized from SciBERT, a pre-trained language model specifically designed for scientific text. To fine-tune the embeddings, a custom loss function is developed based on triplets of papers which is as follows:

$$L = max(d(P^Q, P^+) - d(P^Q, P^-) + m), 0$$

Each triplet consists of a query paper (P^Q), a positive paper (P^+), and a negative paper (P^-). 'm' is the loss margin hyperparameter (we will follow the original paper and choose $m = 1$). For the distance function 'd', we will use L2 norm distance. By training the model on these triplets, SPECTER will create embeddings that capture both the semantic content and citation relationships between papers. Each embedding vector has 768 dimensions, which allows for a more accurate representation of each paper in the embedding space.

3.3.1 Data Pre-processing

Since the SPECTER model (used to generate document embeddings) requires triplets of papers in order to encode the citation information into the semantic encodings, we have to first generate these triplets. In each triplet, there is one query paper, a positive & negative paper. A positive paper is a direct reference of the query paper (strong link) and a negative paper can either be a random paper (no link) or a reference of a reference (weak link). We can assign scores to each of these 3 states and generate Q number of triplets per query paper. In our case, there are around 8000 training papers with 1000 for validation and testing (with the standard 80-10-10 train-val-test split). With $Q = 5$ for our case, we are left with around 29,000 training triplets and around 3600 validation and testing triplets, after removing papers for whom metadata was not parsed in the initial 10K dataset.

So for each triplet, we get the necessary fields (concatenated title + author) for all 3 papers and make it into an AllenNLP Instance. These instances are then stored in a pickle file to save the pre-processed results.

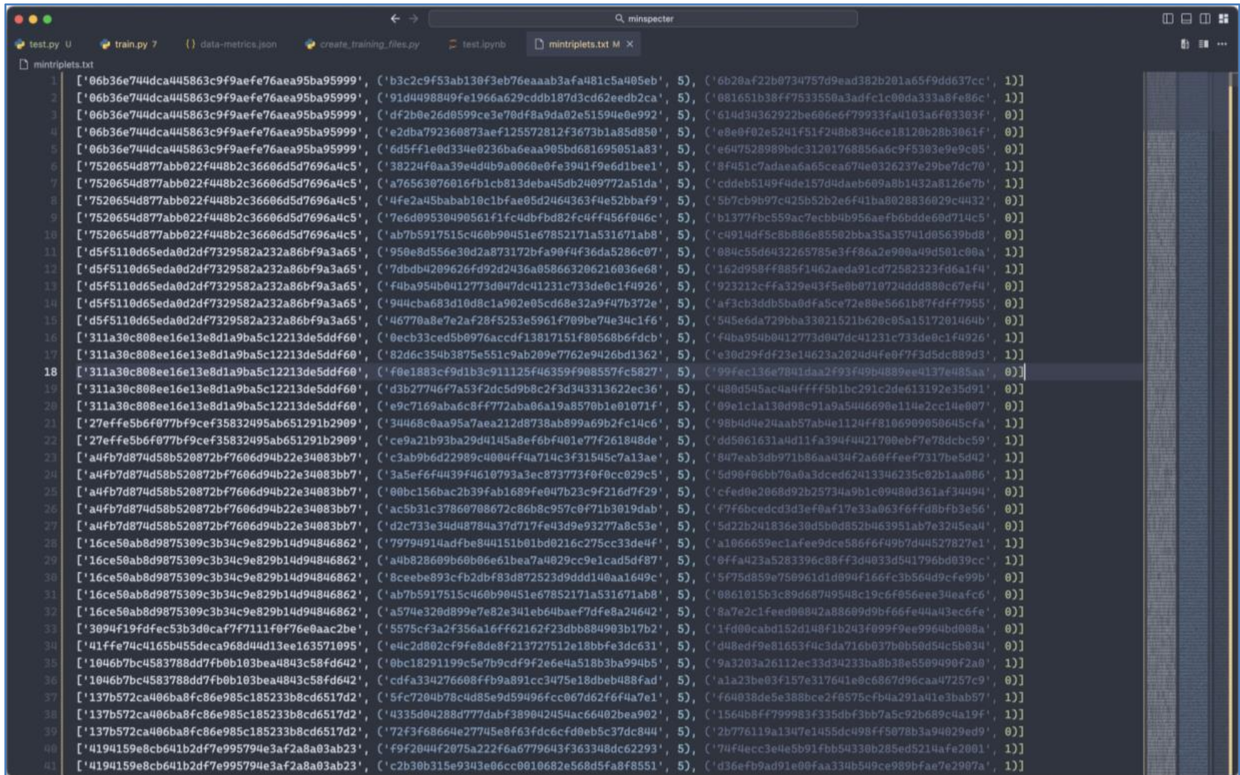


Figure 4: Sample of the triplets used to generate the pickle files (5 = direct reference, 1 = reference of a reference, 0 = random)

3.3.2 Training Model from Scratch

1. Define classes to read dataset from Pickled file (in our case this was class `DataReaderFromPickled` and `IterableDataSetMultiWorkerTestStep`).
2. Define a custom loss function class to implement the triplet loss function above.
3. Define the `Specter` model itself by inheriting from `pl.LightningModule` which helps to streamline the model training process by taking care of boilerplate code.
4. Implement methods like `_get_loader()` to get train-val-test dataloaders.
5. Configure the optimizers and LR-schedulers.
6. Define the initial `model` and `tokenizer` from the pretrained SciBERT model.
7. Implement the training (and val) steps by passing the training data triplets to the model by implementing the `training_step` method and calculating the triplet loss.

8. After implementing some other important functions (**`validation_step`, `test_step`**), **`pytorch lightning`** will handle the rest of the training process (like backward pass).

9. For testing, instead of calculating the loss on the embeddings **`e`**, just return **`e`**.

The original SPECTER model was trained with 630K triplets (compared to our 28K) and was trained for 2 epochs with a batch size of 4 with each epoch lasting around 1.5 days. We tried to retrain the model from scratch with 10K papers and it took around 2 hours on an NVIDIA RTX 5000 (16 GB GPU on Paperspace Gradient Pro) for a single epoch with batch size 4. There is also a pretrained SPECTER model available on HuggingFace which is much better because of being trained for longer with much more compute power. Hence, going forward, we plan to use the embeddings from the pre-trained model for the recommendation engine.

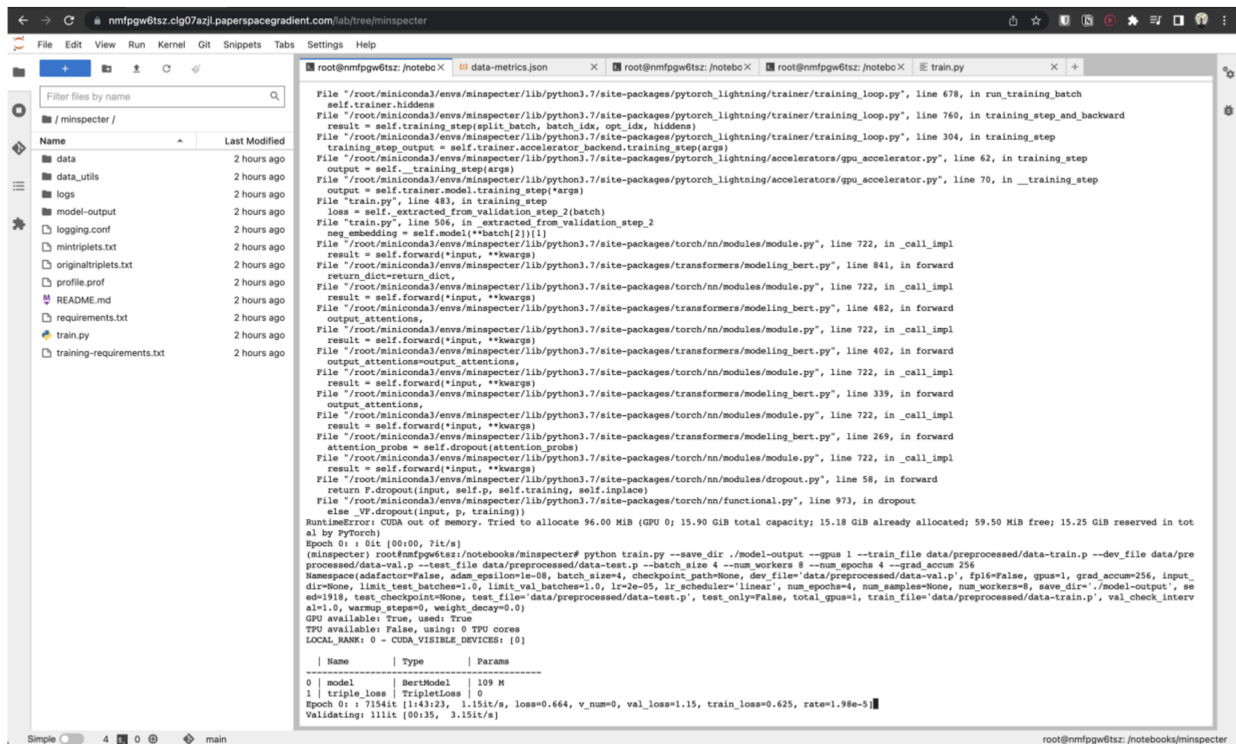


Figure 5: Retraining the model on Paperspace Gradient with RTX 5000 GPU

With the trained model we can get embeddings for the entire dataset where each paper is represented as a 768-dimensional vector which should capture its semantic and citation contents.

3.4 Recommender Engine

The final step in building Refpred is to create a recommendation engine that can take in a new paper's abstract or title and return the most relevant recommendations. The goal of this engine is simple – given a new title and abstract (or an existing paper), predict what papers are most likely (or were) its references.

