Augmenting Academic Research Search And Reading With Richer Context

by

Valay Gaurang Dave

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Master Of Science

Approved April 2021 by the
Graduate Supervisory Committee:

Jia Zou, Co-Chair
Heni Ben Amor, Co-Chair
Kasim Selcuk Candan

ARIZONA STATE UNIVERSITY

May 2021

ABSTRACT

The volume of scientific research is growing at an exponential rate over the past 100 years. With the advent of the internet and ubiquitous access to the web, academic research search engines such as Google Scholar, Microsoft Academic, etc., have become the go-to platforms for systemic reviews and search. Although many academic search engines host lots of content, they provide minimal context about where the search terms matched. Many of these search engines also fail to provide additional tools which can help enhance a researcher's understanding of research content outside their respective websites. An example of such a tool can be a browser extension/plugin that surfaces context-relevant information about a research article when the user reads a research article.

This dissertation discusses a solution developed to bring more intrinsic characteristics of research documents such as the structure of the research document, tables in the document, the keywords associated with the document to improve search capabilities and augment the information a researcher may read. The prototype solution named Sci-Genie(https://sci-genie.com/) is a search engine over scientific articles from Computer Science ArXiv.

Sci-Genie parses research papers and indexes research documents' structure to provide context-relevant information about the matched search fragments. The same search engine also powers a browser extension to augment the information about a research article the user may be reading. The browser extension augments the user's interface with information about tables from the cited papers, other papers by the same authors and even the citations to and from the current article. The browser extension is further powered with access endpoints that leverage a machine learning model to filter tables comparing various entities. The dissertation further discusses

these machine learning models and some baselines that help classify whether a table is comparing various entities or not. The dissertation finally concludes by discussing the current shortcomings of Sci-Genie and possible future research scope based on learnings after building Sci-Genie.

DEDICATION

I dedicate this thesis to three groups of people.

I dedicate it to my parents and family, who have supported me in the best and worst of times.

I dedicate it to Rohan, who has patiently listened to my wild ideas and kept motivating me in during the times of utter failure.

I dedicate it to Misaal, Abhay, Pandu, Anand Sir and Nilesh Masa; My first teachers in the schools of Computer Science, business and life.

ACKNOWLEDGMENTS

I would like to thank my advisors Dr. Jia Zou and Dr. Heni Ben Amor, for guiding me during the course of my thesis. I came up with so many valuable ideas because of our discussions. I am also thankful to Dr. Candan for serving as a member in my thesis committee.

I would like to thank Savin Goyal and the Netflix Metaflow[1] team for creating a wonderful library that helped build many difficult components of the thesis and for providing a sandbox compute environment when building some components for this dissertation.

I would like to thank Hongyu Zeng from University of Iowa and Boyi Qian from Purdue University for helping label information needed in the machine learning task.

I would like to thank the members of CACTUS Lab and Interactive Robotics Lab who provided feedback as initial beta users.

I would like to thank Papers With Code[2], Knowldege Media Institute[3], Pytorch[4] and Elastic[5] for creating libraries which were extreamely essential for some components built in this dissertation.

I would like to thank Misaal Turakia for providing a lot of guidance around design and Rohan Pandya for helping fix a lot of the writing.

[1]https://metaflow.org/

[2]https://paperswithcode.com/

[3]https://cso.kmi.open.ac.uk/

[4]https://pytorch.org/

[5]https://www.elastic.co/

TABLE OF CONTENTS

CHAPTER Page

CHAPTER Page

Page

APPENDIX

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

Chapter 1

INTRODUCTION

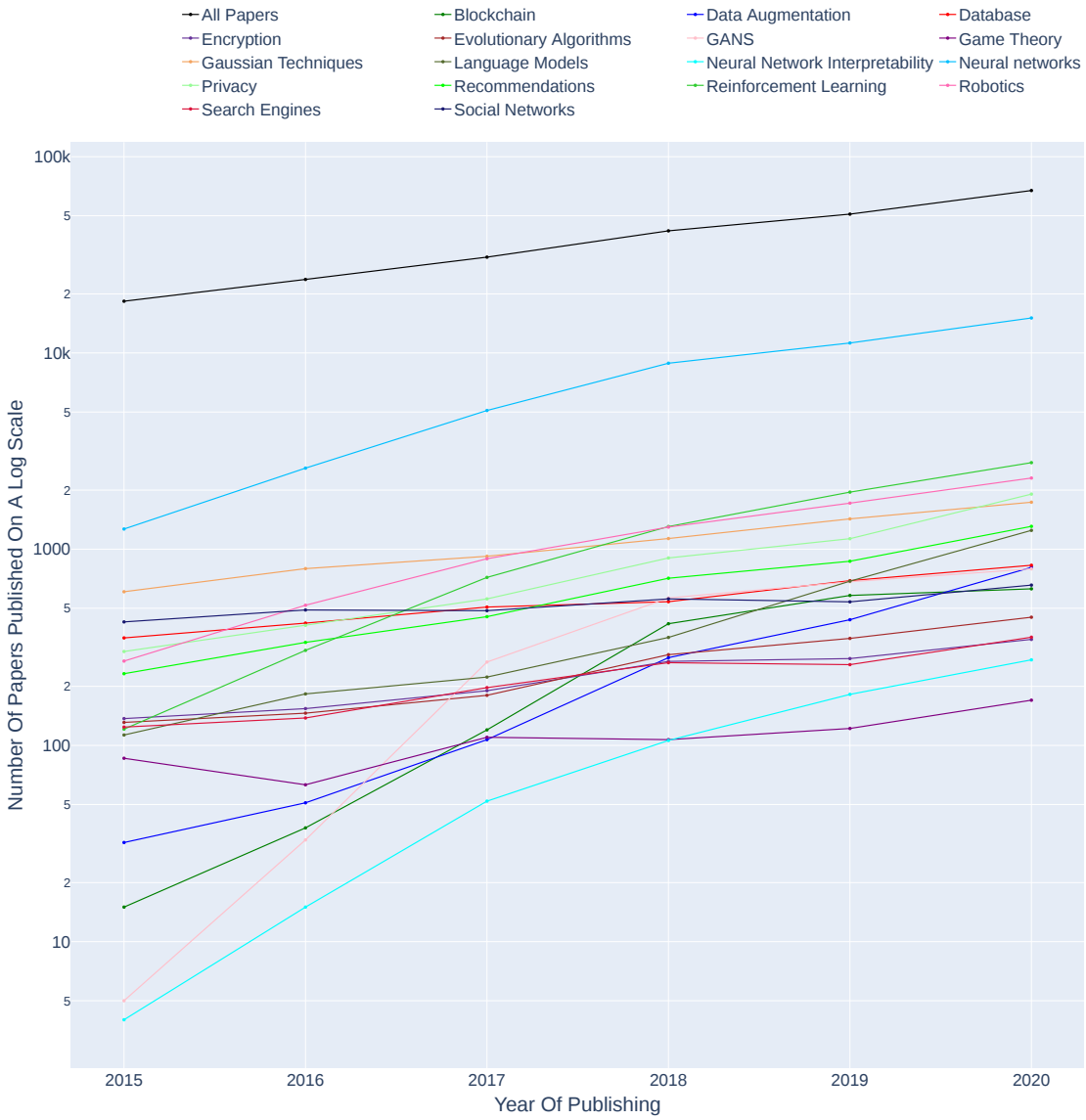


Figure 1. A time series plotting count of papers every year for across various topics in CS from January 2015 to December 2020.

The volume of scientific literature has doubled in the past 20 years[6]. The ubiquity of computing and access to free, open-source software has also enabled exponential growth in publishing for computer science fields. Figure 1 shows a plot of the number of papers published annually on computer science(CS) ArXiv[7] each year 2015 to 2020 on a log scale.

The growth in the research volume has also led to academic research search engines becoming primary mediums of access for scientific literature.

Academic research search engines are different from traditional search engines like Google because they host content specific to scientific literature. Scientific papers generally undergo a process of peer-review. If a paper does not undergo peer review, it may still get cited by other publications/articles. This relational nature of academic research makes it relatively easy to filter credibility based on citations as a metric; Major academic research search engines such as Google Scholar, Microsoft Academic, etc., leverage citations as a metric for ranking documents. Although citations are beneficial, a search query may match many records during the search process. The increase in the number of documents can also result in a sea of noise for a particular search query. In such cases, the researcher gets minimal context information around search matches to infer the usefulness of a search result.

Researchers also tend to look for information embedded deep inside papers. Many papers contain table/s of results for some experiments or comparisons to techniques/methods from other papers. The methods/techniques that are compared are generally cited; But the access to thier information requires researchers to manually hunt for the context behind the comparison. This type of tabular information is not

[6]https://data.worldbank.org/indicator/IP.JRN.ARTC.SC

[7]https://arxiv.org

easily accessible through normal search engines. It is also not readily available at the time of reading research. A platform like PapersWithCode[8] hosts/tracks State-of-the-art(SOTA) leaderboards for Machine Learning(ML) along with their papers. The access to information within this platform is constrained to ML and doesn't provide information outside papers from ML papers.

The next section aims to provide a motivating example to explain the lack of context in search results displayed by three large proprietary search engines.

1.1 Motivating Example: Search Results From Academic Research Search Engines

Machine Learning and deep learning research dominate most CS publications on pre-print servers like ArXiv (Figure 1). Based on this information, consider the use-case of a researcher trying to look for papers where a Transformer(Vaswani et al. 2017) model written in PyTorch(Paszke et al. 2019) is given adversarial examples. The researcher chooses to use the following search query: *pytorch transformer adversarial examples* on the search engines Google Scholar, Semantic Scholar, and Microsoft Academic. Ideally, the experiments or the appendix or the methodology section of a paper would consist of information about code/model in a paper. The following subsections will analyze the search results for this query for three large search engines: Google Scholar, Microsoft Academic, and Semantic Scholar.

[8]https://paperswithcode.com

1.1.1 Search Results From Google Scholar

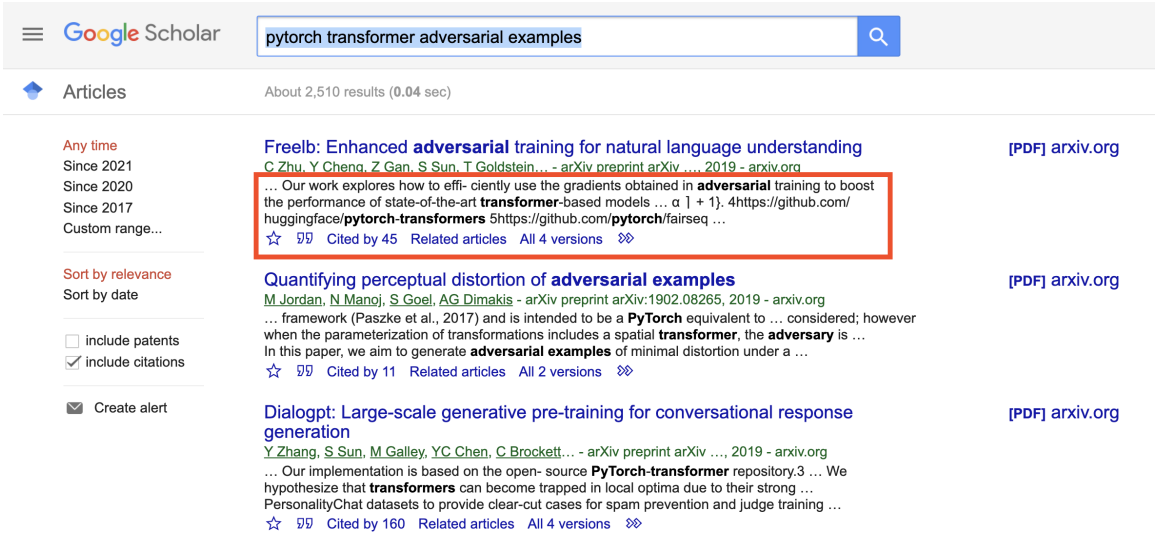


Figure 2. Search Results For Google Scholar.

Although Google scholar found 2000+ search results, the search-result-fragments do not provide any additional context information about where the terms are matching. There is no other context information except for links highlighted in the search results i.e., No mention of why the fragment matched.

1.1.2 Search Results From Microsoft Academic

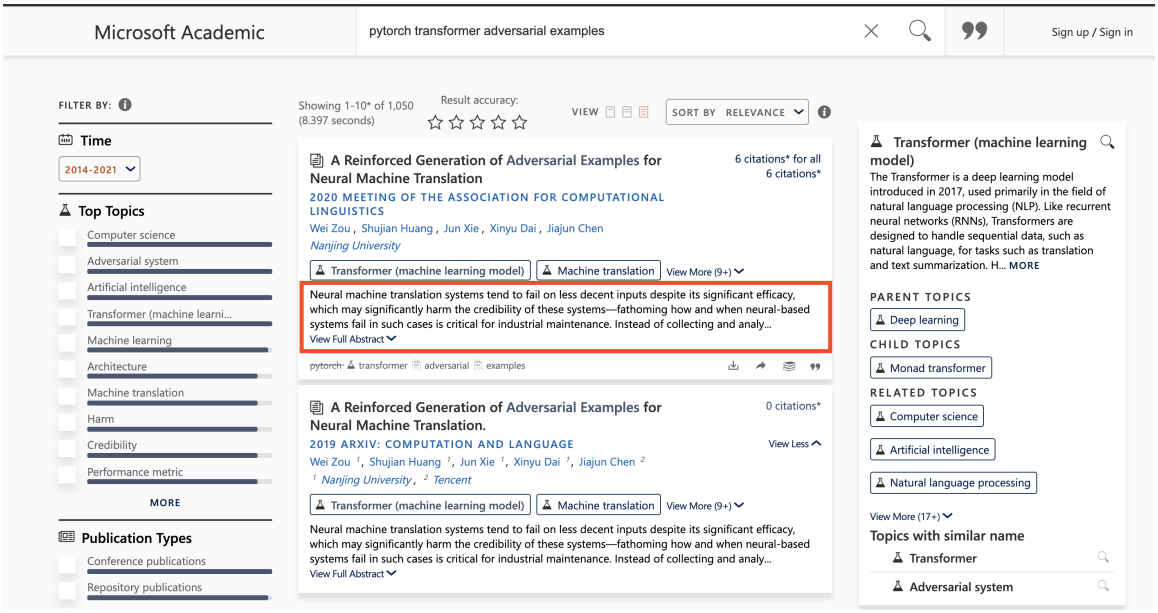


Figure 3. Search Results For Microsoft Academic.

Microsoft Academic provides additional information around context like topics, analytics on types of publication matches etc, It fails to provide information about where the match took place in the paper.

1.1.3 Search Results From Semantic Scholar.

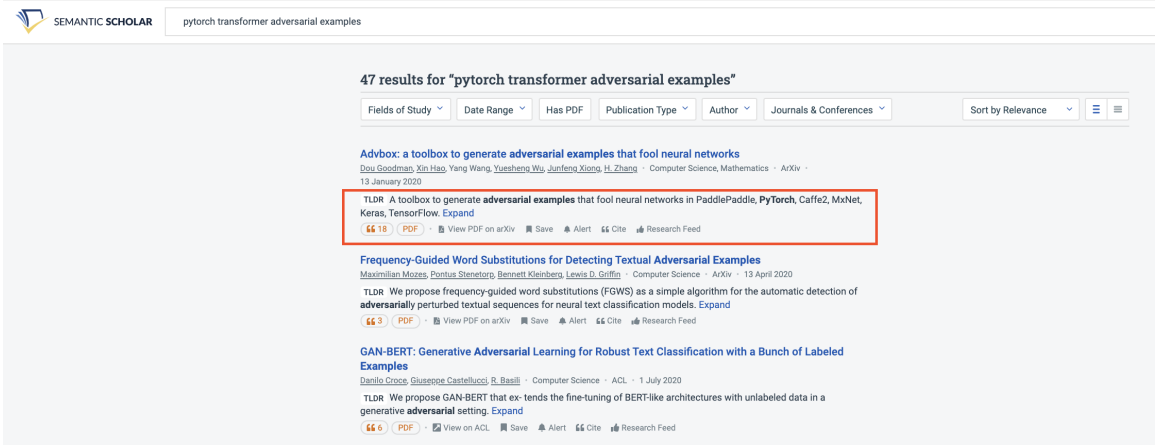


Figure 4. Search Results For Semantic Scholar.

Semantic Scholar highlights few search fragments where results would have matched and presented a smaller result set. But all search results do not consist of context about where the match took place.

1.2 Characterizing Missing Traits

1.2.1 Sparse Context In Search Results About Where Search Terms Matched Inside A Document

All search results shown in section 1.1.1, 1.1.2, 1.1.3, do not provide any additional information about where the match took place inside the research document. An interesting characteristic of research documents is that they possess structure and hierarchy. Research documents have well-laid out sections such as "Introduction," "Related Works," "Methodology," "Experiments," etc. that can provide context about the matched search terms. Such information can provide insights to the researcher about the context around which the keyword was stated in a search result.

1.2.2 Minimal Access To Information When Reading Research

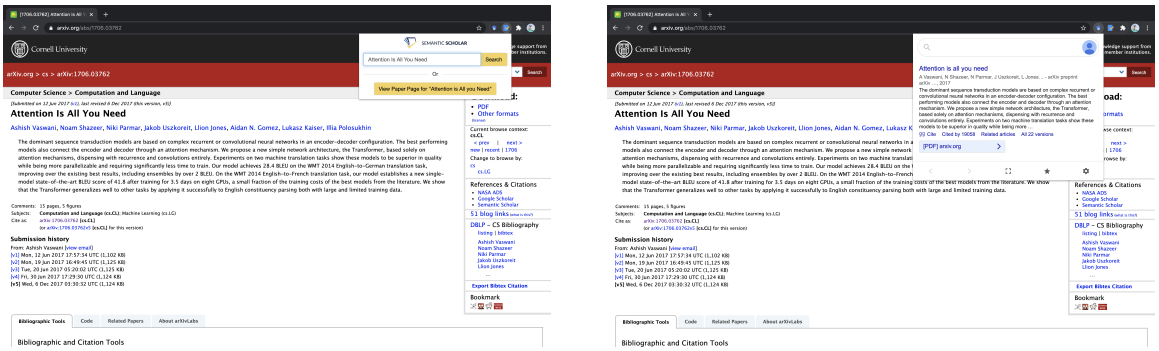


Figure 5. Google Scholar and Semantic Scholar Plugins.

Quite a few scientific literature search engines also offer forward citation search features (Gusenbauer and Haddaway 2020), but these features are not always accessible

outside the search engine's website. Google Scholar and Semantic Scholar provide browser extensions to help the researchers get more context on a research document, as seen from Figure 5. These extensions lead back to the search engine's proprietary websites for further access. They also do not provide information within the browser when the researcher is reading an article online.

1.2.3 No Access To More Fine-Grained Information Such as Tables When Reading Research

Many research papers compare methods from other research papers; such information is not directly accessible to the researcher when reading a research article online. Figure 6 is a good example of this type of information. Figure 6 describes a table showing comparison of methods. This table is derived from Messina et al. 2020. One of the method(VSRN) in the table comes from Li et al. 2019. As comparisons are on the MS-COCO dataset (Lin et al. 2014), the user has to manually hunt for VSRN's performance from its original paper to understand variance in the method's performance from what was initially reported. This task of discovering tables with comparisons is a time-consuming task in research which requires a lot of mannual effort by a researcher.

With the recent growth in Machine Learning publications, new platforms such as PapersWithCode[9] and sotabench[10] help compare State-of-the-art(SOTA) methods from various research papers and link source code to many SOTA ML techniques. Even

[9]https://paperswithcode.com

[10]https://sotabench.com/

	Recall@K			NDCG	
Model	K=1	K=5	K=10	ROUGE-L	SPICE
			1K Test Set		
VSE0 [1]	43.7	79.4	89.7	0.702	0.616
VSE++ [1]	52.0	84.3	92.0	0.712	0.617
VSRN [7]	60.8	88.4	94.1	0.723	0.620
TERN (Ours)	51.9	85.6	93.6	**0.725**	**0.653**
			5K Test Set		
VSE0 [1]	22.0	50.2	64.2	0.633	0.549
VSE++ [1]	30.3	59.4	72.4	0.656	0.577
VSRN [7]	37.9	68.5	79.4	**0.676**	0.596
TERN (Ours)	28.7	59.7	72.7	0.6645	**0.600**

TABLE I
IMAGE RETRIEVAL RESULTS ON THE MS-COCO DATASET, ON 1K AND 5K TEST SETS, FOR BOTH THE RECALL@K AND THE INTRODUCED NDCG METRICS.

