Information Extraction of Technical Details From Scholarly Articles

Kulendra K. Kaushal

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Science and Applications.

Naren Ramakrishnan, Chair
Patrick Butler
Chang-Tien Lu

May 12, 2021
Blacksburg, Virginia

Keywords: Information Extraction, Long Documents, Research Articles, Named Entity Recognition, Hardware Resources, Compute Platform, Programming Language and Libraries, etc.

Copyright 2021, Kulendra K. Kaushal
Researchers have made significant progress in information extraction from short documents in the last few years, including social media interaction, news articles, and email excerpts. This research aims to extract technical entities like hardware resources, computing platforms, compute time, programming language, and libraries from scholarly research articles. Research articles are generally long documents having both salient as well as non-salient entities. Analyzing the cross-sectional relation, filtering the relevant information, measuring the saliency of mentioned entities, and extracting novel entities are some of the technical challenges involved in this research. This work presents a detailed study about the performance, effectiveness, and scalability of rule-based weakly supervised algorithms. We also develop an automated end-to-end Research Entity and Relationship Extractor (E2R Extractor). Additionally, we perform a comprehensive study about the effectiveness of existing deep learning-based information extraction tools like Dygie, Dygie++, SciREX. The research also contributes a dataset containing novel entities annotated in BILUO format and represents the baseline results using the E2R extractor on the proposed dataset. The results indicate that the E2R extractor successfully extracts salient entities from research articles.
Information Extraction of Technical Details From Scholarly Articles

Kulendra K. Kaushal

(GENERAL AUDIENCE ABSTRACT)

Information extraction is a process of automatically extracting meaningful information from unstructured text such as articles, news feeds and presenting it in a structured format. Researchers have made significant progress in this domain over the past few years. However, their work primarily focuses on short documents such as social media interactions, news articles, email excerpts, and not on long documents such as scholarly articles and research papers. Long documents contain a lot of redundant data, so filtering and extracting meaningful information is quite challenging. This work focuses on extracting entities such as hardware resources, compute platforms, and programming languages used in scholarly articles. We present a deep learning-based model to extract such entities from research articles and research papers. We evaluate the performance of our deep learning model against simple rule-based algorithms and other state-of-the-art models for extracting the desired entities. Our work also contributes a labeled dataset containing the entities mentioned above and results obtained on this dataset using our deep learning model.
Dedication

To my beloved parents Dr. Jitendra Kumar, Dr. Neelam Kumari and my sweet sisters
Sudarshana Sharma, Shilpy Shaloni and Shivani Shubham.
Acknowledgments

I would like to start by thanking my advisor Dr. Naren Ramakrishnan for his guidance, support, and efforts without which this research would not have been possible. Your wise words and constant motivation and feedback helped me throughout my Master’s journey. I also thank Dr. Lu and Dr. Butler for their time and valuable suggestions. I would also like to thank CSET research group who allowed me the opportunity to work on this project.

I am indebted to the Computer Science (CS) Department, Statistics Department, and Sanghani Center for Artificial Intelligence and Data Analytics for funding me through my Master’s degree, and all of the staff of the CS Department and Virginia Tech for assisting me when needed.

I would like to express my gratitude to Dr. Sathappan Muthiah for his ideas, support, and for helping me get started on this research. I would also thank Raquib Bin Yousuf for helping me when needed. I am grateful to all of the members of the Discovery Analytic Centre (DAC). I am thankful to my family and friends in India as well as my US friends turned family – Mayank Kishore, Aarohi, Rutwik, and Nihar – for supporting and encouraging me throughout my journey.
Contents

List of Figures .......................................................... ix

List of Tables .......................................................... xi

1 Introduction .......................................................... 1

  1.1 Problem Statement ................................................. 1
  1.2 Motivation .......................................................... 1
    1.2.1 Compute Platform ......................................... 2
    1.2.2 Power Consumption Using Compute Time ............... 2
    1.2.3 Hardware Resources, Programming Languages And Libraries . . . . . . . . . . . . 3
  1.3 Challenges .......................................................... 3
  1.4 Proposed Solution ................................................ 4

2 Review of Literature ................................................. 8

  2.1 Traditional Approaches .......................................... 9
    2.1.1 Rule-Based Approaches .................................... 9
    2.1.2 Unsupervised Learning Based Approaches ............. 10
    2.1.3 Feature-based Supervised Learning Approaches ....... 10
  2.2 Deep Learning Based Approaches ............................. 12
3 Methods

3.1 Weakly Supervised Algorithm for Hardware Extraction

3.1.1 Rules to Extract Patterns and Candidate Seed Words

3.1.2 Dynamic Scoring Model

3.2 Sequential E2R: End To End Research Entity And Relationship Extractor

3.2.1 Binary Sentence Classifier

3.2.2 K-Means Clustering

3.2.3 Named Entity Recognition (NER) Model

3.3 Existing State-of-the-Art Models And Their Architectural Comparison With E2R

3.3.1 DyGIE and DyGIE++

3.3.2 SciREX

3.3.3 E2R and Scirex Comparison

4 Results and Discussion

4.1 Dataset

4.1.1 arXiv Dataset
4.1.2 HPTLL Dataset ............................................. 31
4.1.3 SciREX Dataset ............................................. 32
4.2 Qualitative Analysis of Weakly Supervised Model ...................... 33
4.3 Sequential End To End Research Entity And Relationship Extractor (E2R) . 34
  4.3.1 Binary Sentence Classification ................................ 34
  4.3.2 K-Means Clustering ...................................... 35
  4.3.3 NER Extraction ............................................ 36
4.4 Sequential E2R on SciREX data and comparison with SciREX data .... 38

5 Conclusion .................................................. 41

Bibliography .................................................. 43

Appendices .................................................. 50

Appendix A Dataset Annotation Guideline .................................. 51
  A.1 Dataset Construction ........................................ 51
  A.2 Annotation Agreement ...................................... 51

Appendix B Second Appendix ........................................ 53
  B.1 E2R Extractor .............................................. 53
  B.2 Downloading Trained Models ................................ 53
  B.3 Using E2R Extractor ...................................... 54
List of Figures

1.1 Example PDF Input to E2R with programming library entity highlighted in green color ........................................ 6

1.2 Output of E2R in JSON format for input file 1910.13025 .................. 7

2.1 Related works in the field of information extraction ......................... 9

3.1 Weakly Supervised Algorithm Output ....................................... 19

3.2 Architecture Diagram For Bootstrapping Algorithm ..................... 20

3.3 Sequential E2R: Takes research article as input, extracts sentences having entities, filters out the salient sentences and extracts text spans having entities. 22

3.4 Architecture of the Binary Sentence Classifier ............................. 23

3.5 Architecture of the NER component of E2R model ....................... 24

3.6 Architecture of the SciREX model .......................................... 26

3.7 Fine-grained annotation requirement of the SciREX model ............... 27

3.8 SciREX vs E2R annotation requirement .................................... 28

4.1 Count of full-text research articles in different branches of Computer Science namely Machine Learning (LG), Hardware Architecture (AR), Computer Vision (CV), Artificial Intelligence (AI), Information Extraction (IR) .... 30

4.2 Grobid output format for the arXiv research articles ................. 31
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.3</td>
<td>Count of full-text research articles published over the period of time</td>
<td>32</td>
</tr>
<tr>
<td>4.4</td>
<td>Weakly Supervised Algorithm Output at each iteration for one paper</td>
<td>33</td>
</tr>
<tr>
<td>4.5</td>
<td>Silhouette Score for the hardware entity type after dimensional reduction</td>
<td>35</td>
</tr>
<tr>
<td>4.6</td>
<td>Silhouette Score for the language entity type after dimensional reduction</td>
<td>36</td>
</tr>
<tr>
<td>4.7</td>
<td>Silhouette Score for the Library entity type after dimensional reduction</td>
<td>37</td>
</tr>
<tr>
<td>4.8</td>
<td>Cluster Saliency Analysis</td>
<td>38</td>
</tr>
</tbody>
</table>
List of Tables

4.1 Entity Type and their distribution in HPTLL ........................................... 32
4.2 E2R’s Binary Sentence Classifier F1 Score ............................................... 34
4.3 E2R: Entity and their F1 Scores for each of the tag ................................. 39
4.4 E2R: Overall Performance Metric Scores ............................................... 39
4.5 E2R Performance on predicting the individual entities from the SciREX dataset. 39
4.6 Overall Performance comparison of E2R and SciREX .............................. 39
List of Abbreviations

E2R  End To End Research Entity And Relationship (E2R)

HPTLL  Hardware Resources Compute Platform Time Language and Library

JSON  JavaScript Object Notation

E2R is our deep learning based fully automated End To End Research Entity And Relationship (E2R) Extractor for long documents.