3.4.1 Evaluation

We start with what evaluation metrics we will use to judge how good the recommendations from the recommendation engine are. We will use Precision @ 20 and Recall @ 20 as our primary evaluation metrics. These metrics are chosen over other commonly used metrics like Mean Average Precision (MAP) or nDCG (normalized Discounted Cumulative Gain) for the following reasons:

1. **No inherent order:** In the recommendation system for academic papers, there is no inherent order in the list of references. As long as the most relevant references are at the top, the order of the rest of the references is not crucial. This is unlike some other recommendation systems where the order of the items is significant, such as search engines or product recommendations.

2. **Focus on relevance:** Precision @ 20 and Recall @ 20 focus on identifying the most relevant papers within the top 20 recommendations. Precision @ 20 measures the proportion of the top 20 recommendations that are relevant, while Recall @ 20 measures the proportion of relevant papers that are included in the top 20 recommendations. These metrics highlight the effectiveness of the recommendation system in surfacing relevant papers to the user.

3. **Simplicity:** Precision and Recall are easy to understand and interpret. They provide a clear, intuitive way of assessing the quality of the recommendations. This simplicity can be beneficial when communicating the results to a broader audience, as well as for the developers themselves when interpreting the performance of the recommendation system.

3.4.2 KNN Baseline

The baseline approach is to embed the new `title+abstract` using SPECTER and use K-Nearest Neighbours (KNN) algorithm to find the most similar papers in the embedding space. By calculating the L2 distance between the new paper's embedding and the embeddings of existing papers, the KNN algorithm can return a set of research papers that are semantically and contextually related. However, since the embedding space is high dimensional (768), manually finding KNN neighbours becomes very inefficient. To solve this, we use Ball Trees.

- **Ball Trees**

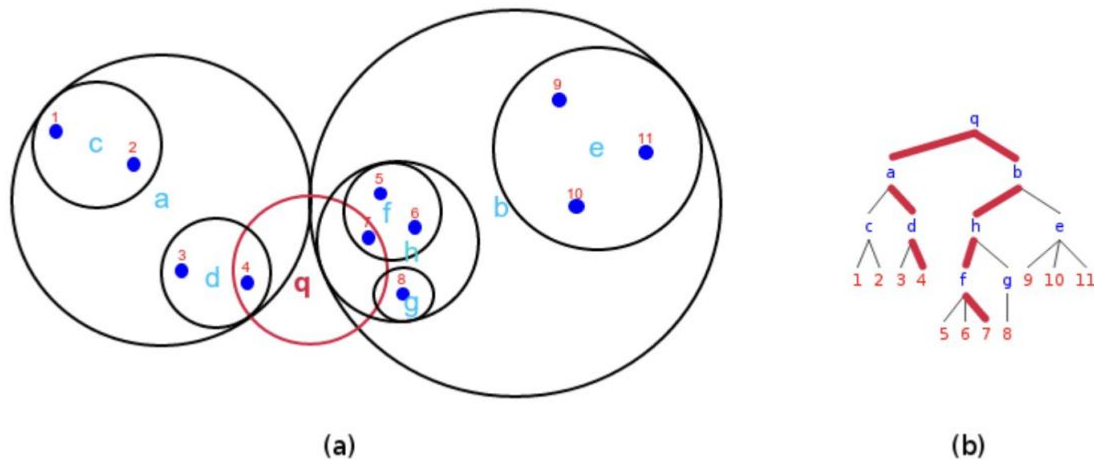


Figure 6: Ball Tree Partitions and the corresponding search tree (Dolatshah 2015)

Ball Trees are a tree-based data structure designed for efficiently solving nearest neighbour search problems in high-dimensional spaces. They recursively partition the input data into non-overlapping hyperspherical regions called "balls," represented by a centroid and radius. The tree is built hierarchically, with each internal node partitioning its parent node's data points into two disjoint subsets.

The construction of a Ball Tree involves selecting a pivot and partition radius to divide the data points into subsets. This process is recursively applied until a stopping criterion is met.

Searching for nearest neighbours in a Ball Tree involves traversing the tree from root to leaf nodes while pruning branches that do not contribute to the final result. This pruning strategy

allows Ball Trees to perform nearest neighbour queries much faster than a brute-force approach, especially in high-dimensional spaces.

In our k-nearest neighbours (KNN) baseline, we use a Ball Tree data structure to efficiently store and search for nearest neighbours in the high-dimensional SPECTER embedding space, providing an efficient and scalable solution for the recommendation task.

- **The Problem of Limited Search Space**

During the development of our KNN-based recommendation system, we encountered a limitation related to the size of our initial dataset of 10K research papers. The restricted dataset size limited the maximum precision and recall scores achievable by our model, as many of the references for the papers were simply not present in the dataset. This issue had the potential to negatively impact the effectiveness of our recommendation system.

To address this problem, we expanded the dataset used for the KNN search by including an additional 100K references from the initial 10K papers. This allowed us to increase the coverage of potential recommendations and improve the chances of finding relevant references for the papers in our dataset. It is important to note that we did not use the entire 100K references for training the neural network reranker, as this would have led to a similar, but larger-scale issue.

By incorporating the additional 100K references into the KNN search (following Section 3.3.1), we were able to mitigate the limitations imposed by the initial dataset size, thus improving the performance of our recommendation system. However, we maintained the training scope of the neural network reranker to the initial 10K papers, ensuring that the model was not affected by the same limitations when evaluating its effectiveness.

- **Getting KNN Recommendations**

In the following code, we describe the procedure to get KNN paper IDs for a given title and abstract. We use a ball tree as our KNN tree formed by the 100,000 reference embeddings parsed in Section 3.2. We simply embed the new paper and find its nearest neighbours by querying the KNN tree.

```python
@torch.no_grad()
def get_embedding(paper, embedding_map=None):
    '''
    Given an input paper (dict with at least 'title' as a key, returns a 768 dimensional embeddings using the SPECTER model from HF.
    '''
    assert type(paper)==dict and 'title' in paper.keys(), "paper must be a dict with at least 'title' as a key"
    if embedding_map and paper['paper_id'] is not None and paper['paper_id'] in embedding_map:
        return embedding_map[paper['paper_id']].view(1,-1)

    title_abs = [d['title'] + specter_tokenizer.sep_token + (d.get('abstract') or '') for d in [paper]]
    inputs = specter_tokenizer(title_abs, padding=True, truncation=True, return_tensors="pt", max_length=512)
    result = specter_model(**inputs)
    cur_embedding = result.last_hidden_state[:, 0, :]
    return cur_embedding

def find_knn(cur_embedding, knn_tree, knn_k):
    '''
    Given a SPECTER embedding, shaped [1,768], a KNN Tree and knn_k (number of nearest neighbours), returns the top indices from the knn_tree
    '''
    scores, top_indices = knn_tree.query(cur_embedding.reshape(1,-1), k=knn_k, return_distance=True)
    scores, top_indices = np.squeeze(scores), np.squeeze(top_indices)
    return top_indices, scores


def get_knn_new_paper(title, abstract, knn_tree, all_paper_ids, knn_k, return_scores=False):
    '''
    Given a new title and abstract, get the knn_k nearest neighbours from the knn_tree. The order of knn_tree and all_paper_ids must be the
      same.
    '''
    new_paper_obj = {'title':title, 'abstract': abstract}
    new_paper_embedding = get_embedding(new_paper_obj)
    top_indices, scores = find_knn(new_paper_embedding, knn_tree, knn_k=knn_k)
    recommended_paper_ids = [all_paper_ids[i] for i in top_indices]
    if return_scores:
        return recommended_paper_ids, scores
    return recommended_paper_ids
```

Figure 7: Code for getting embedding for a new paper, finding KNN for a given embedding and getting KNN paper IDs for a new title and abstract

While the KNN algorithm is a good starting point for the recommendation engine, other approaches can be explored to optimize the predictions. Currently, we look at a neural network approach to learn the features that are most important for making recommendations.

3.4.3 Neural Network Based Reranking of KNN Candidates

As we will see in Section 0, while the recommendation for KNNs seems good, when tested with our evaluation metrics described in Section 3.4.1, it does not score well. Hence to improve the

precision and recall scores, we train a reranking model which takes KNN candidates as inputs and reranks them based on past reference patterns.

The main idea is to prepare pairs for `paper_ids` from existing data (i.e. papers in the dataset and their references) and assigns a label of 1 for papers that cite each other and 0 for those that don't. The model should then learn from these prepared pairs so that when we find the KNN candidates for a new embedding and pass them to the model, we should get better recommendations, which are based on past patterns of citations.

- **Dataset**

This first and most important step of the reranking approach is the data we feed into the reranker, as it determines the preferences the model learns with the labels. First, we define a class called `PaperPairDataset`, which is a custom dataset class that inherits from `PyTorch's Dataset`. It's used to create pairs of academic papers for a reranking model. These pairs will be passed to the model described in the next section to learn from the past patterns of papers referencing each other.

The main steps it follows are:

1. Computes K-nearest neighbours (KNN) for each paper in the dataset, using the KNN tree.
2. Retrieves actual references for each paper & stores the maximum references present (`max_refs_present`) & the total number of references (`total_refs`) for logging purposes.
3. Iterates over the recommendations *randomly* (not in order of KNN) and creates pairs such that the label is 1 if `rec_id` is a reference of `qid`, else 0.

