A literature review on Natural Language Processing Methods on Text and Information Extraction

ABSTRACT

Text extraction and text labeling are important information extraction processes within the field of Natural Language Processing (NLP). With the staggering surge in the volume of information, be it hidden within the likes of web pages, company documents or emails, the need for this subfield of NLP tasks to extract useful information efficiently has proven necessary. Many methods have been proposed to automate this NLP task, however, the lack of structure with text data renders text extraction and labeling a challenging research problem. This literature review is concerned with the different methodologies and approaches used for this branch of information extraction.

1 Introduction

With the world's expanding digital landscape comes a massive proliferation of data and information. Often, it becomes inherently difficult, expensive and time-consuming to identify, extract and process the information available to us, let alone the relevant and useful information. Thus, text analysis and natural language processing (NLP)'s subfield of text extraction and text labeling is vital to alleviate the challenges presented when collecting and filtering data. Using NLP and text mining methods, we are able to convert different kinds of texts, from journal articles to html tags, to structured, readable data that may be used for data analysis, training machine learning models or summarizing information. Some real-world applications of text extraction and text labeling include classifying emails, contact information search and finding product information from websites.

1.1 Problem and Goals

The task of creating a robust system, whether it is supervised or unsupervised, is extremely challenging because a large number of these methods require huge amounts of memory, have large execution times, and do not work for multiple different text types. Besides, when supervised methods are used, labeled data of good quality and highly tuned feature engineering is needed to guarantee a decent performance of these systems, and often this is extremely costly.

The goal of this paper is to present a literature review on the different NLP techniques and tools to conduct text extraction and text labeling.

1.2 Related Work

Many research efforts have been made in the domain of text extraction and text labeling. Among these research work, landmark-based methods, topic boundary segmentation, graph-based methods, bootstrapping methods, machine-learning clustering as well as topic modeling would be the techniques we would like to explore in this paper.

Table 1.1 presents an overview of the related works discussed in this section.

Table 1: Overview of related works

Author	***Title***	***Year***	***Overview***
Pinkesh Badjatiya, Litton J Kurisinkel, Manish Gupta, and Vasudeva Varma	Attention-based Neural Text Segmentation	2018	Text segmentation using Attention-based CNN-BiLSTM to search for topic boundaries
Suresh Parthasarathy, Lincy Pattanaik, Anirudh Khatry, Arun Iyer, Arjun Radhakrishna, Mohammad Raza and Sriram K. Rajamani	Landmarks and Regions: A Robust Approach to Data Extraction	2022	Utilizing Landmarks, Regions and Clusters to extract information

Adi Omari, Sharon Shoham, Eran Yahav	Synthesis of Forgiving Data Extractors	2017	Use of XPath queries that dynamically adjust their precision to handle structural changes to extract web pages
Suresh Parthasarathy, Lincy Pattanaik, Anirudh Khatry, Mohammad Raza	Synthesis and Machine Learning for Heterogeneous Extraction	2019	Use of Machine Learning and Program Synthesis to extract text

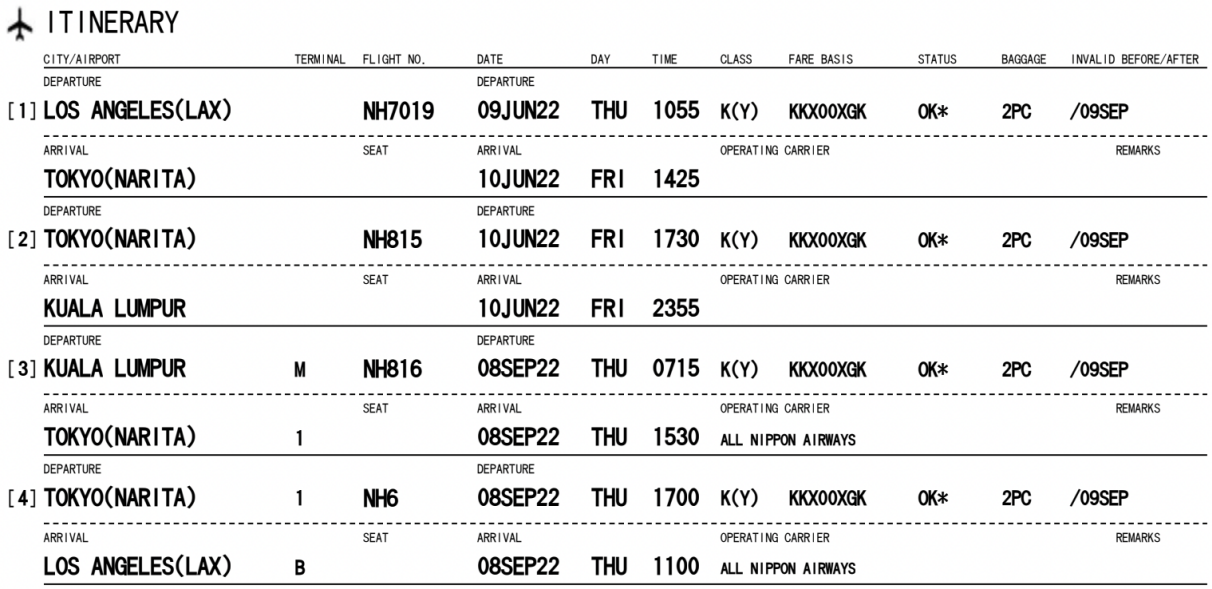
2 Methods

In this section, we explore the mechanics and algorithms of the current methods in text extraction and text labeling, before providing the details of the experiment as well as the results obtained from said experiments.

2.1 LRSyn

S. Parthasarathy et al. (2022) [1] describes a method that utilizes the notion of “landmarks” and “regions” to extract and label semi-structured documents. The intuition behind this method is rooted in the neuroscience of humans when analyzing and searching for specific information in lengthy documents – we simplify the task by first narrowing down and focusing on a small region of where the desired information may possibly be (a landmark), before extracting the information near the landmarks. An intuitive example would be how we navigate dictionaries. If we were to locate the word “spoon” in the dictionary, the most efficient way would be to locate the page where the letter “s” resides, before finding the word “spoon” itself, rather than iterating through the whole dictionary before “spoon” is found. In the case of other text data, such as travel emails as shown in Figure 1, the phrase “DEPARTURE” is a potential landmark to

determine departure location, whereas the phrase “FLIGHT NO.” would be a great landmark to determine the flight number of the plane. The landmark method would thus only work for semi-structured text data where there is an invariance present in all documents.