HPTLL is our proposed annotated dataset. It contains Hardware resources like GPU, RAM, CPU etc., Compute Platform like Cluster, Desktop etc., Compute Time of algorithm, Programming Language like Python, Java, C++ etc, Programming Library like PyTorch, Torch, Caffee, MXNet etc.as entities.
Chapter 1

Introduction

1.1 Problem Statement

Texts from different sources like research articles, the world wide web, and news articles generally contain essential information in an unstructured format. Manually extracting crucial information from these sources involves a lot of human effort. Information extraction is a vital task in Natural Language Processing (NLP) that extracts relevant information from unstructured textual data with the help of powerful computational machines. Information extraction involves encoding texts, identifying sections where relevant information is present, and then extracting the required information in a structured format. Research publications, patents, and other scholarly articles are rich and impactful sources of information. This work focuses on extracting technical entities such as computational platform, hardware resources employed in the research, compute time of the algorithm, and its language/library dependencies.

1.2 Motivation

Analysis of full-text scholarly documents, at scale, would help understand the emerging technologies and their prevalence. The following subsections describe the need to extract each of the aforementioned entities in detail.
1.2.1 Compute Platform

Various machine learning or deep learning algorithms need different compute platforms to run and produce the desired results. Extracting these platforms from a wide range of research articles gives us a mapping between deep learning algorithms and their computational needs. This information also gives a fair idea about the portability of algorithms and their use on different computing platforms to the researchers. For example, some algorithms easily run on portable IoT devices and embedded systems while some algorithms require clusters for their execution.

1.2.2 Power Consumption Using Compute Time

Researchers have shown that complex NLP models like GPT [42], BERT [10], Turing-NLG [43] with billions of parameters are generally more efficient in solving challenging problems like information extraction, machine translation, natural language generation. These complex models are trained on large datasets using powerful hardware resources for a long duration. However, there has always been a pertinent issue about the power consumed in training such complex models. Power consumption depends on the compute time of the algorithm. Extracting compute time would help us measure the power consumed by different algorithms using the formula proposed by Strubell et al. [49]. A fair estimate of power consumption would highlight the need for optimizing the NLP algorithms in terms of accuracy and power consumption.
1.3 Challenges

Extraction of the compute platform, programming language, library dependencies, and hardware resources involves multi fold challenges:

- Usually, the length of a research paper is between eight to fourteen pages, and extraction involves processing long documents. In the research articles, identifying the relevant information sections is challenging since a lot of data is irrelevant.

- Previous works have primarily focused on information extraction from shorter documents like news articles, the world wide web, users’ posts, and comments on different
social media platforms. Many publicly available labeled datasets contain this extracted information. This research aims to extract technical novel entities. Labeled data for such entities is very limited.

- In research articles, researchers often mention entities they have used and entities they have not used. *Salient entities* are entities that have been used in the research. The proposed solution should be able to identify the salient entities from all the mentioned entities. Figure 4.8 shows salient as well as non salient sentences.

- The solution must be scalable to run on millions of full-length papers.

### 1.4 Proposed Solution

This research comprehensively shows that different existing state-of-the-art solutions like DyGIE [28], DyGIE++ [53], and SciREX [18] fail to extract entities of our interest. We present a deep learning based automated tool named End To End Research Entity And Relationship (E2R) to extract compute platform, hardware resources, compute time, programming language, and programming library from long text documents. E2R uses Transformers (BERT) as context encoders and Conditional Random Fields (CRF) as tag decoders. SciBERT [3] extends the BERT model and has been mainly trained on the scientific text data. E2R uses transfer learning to generate scientific contextual embeddings from SciBERT. SciBERT embeddings allow E2R to learn a better representation of scholarly articles and capture cross-sentence contexts. Most of the prior research extracts generic named entities like the name of a person, geographic location, organization from short documents like social media posts, comments, tweets, news articles. To the best of our knowledge, SciREX [18] is the only model aiming to extract similar entities from long scholarly articles. Our solution, E2R, is lightweight as compared to SciREX because E2R requires coarse-grained or less annotation,
and SciREX requires fine-grained or more annotation.

E2R requires input in the JSON format and generates an object containing the extracted entities and their text spans. Figure 1.1 shows a sample input file and Figure 1.2 shows the output of E2R. Following are the research contributions of this thesis:

1. An extensive analysis of existing state-of-the-art solutions in solving the research problems mentioned in the section 1.1.

2. E2R: A deep learning based novel End To End Research Entity And Relationship (E2R) Extractor for long documents.

3. HPTLL: A novel Hardware Resources Compute Platform Time Language and Library (HPTLL) dataset containing 600 annotated sentences in BILUO format.

4. Extraction of the novel entities, viz compute platform, hardware resources, compute time, programming language, and library.

HPTLL dataset and the source codes of this work are available at https://github.com/DiscoveryAnalyticsCenter/csetproject/tree/master/hardware_language_library_extractor

Appendix A briefs about the annotation guideline followed and Appendix B contains a detailed guide to run the source code and reproduce the result.
where $s_{t,t}^k = (-1)^t s_0 \gamma^t$, $t \in \{1, -1\}$, $s_0$ is the initial stepsize, $\gamma < 1$ is the decrease ratio, and $v_{t,t}^{k+1} = \text{proj}(v^{k} + s_{t,t}^{k+1} d_t^k)$. If such a stepsize $s^k$ exists, we update $v^{k+1}$ by (4.6) and repeat the process. Otherwise, we record the number of failures and stop the algorithm when the number of failure is greater than a threshold.

The overall flow is summarized in Algorithm 4.1. In practice, instead of using the whole dataset to train this attack vector, we use a subset $D^b$. The impact for different number of samples is discussed in section 5.2.2.

5. Numerical Experiments. In this section, we show the power of active-subspace in revealing the number of active neurons, compressing neural networks, and computing the universal adversarial perturbation. All codes are implemented in PyTorch and are available online.