Note: It is very important to iterate over the recommendations randomly as otherwise we penalize the top KNN neighbours every time by giving them 0 labels, which results in 0 Precision and Recall scores. (As tested in Section 0)

4. Checks whether the recommended paper was published after the query paper.
5. The `getitem` method returns embeddings, a flag indicating if the recommended paper was published after the query paper (`is_after`), and the label.

```
class PaperPairDataset(Dataset):
    def __init__(paper_ids, embedding_map, reference_map, metadata, all_paper_ids, knn_k, knn_tree, check_year, data_file):
        '''
        Initializes the dataset with paper IDs, embeddings, reference maps, metadata, KNN tree, and other required data. If a data file is
        provided and exists, it loads the saved data.
        '''
        Initialize dataset variables and load data from file if available, else prepare_data_knn()

    def prepare_data_knn(knn_k):
        '''
        Prepares the dataset by computing KNN recommendations, and generating positive and negative pairs of paper IDs. It also keeps track of
        the number of positive and negative pairs, and calculates the positive-to-negative ratio.
        '''
        Initialize data, total_pos, total_neg, max_refs_present, total_refs, and zero_ref_papers
        For each paper_id in paper_ids:
            Get original_year from metadata
            Compute KNN recommendations for the paper
            Get actual_references from reference_map
            Calculate num_positives and num_negatives
            Update max_refs_present and total_refs
            Check if the paper has zero references, increment zero_ref_papers
            For each ref_id in recommendations:
                Check if ref_id is a reference (positive example), append (paper_id, ref_id, 1) to data
                Else if ref_id is after original_year and not a reference (negative example), append (paper_id, ref_id, 0) to data
        Calculate pos_neg_ratio and log the results

    def is_published_after(year1, year2):
        '''
        Returns 1 if year1 is less than or equal to year2, indicating that the paper with year2 is published after the paper with year1.
        '''
        Return 1 if year1 ≤ year2, else return 0

    def __len__():
        '''
        Returns the total number of paper pairs in the dataset.
        '''
        Return the length of data

    def __getitem__(idx):
        '''
        Given an index, returns the embeddings of the two paper IDs, the label, and a flag indicating if the second paper was published after
        the first one.
        '''
        Get paper_id1, paper_id2, and score from data at idx
        Get embeddings and years for both paper IDs
        Compute is_after flag using is_published_after()
        Return embedding1, embedding2, score (as a tensor), and is_after (as a tensor)
```

Figure 8: Pseudo code for **`PaperPairDataset`**

- **Model**

Next, the main neural network architecture driving our recommendation engine, which is designed to process pairs of research papers and predict their new similarity scores. The architecture is implemented as a PyTorch module called **`PaperPairModel`**, which inherits from the **`nn.Module`** class. It takes as input 2 paper embeddings along with a flag (**`is_after`**) and determines a similarity score between 0 and 1. We also pass cosine similarity between the 2 embeddings along with **`x_diff`** and **`x_mul`** as additional features to help the model learn representations between similar embeddings (labelled 1) faster.

```python
class PaperPairModel(nn.Module):
    def __init__(self, embedding_dim=768, hidden_dims=None, dropout_prob=0.3, weight_decay=0.01, use_bn=True, bn_momentum=0.9):
        '''
        • embedding_dim: The dimension of the input embeddings for each paper, defaulting to 768.
        • hidden_dims: A list of integers specifying the dimensions of the hidden layers in the feedforward neural network, defaulting to [2048,
          1024].
        • dropout_prob: The dropout probability used in dropout layers to prevent overfitting, defaulting to 0.3.
        • weight_decay: The weight decay parameter used for regularization, defaulting to 0.01.
        • use_bn: A boolean flag indicating whether to use batch normalization, defaulting to True.
        • bn_momentum: The momentum parameter for batch normalization, defaulting to 0.9.

        '''
        super(PaperPairModel, self).__init__()
        input_dim = embedding_dim * 2 + 2
        layers = [nn.Linear(input_dim, hidden_dims[0])]
        if use_bn:
            layers.append(nn.BatchNorm1d(hidden_dims[0], momentum=bn_momentum))
        layers.extend([nn.Tanh(), nn.Dropout(dropout_prob)])

        for i in range(1, len(hidden_dims)):
            layers.append(nn.Linear(hidden_dims[i - 1], hidden_dims[i]))
            if use_bn:
                layers.append(nn.BatchNorm1d(hidden_dims[i], momentum=bn_momentum))
            layers.extend([nn.Tanh(), nn.Dropout(dropout_prob)])

        layers.extend([nn.Linear(hidden_dims[-1], 1)])#, nn.Sigmoid()])
        self.fc_layers = nn.Sequential(*layers)

    def forward(self, x1, x2, is_after):
        '''
        x1: embedding #1 (query)
        x2: embedding #2 (a recommendation of query)
        is_after: a flag indicating whether x2 was published after x1 (1 if true else 0)
        '''
        cosine_sim = F.cosine_similarity(x1, x2, dim=1, eps=1e-8).unsqueeze(1)
        x_diff = torch.abs(x1 - x2)
        x_mul = x1 * x2
        x = torch.cat((x_diff, x_mul, is_after.unsqueeze(1), cosine_sim), dim=1)
        x = self.fc_layers(x)
        return x.squeeze()
```

Figure 9: Code for the **`PaperPairModel`** model

A sample model architecture can be seen below:

```
PaperPairModel(
  (fc_layers): Sequential(
    (0): Linear(in_features=1538, out_features=1024, bias=True)
    (1): BatchNorm1d(1024, eps=1e-05, momentum=0.9, affine=True, track_running_stats=True)
    (2): Tanh()
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=1024, out_features=2048, bias=True)
    (5): BatchNorm1d(2048, eps=1e-05, momentum=0.9, affine=True, track_running_stats=True)
    (6): Tanh()
    (7): Dropout(p=0.5, inplace=False)
    (8): Linear(in_features=2048, out_features=1, bias=True)
  )
)
```

The architecture is composed of a series of fully connected layers, batch normalization layers (if enabled), activation functions, and dropout layers. The input to the model is formed by concatenating the element-wise absolute difference, element-wise product, temporal information

(whether one paper is published after the other), and cosine similarity between the embeddings of the two input papers.

The `forward` method takes three input tensors, `x1`, `x2`, and `is_after`, representing the embeddings of the first paper, the embeddings of the second paper, and the temporal information, respectively. The cosine similarity between `x1` and `x2` is computed using the `F.cosine_similarity` function. The element-wise absolute difference (`x_diff`) and element-wise product (`x_mul`) between `x1 and x2` are also computed as extra features to learn the similarity effectively.

The input tensor `x` is formed by concatenating `x_diff`, `x_mul`, `is_after`, and the cosine similarity along dimension 1. The concatenated input tensor is then passed through the fully connected layers, batch normalization layers (if enabled), activation functions (`Tanh`), and dropout layers in the `self.fc_layers` sequential model.

The output of the model is a single scalar value obtained by applying a sigmoid activation function to the final layer, which represents the predicted similarity score between the input pair of research papers. The output tensor is then squeezed to remove any singleton dimensions.

- **Training**

Before training we currently will have the following things with us:

1. `embedding_map` - maps paper id to the 768-dimensional embedding
2. `reference_map` - maps one paper id to a list of paper ids which are the actual references of the original paper id
3. `metadata` - provides metadata such as title, abstract, and year of each `paper_id`
4. `all_paper_ids` - a list of all `paper_ids`
5. `knn_tree` – KNN Ball Tree formed from the embeddings of `all_paper_ids`. The order of the `knn_tree` is the same as `all_paper_ids` (important to fetch `paper_ids` as the nearest neighbours).

(1) was computed from the SPECTER embeddings, (2,3,4) were computed from the 100K paper dataset and (4) was computed by feeding (1) in `sklearn’s` Ball Tree.

Now, the training loop is as follows:

1. Create **train_dataset** and **val_dataset** instances of **PaperPairDataset** with the respective **paper_ids**, **embedding_map**, **reference_map**, **metadata**, and **all_pids**.

2. Create **train_dataloader** and **val_dataloader** instances of **DataLoader** with the respective datasets, **batch_size**, and shuffle settings.

3. Instantiate the **PaperPairModel** with **hidden_dims**, **dropout_prob**, **use_bn**, and **bn_momentum** from hyperparameters.

4. Move the model to the appropriate device (CPU or GPU).

5. Set up the loss function (**BCEWithLogitsLoss**) and optimizer (**AdamW**).

6. Initialize the learning rate scheduler with early stopping (**ReduceLROnPlateau**).

7. Train the model using the following steps: a. Loop through the epochs. b. Set the model to training mode. c. Iterate over the **train_dataloader**, getting batches of data (**embeddings**, **labels**, and **is_after** flags). d. Move the data to the appropriate device (CPU or GPU). e. Zero the gradients of the optimizer. f. Get the model's output by passing the input data through the model. g. Calculate the training loss using the criterion. h. Perform backpropagation and update the model's parameters with the optimizer. Accumulate the training loss.

8. Evaluate the model on the validation set using the following steps: a. Set the model to evaluation mode. b. Iterate over the **val_dataloader**, getting batches of data (**embeddings**, **labels**, and **is_after** flags). c. Move the data to the appropriate device (CPU or GPU). d. Get the model's output by passing the input data through the model. e. Calculate the validation loss using the criterion. f. Accumulate the validation loss.

9. Evaluate the model's recommendations on a specific test paper (AIAYN) and calculate the precision, recall, and F1 score.

10. Update the learning rate scheduler based on the validation loss.

11. Check for early stopping criteria and save the best model's state.

This loop covers the training and evaluation of the **`PaperPairModel`** on the given dataset. It includes data preparation, model instantiation, training, validation, and early stopping to prevent overfitting.