ITINERARY

	CITY/AIRPORT	TERMINAL	FLIGHT NO.	DATE	DAY	TIME	CLASS	FARE BASIS	STATUS	BAGGAGE	INVALID BEFORE/AFTER
	DEPARTURE			DEPARTURE							
[1]	LOS ANGELES(LAX)		NH7019	09JUN22	THU	1055	K(Y)	KKX00XGK	OK*	2PC	/09SEP
	ARRIVAL		SEAT	ARRIVAL			OPERATING CARRIER				REMARKS
	TOKYO(NARITA)			10JUN22	FRI	1425					
	DEPARTURE			DEPARTURE							
[2]	TOKYO(NARITA)		NH815	10JUN22	FRI	1730	K(Y)	KKX00XGK	OK*	2PC	/09SEP
	ARRIVAL		SEAT	ARRIVAL			OPERATING CARRIER				REMARKS
	KUALA LUMPUR			10JUN22	FRI	2355					
	DEPARTURE			DEPARTURE							
[3]	KUALA LUMPUR	M	NH816	08SEP22	THU	0715	K(Y)	KKX00XGK	OK*	2PC	/09SEP
	ARRIVAL		SEAT	ARRIVAL			OPERATING CARRIER				REMARKS
	TOKYO(NARITA)	1		08SEP22	THU	1530	ALL NIPPON AIRWAYS				
	DEPARTURE			DEPARTURE							
[4]	TOKYO(NARITA)	1	NH6	08SEP22	THU	1700	K(Y)	KKX00XGK	OK*	2PC	/09SEP
	ARRIVAL		SEAT	ARRIVAL			OPERATING CARRIER				REMARKS
	LOS ANGELES(LAX)	B		08SEP22	THU	1100	ALL NIPPON AIRWAYS				

Figure 1: Flight Ticket and its possible Landmarks for Text Extraction

The author implemented the Landmark-based Robust Synthesis Program (LRSyn), where the concept of landmarks, regions of interests (ROI), and blueprints are introduced. The problem is as follows: given a list of user-provided locations $[l_1, l_2, \ldots l_n]$ used for training, identify the landmark *m*, a value that can be used to identify a location within a document, examine the region of interest, which is the area surrounding the landmark that encloses the field value, and output the field value corresponding to said landmark. The program thus has a blueprint to follow as it iteratively searches for landmarks, ROIs and field values for the remaining documents. For the LRSyn algorithm to work, the authors assume that there exists a computationally inexpensive function *Locate* that is able to locate the landmark *m* in a document easily, as well as the function *EncRgn(L,doc)*, where *L* is a set of locations in document *doc,* that returns the smallest region that contains all

locations in L. Accordingly, LRSyn is split into 3 components: finding landmarks and clusters, region extraction and value extraction.

Vocabulary	Meaning	Function
Landmark	Value used to identify location in a document	*Locate(*doc, *m)* → *l* → returns landmark
Region	Set of contiguous locations that enclose landmarks and the corresponding field values	*EncRgn(L,doc)*→ *R* → returns region of interest surrounding landmark
Blueprints	A "hash" of all parts that are "common" to regions in a cluster	*BP(R)* → *b* → returns blueprint of region
Annotations	A list of locations and the data present in each location, of which are used as training data	*A*(doc) → $[l_1,l_2,...,l_n]$, (Data$[l_1]$, Data$[l_2]$,...Data$[l_n]$) → returns location and field values

Table 2: Vocabulary of the LRSyn Program

Given an annotated training document, the first task of LRSyn is to find landmarks and clusters. We pass each training document into a blueprint function *BP* to obtain the blueprint *b* of each document. Initial clustering is performed by comparing the blueprints of the documents. If the structures of the documents are similar, the distance between the documents, $\delta(b_1,b_2)$, should be a small value. Thus, we obtain many fine-grained clusters C_i of documents with the same format with little to no variation. Afterwards, the procedure *LandmarkCandidates* identifies common values in different *Ci* as landmark

candidates and orders them by a scoring function that is based on the distance between the landmark candidate and annotated values and the size of the region enclosing the landmark candidates and annotated values. Only candidates with a score greater than a certain threshold is returned. The clusters are then repeatedly merged based on their average document distance until no further merging is possible. Thus, we now have the landmark candidates with the associated coarse-grained clusters, which will then be processed to calculate and synthesize extraction programs.

The author then introduces the *SynthesizeExtractionProgram* procedure, which takes in: (a) a cluster *C* and its corresponding landmark *m*, (b) the annotations *A* for documents in *C*, and (c) domain-specific languages (*DSLs*) for synthesizing region programs and extraction programs. The first step of the procedure is to compute the region of interest (ROI) for each document using the *EncRgn* function, which will eventually be used as examples for the synthesis of the region program RProg. Afterwards, the specification of the regions would be used for the computation of the average blueprint *b* for documents in cluster *C*. When conducting text extraction for unseen data, RProg will be used alongside *b* as a robust system to detect ROIs. Subsequently, a value extraction program EProg will be learnt using annotations and synthesized using the examples where the inputs are ROIs for each document and the outputs are the expected field values. The *SynthesizeExtractionProgram* procedure thus returns (RProg, *b*, EProg) to complete the extraction program of LRSyn.

Since LRSyn relies on landmarks for data extraction, it is extremely robust against format variations that change document structure outside the ROIs, position of ROIs, and the number of ROIs. For example, if an advertisement banner is added to the document as a new part, the program will not be misled. However, if an extra part similar to the landmark is added, spurious output may be generated. Besides, if the actual format of the ROI changes, the underlying assumption that ROIs have invariant local structures is violated, and a poor program may be generated.