5.1. Structural Analysis and Compression. We test the ASNet constructed by Algorithm 3.1, and set the polynomial order as $p = 2$, the number of active neurons as $r = 50$, and the threshold in Equation (3.4) as $\epsilon = 0.05$ on default. Inspired by the knowledge distillation [28], we retrain all the parameters in the ASNet by minimizing the following loss function

$$
\min_{\theta} \sum_{t=1}^{m} \beta H(\text{ASNet}_\theta(x^t_0), f(x^t_0)) + (1 - \beta) H(\text{ASNet}_\theta(x^t_0), y^t).
$$

Figure 1.1: Example PDF Input to E2R with programming library entity highlighted in green color
Figure 1.2: Output of E2R in JSON format for input file 1910.13025
Chapter 2

Review of Literature

Information extraction is a multi-step process involving filtering relevant text and then extracting named entities from the unstructured text into a structured format. Named Entity was introduced in the MUC-6 (Sixth Message Understanding Conference) \cite{13} to define the task to extract person, geographic locations, organizations, currency, time, and percentage expressions. Since MUC-6, researchers have done much work in the field of information extraction and Named Entity Recognition (NER). NER methods can be broadly classified into two types based on the type of entities they are extracting, namely i) Generic: if extracting generic Named Entities like person, location \cite{47} \cite{16} \cite{30} ii) Domain-Specific: if extracting domain-specific entities like proteins, enzymes, hardware resources etc. \cite{50} \cite{9} \cite{54}. Further, named entity extraction can be broadly classified into traditional approaches, namely, i) Rule-Based, ii) Unsupervised Learning, iii) Feature-based Supervised Learning, and Deep Learning-based approaches. Figure 2.1 summarizes popular techniques used for information extraction in a tree diagram.
2.1 Traditional Approaches

2.1.1 Rule-Based Approaches

Rule-based approach is the easiest one to implement and gives an insight into the dataset. Researchers create hand-crafted rules using lexical patterns like capital letters, sequence of certain words, parts of speech tags like noun, verbs, etc., domain-specific rules to extract a predefined set of named entities. Humphreys et al. proposed one such hand-crafted rule-based named entity extractor known as LaSIE-II to extract generic NERs like a person, organization, locations, etc. [17]. In extracting named entities from electronic medical records, Quimbaya et al. proposed a named entity extractor tool using knowledge base or dictionary-based approaches. [40]. Rule-based approaches work when the set of syntactic-
lexicon patterns are exhaustive or small, but when the patterns are non-exhaustive, they perform poorly. Hence, rule-based systems often have high precision but low recall.

2.1.2 Unsupervised Learning Based Approaches

Unsupervised learning is based on the contextual similarity between named entities. In a large corpus, named entities have similar lexical patterns, lexical resources and often form well-separated clusters. Unsupervised approaches start from a set of seed words, seed rules to extract candidate named entities. Finally, candidate named entities are grouped using clustering for extraction. Etzioni et al. proposed such an unsupervised based system known as KNOWITALL to extract city, country, actor, film, and their relationships using the initial set of seven seed words. [12]. Further, to improve unsupervised learning based tools, disambiguation based on the threshold score of extracted named entities is used. Nadeau et al. proposed a tool that uses disambiguation in the unsupervised learning approach [35]. Disambiguation is often based on simple general or domain-based heuristics.

2.1.3 Feature-based Supervised Learning Approaches

Named Entity Recognition (NER) can be modeled as a multi-class classification problem or sequence labeling problem for supervised learning. Each training example is annotated carefully with a class, features are crafted, and a model is trained to learn such patterns in the features. Feature vectors are an abstract combination of parts-of-speech (POS) tags, word-level features such case, morphology, etc., converted as boolean or nominal values. A Few popular supervised learning approaches are Hidden Markov Models (HMM) [11], Decision Trees [41], Maximum Entropy Models [19], Conditional Random Fields (CRFs) [23] and Support Vector Machines [14]. Bikel et al. proposed the first hidden Markov model-based named
entity recognition tool named Nymble and IdentiFinderTM using a combination of text for English language and speech for Spanish data to extract names, percentage expressions, time, and other numerical values like a year, product code, etc. [4] [5]. Supervised learning-based approaches have the advantage that for a different combination of subsets of features, different models can be trained, and a majority voting algorithm can be used to obtain the best class or tag for the word or text spans. Borthwick et al. [7] used different knowledge sources like lexical features, capitalization features, and features denoting the current text type, i.e., header or body, to propose the Maximum Entropy-based Named Entity Recognition (NER) extractor called MENE. Support Vector Machine (SVM) learning based approach work on predicting the class of each participating token. Mayfield and McNamee used many computationally inexpensive binary features to design a language-independent SVM-based tool called SNOOD [32]. SNOOD comprises of multiple binary classifiers where each classifier predicts the class for each token out of a total of eight classes, i.e., B- (Beginning), I- (Inside) for PERSON, LOCATION, MIS, and ORGANIZATION tags [32]. The classifier using SVM ignores the information present in the neighboring tokens and is thus context-independent. To deal with the shortcomings of SVM-based approaches, McCallum and Li [31] proposed a conditional random field-based Feature Induction technique and Web-Enhanced Lexicons with an overall F-Score of 84.04% on CoNLL-2003 named entity recognition (NER) dataset. As an extension of CRFs based approaches, Krishnan and Manning describe an approach using two coupled CRF classifiers to incorporate information from non-local dependencies for extracting NERs [21]. The first CRF classifier uses local information to make predictions, and then the second CRF classifier uses the feature produced from the first CRF classifier and local information to predict tags for each token. CRF based tools has been widely used in predicting NERs in biomedical domains, chemical-related datasets, social media tweets [48] [27] [45].
2.2 Deep Learning Based Approaches

Deep Learning has gained immense popularity in solving problems related to Computer Vision (CV), robotics, Natural Language Processing (NLP), etc. Deep learning has multiple artificial neural network layers where forward pass calculates the weighted sum of inputs from different layers. The output is then passed through nonlinear activation functions. Backward pass computes the gradient of an objective function concerning weights of different layers using the chain rule of derivatives [24]. Deep learning-based NER extraction approaches have multiple advantages over traditional machine learning-based approaches. i) Deep learning layers perform nonlinear transformation and hence compute the nonlinear mapping between input and output NER tags. Compared to traditional approaches that compute the linear mapping between input and output (e.g., linear CRFs, linear HMMs), deep learning-based approaches can learn hidden and intricate features naturally, making them more efficient. ii) With deep learning, we do not need to design hand-crafted features and it is thus superior to traditional approaches as it takes less time while traditional methods take time and no domain expertise is required. iii) Deep learning paradigm allows us to design more complex models by increasing the number of layers and parameters that can capture context more efficiently when trained on a large dataset. Deep Learning-based approaches comprise three components: input representation for representing the data like word-level representation, character level representation, hybrid representation, context encoders like CNN, RNN, GRUs, Transformers, and tag decoders, e.g., Multi-Layer Perceptron + Softmax, Conditional Random Fields (CRFs), RNNs.
2.2. DEEP LEARNING BASED APPROACHES

2.2.1 Input Representation

Word Level Representations

Word level representations are pre-trained embeddings for different words and are learned using different unsupervised algorithms like Continuous Bag of Words (CBOW) models, N-grams model, Continuous skip-gram models. Google proposed CBOW and Skip-gram based Word2Vec embeddings [34] [33], Facebook proposed N-gram based fastText embeddings [6] and Stanford research group proposed Global Vectors (GloVe) embeddings [36] trained using large dataset with more than billion words collected from different sources. Word2Vec considers each word as an atomic unit and learns the vector representation for the entire word, while fastText breaks the word into multiple subwords and learns vector representations for each subword. Both word2vec and fastText have been effective in getting accurate word representations depending upon the use cases. GloVe uses both local context window, i.e., a bag of words approach, and global matrix factorization to find relative term frequencies for computing word embeddings.