- **Getting Reranked Recommendations**

To get new recommendations for a given title and abstract, we have defined a function called **`get_recommendations`**.

```python
def get_recommendations(model, paper_id, knn_tree, embedding_map, metadata, all_pids, top_k, knn_k):
    '''
    model: The trained PaperPairModel,
    title: The title of the new paper,
    abstract: The abstract of the new paper,
    knn_tree: The k-NN tree created from the paper embeddings,
    embedding_map: The dictionary mapping paper IDs to their embeddings,
    metadata: The dictionary containing metadata about each paper,
    all_pids: The list of all paper IDs,
    top_k: The number of top recommendations to return (IN THE ORDER OF all_pids),
    knn_k: The number of nearest neighbours to consider in the k-NN tree,
    return_scores (optional): Whether to return the scores along with the recommendations
    '''
    model.eval()
    paper_obj = {'title': title, 'abstract': abstract, 'year': 2023}

    paper_embedding = get_embedding(paper_obj)
    top_indices, scores = find_knn(paper_embedding, knn_tree, knn_k=knn_k)
    recommended_paper_ids = [all_pids[i] for i in top_indices]

    other_paper_embeddings = torch.stack([embedding_map[pid] for pid in recommended_paper_ids])

    # Get years from metadata and create is_after tensor
    paper_year = paper_obj['year']
    other_paper_years = [metadata[pid]['year'] for pid in recommended_paper_ids]
    is_after = torch.tensor([int(paper_year < other_year) for other_year in other_paper_years], dtype=torch.float32).to(DEVICE)

    with torch.no_grad():
        paper_embedding = paper_embedding.expand_as(other_paper_embeddings).to(DEVICE)
        other_paper_embeddings = other_paper_embeddings.to(DEVICE)
        # Pass the is_after tensor to the model
        scores = model(paper_embedding, other_paper_embeddings, is_after)
    model.train()
    top_k = torch.topk(scores, k=top_k)
    top_k_indices = top_k.indices
    top_k_scores = top_k.values
    reranked_pids = [recommended_paper_ids[idx] for idx in top_k_indices]
    if return_scores:
        return reranked_pids, top_k_scores
    return reranked_pids
```

Figure 10: Code for the recommendations function

Inside the function, we first set the model to evaluation mode. Then, we create a paper object with the given title, abstract, and a fixed year (2023 in this case). We use the **`get_embedding`** function to generate the SPECTER embedding for this new paper object using the HF model.

Next, we find the k nearest neighbours in the embedding space using the `find_knn` function. We then retrieve the paper IDs of the recommended papers using the indices returned by `find_knn`. We create a tensor of the embeddings for these recommended papers.

We also create an `is_after` tensor that indicates if the new paper was published after each of the recommended papers. This tensor is created based on the years of publication of the recommended papers and the new paper's year (2023).

With the model in evaluation mode, we expand the new paper's embedding to match the shape of the recommended papers' embeddings tensor. We move all tensors to the appropriate device (CPU or GPU) and pass them through the model, along with the `is_after` tensor.

After obtaining the scores for each recommended paper, we return the model to training mode. We then select the top-k recommendations based on their scores using the `torch.topk` function. We extract the indices and values (scores) of the top-k recommendations and return the reranked paper IDs. If the `return_scores` parameter is set to True, we also return the top-k scores alongside the recommendations.

- **Evaluating Reranked Results from Model**

```python
def evaluate_model(model, paper_ids, knn_tree, embedding_map, reference_map, metadata, all_pids, top_k, knn_k):
    '''
    model: The trained PaperPairModel,
    paper_ids: the list of Paper IDs for the model to be evaluated on,
    knn_tree: The k-NN tree created from the paper embeddings,
    embedding_map: The dictionary mapping paper IDs to their embeddings,
    reference_map: The dictionary mapping paper IDs to the list of IDs of their actual references,
    metadata: The dictionary containing metadata about each paper,
    all_pids: The list of all paper IDs,
    top_k: The number of top recommendations to return (IN THE ORDER OF all_pids),
    knn_k: The number of nearest neighbours to consider in the k-NN tree,
    '''
    precisions = []
    recalls = []
    for paper_id in tqdm(paper_ids):

        recommendations = set(get_recommendations(model, paper_id, knn_tree, embedding_map, metadata, all_pids, top_k=top_k, knn_k=knn_k))
        true_references = set(reference_map.get(paper_id, []))
        if not true_references:
            continue

        intersect = recommendations.intersection(true_references)
        precision = len(intersect) / len(recommendations)
        recall = len(intersect) / len(true_references)

        precisions.append(precision)
        recalls.append(recall)

    mean_precision = sum(precisions) / len(precisions)
    mean_recall = sum(recalls) / len(recalls)
    return mean_precision, mean_recall
```

Figure 11: Code for evaluating the reranking model

The `evaluate_model` function is designed to assess the performance of a trained `PaperPairModel` on a set of paper IDs. This function computes the mean precision and mean recall for these papers, which are used as evaluation metrics to measure the effectiveness of the recommendation system.

To do this, the function iterates through each paper ID in the provided list of `paper_ids`. For each paper, the function generates a set of recommendations using the `get_recommendations` function, which returns the `top_k` most relevant papers based on the model, paper embeddings, and metadata. Then, the function retrieves the set of true references from the `reference_map` using the current paper ID. If there are no true references for the paper, the loop continues to the next paper ID.

For each paper, the function calculates precision and recall by comparing the recommendations and true references. Precision is calculated as the ratio of the number of intersecting papers (i.e., the papers present in both recommendations and true references) to the total number of recommendations. Recall is calculated as the ratio of the number of intersecting papers to the total number of true references. The function accumulates the precision and recall values for each paper in separate lists. Finally, the function computes the mean precision and mean recall by dividing the sums of the respective lists by their lengths and returns these values as the evaluation metrics.

3.5 System Analysis & Design

The proposed system, RefPred, aims to provide a comprehensive and efficient solution for researchers to identify relevant literature for their review process. To achieve this, the system leverages a citation-informed transformer model (SPECTER) and a recommendation engine. The design is focused on optimizing the performance and user experience, with the goal of helping researchers save time and effort during the literature review process.

3.5.1 Introduction

RefPred is designed to address the challenges faced by researchers in identifying and prioritizing relevant research papers. By employing state-of-the-art NLP techniques and a well-structured recommendation engine, RefPred aims to provide a seamless and efficient way to recommend

research papers based on user input, while accounting for factors like citation count and similarity score.

3.5.2 Requirement Analysis

- **Functional Requirements**

1. Product Perspective: RefPred is a standalone software product that integrates with existing research workflows to assist researchers in their literature review process. By recommending relevant research papers based on the input title or abstract, it aims to enhance the overall research experience and facilitate informed decision-making.
2. Product Features:

   a) Literature ingestion: RefPred can crawl and extract research paper metadata from Semantic Scholar (S2) using their API.

   b) Data storage: The collected metadata is stored in a MongoDB database.

   c) Paper embedding: RefPred uses the SPECTER model to generate embeddings for each paper in the dataset, capturing its citation and semantic meaning.

   d) Embedding space: The system constructs an embedding space of papers based on the generated embeddings.

   e) Recommendation engine: A KNN-based recommendation engine serves as a baseline to provide relevant recommendations for a new paper.

   f) Neural network reranking: RefPred includes a neural network model to rerank the initial KNN candidates, resulting in improved Precision and Recall @ 20 scores.

   g) Input handling: The system accepts a new title or abstract as input and provides relevant paper recommendations based on the input.

   h) Sorting mechanism: RefPred can sort recommended papers based on chosen metrics (e.g., citation count or similarity score).

3. User Characteristics: RefPred caters to researchers, academicians, and students who require assistance in identifying, evaluating, and prioritizing relevant research papers for their literature review process, regardless of their technical expertise.
4. Assumption & Dependencies: RefPred assumes that user-provided data is accurate and complete, and the system's performance depends on the quality of data. The system relies on the continuous availability and accuracy of data sourced from the Semantic Scholar (S2) API and presumes that the underlying technologies remain supported and functional.
5. Domain Requirements: RefPred is designed to cater to research domains with vast and rapidly-evolving bodies of literature, such as Medicine, Generative AI, and other emerging fields. The system can be adapted to suit the specific needs of different domains, ensuring a high degree of relevance and usefulness.
6. User Requirements: Users need to provide a title or abstract for the system to process, and they must have access to the internet to ensure real-time updates and recommendations.

- **Non-functional Requirements**

1. Scalability: RefPred is designed to handle a large and growing number of research papers and embeddings efficiently.
2. Performance: The recommendation engine provides fast and accurate recommendations to researchers, ensuring minimal wait times.
3. Usability: RefPred's user interface is intuitive and easy to use for researchers without extensive technical knowledge.
4. Maintainability: The system is modular and well-documented, allowing for easy updates, bug fixes, and the addition of new features.
5. Reliability: RefPred provides consistent and accurate recommendations across different research fields and queries.
6. Security: The system ensures the privacy and security of user data and intellectual property.

7. Interoperability: RefPred is compatible with various platforms and devices, allowing researchers to access recommendations seamlessly.

- **Organizational Requirements**

    1. Implementation Requirements (in terms of deployment): RefPred requires a scalable and reliable infrastructure for deployment, ensuring high availability, and accommodating growing data and user base. Cloud-based solutions or dedicated servers can be utilized to meet these requirements, with proper backup and disaster recovery plans in place.
    2. Engineering Standard Requirements: RefPred should adhere to industry-standard software engineering practices, including version control, code reviews, continuous integration, and testing. This ensures maintainability, stability, and ease of collaboration among team members.