Source: From Messina et al. 2020

Figure 6. A table showing comparison of methods.

though these platforms host parsed information from raw research about comparsions with SOTA methods, there is no way to access that information when reading a research paper online.

1.3 Contributions

There are two main contributions of this thesis. This thesis's first contribution is that it proposes a system which collects, parses and stores data about research papers

from computer science ArXiv. The proposed system stores the research paper's parsed content structure, tables, and citations. This system then uses this data to power various interfaces such as a search interface or a browser extension.

This thesis's second contribution is that it proposes and analyzes various ML models which can be formulated to classify tables describing comparisons using the data collected by the system. The disseration further shows the boost in classification performance when the models undergo pretraining with the large amounts of tabular data collected by the system.

1.4 Proposed Approach And Overview

This dissertation focuses on the solutions developed to tackle the missing characteristics described in Section 1.2.

Section 2.1 provides some preliminary background on inverted-index based search engines, open-source search engines, and state-of-the-art machine learning techniques. Section 2.2 discusses the different studies conducted about academic search engines; Section 2.3 discusses the different techniques used for mining academic literature; Section 2.4 discusses previous research regarding table type classification.

Section 3 provides an overview of the proposed solutions i.e., a search engine over CS ArXiv and a browser extension to augment research at the time of reading.

Section 4 discusses the core components of the search engine and the browser extension; Section 5 discusses the problem, solution and experiments for the machine learning task of classifying table of comparisons.

Finally, Section 6 will provide some insight to the current shortcomings of Sci-Genie, highlight some future directions of research based on the shortcomings of Sci-Genie.

Chapter 2

RELATED WORKS

2.1 Background Literature

2.1.1 Lucene Based Inverted Search Index

An inverted index lists every unique word in any document and identifies all the documents each word occurs in. Since the late 1990s, search engines have been using an inverted index as the core structure to store and access information; Even Google, in its earlier stages, leveraged inverted indexes as means of accessing text documents for search(Brin and Page 1998). Sergey Brin and Larry Page identified three critical tasks for web search: Parsing, Indexing, and Ranking.

Opensource search engines such as the Apache Lucene(Lucene 2010) help provide hooks to index and rank documents for search. Lucene has become a popular search engine upon which many other search engines are built to enrich functionality. Solr[11], Elasticsearch[12] as some opensource search engines built on top of Lucene.

2.1.1.1 Indexing With Lucene

As Lucene leverages an inverted index to store information, it defines two fundamental units for indexing:

[11]https://solr.apache.org/

[12]https://www.elastic.co/elasticsearch/

- Document : A container for various Fields
- Field : Stores the terms that need to be indexed. It is key-value form of mapping.

The Document allows for storing large hierarchical tree-like structures when indexing JSON objects. Every index consists of Documents. The index is similar to Database in the language of RDBMS.

2.1.2 Transformers and Machine Learning SOTA

The Transformer model (Vaswani et al. 2017) has recently proven to show SOTA performance on various domains ranging from natural language (Brown et al. 2020), vision(Radford et al. 2021, Dosovitskiy et al. 2020), music (Huang et al. 2018), image-generation (Ramesh et al. 2021) and even Web Table mining (Deng et al. 2020). These models can be trained with large amounts of data using self supervised training objectives (Chen et al. 2020), (Kolesnikov, Zhai, and Beyer 2019), (Goyal et al. 2019), (Gidaris, Singh, and Komodakis 2018), (Doersch, Gupta, and Efros 2015). Once trained, these models can transfer their learning to different downstream tasks by making minor adjustments to their weights (Howard and Ruder 2018) based on the task.

2.1.2.1 Transformer Architecture

The vanilla Transformer architecture (Vaswani et al. 2017) was created for machine translation. The model consists of an Encoder-Decoder Architecture using self-attention/cross-attention layers. The input to the transformer requires a sequence of

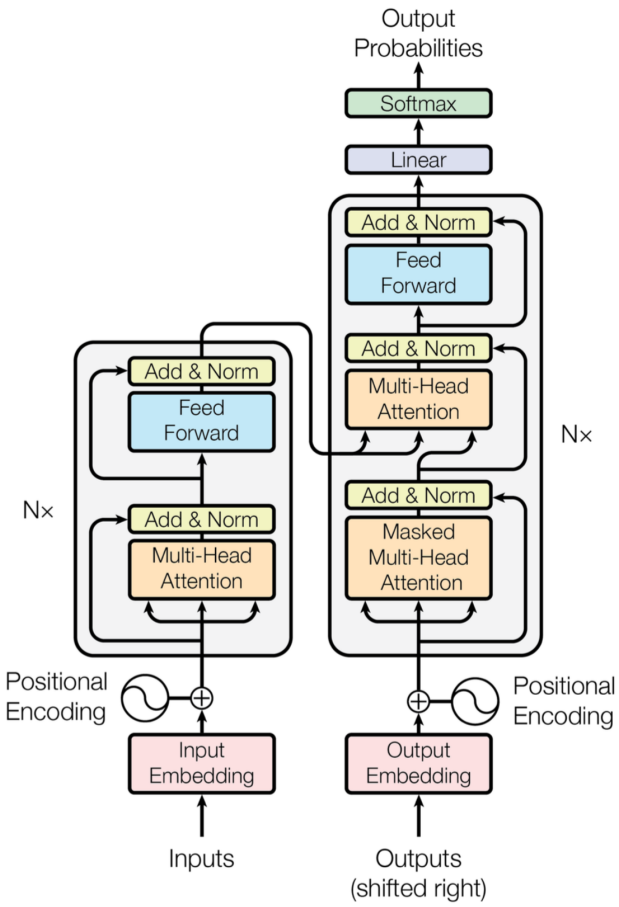


Figure 1: The Transformer - model architecture.

Source: From Vaswani et al. 2017

Figure 7. Transformer Architecture.

input and target symbols. The sequence of input and target symbols are created via a word piece tokenizer. The tokenized sequences are transformed into sequences of embeddings. Positional embeddings are added to the sequences, and then they are passed through either the encoder or decoder layer.

The self-attention/cross-attention layers consist of a multiheaded self-attention/cross-attention operation followed by pointwise feedforward layers with batch normalization (Ioffe and Szegedy 2015) and residual connections (He et al. 2016).

Multiple such layers are stacked together to create an encoder or decoder of the transformer.

The encoder layer encodes a sequence into latent representations using the self-attention operation; the decoder layer decodes the latent representation from the encoder with the cross attention for language translation. Figure 7 visualizes the Transformer architecture.

Multiple variants of this architecture have been created such as the ones by Radford et al. 2019, Devlin et al. 2018. The key insight behind the attention operation is provided in Section 2.1.2.1.1, Section 2.1.2.1.2,

2.1.2.1.1 Self-Attention

The self attention operation operates on a sequence $S \in \mathbb{R}^{s \times d}$ of s length and d dimensions. The self attention operation comprises three key components. A key matrix K, A query matrix Q, and value matrix V;each created from a linear transformation of S.

$$A = softmax(\frac{QK^T}{\sqrt{d}})V$$

The $softmax(\frac{QK^T}{\sqrt{d}})$ operation creates a $s \times s$ dimensional matrix. Element i, j in the matrix correspond to the weight of index i in the sequence S to index j in sequence S. The operation is called self attention because the key matrix and the query matrix are derived from same sequence. The operation helps derive relationships between items in the sequences at each attention head in each layer.

2.1.2.1.2 Cross Attention

Self attention operation applies the attention operation over the same sequence; Cross attention (Tsai et al. 2019) operation applies the attention operation over different sequences. The cross attention operation operates on sequences $S_\alpha \in \mathbb{R}^{s \times d}$ and $S_\beta \in \mathbb{R}^{k \times d}$ where S_α is of s length and d dimensions and S_β is of k length and d dimensions. The query matrix Q_α is created from a linear transformation of S_α and key matrix K_β, value matrix V_β are created from a linear transformation of S_β.

$$A_\alpha = softmax(\frac{Q_\alpha K_\beta^T}{\sqrt{d}})V_\beta$$

The $softmax(\frac{Q_\alpha K_\beta^T}{\sqrt{d}})$ operation creates a $s \times k$ dimensional matrix. Element i, j in the matrix correspond to the weight of index i in the sequence S_α to index j in sequence S_β. This operation is useful in the decoder layer for language translatation where the attention operation is helping correlate relationships between source language S_α with target language S_β.

2.1.2.2 Pretraining and Self Supervised Learning

In NLP, Transformer models possess properties to scale and learn with massive amounts of data using generalized tasks such as language modeling or masked language modeling. The objective of language modeling is to predict the token $n + 1$ given n tokens in the sequence. The masked language modeling objective is to predict a masked out token in the sequence correctly given the other tokens. These training objectives allow the usage of large amounts of data from the web without going

through the effort to clean the data. Recent State-of-the-art Language models such as GPT-3 (Brown et al. 2020) are trained using such self-supervised training objectives.

2.1.2.3 Advantages To Transfer Learning After Pretraining Models

Recent studies such as the one by Hernandez et al. 2021, showed that transformer models pretrained on large data with self-supervised learning objectives transfer learn better to downstream tasks with much less labeled data. The finetuning step(Howard and Ruder 2018) of transfer learning involves adding a new linear layer and then training the model with small learning rates for the task-specific optimization.

2.1.2.4 Machine Learning For Tabular Data

Deng et al. 2020 applied structurally aware Transformers with self-supervised learning for web table data. The model is structurally aware because of the usage of cell specific entities and row/column specific embeddings in the pretraining and finetuning tasks. Figure 8 shows the input representation and the structural embeddings for the representations from the model by Deng et al. 2020. The structural embeddings include cell specific type, and row/column positional embeddings.

Deng et al. 2020 is also the first application of pretraining/finetuning based transfer learning paradigm in the domain of Web Tables.

The learned models possessed rich generalization capabilities for transfer into downstream tasks such as row filling based on headers, Entity Linking to the knowledge base, column type annotation etc.

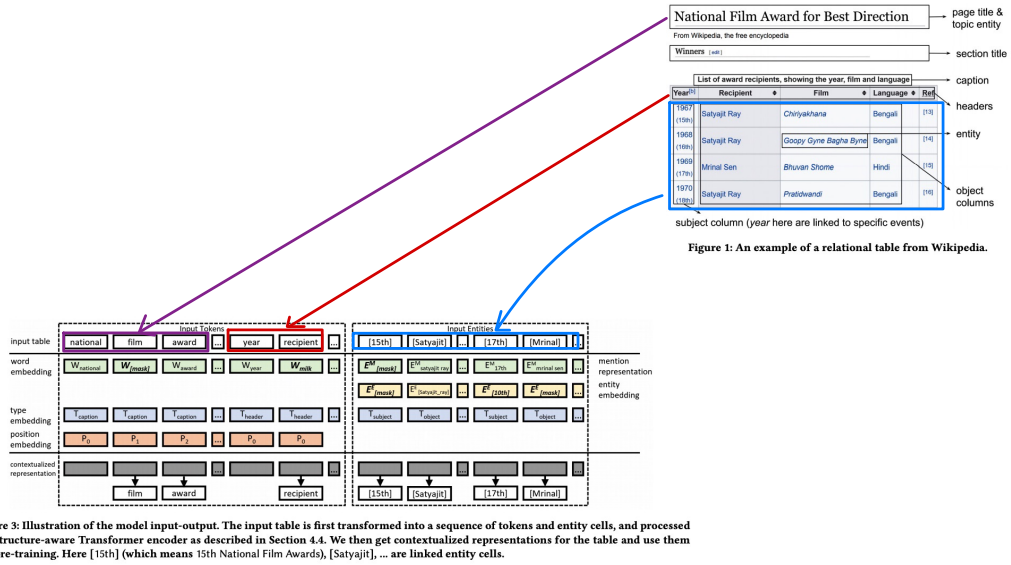


Figure 1: An example of a relational table from Wikipedia.

Figure 3: Illustration of the model input-output. The input table is first transformed into a sequence of tokens and entity cells, and processed for structure-aware Transformer encoder as described in Section 4.4. We then get contextualized representations for the table and use them for pre-training. Here [15th] **(which means** 15th National Film Awards), [Satyajit], **... are linked entity cells.**

Source: From Deng et al. 2020

Figure 8. TURL Architecture and Input Representations correlated.

2.2 Academic Research Search Engines

Gusenbauer and Haddaway 2020 conducted an analysis of search engines based on criteria described in Table 2.2, Table 2, Table 3. The criteria described by Gusenbauer and Haddaway 2020 try to objectively evaluate the usefulness of a search engine to do systematic reviews.

Li, Schijvenaars, and Rijke 2017 performed a detailed analysis on the patterns and failures in search for academic search. Their analysis concluded that search queries in academic search had substantially more entity-based queries than web search and null queries attributed to a significant fraction of failures, i.e., search engine found no

Criteria	Meaning
Subject coverage	This criteria determines whether a search system is multidisciplinary or not.
Size	This criteria assesses the number records in the search engine's database.
Record type	This criteria assesses the types of records offered by a search system like journal papers/books etc.
Retrospective coverage	This criteria assesses the timeperiod of the oldest records is the search engine.
Open Access	This criteria assesses the usage rights of the records offered. This criteria helps contrast between search engines that offer proprietary content or open access content.
Controlled Vocabulary	This criteria assesses if controlled search options provided by the search system.
Field Code Search: Query Refinement	Field Code Search if a criteria that assesses if the search engine offers additional filters to improve search with high precision and recall. Field Code can be attribute on which search can be targeted like title/abstract/keywords etc.

Source: From Gusenbauer and Haddaway 2020

Table 1. Table explaining various criteria For Comparing Search Engines.

search results. This analysis is quite helpful in correlating the need for a "Controlled Vocabulary" (Table 2.2) in search to improve precision.

Kacem and Mayr 2018 analyzed the improvement of search precision based on the usage of *search stratagems*. Search stratagems are additional filters along with the search terms. They provide further context around filtering a search query. Some examples of such filters can Authors of a paper, Forward Citation Search, Footnote Chasing, etc. The study by Kacem and Mayr 2018 concluded that search stratagems could improve search precision for the search queries.

Rovira et al. 2019 conducted a study to analyze the impact of citation count on ASEO(Academic Search Engine Optimization). Their study concluded that Microsoft Academic and Google Scholar seem to leverage received citations as a part of the

Criteria	**Meaning**
Maximum Search String Length	This criteria assesses the size of the longest search query.
Search String Language Support	This criteria assesses tested search systems for support of other languages.
Boolean Functionality	Boolean operators such as AND/OR/NOT etc. can be useful in improving search precision for many search queries. This criteria assessed weather search engines possessed such functionality or not.
Literal vs Expanded Queries	This criteria assesses if a search system automatically expands queries impacting on precision and recall.
Truncation/Wildcards	This criteria assesses if different frequently used truncation or wildcard symbols were functional.
Exact Phrase Search	This criteria assesses if the use of quotation marks—symbols typically used to deem an expression should be searched literally—would result in fewer results than for terms lacking them.
Parentheses	This criteria assesses if parentheses can be used to group concepts in queries.

Source: From Gusenbauer and Haddaway 2020

Table 2. Table explaining various criteria for comparing search engines.

relevance ranking algorithm. This study also noted many unknown attributes in the relevance ranking algorithms for Google Scholar and Microsoft Academic.

Criteria	Meaning
Filtering: Post-Query Refinement	This criteria assesses a search system's capacity for post-query refinement through a so-called faceted search.
Forward Citation Search	This criteria assesses a search system's capacity for forward citation search. Forward citation search allows search to be constrained to the citations of a paper.
Advanced Search String Input Field	This criteria assesses if the search engine provide advanced features along with simple string search to filter content.
Search Help	This criteria assesses if the search engine offers documented help to support the access to the search engine.
Maximum Number of Accessible Hits	This criteria assesses the maximum number of hits made accessible by the search system with a single search.
Bulk Download Supported	This criteria assesses if the search engine allows bulk download.
Reproducibility of Search Results at Different Times	This criteria assesses if the search queries show signs of bias.
Reproducibility of Search Results at Different Locations	This criteria assesses whether the changes in the place from which searches were performed influenced the search results.

Source: From Gusenbauer and Haddaway 2020

Table 3. Table explaining various criteria for comparing search engines.

2.3 Academic Research Literature Mining

Academic Literature mining has many applications such as document structure analysis, citation analysis, document summarization, table extraction, and more. Deep learning-based methods dominate many of the recent SOTA academic literature mining methods.

Beltagy, Lo, and Cohan 2019 created Sci-Bert by training a BERT model (Devlin et al. 2018) using a large corpus of research data. Sci-Bert can be finetuned for

downstream tasks such as citation intent detection, relation extraction, and even named entity detection.

Safder et al. 2020 leveraged deep learning techniques extract metadata from research documents. Zha et al. 2019 proposed methods to track the evolution of algorithms and techniques to make research more understandable. Salatino et al. 2019 developed ontology detection models for Computer science literature. The model can classify ontology from 14K topics.

Table extraction and parsing have also seen a lot of recent advancements. Methods either work on PDF-based documents or LaTeX based documents. Some such advances can include Milosevic et al. 2019's framework for extracting tables from biomedical literature articles or Kardas et al. 2020's methods for extracting tables describing SOTA comparisons from machine learning papers.

All of these advances are because of curation and opensource availability to many datasets some of such being Arxiv Tables[13] or Semantic Scholar Open Research Corpus(Ammar et al. 2018) or the Sci-Cite dataset (Cohan et al. 2019) or the PaperWithCode datasets[14].

2.4 Types Of Tables in Scientific Research

Tables in scientific research are different from web tables. Web-Tables can generally have well-defined entities and type of information being described by a table can have a much broader scope. Scientific research tables on the other hand can be categorized under specific types.

[13]http://boston.lti.cs.cmu.edu/eager/table-arxiv/

[14]https://paperswithcode.com/

Earlier work done by Kim et al. 2012 defined the types of tables based on their content and purpose. Kim et al. 2012 sampled 25 papers from the domains of CS, Biomedical, Life Science, Chemistry, Material Science, Electrical Engineering and Medicine. From these domains, Kim et al. 2012 classified the following types of tables for scientific documents:

- **Definition Table** : *These tables describe equation definitions or symbol definitions.*
- **Statistics Table** : *These tables describe some statistical distribution which is not related to experiment results.*
- **Survey Table** : *These tables describe questionaire surveys and results.*
- **Example Table** : *These tables provide emphasis of something that needs to be clearly explained.*
- **Procedure Table** : *These tables describe a flow or a process.*
- **Experiments Setting Table** : *These tables describe some experiment parameters.*
- **Experiments Results Table** : *These tables describe the results of some experiments discussed in the paper.*

Kim et al. 2012 also found that most of the tables in scientific documents describe results from experiments based on the samples they annotated.