Character Level Representations

Character-based representations are useful because they can learn representations for new or unseen words and transfer information of morpheme-level regularities. CNN’s [29] and RNNs [25] based models are used to obtain character-level word embeddings for extracting named entities. Peters et al. [38] proposed character-level word representation known as Embeddings from Language Models (ELMo), where word representations are learned from the top two layers of CNN-based deep bidirectional language model (biLM). Two types of RNNs, Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRUs), are used for extracting character level representations. Kuru et al. [22] proposed a methodology
CHAPTER 2. REVIEW OF LITERATURE

for tagging at a character-level for language-independent Named Entity Recognition (NER) based on stacked bidirectional Long Short Term Memory (LSTM) model. This work proposes considering the sentence as a sequence of characters instead of words as an input to the model.

Hybrid Level Representations

Hybrid representation of input is a combination of character-level encoding, word-level encoding, and other features like lexical similarity, parts of speech tags, visual features, etc.

2.2.2 Context Encoders

Context encoders are used to capture context dependencies in the input representation using CNNs, RNNs, language models, and transformers.

Convolutional Neural Networks

A sentence can be considered a sequence of words; the embedding for each word can be obtained using standard techniques of the word representation. Convolutional layers are used to learn local features from these word embeddings, and max-pooling or average-pooling is used to generate a context-dependent global input feature vector. Collobert et al. [8] used CNN to learn task-independent, adequate internal representations of sentences as input feature vectors to perform different NLP tasks, e.g., POS tagging, NER detection, etc.

Recurrent Neural Networks

Bidirectional RNNs can retain past and future information using forward and backward states and thus better capture the context from the whole sentence than CNN. LSTMs and
GRUs have been used to learn sentence-level feature vectors for performing different tasks like chunking, POS tagging, NER detection. Huang et al. [16] were among the first few researchers who used RNN to generate sentence-level contextual feature vectors. Huang et al. [16] used a combination of LSTM, LSTM with a CRF, bidirectional LSTM, and bidirectional LSTM with CRF layer to perform sequence tagging task.

Natural Language Models

Language models are used to generate sequences. Given a sequence of tokens \((t_1, t_2, t_3, ..., t_n)\), for any given position \(k\), the language model generates two contextual embeddings, i.e., forward and backward, and combines them to get the final embedding. Forward embedding is generated by modeling the probability of token \(t_k\) given the previous of tokens \((t_1, t_2, t_3, ..., t_{k-1})\) and backward embedding is generated by modelling the probability of token \(t_k\) given the future context of tokens \((t_{k+1}, t_{k+2}, t_{k+3}, ..., t_n)\). [37]. Many researchers have used language model and empirically verified its efficiency in solving standard NLP problems like NER detection, sentiment analysis etc. [37] [38] [44] [26]. The language models have an advantage that they can also learn task-specific knowledge. Researchers have used multi-task learning to learn task-specific contextual embeddings using language models. [44].

Transformers

Language models are based on CNNs and RNNs. In 2017, Vaswani et al. [52] proposed a new architecture solely based on attention mechanism, avoiding CNN and RNN altogether. This work has shown the superiority of transformers in solving complex problems like machine translation compared to existing language models. The proposed model is more parallelizable
and needs significantly less time to train. Radford et al. [42] from the OpenAI community proposed a transformer-based Generative Pre-trained Transformer (GPT) model for different language understanding tasks. The proposed model has a two-step training paradigm. In the first training stage, the GPT model learns initial parameters, and in the second stage, it learns task-specific contextual embeddings [42]. Later, researchers from Google led by Devlin et al. [10], propose a transformer-based Bidirectional Encoder Representations from Transformers (BERT) model. BERT uses conditions jointly learning context from both sides, i.e., beginning and end of the document, while GPT learns context from left to right in the document. [10]. Transformer based contextual embeddings like BERT, GPT, ELMO has been effective as compared to traditional embeddings like Word2Vec, GloVe [10] [42] because transformers can learn task-independent contextual embeddings from unlabelled data and then can be adapted or fine-tuned using additional task-dependent layers and objective functions.

2.2.3 Tag Decoders

Tag decoders are the last component in the named entity detection. Tag decoders take deep contextual embeddings as an input and produce final tags as an output for each token. Typical tag decoders are based on Multi-Layer Perceptron + Softmax, Conditional Random Fields (CRFs), and Recurrent Neural Networks (RNNs).

Multi-Layer Perceptron + Softmax

Identification of named entities can be modeled as a multi-class classification task where multi-layer perceptron and softmax classifier takes contextual embeddings as an input and predicts tags for each of the words independent of neighboring tokens. Tomori et al. [51]
modeled the moves in chess as a NER task and used multi-layer perceptron and softmax-based tag decoder to predict the next valid moves. The right moves are represented by the BIO tag scheme [51].

**Conditional Random Fields**

Conditional Random Field-based (CRF-based) decoder is the most popular choice tag decoder and has been used to obtain the state of the results on different datasets like CoNLL03 and OntoNotes [47] [39] [15]. Viterbi algorithm is used along with CRF to come up with tags for each of the tokens.
Chapter 3

Methods

3.1 Weakly Supervised Algorithm for Hardware Extraction

A set of seed words for each of the entity types such as hardware resources, compute platform is created. Seed words for computational platform are {Server, CPU Cluster, Standard PC, Workstation}; Set of seed words for language/library dependencies are {Matlab, Pytorch, Tensorflow, Caffee, Scikit} and set of seed words for compute time and resources consumed are {Average time, running time, GPU, CPU, NVIDIA, Intel Xeon, Cuda, GFlops, memory}. The sentences having an initial set of seed words are filtered, and then Natural Language Tool Kit (NLTK) is used to tokenize, lemmatize and remove stop words from the sentences. Figure 3.1 shows the example sentences to demonstrate that Parts of Speech (POS) tags can be used to extract salient features. In the first example, the verb is “trained”, indicating that the hardware “GPU” has actually been used and not just mentioned, and duration was “3 months”. Similarly, for the second example, “collected” is the verb showing that the hardware “Intel Xeon CPU” present in the computing platform “socket server” has been used. The algorithm extracts patterns using the POS tags and handcrafted rules. The algorithm identifies new candidate seed words based on these patterns, and a score is assigned to each of those candidate seed words. If the score is over a certain threshold, the candidate seed
3.1. Weakly Supervised Algorithm for Hardware Extraction

A word is appended to the original set of seed words, else discarded. Figure 3.2 shows the architecture diagram of our pipeline.

```
VGG/NNP was/VBD trained/VBN for/IN around/IN `\`` 3/CD months/NNS for/IN a/DT single/JJ GPU/NNP time/NN ./.

The/DTD traces/NNS are/VBP collected/VBN on/IN a/DT dual/JJ -: socket/NN server/NN CPU/NNP (/NNP Intel/NNP Xeon/NNP E5/NNP -/: 2650/CD )

A/DT multicore/JJ CPU/NNP optionally/RB paired/VBD with/IN GPU/NNP is/VBZ equipped/VBN with/IN one/CD APELink/NNP board/NN and/CC made/VBD into/IN a/DT node/NN of/IN the/DTD cluster/NN.

The/DTD delivery/NN of/IN RDMA/NNP events/NNS by/IN APELink/NNP hardware/NN in/IN CPU/NNP memory/NN

Using/VBG Torch/NNP 7/CD ,/, the/DTD architectures/NNS in/IN this/DTD section/NN apply/RB the/DTD latest/JJS techniques/NNS in/IN deep/NN learning/NN to/TO check/VB their/PRP$ synergy/NN with/IN COOL/NNP
```

Figure 3.1: Weakly Supervised Algorithm Output

3.1.1 Rules to Extract Patterns and Candidate Seed Words

Entities are proper nouns, often present around other nouns. To further extract salient entities from all the mentioned entities, a set of verbs trained, collected, used, increased, shows and a set of adjectives high-end, multicore are used. The algorithm uses following rules:

- **Rule 1**: If a verb from the predetermined set connects to a proper noun, it is a candidate seed word.
Figure 3.2: Architecture Diagram For Bootstrapping Algorithm

Extract sentences based on seed word => Tokenize into words => Assign weight to each token and dynamically estimate the score threshold for a given sentence => If sentences crosses threshold, tokens matching seed word list are output hardware components being used and rest of them are candidates. => Select the candidate that can extend the seed word and repeat the process.