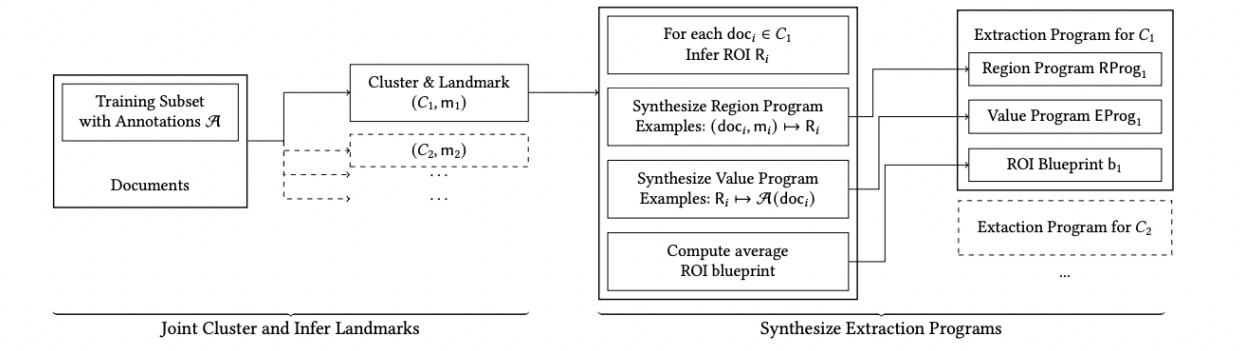


Figure 2: Outline of the LRSyn method

The author discusses the LRSyn algorithm in the context of a HTML document and document image, but we will focus on the instantiations of LRSyn on the HTML document in this section. For HTML documents, blueprints are defined as the set of XPaths to the common value DOM nodes in the region, n-grams are used as landmarks ($n \leq 5$), and the *Locate* function returns the landmark within a DOM node. The *LandmarkCandidates* procedure lists all n-grams in the document, filters out the n-grams containing stop words before performing the initial clustering. The procedure is then conducted as discussed above. The scoring function for HTML documents is the weighted sum of the (a) the number of nodes in the path from the DOM nodes corresponding to *m* and the field value *v*, (b) the number of nodes in the smallest region enclosing both *m* and *v,* and (c) the Euclidean distance between *m* and *v*.

We then move onto the region extraction program of HTML documents, where the algorithm could be simplified into three steps including a pair of integers – parentHops and siblingHops. Given landmark location ℓ and the annotated location Locs, we take the lowest common ancestor (LCA) *n* of ℓ and all nodes in Locs. ParentHop is the difference between *n* and ℓ-1; siblingHops is the difference between the index of the leftmost and rightmost child of *n* that has either ℓ or Locs as a descendant. RProg is then as such: first, given a landmark ℓ, we go up the DOM tree parentHops times to get node n_1, then go

siblingHops times to the right to obtain node n_2, the region is the set of all descendants of all sibling nodes between n_1 and n_2 (inclusive).

The author does not discuss the mechanics of the value extraction program for HTML documents in detail, but briefly summarizes the program into two parts, a web extraction program that extracts the DOM node that contains the field value, and a text extraction program that obtains the field value from the extracted DOM node. For example, given synthesis program R, it first finds the DOM node n which contains the text v before extracting the text from v.

In the case of extracting text from document images, which are first processed using Optical Character Recognition (OCR) techniques to obtain a list of text boxes with their coordinates, the LRSyn approach is significantly more challenging, since OCR outputs can be very noisy and there are no hierarchical structures that define natural regions. For images, n-grams are used as landmarks as well, and the *LandmarkCandidates* takes into account the Euclidean distance between landmarks *m* and field values *v*, and the area of the smallest rectangle that encloses both landmarks and field values. Blueprints, or *BoxSummary(s)*, are defined as the text boxes containing the top 50% most frequent n-grams, and is comprised of 2 parts: (a) the frequent n-gram that is present in the box, and (b) its immediate neighbors' content type. The content type of a box is either: (a) $\bot$ if the box does not exist, (b) $\top$ if the box exists, but does not contain a frequent n-gram or (c) the frequent n-gram is present. For example, the text box enclosing *Engine number* in Figure 3 has the *BoxSummary* <ngram ↦ *Engine number*, Top ↦ $\bot$, Left ↦ *Chassis number*, Right ↦ *Reg Date*, Bottom ↦ $\top$>.

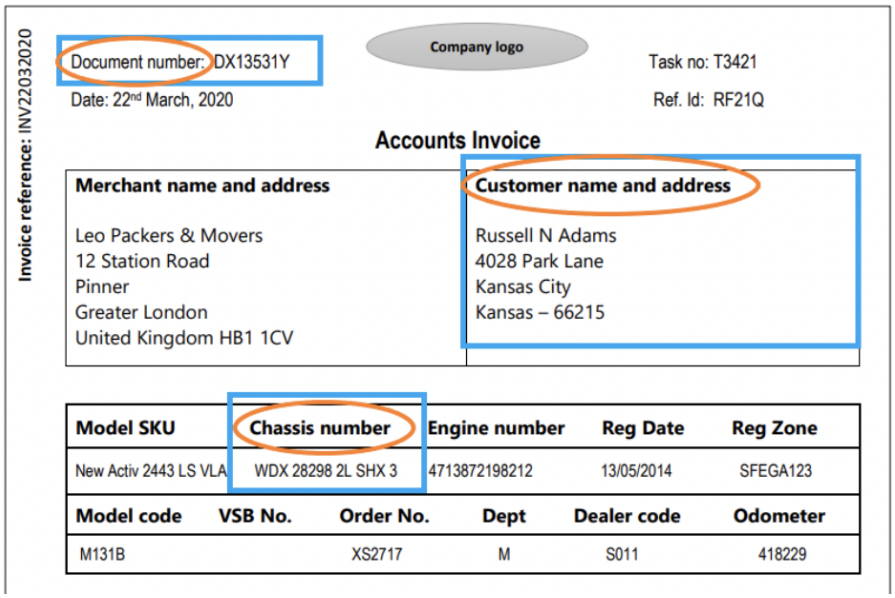


Figure 3: Example document image

The region extraction program for document images is shown as in Figure 4. The top operator is a disjunction of path programs which are executed in sequence and the first non-null result is returned. Each path program begins with the input landmark and is extended in steps until the path's bounding box covers all the annotated values. Each step is defined by a direction and a motion. The motion may be *absolute* (move right by 4 boxes) or *relative* (move down until you hit a text box that matches the regex [0-9]{5}). The *inclusive* parameter specifies whether the box that matches the pattern should be included or excluded; the parameter *k* represents the maximum number of path programs. For the landmark *Chassis number* in Figure 3, let's consider the two region extraction program:

(a) *Ext(Ext(input, Abs(down, 1)), Rel(Right, [0 − 9]{13}, false))*

(b) *Ext(Ext(input, Abs(down, 1)), Rel(Right, DATE, false))*

Both programs first move one step down from the landmark *Chassis number,* but (a) moves right till it hits a 13 digit number (not included in region), while (b) moves right till it hits a date. The programs are then combined disjunctively to cover all possible cases. The synthesis of a region program is similar to *NDSyn* and consists of two parts: (a) generating path programs and (b) selecting path programs. Candidate path programs

are first synthesized using *enumerative synthesis*, where all possible path programs are listed and pruned until a program that is consistent with all the examples is found, then they are filtered by whether they cover the annotated values. Then, a finite set of regular expression patterns are generated using string profiling techniques over all the text values present in the cluster. We are then left with a collection of path programs that are synthesized from a small subset of input examples. The NDSyn algorithm will then be used to synthesize the disjunctive program. This program will then return a concatenation of all the text values in the boxes, which will be fed to the value extraction program, which uses FlashFill.

```
RProg     := Disjunct(path, path, ...)
path      := input | Expand(path, motion)
motion    := Absolute(dir, k)
           | Relative(dir, pattern, inclusive)
dir       := Top | Left | Right | Bottom
```

Figure 4: Region Extraction program for images

The authors compared LRSyn to NDSyn and ForgivingXPaths, both of which are considered state of the art techniques, using a dataset of flight-reservation emails. The experiments are performed in two settings: contemporary, where documents are authored and collected at the same time, and longitudinal, where test data was collected months after training data, allowing for new formats to organically enter the dataset. On the HTML documents, LRSyn outperforms the other techniques, scoring a 1.00 on average precision, recall and F1.

Contemporary			
Metric	ForgivingXPaths	NDSyn	LRSyn
Avg. Precision	0.17	0.96	1.00
Avg. Recall	0.99	0.91	1.00
Avg. F1	0.22	0.93	1.00
Longitudinal			
Metric	ForgivingXPaths	NDSyn	LRSyn
Avg. Precision	0.15	0.99	1.00
Avg. Recall	0.98	0.89	1.00
Avg. F1	0.20	0.92	1.00

Table 3: Results of the Experiment on M2H dataset

For document images, the author compares LRSyn to a neural model AFR, and even with little training data, LRSyn matches AFR on contemporary data, and outperforms AFR for longitudinal data.

Metric	AFR	LRSyn
Avg. Pre.	0.98	0.99
Avg. Rec.	0.96	0.99
Avg. F1	0.97	0.99

Finance dataset

Metric	AFR	LRSyn
Avg. Prec.	0.90	0.97
Avg. Rec.	0.93	0.97
Avg. F1	0.91	0.97

M2H-Images dataset

Table 4: Results of the Experiments on document images

2.2 HDEF

A. Iyer et al [2] (2019) describes a Heterogeneous Data-Extraction Framework (HDEF) that uses a combination of machine learning and program synthesis [5] to extract semi-structured heterogeneous text data with many different formats. Let a heterogeneous dataset D be modeled as a tuple (D, $\{F_0, \ldots, F_n\}$) where D is a finite set of inputs and $\{F_0, \ldots, F_n\}$ is a partition of D into different formats such that $Fi \cap Fj = \varnothing$. Suppose there is a function f : D→T that maps inputs to values of type T that correspond to each of the partitions $\{F_0, \ldots, F_n\}$.

Using the *TrainML* function, a set of annotated inputs, where annotation(i) of an input $i \in D$ is a tuple(o,e) such that o are ground truth outputs whereas e provides auxiliary information (additional information that explain the output), is used to produce

an ML model *M* that yields the candidate output labels, including for input formats with no training data. The output for the initial ML model is usually noisy. The authors then define field constraints *Cf (i, o)*, where *Cf (i, o)* is true if and only if o = *f(i)*, that is if the extracted outputs match the specification of the format. In reality, this would not be feasible to write, thus soft constraints, which are noisy approximations of the actual constraints, are allowed.

The noisy ML output, a set of inputs *D,* is then fed to NoisyDisjSynth (NDSyn), a program synthesizer that generates a number of candidate disjunctive programs $P = (P_0, \ldots, P_n)$ for text extraction. The algorithm begins by repeatedly sampling small subsets (1-4 examples) of the ML output to get SubSpec= $\{(i_0, o_0), \ldots, (i_n, o_n)\}$ which becomes a hard specification for example-based synthesis engine to produce candidate program *P*. There are a number of restrictions when it comes to the program synthesis. If the sampled outputs are of different formats, the program will not be able to produce a candidate program. Besides, if the sampled outputs are inconsistent or incorrect, there is no generalization and the programs are not to produce a candidate program. Then, if none of the candidates' outputs satisfies a certain field constraint, sampling is guided to select inputs that would generate a program that produces a valid output that satisfies the said field constraint. Ideally, NDSyn program is able to maximize the number of programs *P* that satisfies the field constraints and to maximize the number of programs *P* in agreement with the ML model, but these can conflict, so the following scoring functions are introduced: (a) contraintScore(*P*): the number of inputs where *P'*s outputs satisfy constraints and (b) conformanceScore(*P*): the number of inputs where *P* conforms to ML model *M*. The two scores are then combined to get:

$$\text{Score} = w.\text{constraintScore} + (1-w).\text{conformanceScore}$$

where *w* is a weight factor that controls which of the two noisy artifacts (*M* and *Cf*) is given greater importance such that $0 <= w <= 1$. Intuitively, the greater the value of *w,*

greater importance is given to *M*; the smaller the value of *w,* tighter *Cf* would be of greater importance. The value of *w* can be determined by either human intervention or validation data. The threshold *t* where 0<=t<=1 is then introduced by the authors. It is not required to fix the value of *t* in advance, *t* could be given the value 1 and lowered subsequently if no disjunctive program is returned after sufficiently many iterations. Ultimately, the goal of the NDSyn procedure is to produce a program *P* such that Score(*P*) > t.*card*(*D*), where cardinality (number of component programs) is the smallest.