- **Operational Requirements**

1. Economic: RefPred has the potential to save researchers time and effort in the literature review process, contributing to overall research efficiency and cost-effectiveness.
2. Environmental: As a software product, RefPred has a low environmental impact, reducing the need for physical resources such as paper.
3. Social: RefPred facilitates knowledge sharing and collaboration among researchers, fostering a more inclusive and connected research community.
4. Political: By providing unbiased recommendations, RefPred promotes transparency and fairness in research, avoiding undue influence of specific organizations or individuals.
5. Ethical: RefPred adheres to ethical considerations by ensuring the privacy and security of user data, and recommending papers based on merit and relevance.
6. Health and Safety: RefPred indirectly contributes to health and safety by supporting research efforts that may have implications in these areas.
7. Sustainability: As a digital solution, RefPred supports long-term sustainability by minimizing resource consumption and environmental impact.

8. Legality: RefPred complies with applicable laws and regulations related to data privacy, security, and intellectual property.
9. Inspectability: RefPred's modular and well-documented design enables easy inspection and evaluation by relevant stakeholders, ensuring compliance with industry standards and best practices.

- **Hardware & Software Requirements**

The hardware and software requirements for building this project will depend on the scale of the dataset and the desired performance of the recommendation system. Here are some general requirements for each step:

a) Data Acquisition:

- A machine with a fast internet connection to crawl and scrape the Semantic Scholar API.
- Sufficient storage capacity to store the metadata of the research papers (title, abstract, year, citation count, references, etc.) in a MongoDB database.
- Depending on the size of the dataset, a machine with high RAM and CPU processing power to handle large volumes of data efficiently.
- In our case, we've used a Macbook Pro (M1 Pro with 16 GB of RAM) to scrape approximately 50K papers in 1 hour using the S2 API. For the retraining of SPECTER, we have used Paperspace Gradient Pro with NVIDIA RTX5000 GPU.

b) Paper Embedding Generation:

- A machine with a fast CPU, enough RAM, and preferably, a dedicated GPU, such as an NVIDIA graphics card, (in our case RTX 5000 via cloud service Paperspace Gradient) for quicker training or embedding using the pretrained model.

c) Recommendation Engine:

- A machine with a fast CPU, enough RAM, and preferably, a dedicated GPU, such as an NVIDIA graphics card to perform the KNN search in the high-dimensional embedding space.
- If a more sophisticated recommendation algorithm is desired, such as graph neural networks, then a machine with a more powerful GPU (again with Paperspace Gradient) can help speed things up.

Software:

- Python 3.7 is used with Miniconda, as it provides proper dependency management and virtual environments.
- The Semantic Scholar API and MongoDB database are used to collect and store research paper metadata respectively.
- The SPECTER model and AllenNLP library are used to generate paper embeddings.
- For the recommendation engine, we use the scikit-learn library which provides KNN Trees, and PyTorch for the reranking model.

The list of libraries needed are: **`pymongo, requests, aiohttp, asyncio, pandas, numpy, scikit-learn, PyTorch, AllenNLP, matplotlib, transformers, tqdm, pytorch-lightning, optuna, ujson, pickle, flask`**

The entire code for this project can be found at: *https://github.com/siddharth-Gandhi/refpred/.*

4 RESULTS & DISCUSSION

4.1 Data Acquisition & Parsing data from S2:

The following table shows the time it took to parse and store different numbers of papers both synchronously and asynchronously:

Table 2: Time taken to crawl & store paper metadata locally (from S2 API)

Number of Papers	Time Taken
100 papers (Synchronously)	2 minutes
100 papers (With Async Crawler)	3 seconds
10,000 papers	~10 minutes
50,000 papers	~1 hour
100,000 papers (with batch processing)	~ 1 hour

For all further reranking experiments we will be using the 10,000 paper datasets to construct the train, val and test datasets. We will also use the 100,000 paper dataset to populate our KNN Tree to ensure most references of the original 10,000 papers are present. However, from both datasets, we remove some paper IDs for which abstracts or list of references is null.

4.2 SPECTER Retraining

Retraining SPECTER model requires triplets in the form (Q, Q^{+}, Q^{-}) as input. To do this we need first divide the 10,000 initial paper IDs into train, val and test sets with 80, 10, 10 split. This gives:

Table 3: Train, Val, Test Split

Dataset	Number of IDs
Train	7455
Val	932
Test	933

We will be using this same 80-10-10 split throughout all of our experiments.

Using these datasets, we generate the triplets for training SPECTER:

Table 4: Number of Generated triplets for SPECTER

Dataset	Number of triplets
Train	31160
Val	3920
Test	3950

Finally, when training the SPECTER model with the above triplets, we get the following:

Table 5: Comparison of retraining SPECTER vs Original Paper

	Our Retrained Model	Original SPECTER Model
Epochs	1	4
Batch Size	4	4
Number of Train Triplets	31,160	684,000
Number of Val Triplets	3920	145,000
Time Taken / epoch	2 hours	1-2 days
GPU Used	RTX 5000 (16 GB)	Titan V (12 GB)

Using SPECTER, we can embed a research paper (specifically the title + abstract of a paper), into a 768-dimensional embedding. We use tSNE (t-distributed stochastic neighbour embedding) to visualize these high-dimensional embeddings and see the grouping of papers (Figure 12). We compare our retrained model against the pre-trained SPECTER model available on HuggingFace (HF). The biggest circles represent highly cited papers like the original papers of Adam, R, ImageNet, Transformers, etc.

Using a tool like Plotly, where we can interactively see what paper each dot represents, we can see that both versions are great at grouping closely related papers. For example, papers for Image Segmentation are grouped in one place while Markov Model papers are grouped in another. However, the pretrained version seems to be better at modelling distant references (probably due to the bigger training set) and thus might produce more accurate embeddings.

Ultimately, due to the intensive compute and time requirements, and the maximum usage limitations provided in Paperspace Gradient (the cloud platform we used to retrain SPECTER), we decided it would be best to use the HF embeddings for further experiments to get the best quality embeddings.

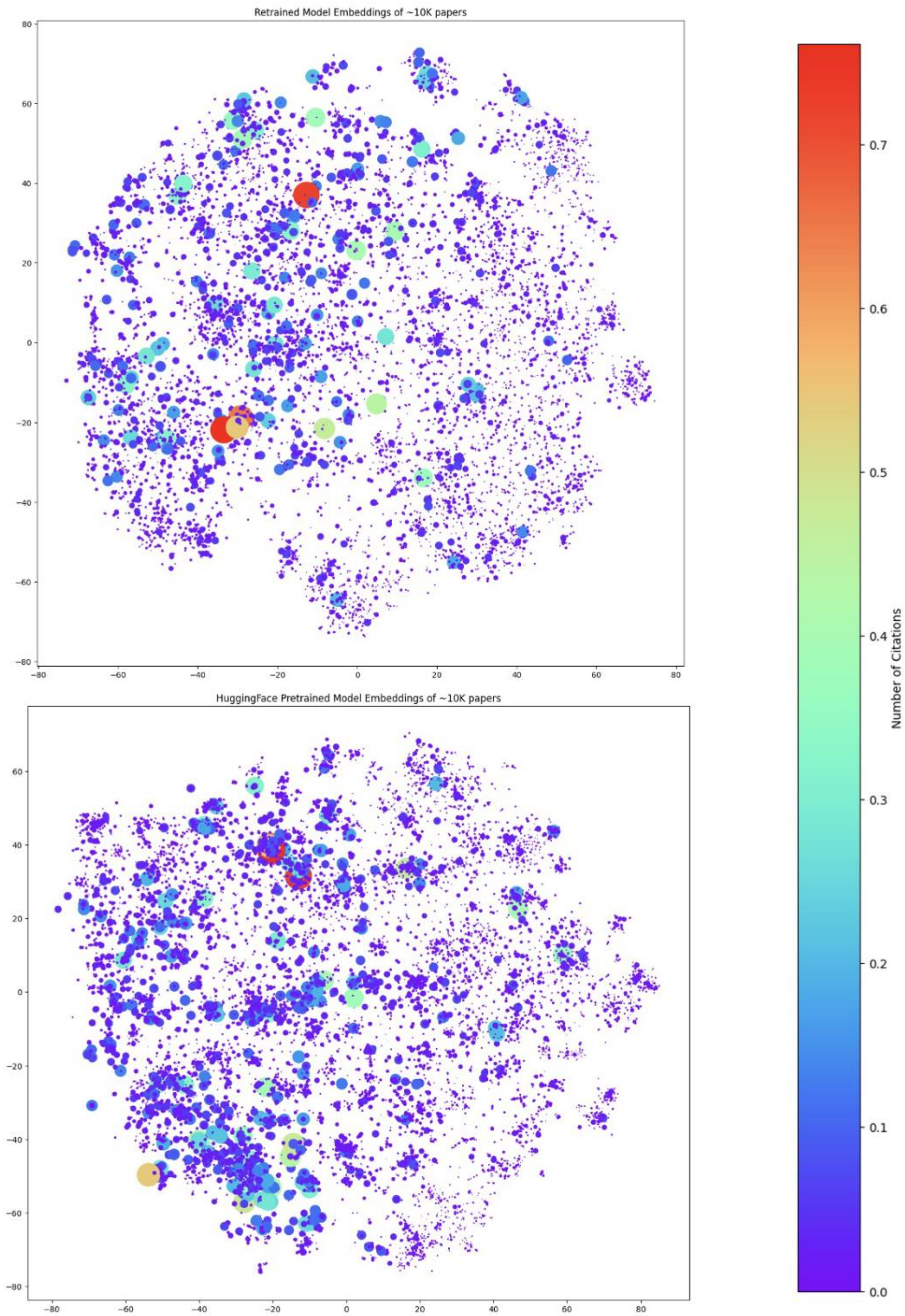


Figure 12: 2D tSNE Visualization of 768-dimensional embedding vectors of both retrained & HF model (size & colour based on number of citations)

4.3 Neural Network-based Reranking

At this point, we have the high dimensional SPECTER embeddings for all the 9319 original papers (excluding papers with null references or abstracts) + 107833 reference papers (to populate the KNN Tree).