- **Rule 2:** If an adjective from the predetermined set precedes a proper noun, it is a candidate seed word.

- **Rule 3:** Digits or cardinal numbers that occur before time unit like ms, µs, days are identified as a potential compute time seed word candidate.

- **Rule 4:** A proper noun superseded by a cardinal number, and the keyword "language" is a potential programming language or a library candidate seed word.
3.1.2 Dynamic Scoring Model

The scoring model is based on multiple factors, such as the section of the research articles from which sentence has been selected. For instance, the Method and the Result section sentences of the research article have more weightage than the Abstract, Literature Survey, or Conclusion section sentences. The entities occur more frequently in the former sections as compared to the latter ones.

Another criterion is the length of a sentence. Based on the Parts of Speech (POS) tags and the rules described, a weight is assigned to each of the words. The final sentence score is computed as the weighted sum of the score of each word. A dynamic score threshold is calculated based on the ideal combination of the above-mentioned rules, and a length normalized threshold score for each sentence is assigned. Based on the threshold scores, candidate seed words are added to the original list of seed words, and verbs and adjectives are added to the list of candidate verbs and words. After each iteration, verbs and adjectives are ranked based on the score and their frequency of occurrence. The top 5% are selected and added to the original list of seed verbs and adjectives.

3.2 Sequential E2R: End To End Research Entity And Relationship Extractor

Since none of the standard tools currently extracts the entities such as (compute platform, hardware resources, compute time, programming language, and libraries), an end-to-end automated information extractor tool for long documents is implemented. The tool selects text having entities and discards the remaining irrelevant text. The system uses contextual Bidirectional Encoder Representation from Transformers (BERT) embeddings to generate a
Figure 3.3: Sequential E2R: Takes research article as input, extracts sentences having entities, filters out the salient sentences and extracts text spans having entities.

contextual representation for the sentences. These contextual embeddings help filter out the relevant text from the long document and do a token-level classification to extract entities. The model consists of three components: a sentence (binary) classifier, a K-Means clustering model, and a standard BERT + CRF (Conditional Random Field) tag decoder. Since there is no annotated data available for the hardware entities, Hardware Resources Compute Platform Time Language and Library (HPTLL) dataset has been created and used for further evaluation. Details about the HPTLL dataset are present in the section 4.1.2. Figure 3.3 shows the working of the sequential E2R pipeline. The yellow blocks in Figure 3.3 represent input/outputs, and the blue blocks represent the process used to obtain the output at each step.

3.2.1 Binary Sentence Classifier

The task of filtering relevant text has been modeled as a binary classification problem where positive class indicates sentence having relevant entities, and negative class refers to sentences that do not have any relevant entities. The dataset for training binary classifier has been created by combining sentences having an initial set of seed words and sentences having
entities from the HPTLL dataset as the positive class sentences. The negative class sentences have been created by randomly sampling sentences from the corpus to capture the context from the entire corpus. A BERT + Transformer-based sequence classifier has been trained for the sentence classification task. Figure 3.4 shows the architecture diagram of the binary classifier.

Figure 3.4: Architecture of the Binary Sentence Classifier

### 3.2.2 K-Means Clustering

The dataset has instances where a relevant entity is just mentioned but is not being used to conduct the research. A standard K-Means clustering model has been trained on contextualized BERT embeddings to extract salient features from all the mentioned entities. Salient and non-salient entities are often used in different context. Non-salient entities are mostly used to mention general facts. For e.g. “GPU reduces the training time of deep learning models”. K-Means algorithm learns the context in which non-salient entities are
used and thus, can separate salient and non-salient entities. The sentences belonging to the classifier’s positive class are fed to the K-Means clustering model to get different clusters. The Silhouette score has been used to find the number of optimal clusters. Further, to validate our hypothesis, we analyzed sentences belonging to different clusters and observed that sentences having salient entities form a distinct cluster.

3.2.3 Named Entity Recognition (NER) Model

The task of extracting entity spans has been modeled as Named Entity Recognition (NER) task. A BERT + CRF model has been trained on the HPTLL dataset to extract entity spans. The E2R model uses Spacy as a tokenizer and BERT + CRF model to predict output tags in BILUO format for each token based on the Viterbi Algorithm. Figure 3.5 shows the architecture diagram of the NER model in detail.

Figure 3.5: Architecture of the NER component of E2R model
3.3 Existing State-of-the-Art Models And Their Architectural Comparison With E2R

Most of the prior research works deal with short texts, and limited work has been done in extracting entities from scholarly research articles. The DyGIE, DyGIE++, and SciREX models are three state-of-the-art solutions that focus on information extraction from scholarly research articles. The following subsections discuss details about the working of these systems and their limitations in extracting hardware entities.

3.3.1 DyGIE and DyGIE++

Dynamic Graph Information Extraction (DyGIE) is a graph-based information extraction framework to perform various tasks such as named entity extraction, relation extraction between entities, and co-reference resolution. The DyGIE system enumerates all the text spans and models them as graph nodes. The edges between the nodes represent the relationship between entities and their co-reference links. The edges of the graph are confidence-weighted. The DyGIE model performs all three tasks jointly, i.e., entity extraction, relationship extraction, and co-reference resolution in a multi-task learning setup. The model dynamically prunes and refines the graph at each iteration.

DyGIE++ [53] is the extension of DyGIE. The DyGIE model uses GloVE and ELMO embeddings [28] whereas DyGIE++ uses BERT embeddings to capture the context. Additionally, the DyGIE++ system adds event-based entity extraction to the DyGIE model, i.e., DyGIE++ uses events triggering mechanism to improve NER extraction. The core idea of both the systems remains the same; i.e., they enumerate all text spans and build a dynamic graph that limits its capabilities of extracting NER to short documents.
These systems work well in extracting entities from multiple domains. Co-reference resolution and shared span representation allow the model to capture entities that are not just proper nouns. DyGIE and DyGIE++ fail to extract entities (such as hardware resources and compute platforms) for the following two reasons:

1. Both the models attempt to enumerate all the possible text spans in the research articles and do co-reference resolution and relation extraction. Hence, the systems perform well for shorter documents, but they do not scale well for longer documents. The models fail to extract the hardware entities even if co-reference resolution and relation extraction are disabled.

2. The systems fail to differentiate between salient and non-salient entities. For instance, DyGIE++ could not capture the relevant entity in “The design in b100 embeds custom RTL level optimizations in OpenCL kernels to boost the performance and avoid the current limitations of the OpenCL programming model.” Here, b100 is the hundredth referenced article. DyGIE++ reports the highlighted terms as a scientific entity.

### 3.3.2 SciREX

![Figure 3.6: Architecture of the SciREX model](image)

Jain et al. [18] proposed a baseline model, SciREX, along with the document level dataset for
information extraction from scholarly articles. The SciREX model performs three subtasks: (1) identifying entities, (2) their document level relation, and (3) saliency of the extracted entities in the document. Figure 3.6 shows the overall working of the SciREX model. The SciREX model extracts all the mention spans of entities from the research article and then jointly models co-reference resolution, salient entity classification, and clustering of the mentions in a multitask learning setup. In the next step, the SciREX model uses salient entities to find the salient cluster and pairwise co-reference scores of all the mentions to find the standard term for each cluster. In the final step, it computes the probability of all possible tuple combinations.