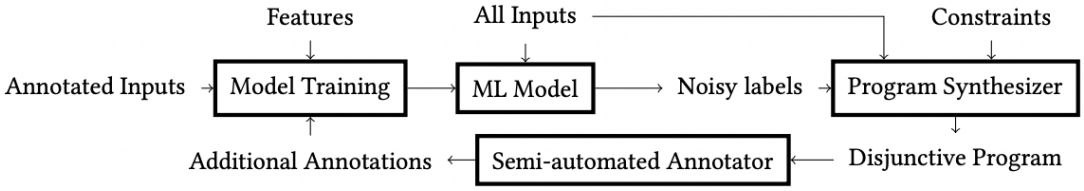


Figure 5: Outline of the HDEF procedure

To ensure robustness, the authors employ a feedback loop, through the function *ChooseInputs*, to iteratively improve the ML model and the synthesis output. When *M* and *P* agree on a label, the confidence value of the label is increased. When there are disagreements, the programs are ranked based on the number of inputs that are correctly processed, and if *M* disagrees with a highly-ranked program, the programs are declared the winners, and the outputs of these programs are used as training data to retrain the ML model. If *M* disagrees with a program that is not highly ranked, the authors resort to user intervention, where users will either accept or reject programs, or provide annotations to fix the errors. The three functions *TrainML, NDSyn* and *ChooseInputs* are repeated until the ML model and *P* agree. However, in practice, it would be more economical to fix a number of iterations (2-3 loops) or run until there is sufficient agreement between *M* and *P.*

The authors performed experiments on two datasets: (a) Political dataset with the goal of extracting the politician's names, birthdays and place of birth and (b) M2H flight emails dataset. A Conditional random field (CRF) model is used to train the political

dataset with 25 annotated inputs; an LSTM-CRF is used to train 570 annotated emails. On the political dataset, CRF training took 20 minutes, synthesis took less than 30 seconds; on the flight emails dataset, LSTM-CRF took 90 seconds and synthesis took less than 2 minutes. The results of the program were compared to the results of its corresponding ML models. Out of the 30 extraction tasks, 24 have seen performance improvements. The authors analyzed the fields that performed worse with HDEF and found that its deteriorating performance may be caused by some fields where the values are semantically distinct but have the same formats. For example, Booking Id (6 alpha-numeric characters) and ticketing Id (13 digits) may be conflated as the same format field. Furthermore, there were also fields where the ML model only produced a prefix of the true output, thus the HDEF only consistently amplified the error, which led to worse results. Thus, it can be concluded that HDEF does not work well with systematic noise.

The results of the experiment are as follows:

Domain Field	Iter	ML model			Synthesis		
		Pre.	Rec.	F1	Pre.	Rec.	F1
Political Father's Name	1	0.99	0.96	0.98	1.00	0.99	**0.99**
	2	0.99	0.96	0.98	1.00	0.99	**0.99**
Political Spouse's Name	1	1.00	0.91	0.95	0.99	0.95	**0.97**
	2	1.00	0.91	0.95	0.99	0.95	**0.97**
Political Date of Birth	1	0.99	0.81	0.89	1.00	1.00	**1.00**
	2	0.99	0.80	0.88	1.00	1.00	**1.00**
Political Place of Birth	1	0.99	0.77	0.87	0.99	0.71	0.83
	2	0.99	0.76	0.86	0.99	0.71	0.83

Table 5: Results of HDEF on Political Dataset

Domain Field	Iter	ML model			Synthesis		
		Pre.	Rec.	F1	Pre.	Rec.	F1
FlyFrontier AirportIata	1	0.06	0.04	0.05	0.55	0.39	**0.46**
	2	0.36	0.01	0.01	0.55	0.39	**0.46**
	3	0.30	0.36	0.33	0.37	0.99	**0.54**
FlyFrontier Airport	1	0.26	0.28	0.27	0.54	0.84	**0.66**
	2	0.37	0.69	0.48	0.43	0.95	**0.59**
	3	0.62	0.72	0.67	0.59	0.98	**0.73**
FlyFrontier FlightNumber	1	0.01	0.01	0.01	0.00	0.00	nan
	2	0.37	0.66	0.47	0.47	1.00	**0.64**
	3	0.47	1.00	0.64	0.47	1.00	**0.64**
FlyFrontier Name	1	0.46	0.61	0.52	0.44	1.00	**0.61**
	2	0.52	0.97	0.68	0.51	1.00	0.66
	3	0.54	0.89	0.67	0.51	1.00	0.67
FlyFrontier ReservationId	1	0.23	0.69	0.35	0.21	1.00	0.35
	2	0.81	0.87	0.84	1.00	1.00	**1.00**
	3	0.92	0.79	0.85	1.00	1.00	**1.00**
FlyFrontier Time	1	0.96	1.00	0.98	1.00	1.00	**1.00**
	2	0.96	1.00	0.98	1.00	1.00	**1.00**
	3	0.96	1.00	0.98	1.00	1.00	**1.00**
FlyFrontier Date	1	0.81	0.95	0.87	0.78	0.93	0.85
	2	0.52	0.96	0.67	0.70	0.93	**0.80**
	3	0.55	0.96	0.70	0.75	0.96	**0.84**
OmanAir FlightNumber	1	0.96	1.00	0.98	0.99	0.92	0.95
	2	0.99	0.98	0.98	0.99	0.92	0.95
	3	1.00	0.98	0.99	0.99	0.92	0.95
OmanAir Name	1	0.66	0.68	0.67	0.80	0.57	0.67
	2	0.56	0.41	0.47	0.86	0.64	**0.73**
	3	0.52	0.32	0.40	0.86	0.64	**0.73**
OmanAir ReservationId	1	0.69	0.91	0.78	0.62	1.00	0.77
	2	0.76	0.99	0.86	1.00	1.00	**1.00**
	3	0.93	0.99	0.96	1.00	1.00	**1.00**
OmanAir Time	1	1.00	1.00	1.00	1.00	0.82	0.90
	2	1.00	0.69	0.82	1.00	0.82	**0.90**
	3	1.00	0.66	0.80	1.00	0.82	**0.90**
OmanAir Date	1	0.60	0.63	0.61	0.47	0.48	0.47
	2	0.69	0.60	0.64	0.91	0.48	0.63
	3	0.73	0.60	0.66	1.00	0.48	0.65
Expedia AirportIata	1	0.98	0.93	0.95	1.00	0.97	**0.98**
	2	0.96	1.00	0.98	1.00	0.97	**0.98**
	3	0.97	1.00	0.98	1.00	0.97	**0.98**
Expedia Airport	1	0.78	0.34	0.47	0.99	0.99	**0.99**
	2	0.96	0.97	0.96	0.99	0.99	**0.99**
	3	0.93	0.99	0.96	0.99	0.99	**0.99**
Expedia FlightNumber	1	0.91	0.99	0.95	1.00	1.00	**1.00**
	2	0.97	0.97	0.97	1.00	1.00	**1.00**
	3	0.96	1.00	0.98	1.00	1.00	**1.00**
Expedia Name	1	0.45	0.99	0.62	0.59	0.99	**0.74**
	2	0.48	1.00	0.65	0.50	0.99	**0.66**
	3	0.39	1.00	0.56	0.50	0.99	**0.66**
Expedia ReservationId	1	0.85	0.83	0.84	0.98	1.00	**0.99**
	2	0.91	1.00	0.95	0.98	1.00	**0.99**
	3	0.79	1.00	0.88	0.98	1.00	**0.99**
Expedia Time	1	0.80	0.99	0.88	0.85	0.96	**0.90**
	2	0.77	1.00	0.87	0.85	0.96	**0.90**
	3	0.78	1.00	0.88	0.85	0.96	**0.90**
Expedia Date	1	0.45	1.00	0.62	0.98	1.00	**0.99**
	2	0.93	1.00	0.96	1.00	1.00	**1.00**
	3	0.91	1.00	0.95	1.00	1.00	**1.00**
Orbitz AirportIata	1	0.79	0.99	0.88	0.79	1.00	**0.88**
	2	0.93	1.00	0.96	0.93	0.99	**0.96**
	3	0.96	1.00	0.98	0.98	0.99	**0.98**
Orbitz Airport	1	0.82	0.61	0.70	0.91	1.00	**0.95**
	2	0.89	0.90	0.89	1.00	1.00	**1.00**
	3	0.86	0.94	0.90	1.00	1.00	**1.00**
Orbitz FlightNumber	1	0.94	0.94	0.94	0.77	0.92	0.84
	2	0.97	0.87	0.92	0.76	0.88	0.82
	3	0.96	0.87	0.91	0.76	0.87	0.81
Orbitz Name	1	0.51	0.93	0.66	0.70	0.94	**0.80**
	2	0.61	0.99	0.75	0.80	0.94	**0.86**
	3	0.55	0.97	0.70	0.84	0.87	**0.85**
Orbitz ReservationId	1	0.80	0.98	0.88	0.69	0.68	0.68
	2	0.79	0.97	0.87	0.78	0.91	0.84
	3	0.83	0.96	0.89	0.75	0.74	0.75
Orbitz Time	1	0.86	0.89	0.87	1.00	0.92	**0.96**
	2	0.94	0.99	0.96	1.00	0.99	**0.99**
	3	0.94	1.00	0.97	1.00	0.99	**0.99**
Orbitz Date	1	0.75	0.90	0.82	1.00	0.88	**0.94**
	2	0.88	0.97	0.92	1.00	0.97	**0.98**
	3	0.87	0.99	0.93	1.00	0.97	**0.98**