4.3.1 Exploratory Data Analysis

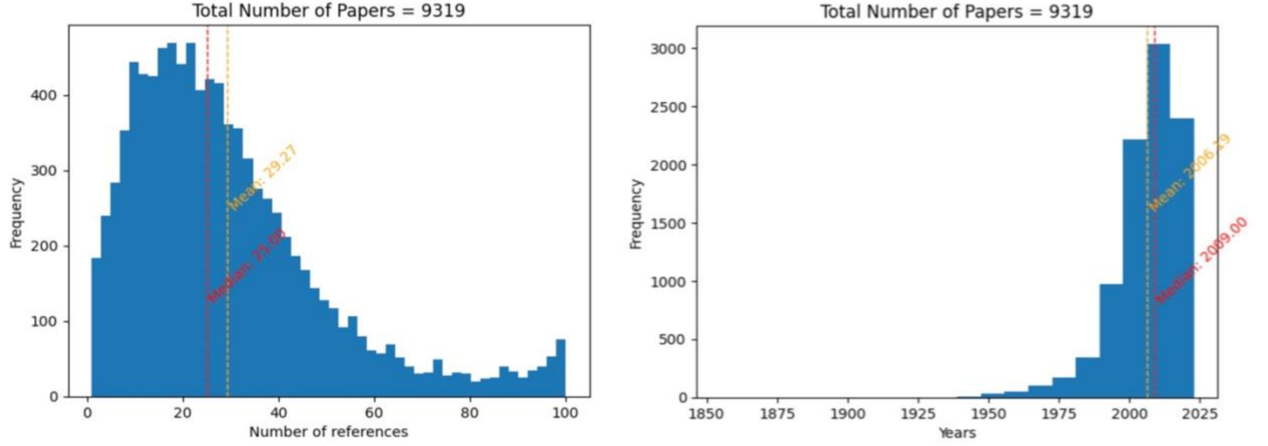


Figure 13: EDA of the paper data

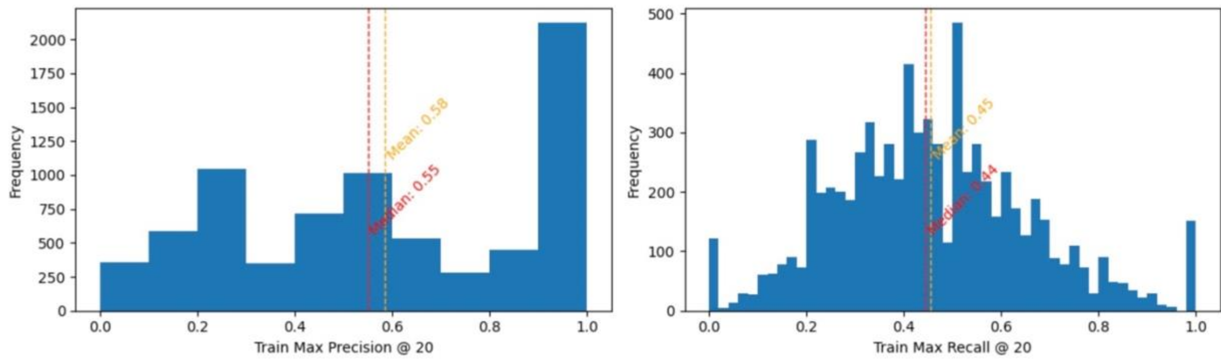
As we can see most papers have on average 29 papers as references and our dataset has most papers published around 2006-07.

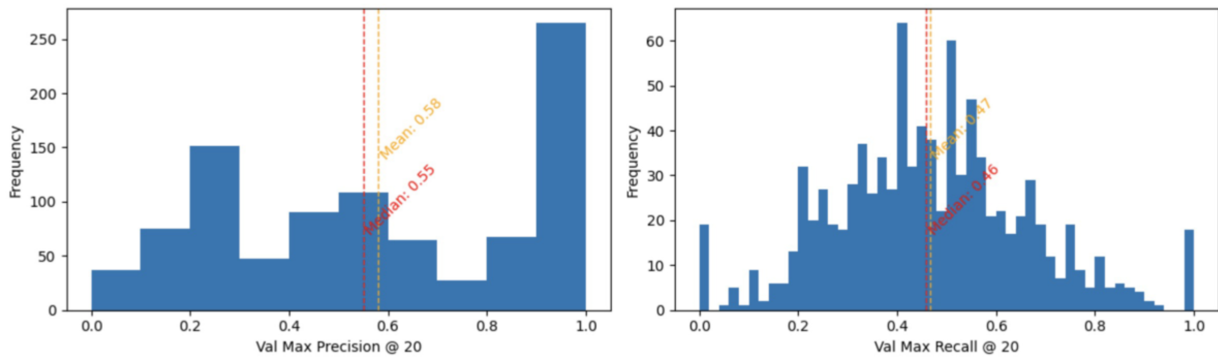
Now for all train, val and test datasets (see Dataset), we store `max_refs_present` and `total_refs` for logging. Using this we can calculate the maximum possible precision and recall for each of the train, val and test datasets. The formula for calculating the maximum precision and recall for a paper at index i in the dataset is as follows:

$$max_precision_i = \frac{min(max_ref_i, top_k)}{top_k}, \quad \text{for } i = 1,2,\dots,n$$

$$max_recall_i = \frac{min(max_ref_i, top_k)}{total_ref_i}, \quad \text{for } i = 1,2,\dots,n$$

where top_k is the k in $Precision @ k$ and $Recall @ k$ (in our case we take $top_k = 20$). We plot the histograms for max precision and recall for each of the train, val and test sets respectively:





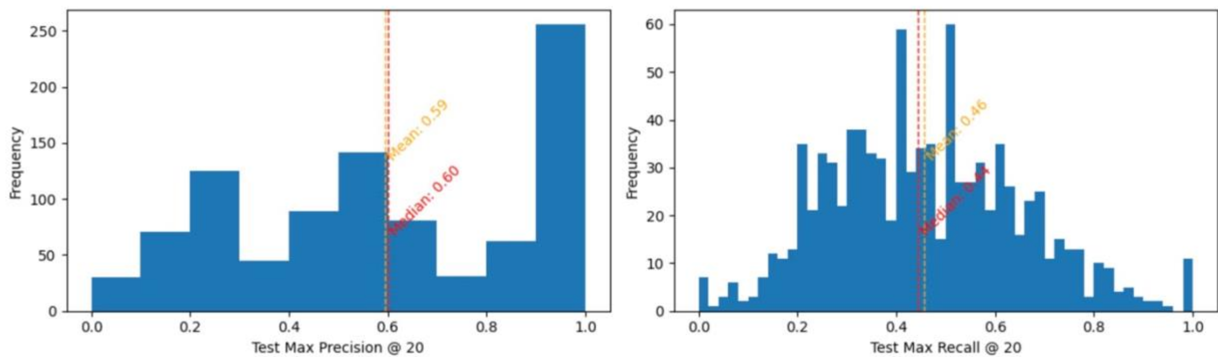


Figure 14: Histograms of Max Precision @ 20 and Max Recall @ 20 for train, val and test paper ID sets.

As we can see with the current KNN Tree (with ~100,000 papers), we can have on average 0.58 as maximum $Precision$ @ 20 and 0.46 as maximum $Recall$ @ 20.

4.3.2 Generating Pairs for the Reranking Model

Table 6: Information about preparing +ve and -ve pairs

Dataset	Total Pairs	+ve Pairs	-ve Pairs	Time Taken
Train	208346	104173	104173	16 minutes
Val	25440	12770	12770	2 minutes
Test	25440	12770	12770	2 minutes

As can be seen in the table above, the Dataset is constructed in a way to maintain $+/-$ ratio to be 1. This is done by selecting as many positive and negative papers as there are references present in the KNN recommendations of a particular paper. For example, if a query paper Q has 37 references in total, with only 26 papers present in the top 1000 nearest neighbours of Q, then we make 26 positive pairs (of all the references present in KNN neighbours) and 26 negative pairs (selected **randomly**). The impact of being random (and not in the order of KNN) we will see in the following sections.

- **Hyperparameters**

As mentioned in Section 3.4.3/Model, we have a feed-forward neural network with batch norm, weight decay, early stopping, learning rate schedulers and variable hidden layer sizes. As a result, the various hyperparameters which can be tweaked as listed below. The ones of interest are **hidden_dims**, **weight_decay**, **dropout_prob**, **use_bn** and **bn_momentum**. We will be tweaking with them in the following section with an Ablation study and hyperparameter optimization.

```
hparams = {
    'batch_size': 2048,
    'hidden_dims': [1024, 2048],
    'lr': 1e-2,
    'min_lr': 1e-7,
    'patience': 3,
    'factor': 0.1,
    'weight_decay': 1e-3,
    'dropout_prob': 0.5,
    'stop_after': 5,
    'knn_k': 1000, # for getting nearest neighbours
    'top_k': 20, # for evaluation
    'use_bn': True,
    'bn_momentum': 0.9,
    'num_epochs': 30
}
```

- **Varying the Methods for preparing the Dataset**

An important element in the reranking model is the data which is fed into it. The dataset (whose construction is explained in Section 3.4.3/Dataset) creates pairs (Q, R) such that R is one of the (say) 1000 KNN recommendations of Q and the label for the pair is 1 if R is a direct recommendation of Q, else 0. However, selecting the negative distractors (with label 0) is an important experiment. We could:

3. Select negative elements in the order of KNN neighbours, i.e. in the order of similarity to the Q paper.
4. Randomly select R papers from the 1000 KNN recommendations.
5. Select R from the inverse order of KNN recommendations (i.e. least similar first).
6. Inspired by (Cohan 2020), we take 50% negative examples randomly and 50% negative examples which are citations of citations (so similar to Q, but not a citation).