3.3.3 E2R and Scirex Comparison

The SciREX model requires fine-grained annotation for performing each step in the Figure 3.6. Figure 3.7 shows the annotation requirement for training the SciREX model using one unit of a document. Key-value pairs in black color in the figure 3.7 can be retrieved programmatically, and key-value pairs in red color need to be annotated manually. An annotator needs to annotate the text span containing the NER and then classify the annotated NER.
CHAPTER 3. METHODS

Figure 3.8: SciREX vs E2R annotation requirement

<table>
<thead>
<tr>
<th>Model</th>
<th>Annotation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SciREX</td>
<td>Annotates all the NER spans and calls them mentions&lt;br&gt;Annotates the salient entities out of all the mentions&lt;br&gt;Annotates all the mentions coreferring the salient entity and comes up with the standard term&lt;br&gt;Annotates all possible combinations of salient entities as 4-ary tuples and a probabilistic score for the occurrence of any single tuple. They call 4-ary tuples as relations&lt;br&gt;Annotates the text spans as sub relations to further improve the score</td>
</tr>
<tr>
<td>E2R</td>
<td>Annotates sentence as a salient or non salient&lt;br&gt;Annotates text span as salient entity in BILUO or ConLL2003 format</td>
</tr>
</tbody>
</table>

into the salient or non-salient category. In addition, the annotator needs to annotate the coreferring terms for all the pair of annotated NER spans, the relation between annotated NER spans.

On the other hand, the E2R model requires coarse-grained annotation; i.e., it contains salient sentences and the NER span in the BILUO format. Figure 3.8 summarizes the differences between annotation requirement of SciREX and E2R. For more details on the HPTLL dataset, please refer the section 4.1.2. Hence, the E2R model can be trained on the SciREX dataset, but the SciREX model cannot be trained on the HPTLL dataset.
Chapter 4

Results and Discussion

4.1 Dataset

4.1.1 arXiv Dataset

arXiv [1] is an open-source research article distribution service maintained by researchers from Cornell University. arXiv dataset corpus contains about 2 million scholarly articles in several fields like physics, mathematics, computer science, quantitative biology, quantitative finance, statistics, electrical engineering and systems science, and economics. To extract compute platform, time, hardware resources, and other related entities, we focus on the full-text computer science related articles in various domains, including Machine Learning (LG), Hardware Architecture (AR), Computer Vision (CV), Artificial Intelligence (AI), Information Extraction (IR) as unlabeled input data. Figure 4.1 shows the distribution of research articles in the above-mentioned streams.

Dataset Processing

arXiv distribution service renders research articles in PDF format. Several tools like Grobid [2], Science Parse, Apache Tika, PDFMiner are available to parse and convert PDF into text. Grobid gives the best result for processing scholarly articles [20]. We use Grobid
Figure 4.1: Count of full-text research articles in different branches of Computer Science namely Machine Learning (LG), Hardware Architecture (AR), Computer Vision (CV), Artificial Intelligence (AI), Information Extraction (IR)

to preprocess the data and convert it to JSON format. The JSON keys are the different sections, and values are the text referring to the corresponding section. The metadata about the research article, like authors, unique identifiers, are stored at the topmost level in the JSON file hierarchy. References are prefixed with “b” and stored as separate key-value pairs in the JSON file for modeling and analyzing the citation graph. Figure 4.2 shows the sample output format of Grobid.

**Preliminary Analysis on arXiv dataset**

As a part of our preliminary analysis, we studied whether the increment in computing power has promoted work in related areas. We analyzed the distribution of research articles in
the field of hardware, artificial intelligence, and information extraction over a period. We observed a positive correlation between hardware technology advancement and the number of papers published in the related areas. Figure 4.3 shows the distribution of research articles from 2004 to 2019.

### 4.1.2 HPTLL Dataset

As there is limited work done to extract the hardware entities, this work proposes a novel NER dataset, HPTLL (Hardware Resources Compute Platform Time Language and Library), annotated by domain experts in the standard BILUO (B-Beginning, I-Intermediate, L-Last, U-Unit, O-Outside) format. The details on the annotation process and followed standards are discussed in Appendix A. HPTLL comprises salient sentences, i.e., sentences containing entities that have been used and not just mentioned in the research article, and the text span in which the entity is present in the BILUO format. HPTLL dataset contains over 600 salient sentences and about 1400 annotated entities overall. Table 4.1 shows the
4.1.3 SciREX Dataset

Researchers from Allen Institute For AI community have proposed an open-source document level dataset, SciREX [18], for information extraction from scholarly articles. It comprises annotated salient entities like Task, Method, Dataset, and Metric. Since SciREX also extracts entities from research articles, we use the SciREX dataset to compare the performance...
4.2 Qualitative Analysis of Weakly Supervised Model

An initial set of seed words is selected to identify the candidate entities. The candidate entities, with a score above the threshold, are added to the original seed word set as the extracted entities. This process is done iteratively, and the performance of the model is evaluated after each iteration. Figure 4.4 denotes the output of the model after each iteration, starting with a specific set of seed words for one paper. It is evident that the performance of the model begins to deteriorate after the third iteration. A large number of patterns can be formed by combining different features. Adding checks and balances for every pattern is a tedious task. Though adding rules might help, there might not be a significant improvement in the performance of the model.

Figure 4.4: Weakly Supervised Algorithm Output at each iteration for one paper
4.3 Sequential End To End Research Entity And Relationship Extractor (E2R)

The sequential E2R model performs the following three tasks: Binary Sentence Classification, K-Means clustering, and NER Extraction. Section 3.2 gives the implementation details about these tasks. The following subsections present the results for each task.

4.3.1 Binary Sentence Classification

A Binary sentence classifier takes a sentence as input and classifies the sentence into positive and negative classes. The positive class refers to sentences having relevant entities, and the negative class refers to the sentence that does not have the relevant entities. We use a set of seed words to sample positive sentences from the arXiv dataset, and from the set of remaining sentences, we randomly select sentences to represent the negative sentences. Table 4.2 shows the F1 scores for hardware, language, and library entities. It can be observed that each classifier is performing well. The high performance can be attributed to BERT’s ability to capture the context of each entity type well.

<table>
<thead>
<tr>
<th>Entity Type</th>
<th>F1 Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware Classification</td>
<td>98.2</td>
</tr>
<tr>
<td>Programming Language Classification</td>
<td>97.3</td>
</tr>
<tr>
<td>Programming Library Classification</td>
<td>97.2</td>
</tr>
</tbody>
</table>
4.3. SEQUENTIAL END TO END RESEARCH ENTITY AND RELATIONSHIP EXTRACTOR (E2R)

4.3.2 K-Means Clustering

E2R model uses K-Means clustering to distinguish between salient and non-salient sentences. It uses contextual embedding to capture the context in which salient entities are often used and then uses clustering to separate the salient and non-salient sentences naturally. For finding the optimal number of clusters, Silhouette score [46] has been used.

Saliency Analysis

We use Principal Component Analysis (PCA) for visualizing the overlapping clusters after reducing the dimensions. Figures 4.5, 4.6 and 4.7, show us that three is the optimal number of clusters for each of the entity type. We also qualitatively analyzed the output of clustering model by randomly sampling 1000 sentences from each cluster type. The figure 4.8 shows examples of sentences belonging to different clusters for the hardware entity type. We observed that Cluster Zero sentences contain the salient sentences with relevant hardware entities. Cluster One contains sentences that mention the model’s architecture and related hardware required to run that architecture. Cluster Two contains non-salient sentences, i.e.,
CHAPTER 4. RESULTS AND DISCUSSION

4.3.3 NER Extraction

The E2R model uses Viterbi algorithm on the Conditional Random Field (CRF) output to decode entity tags for each token. NER component takes a sentence as an input, extracts SciBERT embeddings for each token of the sentence, computes the posterior probability estimate, and predicts tags for each of the words in the BILUO format. E2R uses the exact match criteria, i.e., a prediction is considered true positive only if both the start and end positions of span match the ground data. False positive and false negative refer to instances where the model predicts incorrect class compared to ground truth. True negative refers to the instances where our model predicts the correct class. We use the Micro and Macro Avg F1-score to evaluate the performance of the E2R model.