Table 6: Results of HDEF on Flight email dataset

2.3 Text Segmentation

In this section, we will discuss information extraction methods that utilizes the idea of text segmentation and topic boundaries. Text segmentation uses the fundamental notion that natural language texts are not a chaotic set of topics, but a topically structured entity – in real speech, a topic is introduced before moving to the next topic, which often, is related to the previous topic discussed. Thus, the topical continuity and shifts within

natural language texts might present a segment boundary that is able to separate semantically distinct parts of texts, making information extraction tasks more robust.

2.3.1 Attention-based CNN-BiLSTM

Badjatiya et al [3] (2018) introduces a neural framework – the Attention-based CNN-BiLSTM – to tackle the text segmentation problem. The authors modeled the text segmentation problem as binary classification problem: given a sentence s_i, with a left-context of size *K* and a right-context of size *K*, determine whether or not the s_i denotes the beginning of a new text segment. The neural framework used is based on the intuition that there are distinctive features to be learnt for sentences that mark the beginning of a segment, as well as the context surrounding said sentences.

There are two steps when it comes to data preparation: pre-processing and custom batch creation to incorporate neighboring context.. For data pre-processing, the length of sentences are first truncated or padded to *L* words to obtain an appropriate fixed length embedding. Basic text cleaning procedures such as removing punctuations and stopwords are then performed. The words are then embedded using word2vec that is trained on Google News. For words not in the word2vec vocabulary, lemmatization is used and the embedding of the lemmatized word is done. If the embedding is still missing, it is replaced with a token <UNK>. Let *V* be vocabulary, *d* be the word embedding size. Suppose we have a matrix n_i whose *j*th row corresponds to the one-hot representation of the *j*th word of the sentence s_i, thus, n_i has *L* rows and *V* columns. Then $E^{V \times d}$ is the embedding matrix, $\eta^{L \times V}$ is the one-hot encoding matrix of a given word with L rows and V columns and the sentence embedding matrix is thus $e(s_i)^{L \times d} = \eta_i^{L \times V} \times E^{V \times d}$.

The custom batch creation procedure is to incorporate neighboring context, since neighboring sentences may indicate whether a segment boundary is present. Given a sentence s_i (also known as the mid-sentence), let lc_i and rc_i be the one-hot representations

of the left context and the right context respectively, where each context contains $K \times L$ words. The embeddings are thus: $e(lc_i)^{K\times L\times d} = lc_i^{K\times L\times V} \times E^{V\times d}$ and $e(rc_i)^{K\times L\times d} = rc_i^{K\times L\times V} \times E^{V\times d}$. Thus, after this procedure, we get the following embeddings:

$$Si = [e(lc_i)^{K\times L\times d}, e(s_i)^{L\times d}, e(rc_i)^{K\times L\times d}]$$

The authors employed the CNN architecture to obtain rich feature representations for the left context, mid-sentence and the right context. For every sentence, 1D Convolution operations are performed with z filters and feature maps are obtained as follows:

$$f_{kl} = \varphi(\omega_l^{h\times d} \cdot S_{(i,j)}[k - h/2 : k + h/2]^{h\times d} + b_l)$$

$\omega_l^{h\times d}$ and b_l are weights, filters have dimensions $d\times h$ such that d is the input dimensionality and h represents height and *fk1* represents the result of the convolution using a non-linear transformation φ. The set *fk = {fk1, fk2, . . . , fkz}* is thus the feature vectors obtained. Then, the max-pooling operation is carried out for each feature vector *fk*, and the maximum feature value for each filter is derived – thus, the feature map is obtained.