Chapter 3

PROPOSED SOLUTIONS

3.1 Sci-Genie : Search Engine Harnessing Structure Of Scientific Research.

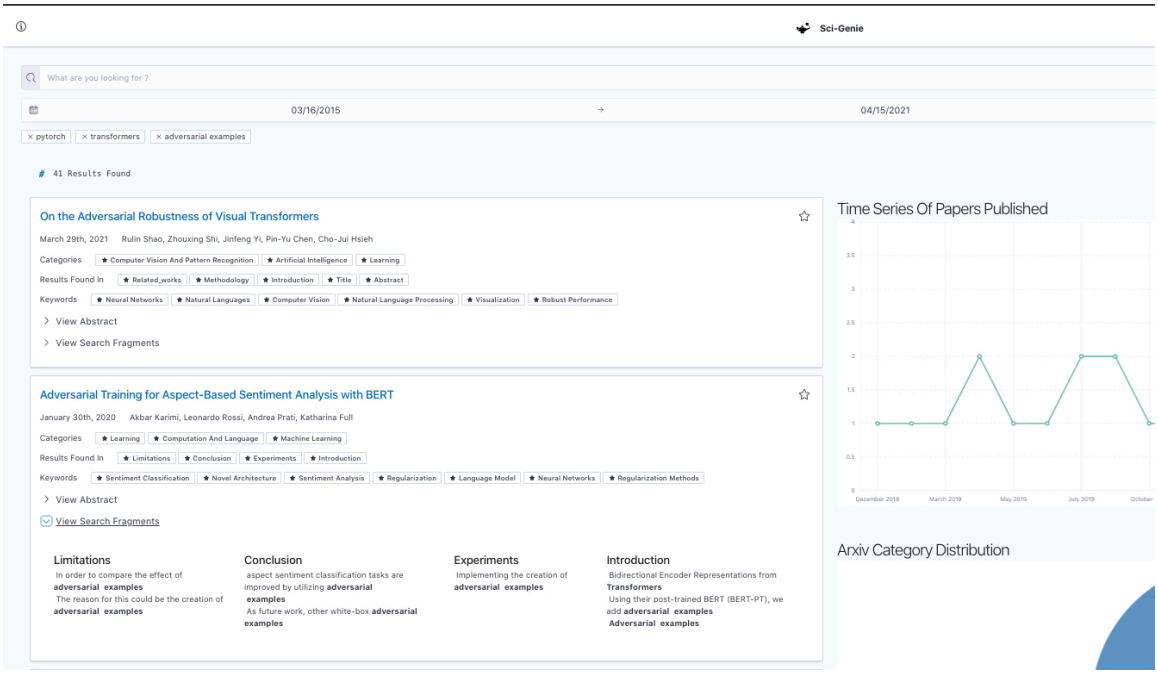


Figure 9. Search Results With Context For Query *pytorch, transformer, adversarial examples* Using Sci-Genie.

Sci-Genie is a prototype search engine over CS ArXiv. Sci-Genie indexes the hierarchical structure of Computer Science research papers to provide more context to search results fragments. Sci-Genie's content mining engine also classifies a paper's ontology using a SOTA ontology classification model developed by Salatino et al. 2019. The mined ontology helps create a "Controlled Vocabulary" for filtering search results.

3.2 Sci-Genie-Extension: In-Browser Extension To Augment Information About Research.

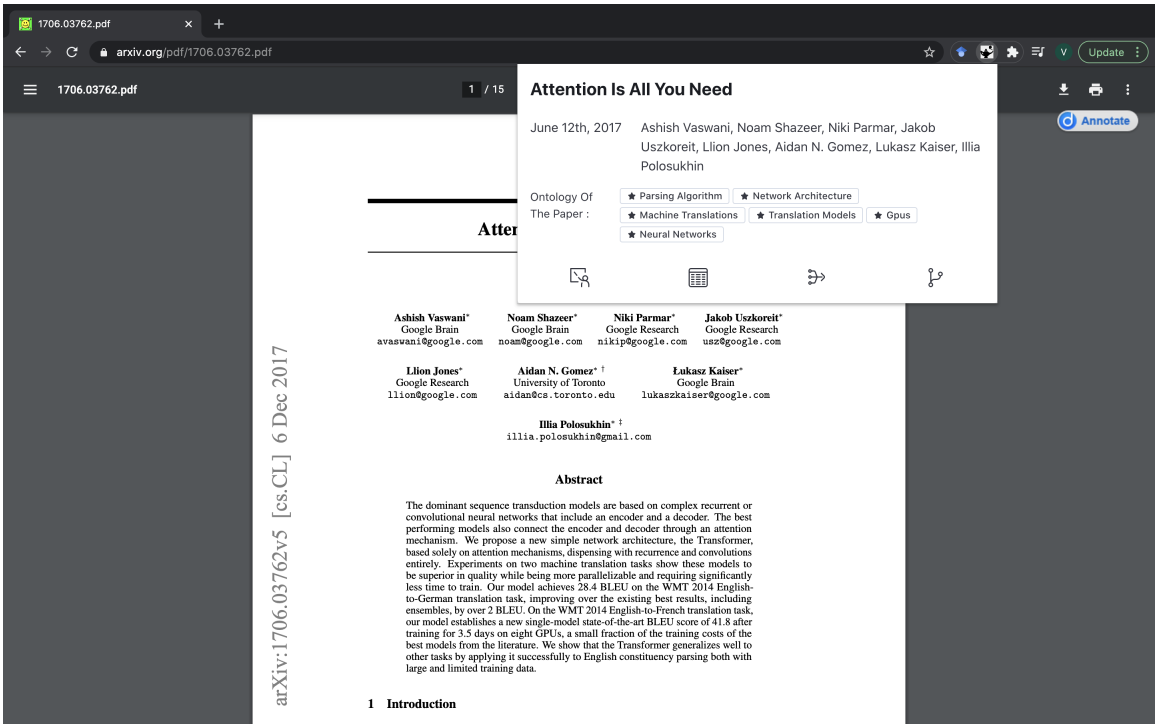


Figure 10. Sci-Genie Browser Plugin

Sci-Genie, at its core, holds research from CS ArXiv, Tables from the paper mined from CS ArXiv and 6.5M papers filtered from the Semantic Scholar Open Research Corpus(Ammar et al. 2018). The Semantic Scholar Open Research Corpus helps create a citation graph supporting information augmentation for the browser plugin.

The citation graph helps surface context-relevant tables from papers cited by the paper the user is reading. The search engine and the citation graph help extract other papers by the same authors and papers citing/cited by the paper the user

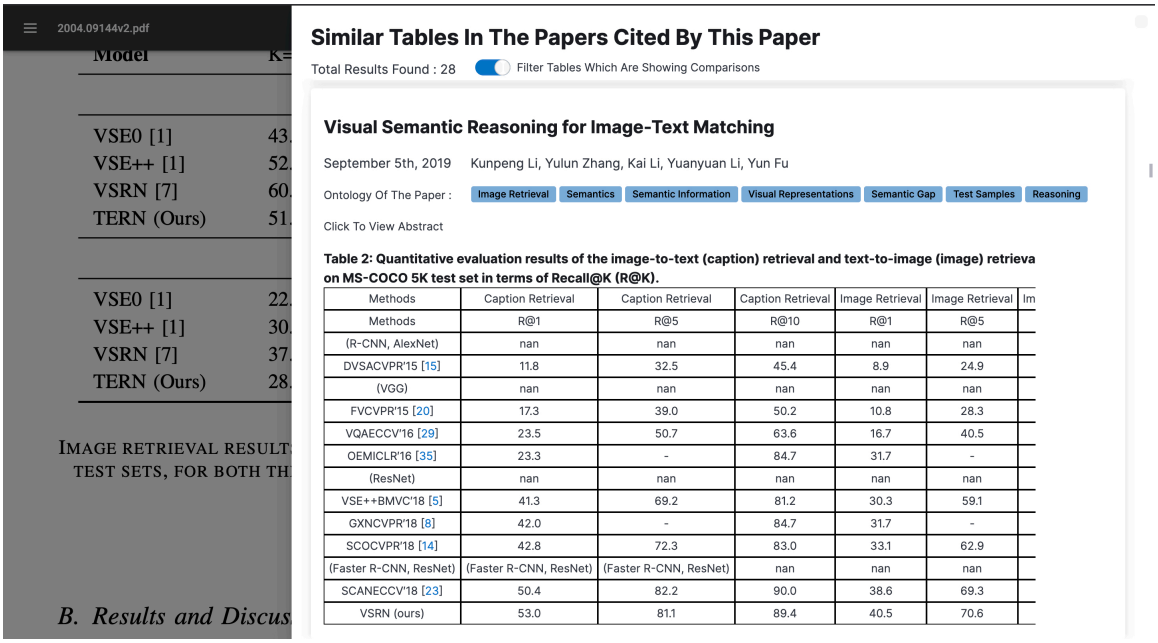


Figure 11. Sci-Genie Browser Plugin surfacing information about tables comparing entities from cited papers.

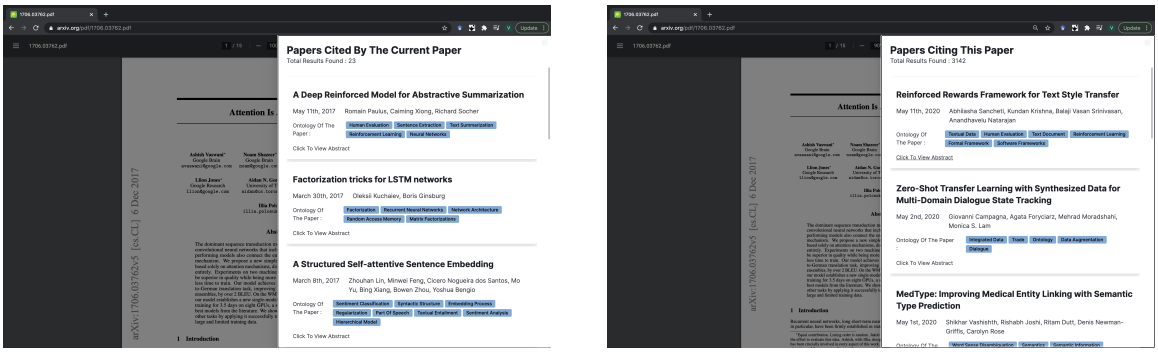


Figure 12. Sci-Genie Browser Plugin surfacing context information about papers citing/cited-by the current paper.

reads. Figure 10,12, shows an example of Sci-Genie extension in action. The browser extension also contains filters for filtering tables comparing entities using a machine learning model. All information provided by the browser extension helps enhance the researcher's understanding without the researcher having to dig up the information independently. Figure 11 is am example of the Sci-Genie extension filtering tables of

comparisons. The filtered tables in Figure 11 are dervied from citations of Messina et al. 2020.

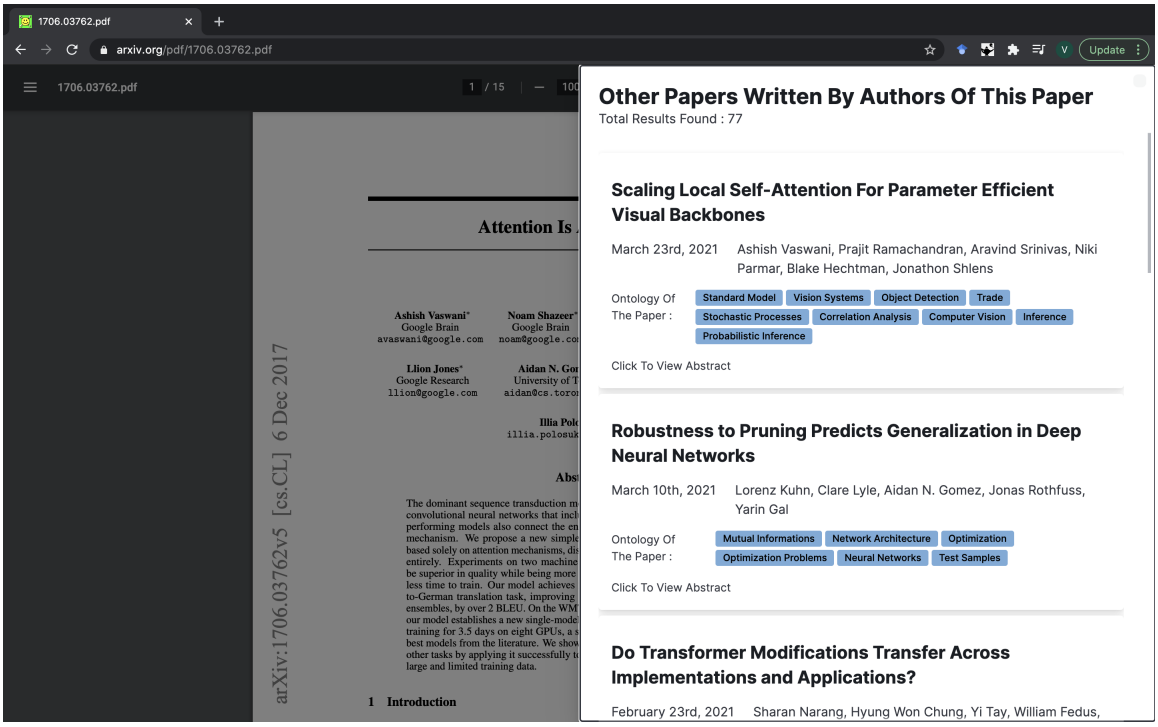


Figure 13. Sci-Genie Browser Plugin surfacing information about other papers from the authors of the paper the user is reading.

3.2.1 Design Choices

Sci-Genie uses ArXiv as a source for papers because ArXiv provides the availability of LaTeX sources. LaTeX based sources are easier to parse compared to PDF documents because of LaTeX's compilability. As LaTeX is Turing complete, the compiled content's parse tree is easily accessible for further downstream processing. Parsing PDFs is beyond the scope of this dissertation.

Chapter 4

SCI-GENIE CORE

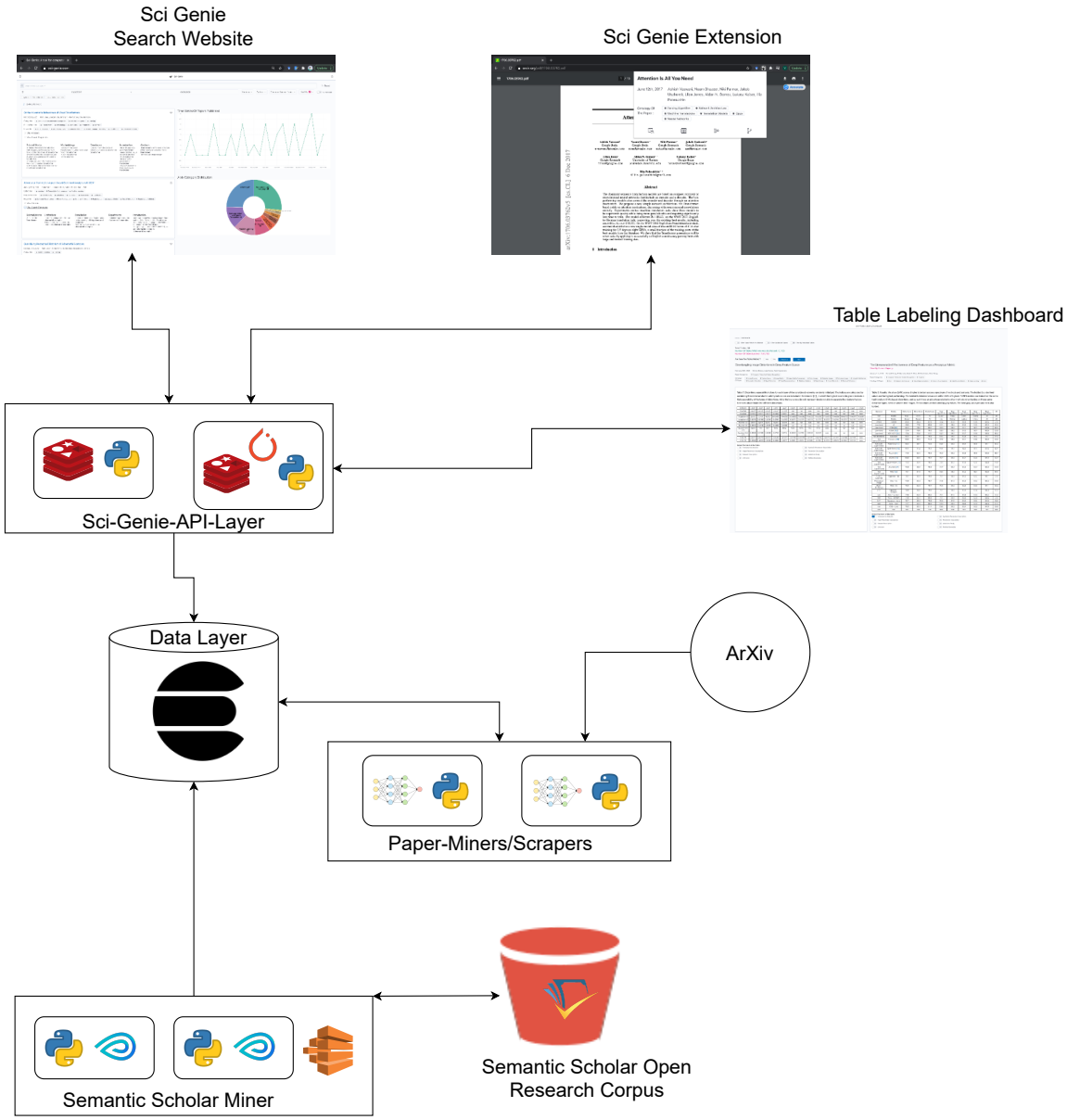


Figure 14. Sci-Genie System Architecture.

4.1 System Architecture

Sci-Genie Core (Figure 14) comprises four key architectural components which help power information for multiple different apps like the browser-extension, the website and the labeling interfaces for machine learning.

- **Data Layer**
- **API Layer**
- **Paper Scraping And Mining Engine**
- **Semantic Scholar Citation Graph Mining Engine**

Paper Scraping And Mining Engine help extract papers from ArXiv in LaTeX format and then parses the structure, mines the ontology, and extracts the tables. The processed information gets stored in the Data Layer.

The Data Layer hosts the information used by the website or the browser extension.

The Semantic Scholar Citation Graph Mining Engine leverages a data processing pipeline to filter CS research and create an index in the Data Layer, containing information corresponding to citations of papers from ArXiv.

The API Layer leverage the general-purpose abstractions built around the Data Layer to access and further intelligently process information at search time. The API Layer connects the Data Layer with various applications like The Sci-Genie Website, The Sci-Genie extension and other apps such as the Table Labeling tool.

4.2 Sci-Genie Data Layer

The Data Layer comprises the open-source search engine named Elasticsearch(Gormley and Tong 2015) which is built on top of Lucene. The data layer aims

to host processed information after scraping, mining, and parsing research documents and tables. Elasticsearch hosts the following indexes[15,16] to filter access of content:

- **Arxiv Parsed Research Index**:*Contains research documents whose structure is explicitly parsed and stored.*
- **Semantic Scholar Index**:*Contains documents from the CS Domains mined and filtered from semantic scholar open research corpus.*
- **Arxiv Table Index**:*stores the tables from research papers. These can be later used for extending context during reading.*
- **Arxiv Semantic Scholar Join Index**:*Stores the citation information filtered from semantic scholar index to map papers in ArXiv.*

4.2.1 Arxiv Parsed Research Index

Documents in the Arxiv Parsed Research Index helps power the search for the Sci-Genie search website. All documents belong to Computer Science ArXiv and the construction of these documents is discussed in Section 4.3. Each document consists of three key pieces of information:

- **ArxivIdentity** *ArXiv associated meta-data information.*
- **ResearchObject** *Parsed Research structure.*
- **Ontology** *Ontology belonging to the article.*

[15]Index is similar to Database in the language of RDBMS

[16]https://www.elastic.co/guide/en/elasticsearch/reference/current/documents-indices.html

4.2.1.1 ArxivIdentity

ArXiv associated meta-data information holds information regarding the abstract, title, authors, categories of the paper etc. The information stored in this object helps provide search stratagems to the Sci-Genie search website.

4.2.1.2 ResearchObject

The ResearchObject is the data structure that holds the parsed text from the research document. The purpose of this object is to fit the research document and its hierarchy into predefined sections which are commonly occurring in research documents. The general pre-identified sections are given below:

- Introduction
- Related Works
- Methodology
- Experiments
- Dataset
- Conclusion
- Limitations

Any section that doesn't fit the predefined sections will be categorized as *Unknown*.

The ResearchObject consists of key value pairs which relate to the given predefined sections. The key is the name of the section eg. Introduction, Related Works etc. and the value is a *Fragment*. Fragment is a tree-like data structure that holds hierarchical information. Fragment is given by:

```
class Fragment:
    title:str = "Introduction"
    text:str = "Text relating to the introduction of a paper"
    children:List[Fragment]
```

The *children* in the Fragment hold the information about the child notes of that Fragment. The Fragment object helps capture a research paper's hierarchy before it gets parsed into a key-value-based ResearchObject. The Fragment object can also be serialized to JSON making it indexable in the Lucene search index.