Finally, since we have KNN neighbours which are published after the current query Q, we can explore if skipping those recommendations is better than labelling it as 0 (in all 4 approaches above). The results for these various methods for preparing dataset are mentioned below:

Table 7: Model Loss w.r.t. various negative distractor selection & skipping future papers

Model	Train Loss	Val Loss	Test Precision @ 20	Test Recall @ 20
KNN Baseline	-	-	0.117	0.095
Negatives selected in KNN order (by L2 distance)	0.274	0.446	0.016	0.014
Randomly Selected Negatives	0.13	0.316	**0.193**	**0.166**
Negatives selected starting for reversed KNN order	0.08	0.25	0.151	0.123
Negatives with 50% random & 50% citation of citations	0.504	0.588	0.168	0.140
Skip Papers published after Query Q	0.35	1.269	0.011	0.008

We can see that using randomly selected negative distractors (including future papers with 0 labels) yields the best results. Going in the order of KNN (by L2 distance) and selecting negatives results in a terrible score, probably because the model learns to rank semantically similar papers (which are the KNN neighbours) lower due to the 0 label. Even skipping future papers yields

terrible results, presumably because skipping those papers leads to the model having no idea of what to do with future papers.

- **Varying Model Architecture**

To test whether the model architecture described in Section 3.4.3/Model performs well, we do an ablation study studying the effects of removing individual components (in isolation) and seeing its effect on evaluation metrics. We also test the effect of replacing **`x_diff`** and **`x_mul`** with **`x1`** and **`x2`**. The results are as follows:

Table 8: Ablation Study (after 30 epochs)

Config	**Batch Norm** (0.5)	**Drop out** (0.5)	**Weight Decay** (1e-3)	**`Is after`**	**`Cosine Sim`**	**`x_mul`**	**`x_diff`**	**Val Loss**	**Test Precision @ 20**	**Test Recall @ 20**
Base Model	Yes	Yes	Yes	Yes	Yes	Yes	Yes	0.316	0.193	0.166
No BN	*No*	Yes	Yes	Yes	Yes	Yes	Yes	0.693	0.118	0.09
No Dropout	Yes	*No*	Yes	Yes	Yes	Yes	Yes	0.731	0.136	0.115
No Weight Decay	Yes	Yes	*No*	Yes	Yes	Yes	Yes	0.321	**0.199**	**0.171**
No **`is_after`**	Yes	Yes	Yes	*No*	Yes	Yes	Yes	0.346	0.138	0.112
No **`cosine sim`**	Yes	Yes	Yes	Yes	*No*	Yes	Yes	0.314	0.186	0.155
No **`x_mul`**	Yes	Yes	Yes	Yes	Yes	*No*	Yes	0.559	0.195	0.16
No **`x_diff`**	Yes	Yes	Yes	Yes	Yes	Yes	*No*	0.28	0.173	0.144
With **`x1`** & **`x2`** instead of **`x_mul`** & **`x_diff`**	Yes	Yes	Yes	Yes	Yes	*Replaced with* **`x1`**	*Replaced with* **`x2`**	0.14	0.11	0.09

With this table, it is clear that BN, Dropout are crucial layers in the model architecture while **`cosine_sim`**, **`x_mul`**, and **`x_diff`** are important features to be added in the forward pass of the model. The only optional argument seems to be weight decay, excluding which leads to better results.

- **Hyperparameter Optimization**

To find out the optimal values for **`hidden_dims`**, **`dropout_prob`**, **`weight_decay`**, and **`bn_momentum`**, we perform a Hyperparameter search using the **`optuna`** python library. This was done by having **`is_after`**, **`cosine_sim`**, **`x_mul`** and **`x_diff`** being passed to the forward pass of the model. Some of the notable results are as follows:

Table 9: Hyperparameter Optimization (after 30 epochs)

Config	Hidden Dims	Drop out	Weight Decay	BN momentum	Val Loss	Test Precision @ 20	Test Recall @ 20
Config #1	[512, 768, 1536]	0.9	0.001	0.3	0.435	0.167	0.139
Config #2	[1024]	0.3	0.001	0.5	0.333	0.177	0.148
Config #3	[512, 512]	0.5	0.001	0.3	0.3140	0.184	0.154
Config #4	[768, 1536]	0.5	0.0001	0.7	0.330	0.178	0.150
Config #5	[768, 256]	0.5	0.0001	0.7	0.322	0.189	0.161
Config #6	[1024, 2048]	0.5	0.001	0.9	0.316	**0.193**	**0.166**
Config #7	[512, 768, 1024, 1536]	0.5	0.001	0.7	0.330	0.175	0.145

As we can see, adding more layers leads to worse evaluation metrics, presumably because of high overfitting. A moderate dropout probability is also important to keep overfitting in check. Finally, weight decay and BN momentum do not seem to have much impact on the final results.

- **Final Model & Comparison With KNN Baseline**

After much experimentation, we find that the best-performing model is the one with the hyperparameters mentioned in Section 4.3.2/Hyperparameters but without any weight decay. Training the model is only effective up to around 30-40 epochs after which learning stagnates. The results can be seen in the graph below.

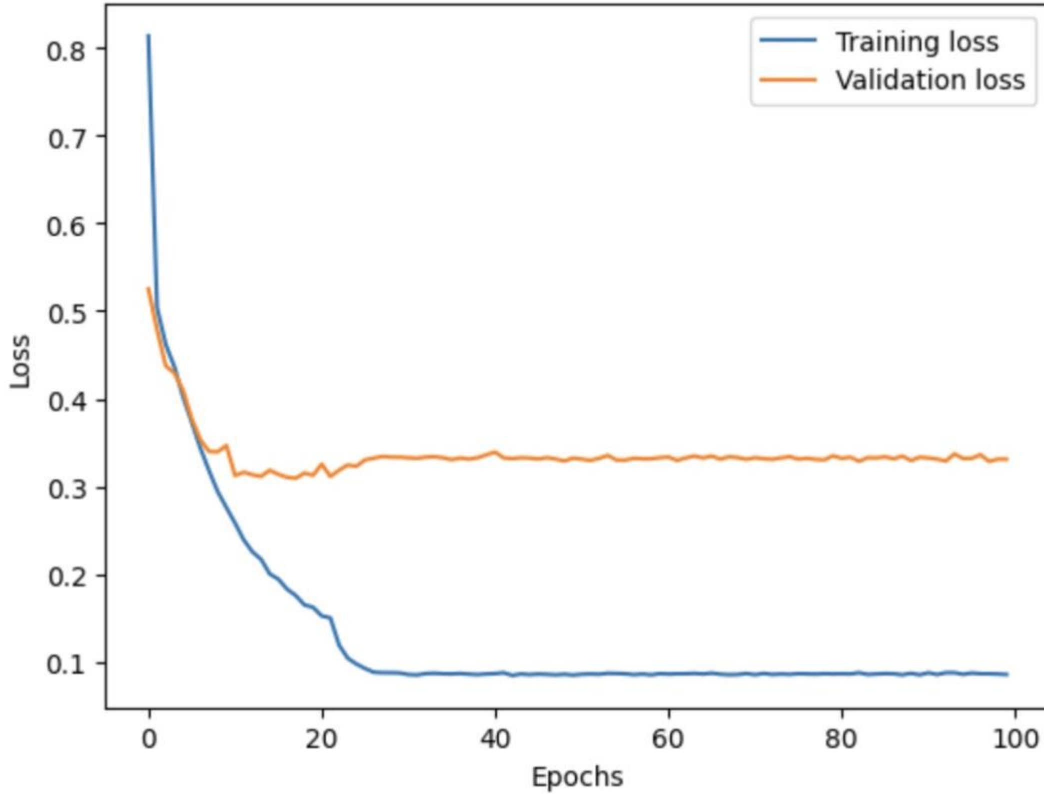


Figure 15: Train & Val Loss vs Epochs for Best Model

The final evaluation results can be seen in the table below:

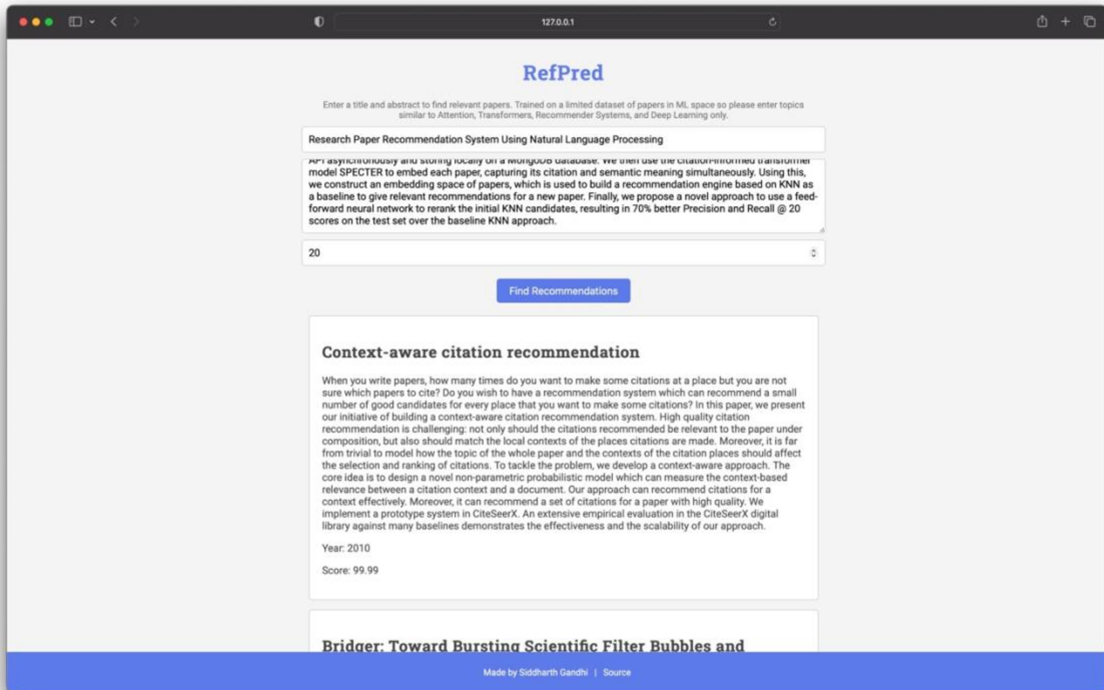
Table 10: Best Model vs KNN Baseline

Model	Train Precision @ 20	Train Recall @ 20	Val Precision @ 20	Val Recall @ 20	Test Precision @ 20	Test Recall @ 20
KNN (baseline)	0.118	0.096	0.117	0.099	0.117	0.095
KNN + Best Reranking Model	**0.1946**	**0.1643**	**0.1987**	**0.1733**	**0.199**	**0.171**