After CNN transformations are completed, a stacked BiLSTM (Bidirectional LSTM) network, followed by soft attention, is used on a smaller sequence that consists of the main sentence and its neighbors to obtain a context representation with rich feature sets. A BiLSTM, where the embeddings are obtained through a forward pass LSTM and a reverse pass LSTM, is used over a regular LSTM network. Therefore, information is captured from both directions using a BiLSTM. The stacked BiLSTM captures information in a single fixed length representation, which has drawbacks as the earlier

hidden states have a significant effect on future states, thus attention would allow the model to give more importance to certain sets of sentences in the context while ignoring others, which in turn allows model to better predict whether a sentence is a segment boundary. The attention method begins with the introduction of an attention vector z_s, and $H_i^{K \times sz}$ such that $H_i = \{h_1, \ldots, h_K\}$, the output of the last BiLSTM layer, where sz is the size of the BiLSTM output corresponding to a sentence in the context. The context embedding for both the left and right context are calculated as below:

$$e_i^{K \times 1} = H_i^{K \times sz} \times W^{sz \times 1} + b_i^{K \times 1}$$

$$a_i = exp(tanh(e_i T z_s)),\ \alpha_i = \frac{ai}{\sum_p ap},\ v_i = \sum_{j=1}^{K} \alpha_j h_j$$

where v_i is the resultant context vector. The resultant embeddings obtained (for left context, right context and mid-sentence) will then be passed to a dense fully connected layer and softmax layer for classification. The predicted probability distribution is then passed to arg max to generate predictions.

The authors performed experiments on three datasets: (a) Medical textbooks, (b) Fiction books and (c) Wikipedia documents. The P_k and WinDiff metric, both loss measures, are used to evaluate the performance of the model. The lower the scores, the better. The model is compared to other competitive methods including *TSM, PLDA and BayesSeg*. The results are as follows:

	Model	Clinical		Fiction		Wikipedia	
		WinDiff	Pk	WinDiff	Pk	WinDiff	Pk
Part A: Competitive Baselines	U&I [25]	0.376	0.370	0.459	0.459	0.368	0.368
	MinCut [12]	0.382	0.368	0.405	0.371	0.389	0.364
	BayesSeg [4]	0.353	0.339	0.337	**0.278**	0.390	0.359
	APS [11]	0.399	0.396	0.480	0.451	0.380	0.392
	PLDA [18]	0.373	0.324	0.430	0.361	NA	NA
	TSM [2]	0.345	**0.306**	0.408	0.325	NA	NA
	Random	0.459	0.441	0.510	0.475	0.486	0.480
Part B: Neural Models without Context	Perceptron	0.338	0.357	0.336	0.335	0.421	0.415
	Stacked-LSTMs	0.381	0.393	0.329	0.394	0.437	0.420
	LSTMs	0.486	0.471	0.366	0.417	0.508	0.455
	CNN	0.309	0.329	0.314	0.386	0.363	0.380
Part C: Context based Neural Models	MeanBoW + BiLSTM	0.349	0.365	0.319	0.389	0.405	0.398
	TF-IDF MeanBoW + BiLSTM	0.345	0.366	0.328	0.382	0.382	0.392
	CNN + BiLSTM	0.334	0.316	0.331	0.324	0.378	**0.328**
	CNN + Attn-BiLSTM	**0.294**	0.318	**0.308**	0.378	**0.315**	0.344

Table 7: Table comparing the accuracy of the different methodologies. **Lower values are better.**

Overall, the attention-based CNN-BiLSTM model provides a performance improvement for all datasets on the Windiff method and compares well with the best methods on the P_k metric. Furthermore, the attention-based CNN-BiLSTM also has a fast runtime, averaging around 0.09 seconds compared to the other methods that usually takes minutes.

2.4 Forgiving XPaths

Omari et al (2017) [3] introduces a novel framework for the synthesis of robust data extractors for a family of websites from example annotated web pages. The method includes the generalization of XPath queries using decision trees and what the authors describe as *Forgiving XPaths*, which are XPath queries that adjust its extracting precision dynamically. The goal of the *Forgiving XPaths* is to extract information from a group of semantically similar sites, which could be, for example, book-selling sites, hotel reservation sites or flight-booking sites, since these sites may have similar patterns. For instance, book-selling sites would have HTML nodes containing information about the author, book prices and book name although having completely different HTML

structures. Thus, a generalized extractor could help immensely with expediting the process of aggregating information from many websites.

Let D be a set of web documents $\{d1, \ldots, dn\}$ such that each d is represented as a DOM tree with set A of nodes. A is thus the global set of DOM nodes for a webpage, and the target set of nodes of interest is represented by N where $N_i \subseteq A_i$ for each d, whereas $\overline{N}$ represents the remainder of nodes. Let set F be a set of features used to classify A into N and $\overline{N}$. We want to generate an XPath that extracts all N in d accurately. Consider the following HTML structures for book-selling websites:

```
Abe
<div id="bookInfo">
 <h1 id="book-title">Rachael Ray 30-Minute Meals 2</h1>
 <h2>
   <a userselected href="/servlet/Se.._-author">
     Ray, Rachael</a>
 </h2>...
</div>

B&N
<section id="prodSummary">
<h1 itemprop="name">History of Interior Design</h1>
 <span>
  by <a userselected href="/s/.._contributor..">
       Jeannie Ireland</a>
 </span>...
</section>

Alibris
<div class="product-title">
<h1 itemprop="name">La Baba del Caracol</h1>
 <h2>by
  <a userselected itemprop="author" itemscope href="..">
       C. Maillard</a>
 </h2>
 </div>
```

Figure 6: Fragments of web pages from three different book-selling sites.

Suppose we want to extract the author attribute of the book-selling sites' web pages, then we first annotate documents by tagging the HTML nodes containing instances of the wanted attribute, which in this case is the *author* attribute. This has already been done in Figure 6 with the "userselected" tag. The annotated documents are then used to learn a decision tree by extracting features that describe the HTML nodes. Feature extraction is done by traversing the paths from tagged nodes to its roots while collecting information about the nodes and its surrounding context. Thus, we would have a large set of structural features for all documents in *D,* of which only the features that are expressible by a valid XPath query would be kept. This means that each feature *f* is defined by an XPath predicate such that the value of *f* is 1 if the XPath predicate returns true on *N*. Given a feature set *f*, a variant of the ID3 algorithm is used to learn a decision tree that classifies *A* into *N* and $\overline{N}$. The tree is constructed by the variation of ID3 algorithm such that its 1-class is *N* and the 0-class is $\overline{N}$. Let *S(t)* be a subset of the samples, whose feature vectors match the path from the root to *t*. If all samples in *S(t)* have the same classification, then *t* becomes a leaf, otherwise it is partitioned based on some value of $f \in F$ and the node *t* is subsequently labeled *f*. The construction of the decision tree continues recursively with its children, where a subset of *S(t)* is a 1-child, denoted by $S(t)_f$, and the others a 0-child, denoted by $S(t)_{\bar{f}}$.