Due to the search engine's scope being limited to CS research on ArXiv, the predefined sections are created from observing writing patterns of research in the CS domain. Section 4.3 provides more details on the creation of the Fragment and ResearchObject.

4.2.1.2.1 Advantages Of Indexing ResearchObject

The ResearchObject is a key value-based object, and each key allows indexing the text according to its corresponding section. This object schema helps provide more context about the search match within each section, as seen from Figure 9.

4.2.1.3 Ontology

The study conducted by Li, Schijvenaars, and Rijke 2017 gave a great indicator regarding the need for entity based queries in academic research search engines to improve precision. As Sci-Genie only operates over computer science research, the

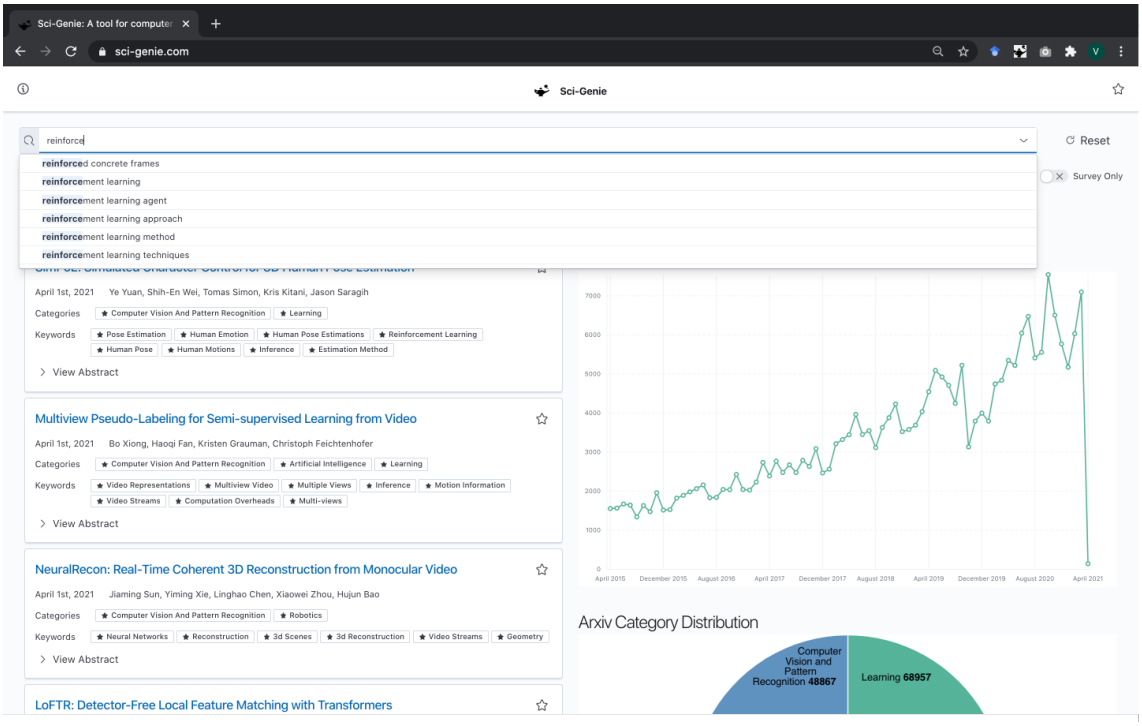


Figure 15. Sci-Genie auto-completing the search terms using the ontology.

research papers' ontology helps power the autocomplete panel in the search bar from Figure 13. Each paper consists of an ontology which is classified via CS specific ontology classifier.[17]. Sci-Genie indexes the entire ontology given by Salatino et al. 2019.

4.2.2 Semantic Scholar Index

This index hosts information filtered from the Semantic Scholar Open Research Corpus(Ammar et al. 2018) post Citation Graph Mining. More details on how it is filtered is described in Section 4.4. This index's core purpose is to help correlate citation to and from papers on ArXiv. The Sci-Genie extension further uses this

[17]https://github.com/angelosalatino/cso-classifier#sample-output-sp

citation information for finding tables from the neighborhood of papers as seen in Figure 11.

4.2.3 Arxiv Table Index

This index holds information regarding the tables from a research paper. The core methodology which is used to extract and serialize/deserialize tables from LateX is based the methodology developed by Kardas et al. 2020. The method created by Kardas et al. 2020 helps convert tables in pandas[18] serializable DataFrames. These DataFrames can be serialized to json for storage in Elasticsearch. The schema for the object can seen below :

```
class ArxivTable:
    semantic_scholar_id:str = ""
    arxiv_id:str=""
    identity:str = ""
    figure_id:str=""
    caption:str=""
    name:str=""
    layout:str=""
    table:str=""
```

The *identity* is the unique identifier for a table. Each record also holds loose relations to the Arxiv Parsed Research Index and the Semantic Scholar Index via respective ids.

[18]https://pandas.pydata.org/

4.2.4 Arxiv Semantic Scholar Join Index

This index holds references of in and out citations for each ArXiv article present in the Arxiv Parsed Research Index. Although the Semantic Scholar Index contains citation data, none of it is correlated with ArXiv articles and hence this index is created to help correlate information at search time. Below is the schema for the documents in the index:

```
    {
        "in_citn_arxiv": [],
        "out_citn_arxiv": [],
        "semantic_scholar_id": "",
        "arxiv_id": "",
}
```

The *arxiv_id* is the unique ArXiv identifier, The *in_citn_arxiv* correspond to the unique ArXiv ids of papers that cited a paper. The *out_citn_arxiv* correspond to the unique ArXiv ids of papers cited by a paper. This information is useful when filtering out tables as seen in Figure 11. The construction of this index is discussed in Section 4.4.

4.3 Paper Scraping And Mining Engine

The Paper Scraping And Mining Engine connects with ArXiv to extract LaTeX source data. The LaTeX source data undergoes three parallel processing steps:

- Content Hierarchy Parsing : The extraction of content based on document structure for storage.
- Ontology Mining : Correlating the ontology of the paper from ArXiv.
- Table Extraction : Leveraging tools to extract tables from the ArXiv document's LaTeX source.

4.3.1 Content Hierarchy Parsing

A structured ResearchObject is parsed from raw LateX content in two steps:

- Step 1 : Extract the structure of the research document and create a Fragment object.
- Step 2 : Correlate the children in the Fragment Object with predefined sections to create the ResearchObject.

4.3.1.1 Fragment Creation

Each LaTeX source repository is parsed to create a "Structure Tree" of the research document. The Structure tree[19] helps correlate the structure of latex document. The structure tree is then used to create Fragment objects described in Section 4.2.1.2. The text within each Fragment is populated by using the opendetex library[20]. The opendex library helps filter text information from individual tex files. The code to match the structure tree is provided in the Appendix.

[19]https://github.com/alvinwan/tex2py

[20]https://github.com/pkubowicz/opendetex

4.3.1.2 ResearchObject Creation From Fragment

The initial Fragment object created via the parsing described earlier will contain the entire research document stored in a single Fragment. This Fragment then undergoes a "heuristic" serialization to the ResearchObject. Meaning the title of the children within the Fragment is matched to predefined sections/classes given in Section 4.2.1.2. The matching process uses heuristics to match the title content. More details on the section name matching strings and algorithm are given in the Appendix A.2 and A.3 respectively.

4.3.2 Ontology Mining

Sci-Genie leverages the ontology mining method described by the Salatino et al. 2019. The research document's abstract and title are only needed for ontology mining. This ontology further helps provide realtime autocomplete filters around searching for research.

4.3.3 Table Extraction and Storage

To extract tables from papers on ArXiv, Sci-Genie uses arxiv-vanity/engrafo[21] to convert LaTeX to HTML and then employ table serialization methods created by Kardas et al. 2020. The parsed tables are stored in the Arxiv Table Index.

[21]https://github.com/arxiv-vanity/engrafo

4.4 Semantic Scholar Citation Graph Mining.

A rich source of context and metadata about the tables cited by a paper comes via the access to citations in a paper and leveraging the citation graph. The Semantic Scholar[22] corpus consists of 136M research papers from various domains along with their citations. This corpus event consists of all urls to a research paper, including the ones on ArXiv.

Metaflow[23] is used to create a data processing pipeline[24] for filtering the subset of CS papers from the Semantic Scholar corpus. The filtered CS papers are stored in the Semantic Scholar Index.

The pipeline further correlates the citations of papers which are present on ArXiv and stores them in the Arxiv Semantic Scholar Join Index.

4.5 API Layer

The API Layer leverages the general-purpose abstractions built over the DataLayer to access and process information for various apps such as the search engine website, the chrome plugin and the table labeling interface.

[22] http://s2-public-api-prod.us-west-2.elasticbeanstalk.com/corpus/

[23] https://metaflow.org/

[24] https://github.com/valayDave/semantic-scholar-data-pipeline

4.5.1 Scientific Table Correlation Via Citation Graphs

With the access to distinctly processed and correlated information about research papers on ArXiv and their associated tables, the API Layer leverages these traits to filter tables from papers that cited the paper the user is reading (as seen in Figure 11 and 12).

4.5.2 Table Of Comparison Classification

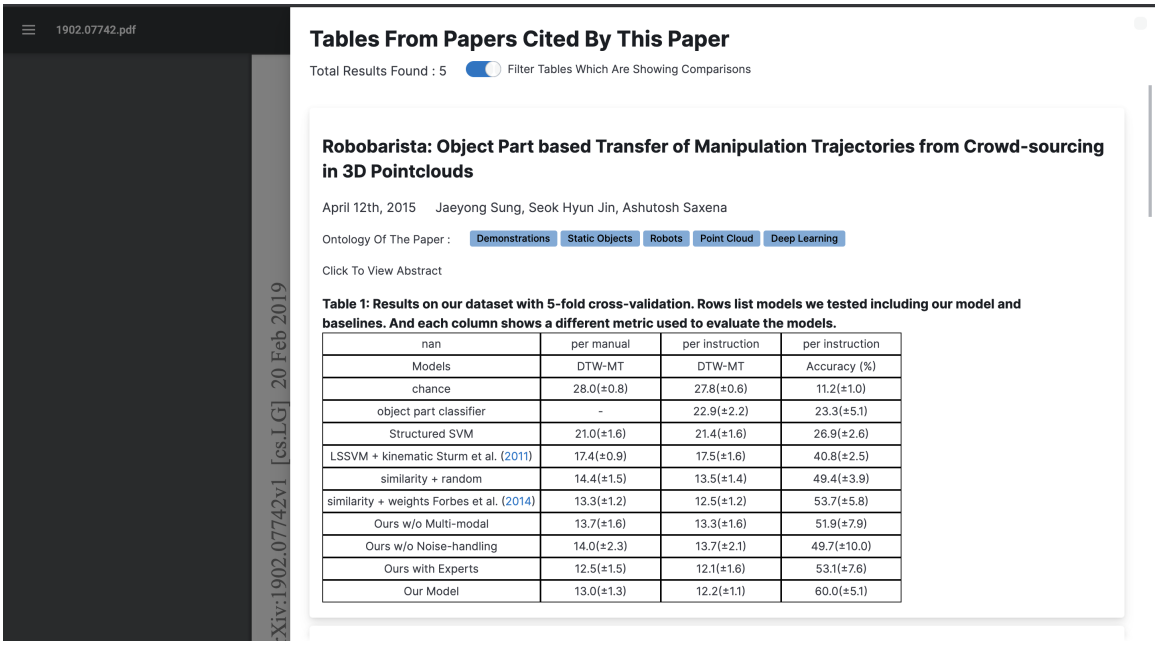


Figure 16. Tables describing comparisons filtered via API-Layer.

The tables extracted from the papers that were cited also undergo a binary classification via a machine learning model to label the tables describing a comparison. This information is useful to provide filters at the time of reading, as seen Figure

11. Chapter 5 provides more details on the table of comparison classification and the choice of model for the API-Layer.

4.5.3 Table Intent Labeling API

The API Layer also provides API's for table annotation and labeling which are used by the labeling interface. Figure 20 visualizes the interface for labeling. These annotations and labels help provide labeled data for the machine learning task which is discussed in Chapter 5.

Chapter 5

CLASSIFYING TABLES DESCRIBING COMPARISONS

A lot of valuable information about comparisons with some baselines and other research is present in the tables of academic research. Researchers tend to spend much valuable time finding the citations and the table of results from those citations. Sci-Genie via Citation Graph and the Machine Learning models filters tables describing comparisons at the time of reading research.

This chapter will cover the approaches applied to tackle the problem of classifying tables describing comparisons.

5.1 Problem Formulation

The structure of the tables in scientific research can fluctuate based on the information presented by the tables. The tables may also contain information whose context cannot be exactly inferred without the caption underneath the table. Tables occurring in scientific research can also be segregated into well-defined types such as the ones described in Chapter 2.4. These tables may also not contain a directly linkable entity from a knowledge base(KB) as entities defined in the research may not be present in the KB. Due to such different intrinsic characteristics, models trained using knowledge bases along with web tables such as the one by Deng et al. 2020 cannot be directly fit to solve this problem.

The problem of classifying a table T as a table describing a comparison can be framed as a binary classification problem. Given a table T and its caption C, the

Symbol	Meaning	Type
Y	Labels	Discrete : $\{0, 1\}$
X_T	Cell Embedding	$\mathbb{R}^{T_l \times d}$
X_C	Caption Embedding	$\mathbb{R}^{C_l \times d}$
P_C	Positional Embedding	$\mathbb{R}^{C_l \times d}$
E_r	Row Embedding	$\mathbb{R}^{T_l \times d}$
E_c	Column Embedding	$\mathbb{R}^{T_l \times d}$
T_l	Number Of Cells	$\mathbb{N}$
T_r	Number Of Rows	$\mathbb{N}$
T_c	Number Of Columns	$\mathbb{N}$
C_l	Length of Caption	$\mathbb{N}$
T_{cell}	Sequnces of Cells	$\mathbb{N}^{T_l}$
f_θ	Neural Network	-

Table 4. Table Of Symbols for Equations.

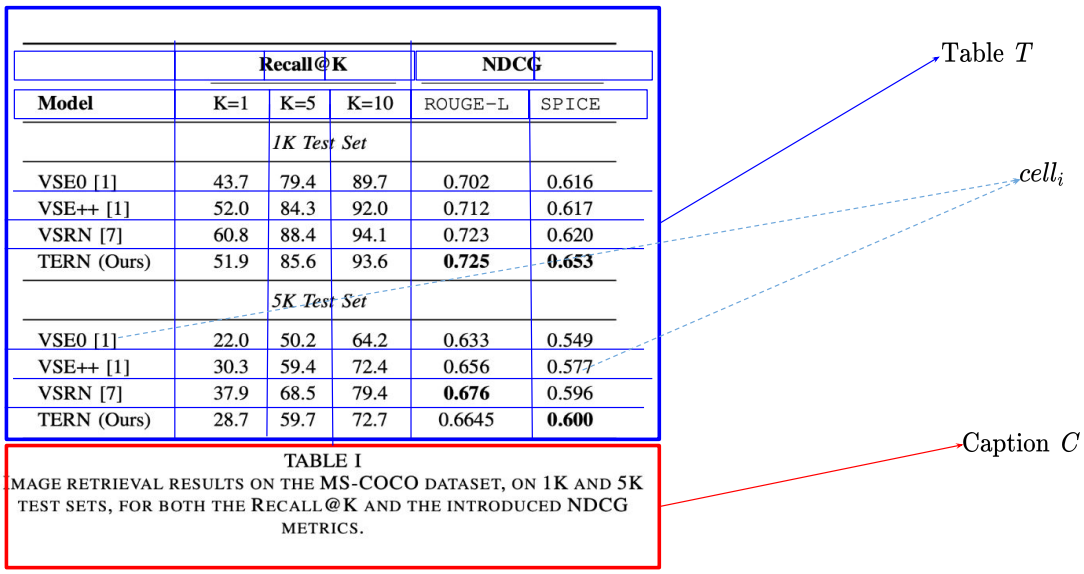
classifier $f_\theta(T, C) \rightarrow Y$ predicts a binary label $Y \in \{0, 1\}$ to denote weather the table is describing a comparison of entities[25].

Based on this formulation, this dissertation analyzes 4 different types of ML models and the representations they use for the table T and caption C for the problem:

- Cross Channel Transformers(CC-T) With/Without Pretraining
- Encoder Only Multi-Channel Transformer(E-MCT) With/Without Pretraining
- Caption only Encoder Transformer With/Without Pretraining
- Finetuning Pretrained Scibert (Baseline)
- SVM and Naive Bayes (Baseline)

Figure 17 visualizes the representation of T and C. Different models are analyzed because the representations of T and C can be treated as seperate variables or as a single variable, based on the choice of the model. Cross Channel Transformer and

[25]Both the table and caption are used because human labeling discovered that these two variables combined are the most prominant indicators of a table describing a comparison.



Model	Recall@K			NDCG	
	K=1	K=5	K=10	ROUGE-L	SPICE
		1K Test Set			
VSE0 [1]	43.7	79.4	89.7	0.702	0.616
VSE++ [1]	52.0	84.3	92.0	0.712	0.617
VSRN [7]	60.8	88.4	94.1	0.723	0.620
TERN (Ours)	51.9	85.6	93.6	**0.725**	**0.653**
		5K Test Set			
VSE0 [1]	22.0	50.2	64.2	0.633	0.549
VSE++ [1]	30.3	59.4	72.4	0.656	0.577
VSRN [7]	37.9	68.5	79.4	**0.676**	0.596
TERN (Ours)	28.7	59.7	72.7	0.6645	**0.600**

TABLE I
IMAGE RETRIEVAL RESULTS ON THE MS-COCO DATASET, ON 1K AND 5K TEST SETS, FOR BOTH THE RECALL@K AND THE INTRODUCED NDCG METRICS.

Figure 17. An example representation of Table T and Caption C. Each Table T consists of sequence of *cell*'s.

Encoder Only Multi-Channel Transformer treats T and C as separate representations. Scibert treats T and C as one joint representation. The caption only transformer only uses the Caption C. The caption only transformer is tested, to validate the hypothesis of wheather only captions themselves can be enough for classifying tables describing comparisons.

Section 5.2 describes the models developed/trained for the input representations of table T and caption C. The data collection and labeling for the machine learning models is described in Section 5.3. The experiment results for different models is described in Section 5.4

5.2 Classification Models

5.2.1 Encoder Only Multi-Channel Transformer (E-MCT)

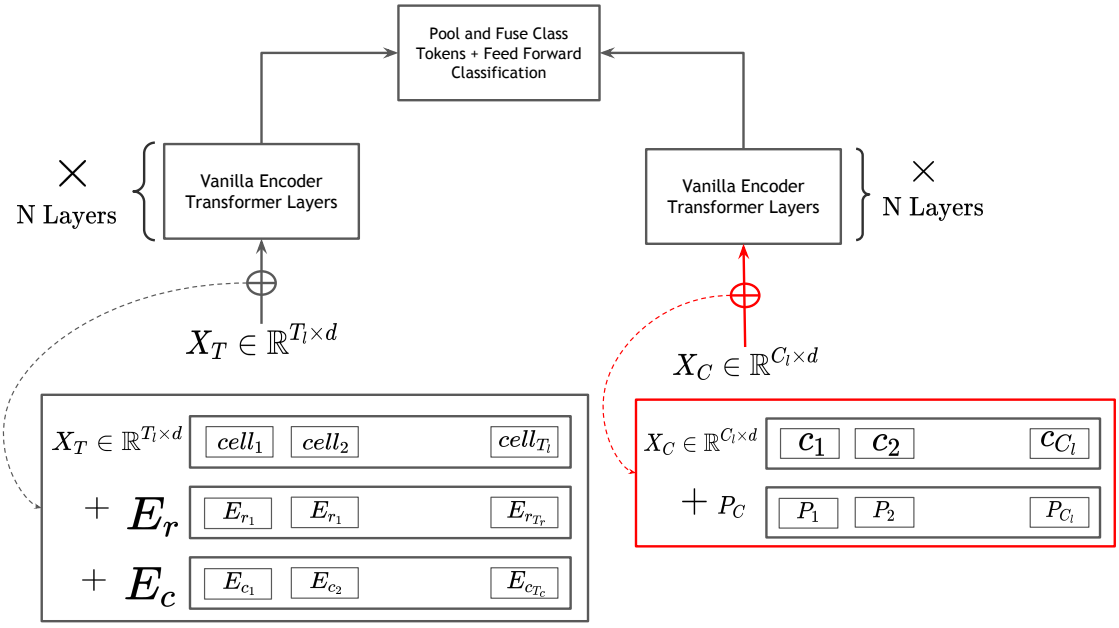


Figure 18. Encoder Only Multi-Channel Transformer For Table Classification.