As we can see there is almost a **70% improvement** in the evaluation metrics across all 3 - train, val and test datasets, over the baseline KNN model. However, we are still quite far from the theoretical maximum Precision @ 20 and Recall @ 20 values we derived in Section 4.3.1.

4.4 Web App

We also implemented a flask web app with the final model, which can be seen below:



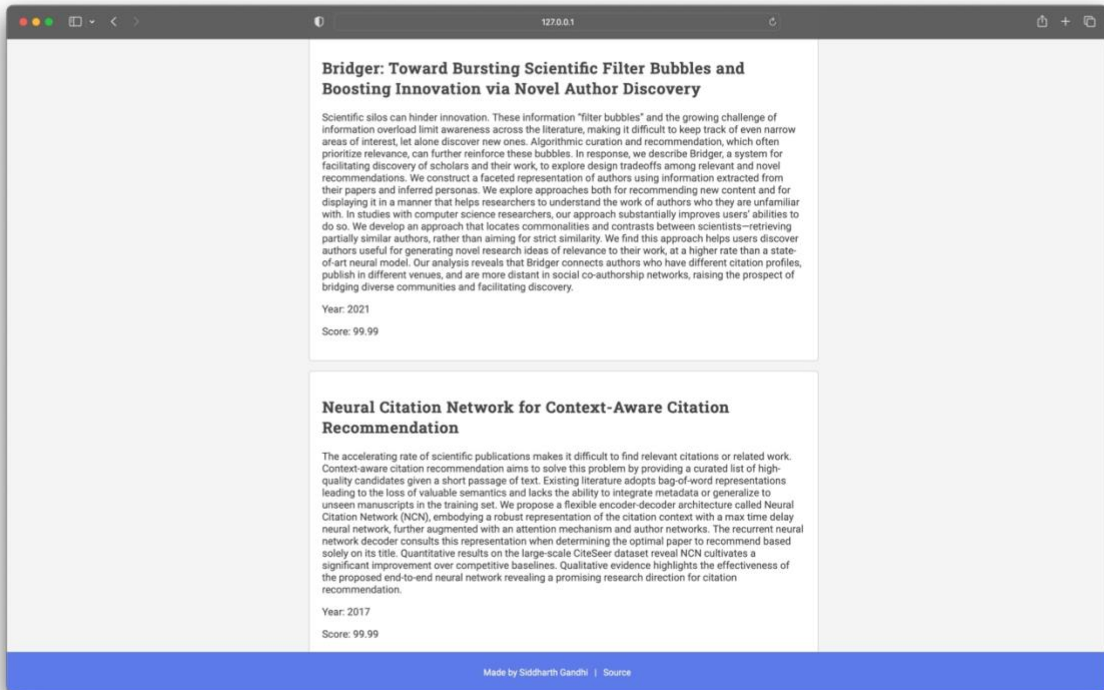


Figure 16: Final Reranked Model implemented into a Flask Web App

5 CONCLUSION & FUTURE WORK

In conclusion, this project investigated the development of a research paper recommendation system that leverages citation-informed transformers (SPECTER) to generate dense paper embeddings. The baseline recommendations were obtained using KNN, and our new deep learning reranking approach improved these recommendations by 70% (measured by Precision @ 20 and Recall @ 20), yielding more accurate results compared to the baseline KNN approach on historical papers and their citation graphs. The primary contributions of the project include:

1. Design and implementation of a fast and efficient asynchronous crawler from scratch, capable of recursively parsing thousands of specific research papers from Semantic Scholar, along with their references, and storing them locally on a MongoDB database.

2. Attempt to retrain and recreate the original SPECTER results for comparison with the pretrained version. However, due to vast compute requirements, it was not possible to fully replicate the results, necessitating the use of the pretrained model for further experiments.

3. Design and implementation of a novel approach to rerank KNN recommendations from the SPECTER model based on historical citation graphs. This was achieved by creating pairs from past citation graphs and allowing the model to determine the new similarity score between two SPECTER embeddings of a query and a KNN recommendation guided by the labels. Various aspects, such as model architecture, dataset preparation (including selecting negative distractors), and hyperparameters, which were extensively tested, influenced the final evaluation metrics. We also implemented a Flask Web App using the final reranking mode.

Despite these advancements, there remains significant room for improvement, as the improved results with reranking (Precision @ 20 = 0.19 and Recall @ 20 = 0.17) are still far from the theoretical maximum of Precision @ 20 = 0.58 and Recall @ 20 = 0.47. Potential avenues for future research include:

1. Establishing a comprehensive pipeline that incorporates SPECTER embeddings and the reranking model, making the entire system trainable. This would enable the capture of

embeddings guided by both semantics and citation graphs, though it may be computationally expensive.

2. Developing more intelligent techniques for selecting negative pairs or preparing the dataset for reranking.

3. Exploring the use of Graph Neural Networks or other advanced neural network architectures (such as attention mechanisms) for the reranking model instead of conventional Feed Forward Neural Networks.

4. Investigating the integration of additional features, such as authorship, publication venue, and temporal information, to further enhance the recommendation system's performance. This could provide more context to the recommendation process and help identify highly relevant papers that share similar authorship or are published in the same domain-specific venues.

5. Exploring transfer learning and fine-tuning techniques to adapt pre-trained language models like BERT, RoBERTa, or GPT for the specific task of research paper recommendation. These models have shown strong performance in various natural language processing tasks and could potentially improve the accuracy and relevance of recommendations when fine-tuned for this application.

6 REFERENCES

[1] Nigram, Nishant. 2021. *Research Paper Analysis using Natural Language Processing.* https://www.cs.cit.tum.de/en/sccs/news/sccs-colloquium/article/nishant-nigam-research-paper-analysis-using-nlp-techniques/.

[2] Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. *Attention is all you need.* Advances in neural information processing systems 30.

[3] Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. *Bert: Pre-training of deep bidirectional transformers for language understanding.* arXiv preprint arXiv:1810.04805.

[4] Beltagy, Iz, Kyle Lo, and Arman Cohan. 2019. *SciBERT: A pretrained language model for scientific text.* arXiv preprint arXiv:1903.10676 .

[5] Jeong, Chanwoo, Sion Jang, Eunjeong Park, and Sungchul Choi. 2020. *A context-aware citation recommendation model with BERT and graph convolutional networks.* Scientometrics 124.

[6] Bhagavatula, Chandra, Sergey Feldman, Russell Power, and Waleed Ammar. 2018. *Content-based citation recommendation.* arXiv preprint arXiv:1802.08301.

[7] Cohan, Arman, Sergey Feldman, Iz Beltagy, Doug Downey, and Daniel S. Weld. 2020. *Specter: Document-level representation learning using citation-informed transformers.* arXiv preprint arXiv:2004.07180.

[8] Lo, Kyle, Lucy Lu Wang, Mark Neumann, Rodney Kinney, and Dan S. Weld. 2019. *S2ORC: The semantic scholar open research corpus.* arXiv preprint arXiv:1911.02782.

[9] Singh, Amanpreet, Mike D'Arcy, Arman Cohan, Doug Downey, and Sergey Feldman. 2022. *SciRepEval: A Multi-Format Benchmark for Scientific Document Representations.* arXiv preprint arXiv:2211.13308 .

[10] Dolatshah, Mohamad, Ali Hadian, and Behrouz Minaei-Bidgoli. 2015. *Ball*-tree: Efficient spatial indexing for constrained nearest-neighbor search in metric spaces* . arXiv preprint arXiv:1511.00628.

[11] Burges, Chris, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. 2005. *Learning to rank using gradient descent.* In Proceedings of the 22nd international conference on Machine learning, pp. 89-96.

[12] Ebesu, Travis, and Yi Fang. 2017. *Neural citation network for context-aware citation recommendation.* Proceedings of the 40th international ACM SIGIR conference on research and development in information retrieval.

[13] Narechania, Arpit, Alireza Karduni, Ryan Wesslen, and Emily Wall. 2021. *Vitality: Promoting serendipitous discovery of academic literature with transformers & visual analytics.* IEEE Transactions on Visualization and Computer Graphics 28, no. 1 (2021): 486-496.

[14] Portenoy, Jason, Marissa Radensky, Jevin D. West, Eric Horvitz, Daniel S. Weld, and Tom Hope. 2022. *Bursting scientific filter bubbles: Boosting innovation via novel author discovery.* In Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems (pp. 1-13).