Just like the ID3 algorithm, the feature *f* with the highest information gain (metric that measures the reduction in uncertainty), *IG(S, f)*, is chosen to partition the nodes. The only difference between the Forgiving XPath's algorithm is that it takes into account feature costs and feature class correlation. Intuitively, the more specific the feature, the higher its cost, since the goal to obtain generalizes features over specific features. The information-gain-to-cost ratio of a feature *f* is as below:

$$IGR(S,f) = IG(S,f)/Cost(f)$$

where *Cost(f)* is the cost of feature $f \in F$.

Feature class correlation is guided by the intuition that the existence of features is important for being classified as part of the 1-class, and is thus obtained by restricting the features to split *S(t)* on such that there are more instances of $S(t)_f$ than $S(t)_{\bar{f}}$ in *N*.

Since generalized features are key in the Forgiving XPaths method, the decision tree is then pruned to limit node precision and increase recall. The precision of a node *t* is defined as:

$$precision(T,t) = \frac{|S(t) \cap N|}{|S(t)|}$$

Given a precision threshold α and a decision tree *T*, the authors use the method $T_{i+1}=prune(T_i, \alpha_i)$ for $0 \leq i \leq k$ to prune every subtree of *T* whose root *t* has a $precision(T,t) \geq \alpha$, and turns *t* into a leaf labeled by 1. Thus, the induced 1-class is increased. After the pruning process, we will be left with a sequence of decision trees $T_0, T_1, \ldots, T_k$ with a monotonically decreasing precision and increasing recall.

After the pruning procedure, the XPath predicate is obtained using the recursive function *GetXPath(t)*. The function works as follows:

1. If *t* is a leaf node, return true if the label is 1, else label it as false
2. If *t* is an inner node labeled by *f,* then *GetXPath(t)* is called on its 0-child, *t0* and 1-child, *t1*. Then, return *(f* and $GetXPath(t_1))$ *or (not(f)* and $GetXPath(t_0))$

Thus, an XPath predicate *p* is returned and the query //*[*p*] is constructed.

Experiments are performed on three different datasets, namely DS1, DS2 and DS3. DS1 contains pages from 30 real-life large scale websites divided into four different categories: books, shopping, hotels and movies; DS2 contains different versions (current and old versions) of pages from archive.org; DS3 contains 5025 product pages from five widely known webpages. For each dataset, around 2-3 common attributes are selected to

be extracted. The *Forgiving XPath method* was compared to other extraction methods, such as C4.5, naive bayes (NB), Non-Forgiving XPaths (NFX) (XPaths generated directly from decision trees) and alignment-based XPath (XA). In the experiment, negative values with smaller absolute value indicate higher robustness to structural differences in the training set, and is thus preferred.

The results of the experiment are shown in the table below:

	$s=1$			$s=2$			$s=3$			$s=4$			$s=5$			$s=6$		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1
Performance on Documents from Seen Sites																		
FX	0.86	0.93	0.87	0.86	0.94	0.87	0.87	0.93	0.88	0.86	0.91	0.86	0.85	0.90	0.86	0.85	0.89	0.85
NFX	0.86	0.90	0.87	0.83	0.87	0.84	0.84	0.87	0.85	0.84	0.85	0.83	0.83	0.85	0.83	0.84	0.84	0.82
C4.5	0.80	0.87	0.81	0.72	0.93	0.79	0.69	0.91	0.76	0.68	0.91	0.76	0.67	0.92	0.75	0.65	0.91	0.72
NB	0.55	0.92	0.60	0.55	0.93	0.64	0.45	0.93	0.56	0.39	0.92	0.49	0.34	0.92	0.45	0.29	0.92	0.40
XA	0.53	0.54	0.52	0.24	0.38	0.25	0.14	0.33	0.15	0.10	0.29	0.11	0.07	0.28	0.09	0.04	0.26	0.06
Performance on Documents from Unseen Sites																		
FX	0.31	0.66	0.35	0.48	0.75	0.53	0.56	0.79	0.60	0.60	0.79	0.62	0.60	0.77	0.61	0.63	0.78	0.62
NFX	0.30	0.31	0.29	0.36	0.38	0.36	0.39	0.38	0.36	0.41	0.40	0.38	0.42	0.40	0.38	0.43	0.41	0.39
C4.5	0.28	0.32	0.28	0.38	0.48	0.40	0.43	0.56	0.45	0.46	0.60	0.49	0.48	0.63	0.51	0.51	0.66	0.53
NB	0.01	0.01	0.01	0.26	0.35	0.27	0.31	0.49	0.34	0.34	0.56	0.37	0.34	0.62	0.38	0.32	0.65	0.35
XA	0.01	0.01	0.01	0.04	0.07	0.04	0.05	0.12	0.06	0.04	0.14	0.04	0.03	0.14	0.03	0.01	0.11	0.01

Table 8: Performance of data extractors on both seen and unseen sites

The results show that forgiving XPath queries beat the accuracy of other approaches for all the three different performance aspects.

3 References

[1] Suresh Parthasarathy, Lincy Pattanaik, Anirudh Khatry, Arun Iyer, Arjun Radhakrishna, Mohammad Raza and Sriram K. Rajamani. 2022. Landmarks and Regions: A Robust Approach to Data Extraction https://arxiv.org/pdf/2204.05021.pdf

[2] Suresh Parthasarathy, Lincy Pattanaik, Anirudh Khatry, Mohammad Raza. 2019. Synthesis and Machine Learning for Heterogeneous Extraction https://www.microsoft.com/en-us/research/uploads/prod/2019/04/HeterogeneousExtraction.pdf

[3] Adi Omari, Sharon Shoham, Eran Yahav. 2017. Synthesis of Forgiving Data Extractors. https://dl.acm.org/doi/10.1145/3018661.3018740

[4] Pinkesh Badjatiya, Litton J Kurisinkel, Manish Gupta, and Vasudeva Varma. 2018. Attention-based Neural Text Segmentation. https://arxiv.org/pdf/1808.09935.pdf

[5] Sumit Gulwani, Alex Polozov, Rishabh Singh. 2017. Program Synthesis. https://www.microsoft.com/en-us/research/publication/program-synthesis/#:~:text=Program%20synthesis%20is%20the%20task,holy%20grail%20of%20Computer%20Science.