Based on the models developed by Deng et al. 2020. A structurally-aware encoder only Transformer model is adapted to fit the function f_θ. Figure 18 visualizes the model and its input representations.

5.2.1.1 Input Representation

Each table T is represented as tuple of (T_{cell}, E_r, E_c); where $T_{cell} \in \mathbb{N}^{T_l}$ are the cells of the table represented as a sequence of integer tokens; E_r is the row embedding where embedding E_{r_i} represents the row of the cell T_{cell_i} as an embedding; E_c is the

column embedding where an embedding E_{c_i} represents the column of the cell T_{cell_i} as an embedding. The caption C is represented as sequence of integer tokens that gets transformed to a sequence of embeddings. The representations for the table T are inspired from the models created by Deng et al. 2020.

The cell sequence tokens T_{cell} and caption C are created using a word-piece tokenizer from the SciBert Model [26]. The tokenizer converts the value of individual cells or the caption into a sequence of integer tokens. These tokens can be used to create embeddings which are passed to the transformer model.

5.2.1.2 Model Description

The table cell sequence T_{cell} and the Caption C are first converted to embeddings X_T, X_C respectively. Post creation of the embeddings, the caption embedding X_C is summed with the position embedding P_C; The cell embeddings X_T are summed with E_r, E_c. After the summings of the respective embeddings, differentiable CLS tokens are concatenated to X_T, X_C.

These embeddings are passed to individual encoder self-attention transformers. The class tokens from the final layer is pooled and sent for classification using a linear feed forward layer and a cross entropy loss is applied on the predictions. Figure 18 visualizes the model for classification.

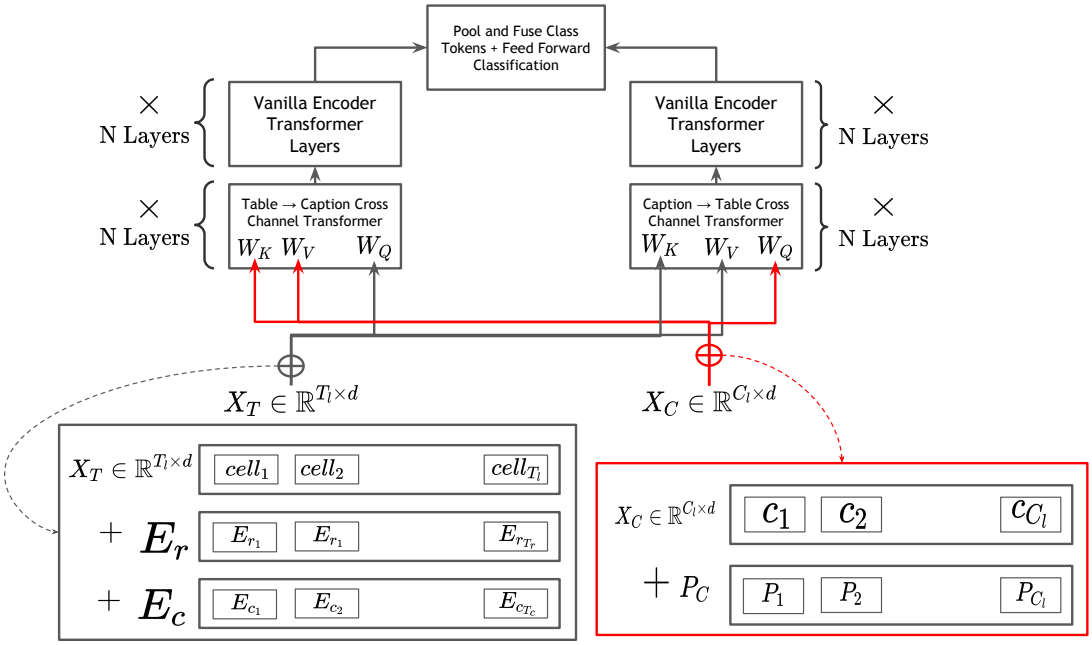


Figure 19. Cross Channel Transformer For Table Classification.

5.2.2 Cross Channel Transformer (CCT)

Based on the models developed by Tsai et al. 2019 and Deng et al. 2020, a structurally-aware cross channel Transformer model is adapted to fit the function f_θ. Figure 19 shows the outline of the model for classification. This model is chosen to validate wheather cross attention based intermediate representations have an influence on the performance of the model or not.

5.2.2.1 Input Representation

The input representations for T and C are the same as given in Section 5.2.1.1

[26]https://huggingface.co/allenai/scibert_scivocab_uncased

5.2.2.2 Model Description

The table cell sequence T_{cell} and the Caption C are first converted to embeddings X_T, X_C respectively. Post creation of the embeddings, the caption embedding X_C is summed with the position embedding P_C; The cell embeddings X_T are summed with E_r, E_c. After the summings of the respective embeddings, differentiable CLS tokens are concatenated to X_T, X_C. These embeddings are passed to separate cross channel attention transformer layers (Tsai et al. 2019) after which they are passed to the vanilla transformer encoder attention layers (Vaswani et al. 2017). Different strategies are chosen for handling the output of the vanilla tranformer encoder based on the type of task.

Embeddings at the CLS token position, are pooled from the sequences and concatenated for the table of comparison classification task. Once concatenated, this joint embedding is passed to feed forward layers for classification using Cross Entropy Loss. For the pretaining task, the last vanilla encoder layer's output will be used for predicting the masked tokens of the input table cells T_{cell} and the caption C.

5.2.3 Finetuning Scibert Transformer Model

Beltagy, Lo, and Cohan 2019 created a model which was trained on large full text corpus of scientific research containing 1.14M research papers. As this model has been subjected large amounts of pretraining, it is chosen for fine-tuning for the same problem as the model may have encountered data of a similar distribution.

5.2.3.1 Input Representation

For the finetuning of SciBert, The Table T and caption C are serialized to strings and one concatenated string is created. An example representation is given below:

```
CAPTION : Table 1: Feature extraction time (Seconds),
number of parameters (Millions), and network size (Megabytes)
for each source on Office + Caltech-10 datasets.
```

```
TABLE :                          0         1          2      3
0                     Task  Net size  Parameters   Time
1           Squeezenet [17]        46        1.24   13.3
2              Alexnet [21]       227          61   13.9
3            Googlenet [40]        27           7   15.9
4           Shufflenet [51]       6.3         1.4   17.0
5             Resnet18 [15]        44        11.7   14.8
6                Vgg16 [36]       515         138   33.6
7                Vgg19 [36]       535         144   37.1
8          Mobilenetv2 [35]        13         3.5   21.4
9         Nasnetmobile [55]        20         5.3   39.3
10            Resnet50 [15]        96        25.6   22.7
```

5.2.3.2 Model Description

The input representation string is converted to a sequence of integer tokens which are capped to the length of 512 tokens because of SciBert's constraint on maximum

sequence length. A class token is appended at the start of the sequence; the sequence is then passed to the Scibert Model. The class token is pooled from after the final layer and is passed to a feed forward neural network for binary classification using cross entropy loss.

5.2.3.3 Model Training

The model is trained using the for 6 epochs with a linear learning rate warm up schedule of 20 epochs.

5.2.4 Caption only Encoder Transformer

5.2.4.1 Input Representation

The input representation for the Caption only Encoder Transformer uses only the Caption C as input. The encoder and the input representations for the transformer is same given in Section 5.2.1.1 and Figure 18.

5.2.5 SVM and Naive Bayes

SVM models are used as baselines as when Kim et al. 2012 conducted a study for table type classification, deep learning was not as popular. Naive Bayes(NB) model are also chosen as an additional baseline for the classification task.

The input representation for the SVM and the NB model are the same as the

string described in Section 5.2.3.1. The string undergoes tokenisation and a classifier is trained based on TF/IDF values of the input strings.

5.2.6 Pretraining Transformers

Based on the insights learned from Hernandez et al. 2021 on the effectiveness of pretrained models for downstream tasks, both transformer models in Section 5.2.2 and Section 5.2.1 are first pretrained with 70K tables from papers from ArXiv. The pretraining consists of the task of Mask Language Modeling(MLM) for the sequence of cells T_{cell} and the caption C. The model is tasked to predict the token where a MASK token is inserted in the table T and caption C. A mask is inserted in 80% of the training samples with 15% of the sequence being masked. The cross entropy losses from MLM losses of the Table and Caption are summed and backpropagated. The pretraining task for the CCT and E-MCT transformer models is run for 35K steps in 5 epochs.

The caption only transformer is trained with MLM loss on the captions for 6K steps over 5 epochs. It is trained for lesser steps due to larger batch size. The loss of the pretaining for the caption only transformer can be seen from Figure 22.

The learning rate for all pretraining tasks is set to 0.0001 with an AdamW (Loshchilov and Hutter 2017) optimizer.

5.2.7 Training Transformers For Classification

After CCT ,E-MCT and the caption only Transformer undergo the pretraining task, the transformer layers from the models are extracted; A new linear classifier

head is added as the last layer and then the models are finetuned on the classification task with a learning rate (LR) of 2e-05, scheduled with a linear LR scheduler for 20 epochs using the AdamW optimizer. All models trained for classification undergo same conditions for training. Even the model trained from scratch on the classification task undergo the same optimization regime for same number of epochs.

5.3 Dataset Collection, Labeling and Construction

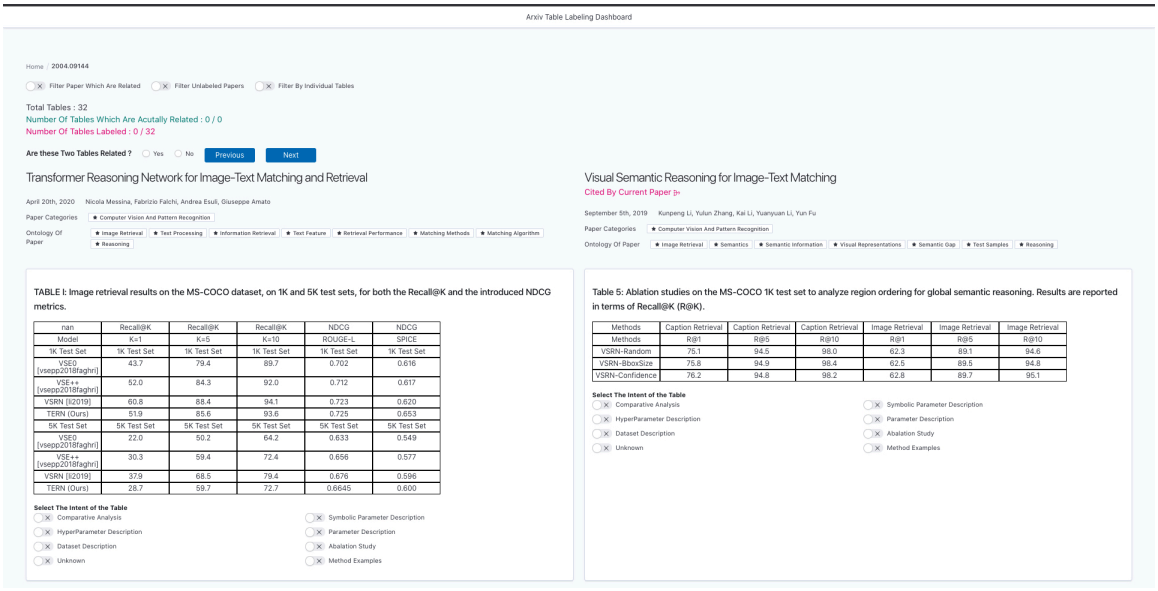


Figure 20. Table Labeling Interface For Table Relation and Type Annotation.

5.3.1 Data Collection

All the tables needed for classification are extracted from the LaTeX source via Sci-Genie. Sci-Genie uses arxiv-vanity/engrafo[27] to convert LaTeX to HTML. The tables are extracted from HTML using CSS selectors. The LaTeX to HTML conversion ensures specific CSS selectors for explicitly stated tabular data in the LaTeX source. Once the tables are extracted from the HTML source, they are serialized to pandas DataFrames and stored in the search index. The schema of the stored data is described in Chapter 4.2.3.

5.3.2 Data Labeling

A labeling interface (Figure 20) is designed to annotate table types. Tables are annotated from 240 research papers. The papers are chosen at random to label. The topics of the papers from which the tables are derived can be seen in Figure 21. To explicitly discover tables which are *not* describing comparisons, the following types of tables are defined:

- Comparative Analysis : The table is describing a comparison between entities based on some metric/metrics. This is the type of table the classifier is supposed to classify.
- Symbolic Parameter Description : The table is describing some symbols which could belong to an equation.

[27]https://github.com/arxiv-vanity/engrafo

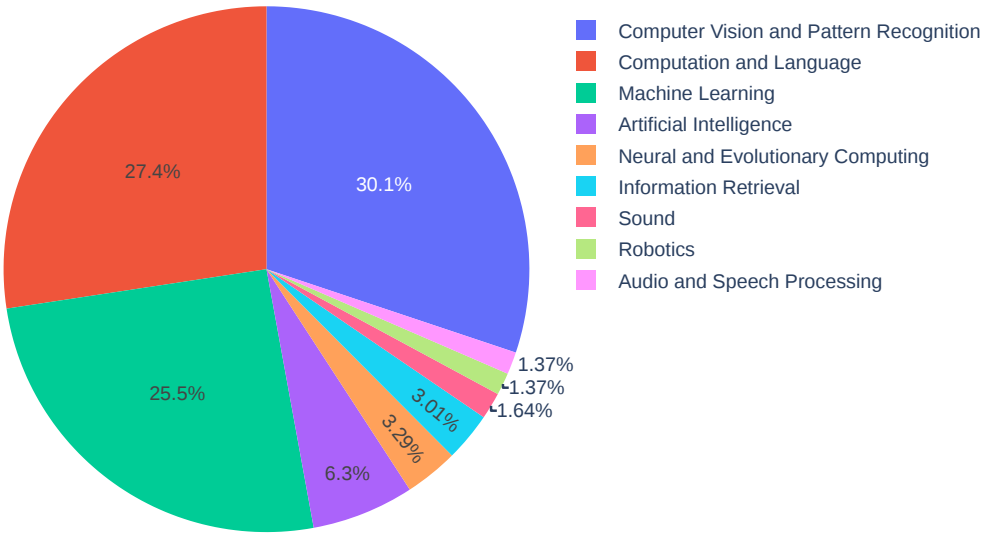


Figure 21. Distribution of the different categories of papers the labeled tables belongs to.

- HyperParameter Description : The table is describing hyper parameters for some experiment.
- Dataset Description : The table is describing statistics/examples from some dataset.
- Ablation Study : The table is describing ablation study or showing some perturbation analysis.
- Parameter Description: The table is describing parameters or entities which are relevant to describing the experiment. The label is more generalised than HyperParameter Description or Dataset Description or Symbolic Parameter Description. This label is provided when labelers are unable to categorize a

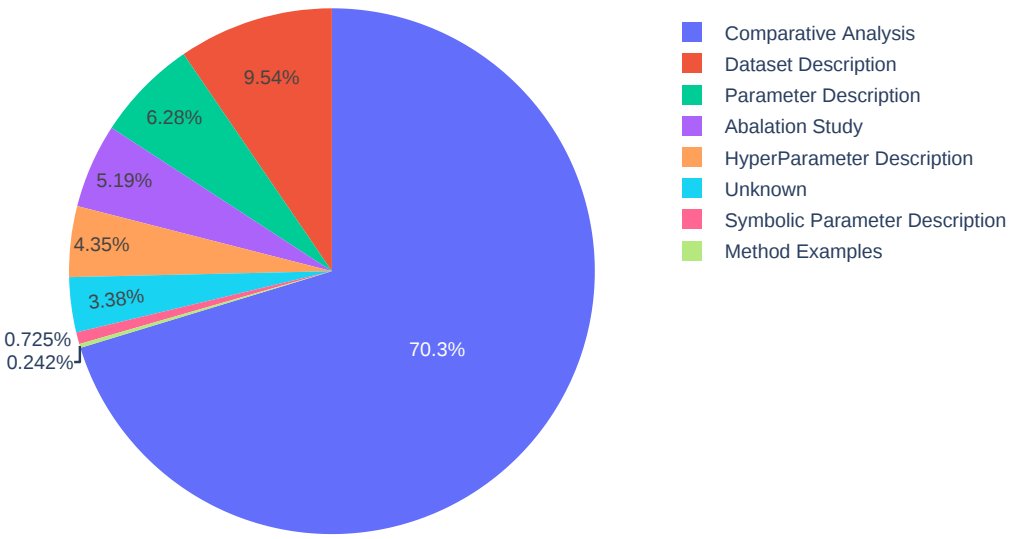


Figure 22. Distribution of the labeled types for different tables identified in the labeling process.

table under a HyperParameter Description or Dataset Description or Symbolic Parameter Description but do observe a form of parametric description necessary to describe an experiment.

- Method Examples : These are tables which are showing input-ouput of some method described in the paper. These types of tables are observed in Machine learning papers where authors show samples various input-output pairs of for an ML model.

The above defined types are similar to Kim et al. 2012 but also help capture some types which are observed distinctly in recent CS literature. The tables which could not be explicitly classified are labeled as *Unknown*. Examples of all tables are provided in

the Appendix B.2. The extra types are added during the labeling phase to ensure a solid justification for a label when a labeler chooses to label a table.

3 labelers are tasked to label the data. 820 tables are labeled out of which 574 tables are describing a comparison and 246 are not. Figure 22 visualizes the distribution of the labeled types of tables as a pie chart.

5.3.3 Test Set Construction

The test set consists of 100 tables describing comparisons and 100 tables which are not describing a comparison. 100 random samples are extracted from tables which are labeled "Comparative Analysis". 100 random samples are taken from tables belonging to the other classes.

5.3.4 Training Set Construction

The training set consists of the 620 tables which remain after the extraction of the test set. As the dataset has a class imbalance, the classes are rebalanced by duplicating data of non-comparison-tables for the classification training process.

Model Name	Loss	F1 Score	Accuracy
Caption Only without Pretraining	0.599 ± 0.017	0.7134 ± 0.001	71.354 ± 0.01
Caption Only with Pretraining	0.676 ± 0.036	0.757 ± 0.086	73.438 ± 8.854
E-MCT without Pretraining	0.567 ± 0.003	0.809 ± 0.016	80.404 ± 1.765
E-MCT with Pretraining	$\mathbf{0.417 \pm 0.044}$	$\mathbf{0.845 \pm 0.017}$	$\mathbf{84.505 \pm 1.722}$
CCT without Pretraining	0.658 ± 0.13	0.596 ± 0.144	63.281 ± 14.844
CCT with Pretraining	0.505 ± 0.008	0.7962 ± 0.001	79.688 ± 0.01
SciBERT Finetuning	0.528 ± 0.013	0.826 ± 0.005	82.292 ± 0.521
SVM	-	0.77 ± 0.007	74.635 ± 0.991
Naive Bayes	-	0.77 ± 0.007	73.125 ± 1.06

Table 5. Test set results of the table of comparison classification models. The presented statistics are after averaging 8 training runs.

The following are the ML models whose results are reported in Table 5.4:

- Cross Channel Transformer (CCT) With/Without pretraining.
- Encoder Only Multi-Channel Transformer (E-MCT) With/Without pretraining.
- Caption Only Transformer Encoder With/Without pretraining.
- SciBert Finetuned on the classification task (Baseline).
- SVM and Naive Bayes (Baseline).