Table 4.3 shows the F1 score for B-tag, I-tag, L-tag, and U-tag for each entity type, i.e., hardware resources, compute platform, compute time, programming language, and libraries. The score for hardware tags shows that the model can extract multi-word hardware entities.
effectively. The F1 scores for U-tag entities are relatively low because hardware entities usually span multiple words, and there are very few one-word hardware entities in the training dataset. Similarly, the score of B-tag and L-tag for programming language and library are low as they occasionally span multiple words, and hence, fewer instances are present in the training dataset. Also, it can be observed from Table 4.3, I-tag F1 scores for some of the entities are N.A. because they rarely span more than two words. The F1 score for O-tag is 97.7. F1-Score of O-tag is high as most of the tokens in the training dataset have a class O-tag, and the model can learn its class distribution well.

Table 4.4 presents the overall performance metric scores of E2R model on HPTLL dataset. Entities such as compute time are mentioned less in the research articles and hence are difficult to extract. This work reports both the macro and micro average F1 scores to address the class imbalance situation. Currently, the training dataset has 51 instances of compute time entities. If the size of annotated data increases, the performance of the model over such entities would increase, and thus, the overall performance may improve.
Presently, SciREX is the state-of-the-art model to extract entities from long research articles. SciREX and E2R predict different sets of entities. SciREX predicts entities like dataset, task, method, and metric, while E2R predicts entities such as hardware resources, compute platform, compute time, programming language, and library. SciREX requires fine-grained annotation while E2R requires coarse-grained annotation, due to which E2R can be trained on the SciREX dataset, but SciREX cannot be trained on the HPTLL dataset. Hence, to compare SciREX and E2R, we train the E2R model on the SciREX dataset.
Table 4.3: E2R: Entity and their F1 Scores for each of the tag

<table>
<thead>
<tr>
<th>Entity</th>
<th>B-Tag</th>
<th>I-Tag</th>
<th>L-Tag</th>
<th>U-Tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware Resources</td>
<td>88.37</td>
<td>92.06</td>
<td>86.82</td>
<td>62.5</td>
</tr>
<tr>
<td>Compute Platform</td>
<td>75.67</td>
<td>58.82</td>
<td>75.67</td>
<td>N.A.</td>
</tr>
<tr>
<td>Programming Language</td>
<td>36.36</td>
<td>N.A.</td>
<td>36.36</td>
<td>76.00</td>
</tr>
<tr>
<td>Programming Library</td>
<td>28.57</td>
<td>N.A.</td>
<td>28.57</td>
<td>72.72</td>
</tr>
<tr>
<td>Compute Time</td>
<td>0.5</td>
<td>N.A.</td>
<td>0.5</td>
<td>54.54</td>
</tr>
</tbody>
</table>

Table 4.4: E2R: Overall Performance Metric Scores

<table>
<thead>
<tr>
<th>Metric Type</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precision Overall</td>
<td>73.15</td>
</tr>
<tr>
<td>Recall Overall</td>
<td>70.77</td>
</tr>
<tr>
<td>Micro avg. F1 Overall</td>
<td>71.94</td>
</tr>
<tr>
<td>Macro avg. F1 Overall</td>
<td>53.37</td>
</tr>
</tbody>
</table>

Table 4.5: E2R Performance on predicting the individual entities from the SciREX dataset.

<table>
<thead>
<tr>
<th>Entity</th>
<th>B-Tag</th>
<th>I-Tag</th>
<th>L-Tag</th>
<th>U-Tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dataset</td>
<td>47.18</td>
<td>46.21</td>
<td>47.78</td>
<td>56.48</td>
</tr>
<tr>
<td>Task</td>
<td>73.06</td>
<td>69.48</td>
<td>74.68</td>
<td>66.04</td>
</tr>
<tr>
<td>Metric</td>
<td>71.05</td>
<td>65.13</td>
<td>72.60</td>
<td>73.13</td>
</tr>
<tr>
<td>Method</td>
<td>78.32</td>
<td>77.44</td>
<td>79.83</td>
<td>78.92</td>
</tr>
</tbody>
</table>

Table 4.6: Overall Performance comparison of E2R and SciREX

<table>
<thead>
<tr>
<th>Metric Type</th>
<th>E2R on SciREX dataset</th>
<th>SciREX on SciREX dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precision Overall</td>
<td>71.91</td>
<td>70.7</td>
</tr>
<tr>
<td>Recall Overall</td>
<td>75.66</td>
<td>71.7</td>
</tr>
<tr>
<td>Micro avg. F1 Overall</td>
<td>73.74</td>
<td>Not Reported</td>
</tr>
<tr>
<td>Macro avg. F1 Overall</td>
<td>69.09</td>
<td>71.2</td>
</tr>
</tbody>
</table>
Table 4.5 denotes the performance of E2R in predicting individual tags of entities belonging to the SciREX dataset. We have not compared E2R and SciREX on entity-level because the SciREX paper has not reported the F1-score on the entity level. Table 4.6 shows the overall performance comparison of E2R and SciREX on predicting entities from the SciREX dataset. SciREX authors have not reported the Micro Avg. F1-Score. E2R is lightweight compared to SciREX in terms of annotation required and has comparable performance on the task of mention prediction.
Chapter 5

Conclusion

Past research has focused on the task of extracting generic entities such as person, organization, geographic location, currency, bacteria from short documents. Extracting novel entities like hardware resources, compute platform, compute time, programming language, and the library from long documents is a relatively new task with its own set of challenges and is technically different from extracting generic named entities. Extracting compute time would allow us to estimate the power consumed by different deep learning algorithms. Extracting hardware resources and compute platforms would allow us to create a mapping between deep learning algorithms and their hardware requirements. Extracting these entities involves processing long documents and distinguishing between salient and non-salient entities. As per our knowledge, these entities have not been extracted before, and hence no direct solution or labeled dataset is available.

Rule-based weakly supervised algorithms require a set of keywords and can be applied on unlabeled dataset to extract hardware entities. However, rule-based weakly supervised algorithms have low recall and are not scalable. Most state-of-the-art models fail to differentiate between salient and non-salient entities and do not work well on long research articles. Thus there is a need for a better approach to extract hardware-related entities.