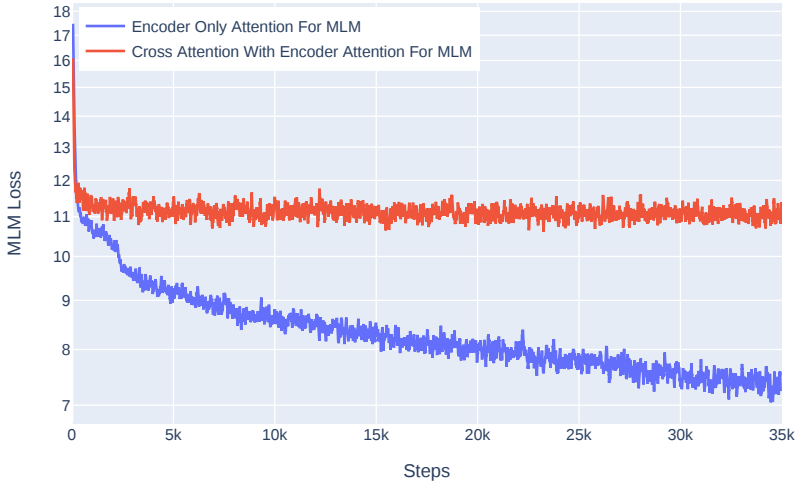


Figure 23. Masked Language Modeling(MLM) Loss For Encoder Style Attention VS Cross Attention. The MLM losses for T and C are summed.

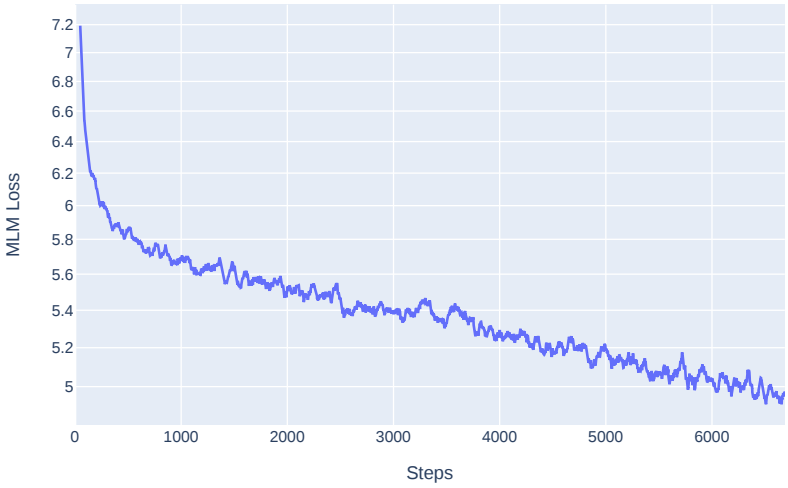


Figure 24. Masked Language Modeling(MLM) Loss For Caption Only Transformer Encoder Model

All experiments are conducted using Google Colab Notebooks on NVIDIA P100 GPUs. The models that undergo pretraining follow the pretraining regime given in Section 5.2.6. The loss plot for the pretraining phases for the E-MCT ,CCT, Caption only transformer are visualized in Figure 23, 24.

All model variants except for SciBert are trained for 20 epochs. SciBert is trained for 6 epochs. The training optimization details are provided in Section 5.2.7.

The optimization method is kept the same for all classification experiments. More details on the hyperparameters for the experiments are found in the Appendix B.

All experiments are conducted on the same frozen test set with evenly balanced samples as described in Section 5.3.3. Table 5.4 describes performance on test set for each model after averaging 8 training runs.

A more detailed view of the test set performance metrics can be seen from Figure 27. Figure 27 visualizes the distribution of performance metrics described in Table 5.4. Figure 25, 26 visualize the metrics like loss, accuracy and F1 score during the training process for all the models.

The metric value at each training step within the plots of Figure 25 and 26 is an average from the 8 experiment runs. Section 5.4.1 discusses the insights inferred from the experiment results.

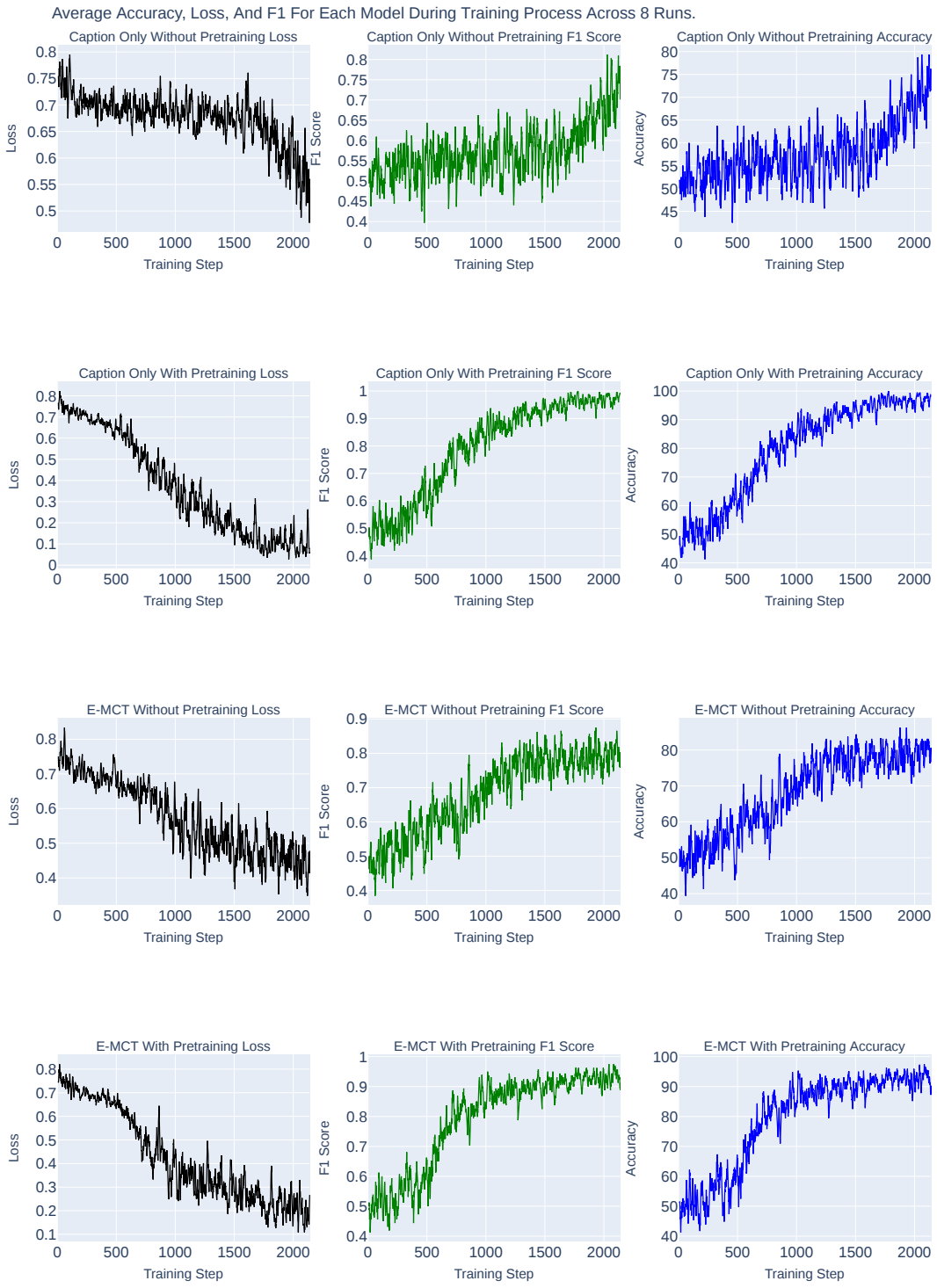


Figure 25. Average training loss, accuracy and F1 scores during the training process for the classification task across 8 runs using caption only Transformer with/without pretraining, and the E-MCT with/without pretraining. The metric value at each training step within the plots is an average from the 8 experiment runs.

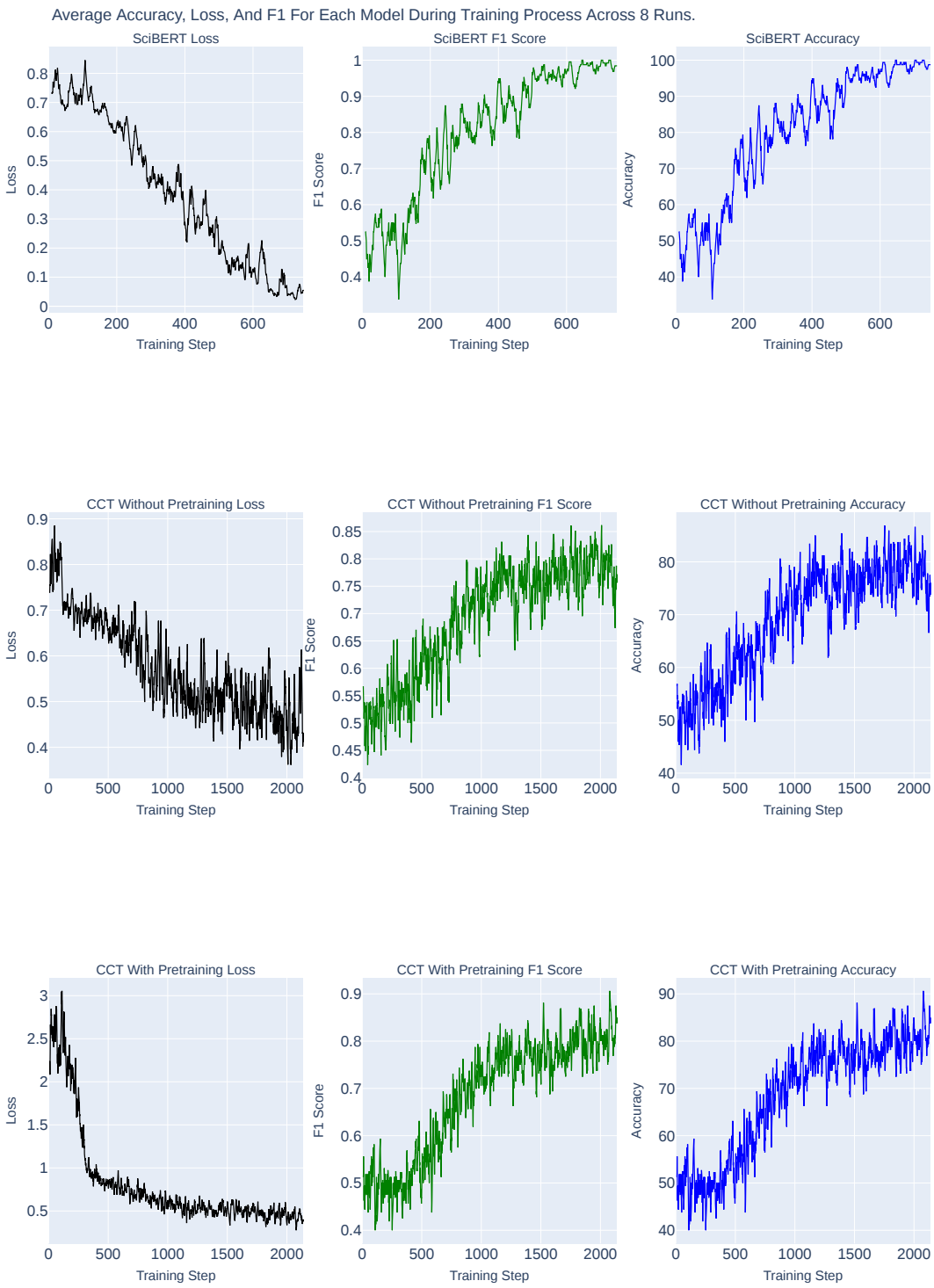


Figure 26. Average training loss, accuracy and F1 Score during the training process for the classification task across 8 runs using CCT with/without pretraining and Finetuning Sci-Bert. The metric value at each training step within the plots is an average from the 8 experiment runs.

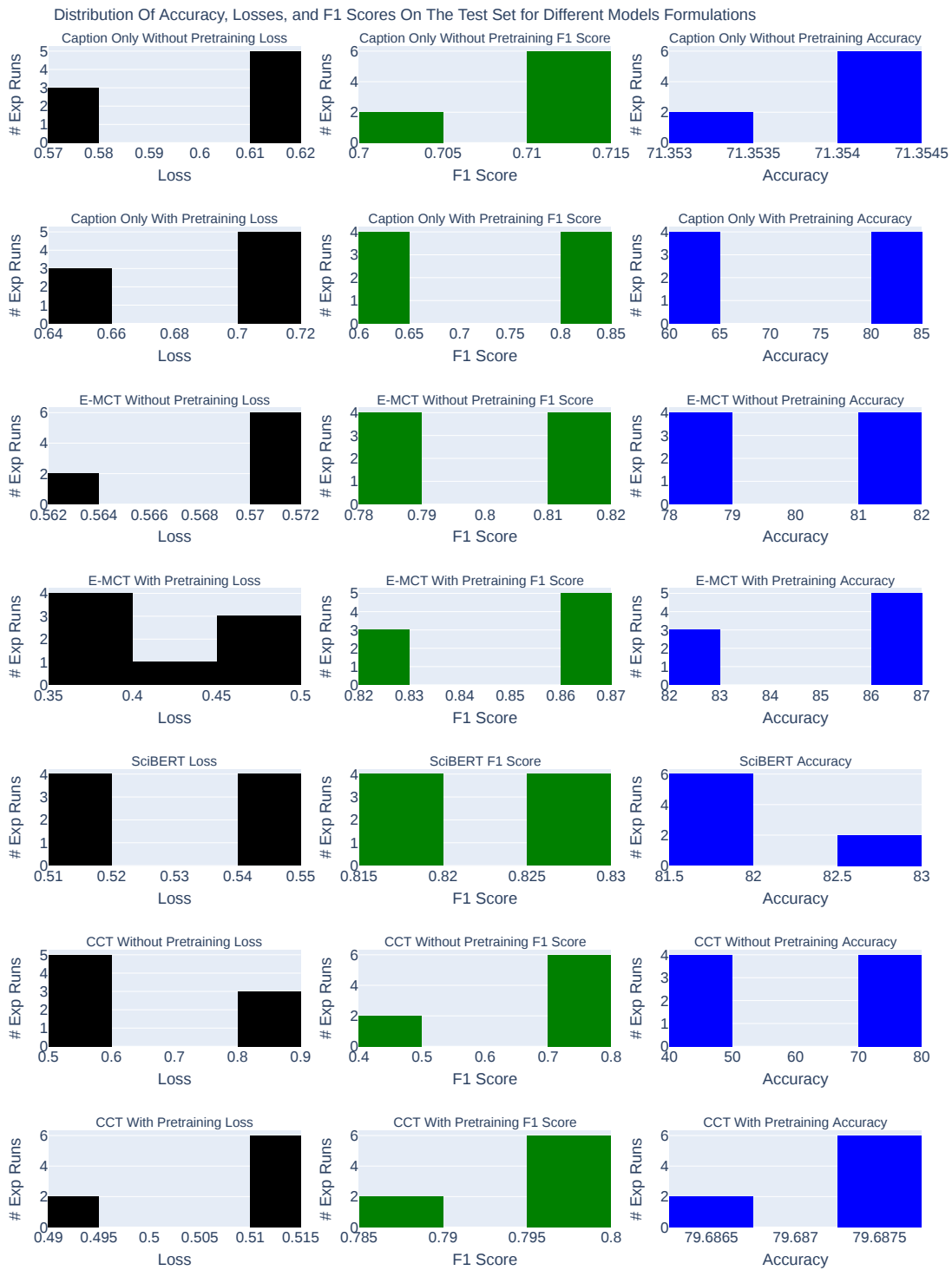


Figure 27. Distribution of loss, accuracy and F1 score on the test set for the classification task after 8 training runs for all experiments. The X-axis in each plot represents the metric. The Y-Axis is the number of experiment which resulted in the value of the metric

5.4.1 Experiment Result Discussion

There following are some distinct observations after training the models for the classification task:

- Pretraining improves the performance of the model on the task specific optimization.
- Using caption only encoder transformer for the classification task gives performance which comes at par with SciBert but with high variance.

The pertrained E-MCT model has the best performance on the test set as seen from Figure 27 and Table 5.4. Pretraining improves the performance of all the models including E-MCT, CCT and Caption only transformer compared to training those models from scratch as seen from Figure 27.

The performance of the pertrained CCT transformer is better than the model trained from scratch because the model trained from scratch has high performance variance in the test set as seen in Figure 27. CCT model has a higer loss on the pretraining task than the E-MCT model as seen in Figure 23. This helps infer the reason for the E-MCT's better performance on the fine-tuning task compared to CCT.

Even though SciBert (Figure 26) has a better accuracy on the train set than E-MCT (Figure 25), E-MCT generalizes better on the test-set.

It can be argued that the difference in performance between Sci-Bert and E-MCT models (only a few percentage points) is a negligible difference due to the size of the test set. But this arguement also sheds light that while E-MCT is a smaller model, it gives performance at par or better than a larger model like SciBert after pretraining.

The pretrained caption only encoder transformer performs as well as the SciBert but with a high variance as seen in Figure 27. Even though this pretrained model

reaches convergence in the training process as seen in Figure 25, its test set performance has high variance and the best model doesn't outperform E-MCT.

Chapter 6

CONCLUSION AND FUTURE SCOPE

6.1 Sci-Genie: Technical Debts and Shortcomings

Although Sci-Genie provides a more context rich access to search results, the components that help enrich context during search i.e., the parsing/indexing can be further improved. This section details the limitations of some components. Based on some of these limitations, future work is described in Section 6.2.

6.1.1 Heuristic Parsing

Sci-Genie uses a heuristic parser for parsing research documents. The heuristic parser correlates the Fragment object to the appropriate keys in the ResearchObject. Figure 28 shows the statistics of parsing using the heuristic parser from a total corpus of 158K papers. The parser misses out many sections and it can be further improved using better methods. The parser also considers the sections based on ones observed in CS papers. Future research directions can include identifying common patterns in the structure of research documents for fields outside CS; Research directions can include identifying domain specific document structural patterns to create a taxonomy that can be used as a part of parsing.

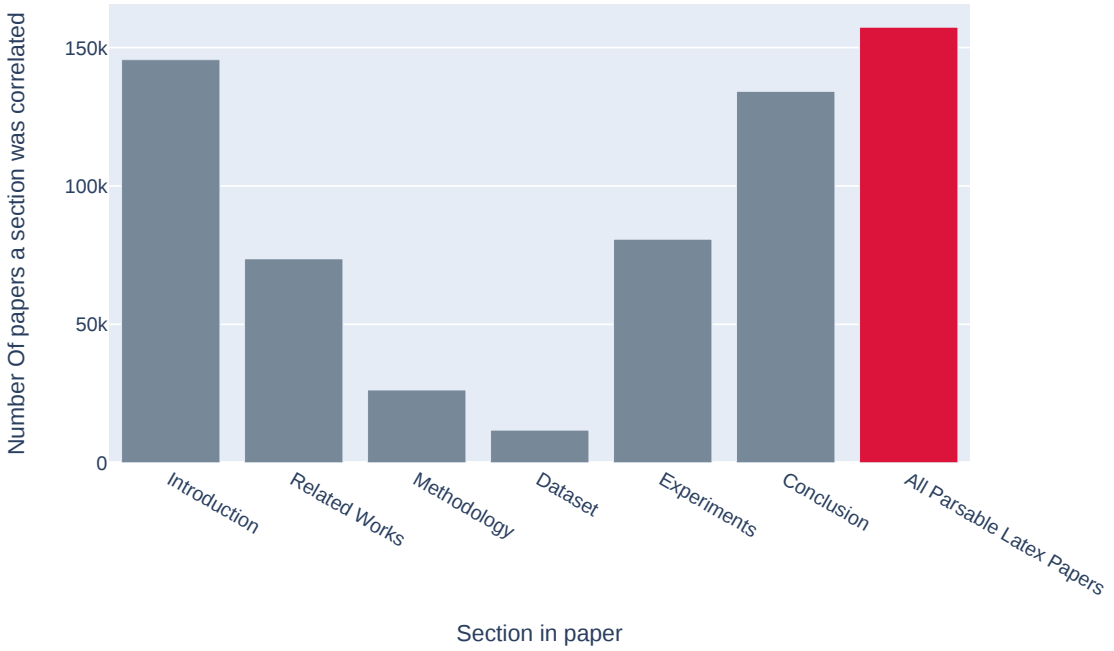


Figure 28. A bar plot of number of documents where a section got parsed by the heuristic parser from 158K papers.

6.1.2 CS ArXiv Only Access & No Specialized Ranking

While ArXiv can be a source of freely available well cited research, it also remains a preprint server where lots documents are not gauranteed to be cited. As Sci-Genie doesn't use citations or any other indicator as a metric for ranking search results, it lacks providing most well cited information. Sci-Genie is also specialized to CS and its ontology classifier uses CS specific domains.

Future versions can extend the search engine to support more fields of research and use ontology for those fields. Future versions can even include better ranking schemes for the documents.

6.2 Future Research Scope

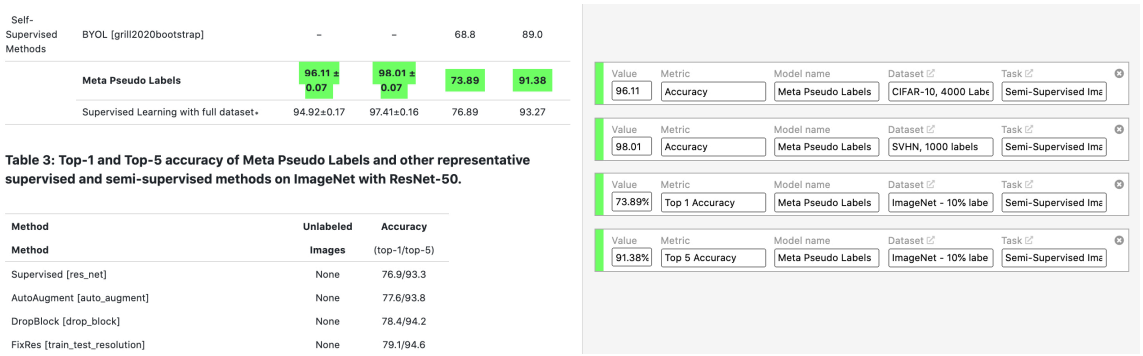
6.2.1 Grannular Table Type Classification

The table types defined by Kim et al. 2012 are general purpose descriptions which can apply to various domains. Although they are quite useful, tables in recent CS literature have more grannular sub-classifications. For example a "Statistics Table" could be describing a Dataset for Machine Learning experiments. The tables which may seem as Experiments Results can also be describing an Ablation Study conducted on a ML model. Due to a very nuanced distinction between a lot of these tables, The task of machine learning to classify table types needs more grannular sub-types within the classes defined by Kim et al. 2012.

Multiple grannular table types such as Ablation Study, Dataset Descriptions, can also be further described based on the field of research the tables belong to. The interface shown in Figure 20, was developed to annotated tables from scientific research. This interface can be extended for many fields outside CS; This inturn can aid the effort of curation and indentification of various types of tables from different avenues of research.

6.2.2 Discovering Related Tables

Organisations such as PapersWithCode(PwC) have developed tools to correlate information about SOTA ML methods from papers on ArXiv. PwC parses LaTeX



Self-Supervised Methods	BYOL [grill2020bootstrap]	–	–	68.8	89.0
	Meta Pseudo Labels	**96.11 ± 0.07**	**98.01 ± 0.07**	**73.89**	**91.38**
	Supervised Learning with full dataset•	94.92±0.17	97.41±0.16	76.89	93.27

Table 3: Top-1 and Top-5 accuracy of Meta Pseudo Labels and other representative supervised and semi-supervised methods on ImageNet with ResNet-50.

Method	Unlabeled Images	Accuracy (top-1/top-5)
Supervised [res_net]	None	76.9/93.3
AutoAugment [auto_augment]	None	77.6/93.8
DropBlock [drop_block]	None	78.4/94.2
FixRes [train_test_resolution]	None	79.1/94.6

Figure 29. Viewing Tables For State of the art benchmarks using https://paperswithcode.com.

based tables and correlates the benchamark metrics manually or via the use of libraries like AxCell[28].

The benchmark correlation methods used by PwC are specific to ML SOTA benchmarks, Future research directions can include the automated extraction, corellation, and storage of tables from research papers into strict relational databases for domains outside ML and CS.

6.2.3 Scientific Research Document Parsing For Search Context

Google search pivoted to ranking results for search using BERT[29] since 2019. The trend of using language models to help rank search results has lead to creation of opensource tools[30] that rerank documents over off the shelf search engines like Elasticsearch. But search ranking alone would not solve the search context problem as described in Chapter 1.2.

[28]https://github.com/paperswithcode/axcell

[29]https://blog.google/products/search/search-language-understanding-bert/

[30]https://github.com/deepset-ai/haystack

Intelligent parsing maybe one part of the solution. Sci-Genie are uses domain specific content parser which is heuristically derived (Chapter 4.3.1.2) to help bring context relevant information in search results. This parser may not be applicable to other domains or document types.

Kashyap and Kan 2020, introduced deep learning based techniques for scientific document parsing. Kashyap and Kan 2020's model generates a general purpose parsing structure. Although very useful, Every domain's document parse structure also have domain specific characteristics; For example, papers in Machine Learning explicitly have sections describing datasets in the paper. Identifying such patterns for generate domain specific parse structures for indexing can be a useful future direction of research for improving search context in academic search.

6.3 Conclusion

In summary, this disseration presented a system to collect, parse and store data about research papers from CS ArXiv. The dissertaion additionally describes the components of the system and methods the system uses to extract various types of data about research like the structured full text content, the tables and the citations. The disseration finally discusses ML models which help classify tables describing comparisons. It further shows the boost in classification performance when the models undergo pretraining with the large amount of tabular data collected by the system.

REFERENCES

Ammar, Waleed, Dirk Groeneveld, Chandra Bhagavatula, Iz Beltagy, Miles Crawford, Doug Downey, Jason Dunkelberger, et al. 2018. "Construction of the Literature Graph in Semantic Scholar." In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 3 (Industry Papers),* 84–91. New Orleans - Louisiana: Association for Computational Linguistics, June. doi:10.18653/v1/N18-3011.

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

Brin, Sergey, and Lawrence Page. 1998. "The anatomy of a large-scale hypertextual web search engine." *Computer networks and ISDN systems* 30 (1-7): 107–117.

Brown, Tom B, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. "Language models are few-shot learners." *arXiv preprint arXiv:2005.14165.*

Chen, Ting, Simon Kornblith, Kevin Swersky, Mohammad Norouzi, and Geoffrey Hinton. 2020. "Big self-supervised models are strong semi-supervised learners." *arXiv preprint arXiv:2006.10029.*

Cohan, Arman, Waleed Ammar, Madeleine Van Zuylen, and Field Cady. 2019. "Structural scaffolds for citation intent classification in scientific publications." *arXiv preprint arXiv:1904.01608.*

Dasigi, Pradeep, Nelson F Liu, Ana Marasović, Noah A Smith, and Matt Gardner. 2019. "Quoref: A reading comprehension dataset with questions requiring coreferential reasoning." *arXiv preprint arXiv:1908.05803.*

Deng, Xiang, Huan Sun, Alyssa Lees, You Wu, and Cong Yu. 2020. "TURL: table understanding through representation learning." *Proceedings of the VLDB Endowment* 14 (3): 307–319.

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.*

Doersch, Carl, Abhinav Gupta, and Alexei A Efros. 2015. "Unsupervised visual representation learning by context prediction." In *Proceedings of the IEEE international conference on computer vision,* 1422–1430.

Dosovitskiy, Alexey, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. 2020. "An image is worth 16x16 words: Transformers for image recognition at scale." *arXiv preprint arXiv:2010.11929.*

Gidaris, Spyros, Praveer Singh, and Nikos Komodakis. 2018. "Unsupervised representation learning by predicting image rotations." *arXiv preprint arXiv:1803.07728.*

Gormley, Clinton, and Zachary Tong. 2015. *Elasticsearch: the definitive guide: a distributed real-time search and analytics engine.* " O'Reilly Media, Inc."

Goyal, Priya, Dhruv Mahajan, Abhinav Gupta, and Ishan Misra. 2019. "Scaling and benchmarking self-supervised visual representation learning." In *Proceedings of the IEEE/CVF International Conference on Computer Vision,* 6391–6400.

Gusenbauer, Michael, and Neal R Haddaway. 2020. "Which academic search systems are suitable for systematic reviews or meta-analyses? Evaluating retrieval qualities of Google Scholar, PubMed, and 26 other resources." *Research synthesis methods* 11 (2): 181–217.

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. "Deep residual learning for image recognition." In *Proceedings of the IEEE conference on computer vision and pattern recognition,* 770–778.

Hernandez, Danny, Jared Kaplan, Tom Henighan, and Sam McCandlish. 2021. "Scaling Laws for Transfer." *arXiv preprint arXiv:2102.01293.*

Howard, Jeremy, and Sebastian Ruder. 2018. "Universal language model fine-tuning for text classification." *arXiv preprint arXiv:1801.06146.*

Huang, Cheng-Zhi Anna, Ashish Vaswani, Jakob Uszkoreit, Noam Shazeer, Ian Simon, Curtis Hawthorne, Andrew M Dai, Matthew D Hoffman, Monica Dinculescu, and Douglas Eck. 2018. "Music transformer." *arXiv preprint arXiv:1809.04281.*

Ioffe, Sergey, and Christian Szegedy. 2015. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." In *International conference on machine learning,* 448–456. PMLR.

Kacem, Ameni, and Philipp Mayr. 2018. "Analysis of search stratagem utilisation." *Scientometrics* 116 (2): 1383–1400.

Kardas, Marcin, Piotr Czapla, Pontus Stenetorp, Sebastian Ruder, Sebastian Riedel, Ross Taylor, and Robert Stojnic. 2020. "Axcell: Automatic extraction of results from machine learning papers." *arXiv preprint arXiv:2004.14356.*

Kashyap, Abhinav Ramesh, and Min-Yen Kan. 2020. "SciWING–A Software Toolkit for Scientific Document Processing." *arXiv preprint arXiv:2004.03807.*

Kim, Seongchan, Keejun Han, Soon Young Kim, and Ying Liu. 2012. "Scientific table type classification in digital library." In *Proceedings of the 2012 ACM symposium on Document engineering,* 133–136.

Kolesnikov, Alexander, Xiaohua Zhai, and Lucas Beyer. 2019. "Revisiting self-supervised visual representation learning." In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition,* 1920–1929.

Li, Kunpeng, Yulun Zhang, Kai Li, Yuanyuan Li, and Yun Fu. 2019. "Visual semantic reasoning for image-text matching." In *Proceedings of the IEEE/CVF International Conference on Computer Vision,* 4654–4662.

Li, Xinyi, Bob JA Schijvenaars, and Maarten de Rijke. 2017. "Investigating queries and search failures in academic search." *Information processing & management* 53 (3): 666–683.

Lin, Tsung-Yi, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. 2014. "Microsoft coco: Common objects in context." In *European conference on computer vision,* 740–755. Springer.

Liu, Pengfei, Xipeng Qiu, and Xuanjing Huang. 2017. "Adversarial multi-task learning for text classification." *arXiv preprint arXiv:1704.05742.*

Loshchilov, Ilya, and Frank Hutter. 2017. "Decoupled weight decay regularization." *arXiv preprint arXiv:1711.05101.*

Lucene, Apache. 2010. "Apache lucene-overview." *Internet: http://lucene. apache. org/iava/docs/[Jan. 15, 2009].*

Ma, Wei-Chiu, Ignacio Tartavull, Ioan Andrei Bârsan, Shenlong Wang, Min Bai, Gellert Mattyus, Namdar Homayounfar, Shrinidhi Kowshika Lakshmikanth, Andrei Pokrovsky, and Raquel Urtasun. 2019. "Exploiting sparse semantic HD maps for self-driving vehicle localization." *arXiv preprint arXiv:1908.03274.*

Maksai, Andrii, Xinchao Wang, and Pascal Fua. 2016. "What players do with the ball: A physically constrained interaction modeling." In *Proceedings of the IEEE conference on computer vision and pattern recognition,* 972–981.

Messina, Nicola, Fabrizio Falchi, Andrea Esuli, and Giuseppe Amato. 2020. "Transformer reasoning network for image-text matching and retrieval." *arXiv preprint arXiv:2004.09144.*

Milosevic, Nikola, Cassie Gregson, Robert Hernandez, and Goran Nenadic. 2019. "A framework for information extraction from tables in biomedical literature." *International Journal on Document Analysis and Recognition (IJDAR)* 22 (1): 55–78.

Paszke, Adam, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. "Pytorch: An imperative style, high-performance deep learning library." *arXiv preprint arXiv:1912.01703.*

Radford, Alec, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. "Language models are unsupervised multitask learners." *OpenAI blog* 1 (8): 9.

Radford, Alec, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. 2021. "Learning transferable visual models from natural language supervision." *arXiv preprint arXiv:2103.00020.*

Ramesh, Aditya, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. 2021. "Zero-shot text-to-image generation." *arXiv preprint arXiv:2102.12092.*

Rovira, Cristòfol, Lluís Codina, Frederic Guerrero-Solé, and Carlos Lopezosa. 2019. "Ranking by relevance and citation counts, a comparative study: Google Scholar, Microsoft Academic, WoS and Scopus." *Future Internet* 11 (9): 202.

Safder, Iqra, Saeed-Ul Hassan, Anna Visvizi, Thanapon Noraset, Raheel Nawaz, and Suppawong Tuarob. 2020. "Deep learning-based extraction of algorithmic metadata in full-text scholarly documents." *Information Processing & Management* 57 (6): 102269.

Salatino, Angelo A, Francesco Osborne, Thiviyan Thanapalasingam, and Enrico Motta. 2019. "The CSO classifier: ontology-driven detection of research topics in scholarly

articles." In *International Conference on Theory and Practice of Digital Libraries,* 296–311. Springer.

Tsai, Yao-Hung Hubert, Shaojie Bai, Paul Pu Liang, J Zico Kolter, Louis-Philippe Morency, and Ruslan Salakhutdinov. 2019. "Multimodal transformer for unaligned multimodal language sequences." In *Proceedings of the conference. Association for Computational Linguistics. Meeting,* 2019:6558. NIH Public Access.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. "Attention is all you need." *arXiv preprint arXiv:1706.03762.*

Zha, Hanwen, Wenhu Chen, Keqian Li, and Xifeng Yan. 2019. "Mining algorithm roadmap in scientific publications." In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining,* 1083–1092.

Zhang, Richard, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. 2018. "The unreasonable effectiveness of deep features as a perceptual metric." In *Proceedings of the IEEE conference on computer vision and pattern recognition,* 586–595.

APPENDIX A

ELASTICSEARCH RELATED INFORMATION

A.1 Arxiv Parsed Research Index Example


```
{
   "identity": {
     "identity": "2104.00682",
     "url": "http://arxiv.org/abs/2104.00682v1",
     "title": "",
     "abstract": "",
     "categories": [
       "cs.CV",
       "cs.AI",
       "cs.LG"
     ],
     "published": "2021-04-01T17:59:48Z",
     "updated": "2021-04-01T17:59:48Z",
     "authors": [
       "Bo Xiong",
       "Haoqi Fan",
       "Kristen Grauman",
       "Christoph Feichtenhofer"
     ],
     "affiliation": [],
     "version": 1,
     "journal_reference": null,
     "created_on": "2021-04-02T04:20:33.981856"
   },
   "research_object": {
     "introduction": {
       "name": "Introduction",
       "subsections": [],
       "text": "",
       "matched": true
     },
     "related_works": {
       "name": "RelatedWorks",
       "subsections": [],
       "text": "",
       "matched": true
     },
     "methodology": {
       "name": "Methodology",
```

```
  "subsections": [],
  "text": "",
  "matched": false
},
"experiments": {
  "name": "Experiments",
  "subsections": [],
  "text": "",
  "matched": true
},
"results": {
  "name": "Results",
  "subsections": [],
  "text": "",
  "matched": false
},
"dataset": {
  "name": "Dataset",
  "subsections": [],
  "text": "",
  "matched": false
},
"conclusion": {
  "name": "Conclusion",
  "subsections": [],
  "text": "",
  "matched": true
},
"limitations": {
  "name": "Limitations",
  "subsections": [],
  "text": "",
  "matched": false
},
"unknown_sections": [
  {
    "name": "",
    "subsections": [
      {
        "name": "",
        "subsections": [],
        "text": ""
```

```
          }
        ],
        "text": ""
      }
    ]
  },
  "parsing_stats": {
    "active": true,
    "section_matches": [
      "Introduction",
      "RelatedWorks",
      "Experiments",
      "Conclusion"
    ],
    "num_un_matched": 2
  },
  "created_on": "2021-04-02T04:20:40.318679",
  "ontology": {
    "syntactic": [
    ],
    "semantic": [
    ],
    "union": [
    ],
    "enhanced": [
    ],
    "mined": true
  }
}
```