In this work, we propose a deep learning-based automated tool, End To End Research Entity and Relationship Extractor (E2R Extractor), to effectively solve the problem of extracting hardware-related entities from technical text. Furthermore, this work also proposes an
open-source dataset, HPTLL, comprising annotated salient hardware entities in the BILUO format. E2R uses transformer-based SciBERT embeddings to capture the context in which hardware entities are generally used. E2R uses a binary sentence classifier to identify sentences containing entities, followed by clustering, to select salient sentences from all the mentioned entities, and finally, uses a CRF-based tag decoder to extract salient entity span. From the results in Table 4.3, it is evident that entities such as compute time are sparse and thus hard to extract. Also, entities such as programming language and library rarely span more than two words, and entities such as hardware resources rarely span less than two words. Since the entity distribution is not uniform, we report the overall Micro average F1-Score in addition to the overall Macro average F1 Score. To evaluate the result of E2R on the entity level, we report F1 scores for the B-tag, I-tag, L-tag, U-tag, and O-tag for each entity. Qualitative and quantitative analysis of the output of the E2R model demonstrates its ability to extract relevant entities effectively. With further increase in the training data size, the overall performance of E2R would increase significantly. SciREX is the state-of-the-art model for extracting entities such as Task, Method, Material, and Metrics from research articles. E2R requires coarse-grained or less annotation Whereas SciREX requires fine-grained or more annotation. Table 4.6 shows that E2R has comparable performance with SciREX. E2R is a light-weight model with comparable performance to the state-of-the-art model. Transfer learning with E2R as a base model can be used to extract domain-centric entities such as disease names and symptoms from biomedical texts, laws and clauses from legal domain etc. Thus, we can use E2R architecture for interdisciplinary entity extraction.
Bibliography


[24] Jing Li, Aixin Sun, Jianglei Han, and Chenliang Li. A survey on deep learning for
named entity recognition. *IEEE Transactions on Knowledge and Data Engineering*, 2020.


Appendices
Appendix A

Dataset Annotation Guideline

A.1 Dataset Construction

Since most labeled datasets contain generic named entities, we decided to annotate and create our dataset for this research work. We used the arXiv dataset as the source of unlabeled data. HPTLL contains salient sentences and the relevant entities annotated in the BILUO format. We use both automatic and manual annotation to build the HPTLL dataset. We use the clusters obtained from the output of the sequential E2R pipeline for identifying salient sentences. We term the cluster having salient sentences as the salient cluster. We sample sentences from the salient cluster and then annotate the entities using a weakly supervised approach, discussed in the section 3.1. Finally, the annotators perform three sets of tasks on the automatically annotated sentences: (1) Verify if the sentence is salient or not. (2) Check whether the automatically annotated spans contain any relevant entity, and (3) Check the start and end boundary of the span, and fix if incorrect. We use a combination of automatic and manual annotation to reduce the overall overhead.

A.2 Annotation Agreement

Since our dataset is domain-specific, the annotators need to have domain expertise to annotate the relevant entities. Our annotators are computer science graduate students and have
the prerequisite knowledge required to perform the annotation task. Since there can be a
difference of opinion among the annotators, we decided to get our annotators’ vote.

We asked three computer science graduate students to perform the annotation task. Each
annotator marks the saliency of each sentence. If a sentence gets at least two votes, then
we include that sentence in our dataset. As a next step, we compare the entities and their
boundaries for each of the annotators. If at least two annotators have the same annotation,
we include it in our final dataset. Finally, we are left with over 600 sentences and 1000
entities.
Appendix B

Second Appendix

B.1 E2R Extractor

E2R Extractor has been trained on scholarly articles from the arXiv corpus

- E2R Extractor has been developed with an aim of extracting Technical Details from Scholarly Literature.
- E2R Extractor takes research paper in JSON format after being processed from GROBID.
- Given a research paper(s) in JSON, E2R extracts the to find the following items.
  - Computational platform utilized in the paper,
  - Language/library dependencies, and
  - Compute time and resources consumed.

B.2 Downloading Trained Models

The E2R extractor is a sequential model and contains the following trained models at the “hardware_language_library_extractor/models” location in the repository.
• Sentence Classifier Model
  – Hardware Sentence Classifier
  – Language Sentence Classifier
  – Library Sentence Classifier

• Clustering Model
  – Hardware Clustering Model
  – Language Clustering Model
  – Library Clustering Model

• NER Model
  – BERT + CRF Model
  – E2R NER Model identifies following entities from a sentence.
    * Compute Platform
    * Hardware Resources
    * Compute Time
    * Programming Language
    * Programming Library

**B.3 Using E2R Extractor**

Complete the following prerequisites for setting the environment:

• Setup the Python 3.6 virtual environment.
• Install the dependencies using the command “pip install -r requirements.txt”.

• For using prediction pipeline:
  
  – Point the variable MODELS_FOLDER_BASE_PATH in “prediction_pipeline/config” to the models folder present inside the repository.
  
  – Point “LIST_OF_PDFS” in “prediction_pipeline/config” to a text file having name of all the research papers one on a line, a sample file has been included in the “sample_files/filelist.txt”.
  
  – Point “OUTPUT_FOLDER” in “prediction_pipeline/config” to the folder where you want to generate outputs (can create an empty folder and point to it).
  
  – Point “PDF_FOLDER_PATH” in “prediction_pipeline/config” to the folder containing all research paper in JSON, sample files has been included in the “sample_files/pdf_folder”.
  
  – Run the “prediction_pipeline/extraction_script”.

• For using training pipeline:
  
  – Point the “TRAINING_DATA_BASE_PATH” in “training_pipeline/config” to the folder which will contain training data.
  
  – Point the ‘MODELS_FOLDER_BASE_PATH‘ in ‘training_pipeline/config’ to the folder containing all the pretrained models.
  
  – Point the ‘OUTPUT_FOLDER_BASE_PATH‘ in ‘training_pipeline/config‘ to the folder, which will contain the training outputs.
  
  – For training Sentence Classifier
* Put all the hardware positive class sentences in a text file named “positive_hardware_sents.txt” inside “TRAINING_DATA_BASE_PATH”, a sample file has been included in the “sample_files/training_data”.

* Put all the hardware negative class sentences in a text file named “negative_hardware_sents.txt” inside “TRAINING_DATA_BASE_PATH”, a sample file has been included in the “sample_files/training_data”.

* Put all the language positive class sentences in a text file named “positive_language_sents.txt” inside “TRAINING_DATA_BASE_PATH”, a sample file has been included in the “sample_files/training_data”.

* Put all the language negative class sentences in a text file named “negative_language_sents.txt” inside “TRAINING_DATA_BASE_PATH”, a sample file has been included in the “sample_files/training_data”.

* Put all the library positive class sentences in a text file named “positive_library_sents.txt” inside “TRAINING_DATA_BASE_PATH”, a sample file has been included in the “sample_files/training_data”.

* Put all the library negative class sentences in a text file named “negative_library_sents.txt” inside “TRAINING_DATA_BASE_PATH”, a sample file has been included in the “sample_files/training_data”.

* Run the “training_pipeline/sentence_classifier_trainer.py” script.

  - For training Clustering Model:

    * Put all the hardware positive class sentence embeddings in a text file named “positive_hardware_sent_embeddings.txt” inside “TRAINING_DATA_BASE_PATH”, a sample file has been included in the “sample_files/training_data”.

    * Put all the language positive class sentence embeddings in a text file named “positive_language_sent_embeddings.txt” inside “TRAINING_DATA_BASE_PATH”, a sample file has been included in the “sample_files/training_data”. 
a sample file has been included in the “sample_files/training_data”.

* Put all the library positive class sentence embeddings in a text file named "positive_library_sent_embeddings.txt" inside “TRAINING_DATA_BASE_PATH”, a sample file has been included in the “sample_files/training_data”.

* Run the script located at “training_pipeline/clustering_trainer.py”.

- For training NER Model:

  * Data present in “BILUO” format should be placed in “ner_trainer/data/cset”.

  * Download Scibert Model and point the BERT_VOCAB and BERT_WEIGHTS variable in “ner_trainer/scripts/train_allennlp_local.sh” to the Scibert folder present inside “training_pipeline/ner_trainer/scibert”.

  * Point “dataset_size” variable in “ner_trainer/scripts/train_allennlp_local.sh” to the length of training data.

  * From “ner_trainer‘ location, run ‘bash ./scripts/train_allennlp_local.sh output_dir’

  * output_dir is the directory where trained model files would be stored (can create an empty directory and point to it).