A.2 Content Parsing Code/Algorithm

The *ArxivLatexParser* is used to create the parsed content tree from which section based correlation is done. Below is the algorithm and source code in Python programming language.

```
import os
from subprocess import Popen, PIPE
from tex2py import tex2py
from typing import List
import json


def get_tex_tree(tex_path):
    with open(tex_path,'r') as f:
        data = f.read()
    tex_root_node = tex2py(data)
    return tex_root_node

class Section():
    """Section
    Section will contain subsections which are of type Section
    """
    def __init__(self,name=None):
        # Core Attributes
        self.name = self.__class__.__name__ if name is None else name
        self.subsections = [] # List[Sections]
        self.text = ''

    def to_markdown(self,tab_counter=0):
        SPACE= ' '
        HEADING='#'
        tab_counter+=1
        heading_val = HEADING*tab_counter if tab_counter <=3 else HEADING*3
        hierarchy = [heading_val+SPACE+self.name+"("+str(len(self.text))+")
            "]+[self._clean_text(self.text)] if len(self.text) > 0 else []
        for subsection in self.subsections:
            hierarchy.append(subsection.to_markdown(tab_counter=tab_counter))
        return '\n'.join(hierarchy)

    def to_quoted_tags(self):
        SPACE= ' '
        QUOTE_SECTION='<SECTION>'
```

```
    UNQUOTE_SECTION = '</SECTION>'
    QUOTE_CONTENT='\n<CONTENT>\n'
    UNQUOTE_CONTENT = '\n</CONTENT>\n'
    # tab_counter+=1
    # heading_val = HEADING*tab_counter if tab_counter <=3 else HEADING*3
    hierarchy = [
        QUOTE_SECTION+\
            self.name+UNQUOTE_SECTION]+[QUOTE_CONTENT+self._clean_text(
                self.text)+UNQUOTE_CONTENT] if len(self.text) > 0 else []
    for subsection in self.subsections:
        hierarchy.append(subsection.to_quoted_tags())
    return '\n'.join(hierarchy)

@staticmethod
def _clean_text(text):
    return text.replace('\\n','\n').replace('\t','').replace('\r','')

def hierarchy_string(self,tab_counter=0):
    TAB = '\t'
    tab_counter+=1
    hierarchy = [TAB*tab_counter+self.name+"("+str(len(self.text.split('
        ')))+")"]
    for subsection in self.subsections:
        hierarchy.append(subsection.hierarchy_string(tab_counter=
            tab_counter))
    return '\n'.join(hierarchy)

def to_json(self):
    serialized_object = {
        'name' : self.name,
        'subsections': [],
        'text': self.text
    }
    for ss in self.subsections:
        serialized_object['subsections'].append(ss.to_json())
    return serialized_object

@classmethod
def from_json(cls,json_object):
    if 'name' not in json_object:
        raise SectionSerialisationException('"name" Attribute missing in
            object')
    generated_obj = cls(name=json_object['name'])
    generated_obj.text = json_object['text']
```

```
        for val in json_object['subsections']:
            generated_obj.subsections.append(cls.from_json(val))
        return generated_obj

    def save_to_file(self,file_path):
        with open(file_path,'w') as f:
            json.dump(self.to_json(),f)

    def _get_hierarchy(self):
        hierarchy = []
        for subsection in self.subsections:
            hierarchy.append(subsection._get_hierarchy())
        return {self.name:hierarchy}

    def __str__(self):
        return self.hierarchy_string()

    def flattened_sections(self):
        flattened_arr = []
        curr_sec = Section(name=self.name)
        curr_sec.text=self.text
        flattened_arr.append(curr_sec)
        for subsection in self.subsections:
            flattened_arr.extend(subsection.flattened_sections())
        return flattened_arr


class LatexParserException(Exception):
    headline = 'Latex Parsing failed'
    def __init__(self, msg='', lineno=None):
        self.message = msg
        self.line_no = lineno
        super(LatexParserException, self).__init__()

    def __str__(self):
        prefix = 'line %d: ' % self.line_no if self.line_no else ''
        return '%s%s' % (prefix, self.message)

class SectionSerialisationException(LatexParserException):
     def __init__(self,ms):
        msg = "Serialisation of Section Object Requires %s"%ms
        super(SectionSerialisationException, self).__init__(msg)

class LatexToTextException(LatexParserException):
```

```
    def __init__(self):
        msg = "Exception Raised From Text extraction at Detex"
        super(LatexToTextException, self).__init__(msg)


class DetexBinaryAbsent(LatexParserException):
    def __init__(self):
        msg = "Exception Raised Because Of No Detex Binary"
        super(DetexBinaryAbsent, self).__init__(msg)

class MaxSectionSizeException(LatexParserException):
    def __init__(self,avail,limit):
        msg = "Number of Sections %d are Larger than Maximum allowed (%d) for
             a Parsed Document"%(avail,limit)
        super(MaxSectionSizeException, self).__init__(msg)



class LatexToText():
    """LatexToText
    This class will manage the conversion of the latex document into text.
    It uses `detex` to extract the text from tex Files.
    """
    detex_path = os.path.join(os.path.abspath(os.path.dirname(__file__)),'
        detex')

    def __init__(self,detex_path=None):
        # check binary existance.
        if not self.binary_exists(self.detex_path):
            if detex_path is None:
                raise DetexBinaryAbsent()
            elif not self.binary_exists(detex_path):
                raise DetexBinaryAbsent()
            else:
                self.detex_path = detex_path

    @staticmethod
    def binary_exists(detex_path):
        try:
            os.stat(detex_path)
        except:
            return False
        return True
```

```
    def __call__(self,latex_document_path):
        try:
            process = Popen([self.detex_path, latex_document_path], stdout=
                PIPE)
            (output, err) = process.communicate()
            exit_code = process.wait()
            return output
        except Exception as e:
            print(e)
            raise LatexToTextException()

def split_match(split_value:str,splitting_string:str,split_upto=0.5,
    split_bins=10):
    """split_match
    Splits a Keep Splitting a `splitting_string` based on the value of `
        split_value`.
    It does so by removing `split_upto`% of the string until there is a
        match. or return no match.

    `split_upto` specifies the size after which it will stop splitting.

    :param split_value: [String]
    :param splitting_string: [description]
    :param split_upto: [float], defaults to 0.5
    :param split_bins: [int] the number bins inside which each `split_value`
         will fall under.
                               eg.
                                   split_value = "Deep Learning Techniques for
                                       ASD Diagnosis and Rehabilitation'"
                                   split_bins=3,
                                   split_upto=0.5
                                   then the text will be checked for matches
                                       against :
                                       - ['Deep Learning Techniques for','Deep
                                           Learning Techniques for ASD','Deep
                                           Learning Techniques for ASD
                                           Diagnosis','Deep Learning Techniques
                                            for ASD Diagnosis and' ....]
                                       - The purpose of doing this is to ensure
                                            a partial match of a string can
                                           help extract the split text
    :returns splitted_text : List[String] : [s1,s2] or []
    """
```

```
    split_value = split_value.split(' ') # This make it remove words instead
        of the characters.
    sb = [i for i in range(split_bins)]
    split_mul = (1-split_upto)/split_bins
    # Spread the `split_bins` according to the how much the split needs to
       happen.
    split_range = [1-float((i)*split_mul) for i in sb]
    # index at which the new `split_value` will be determined. Order is
       descending to ensure largest match.
    slice_indices = [int(len(split_value)*split_val) for split_val in
       split_range]
    # creates the split strings.
    split_values_to_checks = [' '.join(split_value[:index]) for index in
       slice_indices]

    for split_val in split_values_to_checks:
        if split_val == '': # In case of empty seperator leaave it.
            continue
        current_text_split = splitting_string.split(split_val)
        if len(current_text_split) > 1:
            return current_text_split

    return []

class LatexInformationParser(object):
    """LatexInformationParser

    This is the parent class responsible for extraction of Processed
       information from
    Latex Based Documents. Process follows the below steps:

    ## `section_extraction`
    Use the `section_extraction` method to extract the document information
    sections from the single/document Latex setup. This returns a Sequential
        Tree like structure with sub sequences.
    This will use `tex2py` like functions to extract the document structure
       from the tex documents.

    ## `text_extraction`:
    Will extract text from the Latex File. uses `opendetex` to extract the
       text from latex.

    ## `collate_sections` :
```

```
    This will collate the information text and sections extracted based on
        the strategy of the extraction.

    """
    max_section_limit = 30 # Maximum number of sections to allow for
        extraction
    def __init__(self,max_section_limit=20,detex_path=None):
        self.max_section_limit = max_section_limit
        self.text_extractor = LatexToText(detex_path=detex_path)


    def section_extraction(self,tex_file_path) -> List[Section]:
        raise NotImplementedError()

    @staticmethod
    def get_subsection_names(tex_node):
        subsections = []
        try:
            subsections = list(tex_node.subsections)
            subsections = [i.string for i in subsections]
        except:
            pass
        return subsections


    def text_extraction(self):
        raise NotImplementedError()

    def collate_sections(self):
        raise NotImplementedError()

    def from_arxiv_paper(self,tex_files):
        raise NotImplementedError()


class SingleDocumentLatexParser(LatexInformationParser):
    def __init__(self, max_section_limit=20,detex_path=None):
        super().__init__(max_section_limit=max_section_limit,detex_path=
            detex_path)

    def section_extraction(self,tex_file_path) -> List[Section]:
        tex_node = get_tex_tree(tex_file_path)
        if len(tex_node.branches) > self.max_section_limit:
```

```
            raise MaxSectionSizeException(len(tex_node.branches),self.
                max_section_limit)

        sequential_sections = []
        for node in tex_node:
            curr_section = Section(str(node))
            subsections = self.get_subsection_names(node)
            curr_section.subsections = [Section(ss) for ss in subsections]
            sequential_sections.append(curr_section)

        return sequential_sections

    def text_extraction(self,latex_path):
        return self.text_extractor(latex_path)


    @staticmethod
    def split_and_find_section(curr_text,curr_sec_name,prev_section,
        split_upto=0.2,split_bins=10):
        """split_and_find_section
        Helps Recurrsively/iteratively split a Latex document for the Section
             list found from
        `section_extraction`. Splits a text string based on `curr_sec_name`
            and then allocates the
        split[0] to the `prev_section` object.

        :type curr_text: [String]
        :type curr_sec_name: [String]
        :type prev_section: [Section]
        :type split_upto : refer `split_match`
        :type split_bins : refer `split_match`
        :returns curr_text,Find_status : After removal of redundant section
            post splitting.
                                        status points to weather it could set
                                            text of the
                                        `prev_section` object
        """
        current_text_split = split_match(curr_sec_name,curr_text,split_upto=
            split_upto,split_bins=split_bins)
        # print("Found Splits,",curr_sec_name,len(current_text_split))
        if len(current_text_split) == 0:
            # This means no splits were found
            return curr_text,False
```

```
    portion_before_section = current_text_split[0]

    if prev_section is not None:
        prev_section.text = portion_before_section
        # print(ss.name,"added To Section ",prev_section.name,len(
            prev_section.text))
    portion_after_section = current_text_split[1:]
    curr_text = ''.join(portion_after_section)
    return curr_text,True


def collate_sections(self,paper_text,section_list:List[Section],
    split_upto=0.2,split_bins=10):
    """collate_sections
    Gets Latex compiled text string and
    then uses the found section based hierarchy from Latex
    to fill Text content of the `Section` objects which were discovered
        in
    `section_extraction`

    :param paper_text: text in string of from text_extraction]
    :type section_list: List[Section]
    :return: List[Section] : filled with text attributed
    """
    current_text_split = []
    prev_section = None
    curr_text = str(paper_text)
    unfound_sections = []
    some_section_not_found = False
    for index,s in enumerate(section_list):
        curr_text,section_status = self.split_and_find_section(curr_text,
            s.name,prev_section,split_upto=split_upto,split_bins=
            split_bins)
        if not section_status: # If couldn't match section add it here.
            some_section_not_found = True
        # print('\n\t'+s.name)
        prev_section = s
        for ss in s.subsections:
            curr_text,section_status = self.split_and_find_section(
                curr_text,ss.name,prev_section,split_upto=split_upto,
                split_bins=split_bins)
            if not section_status:
                some_section_not_found = True
                # print("Cannot Match For :",ss.name)
```

```
                prev_section = ss
                # print('\n\t\t'+ss.name)
            if index == len(section_list)-1:
                s.text = curr_text
        return section_list,some_section_not_found

    def from_arxiv_paper(self,tex_files:List[str],
        lowest_section_match_percent=0.2,number_to_tries=10):
        """from_arxiv_paper
        Extract Parsable section array from Arxiv Latex Paper which only have
             one paper will all the sections in it.
        :param lowest_section_match_percent [float]: % of the section heading
             that mimimum matches to create the split in text for parsing.
        :param paper: [ArxivPaper]
        :param number_to_tries: [float], number of different matches to make
            with mimimum match strings
        :return: Tuple (
            found_sections:List[Section],
            some_section_not_found:bool,
            latex_files_status_dict:dict
        )
        """
        largest_file = None
        max_size = 0
        file_results = {}
        for file_path in tex_files:
            if os.path.getsize(file_path) > max_size:
                largest_file = file_path
        latex_path = largest_file
        sections = self.section_extraction(latex_path)
        tex_in_text = self.text_extraction(latex_path)
        sections,some_section_not_found = self.collate_sections(tex_in_text,
            sections,split_upto=lowest_section_match_percent,split_bins=
            number_to_tries)
        file_results[latex_path] = some_section_not_found
        return sections,some_section_not_found,file_results

class MultiDocumentLatexParser(SingleDocumentLatexParser):
    def __init__(self, max_section_limit=20, detex_path=None):
        super().__init__(max_section_limit=max_section_limit, detex_path=
            detex_path)
```

```
    def from_arxiv_paper(self,tex_files:List[str],
        lowest_section_match_percent=0.2,number_to_tries=10):
        """from_arxiv_paper
        Extract Parsable section array from Arxiv Latex Paper which only have
             one paper will all the sections in it.
        :param tex_files: [path to files]
        :param lowest_section_match_percent [float]: % of the section heading
             that mimimum matches to create the split in text for parsing.
        :param number_to_tries: [float], number of different matches to make
            with mimimum match strings
        :return: Tuple (
            found_sections:List[Section],
            some_section_not_found:bool,
            latex_files_status_dict:dict
        )
        """
        collected_sections = []
        snf = False
        file_results = {}
        for latex_path in tex_files:
            try:
                sections = self.section_extraction(latex_path)
                tex_in_text = self.text_extraction(latex_path)
                sections,some_section_not_found = self.collate_sections(
                    tex_in_text,sections,split_upto=
                    lowest_section_match_percent,split_bins=number_to_tries)
                file_results[latex_path] = True
                if some_section_not_found:
                    snf = some_section_not_found
                collected_sections+=sections
            except:
                file_results[latex_path] = False
        return collected_sections,snf,file_results

class ArxivLatexParser():
    """
    Parses Arxiv Latex Documents with `LatexInformationParser` according
        - Based on Number of latex Pages (Chooses `SingleDocumentLatexParser`
             | `MultiDocumentLatexParser`)
            - Parsing is Ment to create Tree Like `Section` Datastructures.

    :return: tuple(
        collected_sections:List[Section],
        some_sections_failed:bool,
```

```
    file_results:dict
)
"""
parsing_result_name = 'Symantic Parsing Result'

def __init__(self,max_section_limit=20, detex_path=None):
    self.single_doc_parser = SingleDocumentLatexParser(max_section_limit=
        max_section_limit, detex_path=detex_path)
    self.multi_doc_parser = MultiDocumentLatexParser(max_section_limit=
        max_section_limit, detex_path=detex_path)

def __call__(self,tex_files:List[str],lowest_section_match_percent=0.2,
    number_to_tries=10):
    selected_parser = None
    if len(tex_files) == 0 :
       return None

    # Selected Parser for Latex based On Size.
    if len(tex_files) >=4:
        selected_parser = self.multi_doc_parser
    elif len(tex_files) >=1:
        selected_parser = self.single_doc_parser

    # Run the parser and see the results.
    try:
        collected_sections,some_sections_failed,file_results =
            selected_parser.from_arxiv_paper(tex_files,
            lowest_section_match_percent=lowest_section_match_percent,
            number_to_tries=number_to_tries)
        return collected_sections,some_sections_failed,file_results
    except Exception as e:
        return None
```

A.3 Heuristic Mapping Values For Serialisation of Research Object

```
  INTRODUCTION_SEARCH_CONSTS = [
    "introduction",
    "preliminaries",

]
RELATED_WORKS_SEARCH_CONSTS = [
    "problem statement",
    "background",
    "related work",
    "problem definition",
    "literature review",

]
METHODOLOGY_SEARCH_CONST = [
    "methodology",
    "method",
    "problem formulation",
    "approach",
    "proposed method"
    "implementation",
    "our approach",
    "proposed algorithm",
    "system design",
    "architecture",

]
DATA_SEARCH_CONST = [
    "dataset",
    "data"
]

EXPERIMENTS_SEARCH_CONSTS = [
    "evaluation",
    "experiment",
]
RESULTS_SEARCH_CONSTS = [
    'result',
    "analysis"
]

CONCLUSION_SEARCH_CONSTS = [
```

```
    "discussion",
    "conclusion",
    "future work",
    "concluding remarks"
]

LIMITATIONS_SEARCH_CONSTS = [
    "limitations",
    "ablation study",

]

STRICT_CHECK_CONSTS = [
    "data", # These key words need strict "==" checking to ensure the match
        name
    "method",
    "approach"
]
```

APPENDIX B

TABLE OF COMPARISON CLASSIFICATION

B.1 Hyper Parameters For ML Training

Model Name	# Layers	Embedding Size	Batchsize	Learning Rate	# Parameters	Size In MB	# Epoch
CCT (31M)	6 Layers Per Cross Channel, 6 Layers Per Vanilla Encoder	256	8	2e-05	31M	140	20
E-MCT (25M)	8 Vanilla Encoder	256	8	2e-05	25M	100	20
Caption Only Encoder Transformer (14.5M)	8 Vanilla Encoder	256	8	2e-05	14.5M	58	20
SciBert Fine-tuned	12 Vanilla Encoder Layers	768	8	2e-05	109 M	440	6

Table 6. Hyper Parameters For Training Table Classification Models. All Learning rates are scheduled using a cosine learning rate scheduler with a warm up of 20 epochs.

B.2 Table Type Examples

The examples of the types of tables labeled during the labeling processs are given in Figure 30, 31, 32, 33, 34, 35.

Table 1: Key statistics of Quoref splits.

nan	Train	Dev.	Test
Number of questions	19399	2418	2537
Number of paragraphs	3771	454	477
Avg. paragraph len (tokens)	384±105	381±101	385±103
Avg. question len (tokens)	17±6	17±6	17±6
Paragraph vocabulary size	57648	18226	18885
Question vocabulary size	19803	5579	5624
% of multi-span answers	10.2	9.1	9.7

Source: From Dasigi et al. 2019

Figure 30. Table Type : Dataset Description.

TABLE III: Ablation studies on the impact of each system component

Method	Properties	Properties	Properties	Travelling Dist =2km	Travelling Dist =2km	Travelling Dist =2km	Travelling Dist =2km	Travelling Dist =2km	Travelling Dist =2km
Method	nan	nan	nan	Longitudinal Error (m)	Longitudinal Error (m)	Longitudinal Error (m)	Lateral Error (m)	Lateral Error (m)	Lateral Error (m)
Method	Lane	GPS	Sign	Median	95%	99%	Median	95%	99%
Lane	yes	no	no	13.45	37.86	51.59	0.20	1.08	1.59
Lane+GPS	yes	yes	no	1.53	5.95	6.27	0.06	0.24	0.43
Lane+Sign	yes	no	yes	6.23	31.98	51.70	0.10	0.85	1.41
All	yes	yes	yes	1.12	3.55	5.92	0.05	0.18	0.23

Source: From Ma et al. 2019

Figure 31. Table Type : Ablation Study.

Table 4: Typical patterns captured by shared layer and task-specific layer of SP-MTL and ASP-MTL models on Movie and Baby tasks.

Model	Shared Layer	Task-Movie	Task-Baby
SP-MTL	good, great bad, love, simple, cut, slow, cheap, infantile	good, great, well-directed, pointless, cut, cheap, infantile	love, bad, cute, safety, mild, broken simple
ASP-MTL	good, great, love, bad poor	well-directed, pointless, cut, cheap, infantile	cute, safety, mild, broken simple

Source: From Liu, Qiu, and Huang 2017

Figure 32. Table Type : Method Examples.

Table 1: Notations

T,K	Number of temporal frames and ball states
It	Image evidence at time t
Xt,St	Discrete location and state of the ball at time t
Pt	3D coordinates of the ball at time t
i,j,k,l	Node indices in the ball or players graph
Vb,Vp	Sets of nodes in ball and player graphs
Eb,Ep	Sets of edges in the ball and player graphs
xi,si,ti	Discrete location, state, and time of node i
Sb	Special node for the ball at t=0
Sp,Tp	Source and sink nodes of player trajectories
fji,pji	Number of balls and players moving from i to j
cjbi,cjpi	Ball and player transition costs from i to j

Source: From Maksai, Wang, and Fua 2016

Figure 33. Table Type : Symbolic Parameter Description.

TABLE II: TIMELY parameters.

Component	Params	Spec	Energy (fJ)	Area (μm2)
nan	nan	nan	/compo.	/compo.
TIMELY sub-Chip	TIMELY sub-Chip	TIMELY sub-Chip	TIMELY sub-Chip	TIMELY sub-Chip
DTC	resolution	8 bits	37.5	240
DTC	number	16×32	37.5	240
ReRAM	size	256×256	1792	100
crossbar	number	16×12	1792	nan
nan	bits/cell	4	1792	nan
Charging+	number	12×256	41.7	40
comparator	number	12×256	41.7	nan
TDC	resolution	8 bits	145	310
TDC	number	12×32	145	nan
X-subBuf	number	12×16×256	0.62	5
P-subBuf	number	15×12×256	2.3	5
I-adder	number	12×256	36.8	40
ReLU	number	2	205	300
MaxPool	number	1	330	240
Input buffer	size/number	2KB/1	12736	50
Output buffer	size/number	2KB/1	31039	50
Total	nan	nan	nan	0.86 mm2
TIMELY chip (40 MHz)	TIMELY chip (40 MHz)	TIMELY chip (40 MHz)	TIMELY chip (40 MHz)	TIMELY chip (40 MHz)
sub-Chip	number	106a	nan	0.86 mm2
Total	nan	nan	nan	91a mm2
Inter chips	Inter chips	Inter chips	Inter chips	Inter chips
Hyper link	links/freq	1/1.6GHz	1620	5.7 mm2
Hyper link	link bw	6.4 GB/s	nan	5.7 mm2

Source: From Maksai, Wang, and Fua 2016

Figure 34. Table Type : HyperParameter Description.

Table 2: Our distortions. Our traditional distortions (left) are performed by basic low-level image editing operations. We also sequentially compose them to better explore the space. Our CNN-based distortions (right) are formed by randomly varying parameters such as task, network architecture, and learning parameters. The goal of the distortions is to mimic plausible distortions seen in real algorithm outputs.

Sub-type	Distortion type
Photometric	lightness shift, color shift, contrast, saturation
nan	uniform white noise, Gaussian white, pink,
Noise	& blue noise, Gaussian colored (between
nan	violet and brown) noise, checkerboard artifact
Blur	Gaussian, bilateral filtering
Spatial	shifting, affine warp, homography,
nan	linear warping, cubic warping, ghosting,
nan	chromatic aberration,
Compression	jpeg

Source: From Zhang et al. 2018

Figure 35. Table Type : Parameter Description.